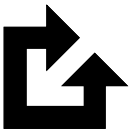
SQT1-12913-00



QPAC for Batch

Referenzhandbuch

Version 9 Release 13

Erste Ausgabe (Dezember 2025)

Diese Ausgabe bezieht sich auf die Version 9 Release 13 des lizenzierten Programms QPAC for Batch der Osys Software AG, Programmnummer 8050-QP-913-T10, und alle darauffolgenden Versionen, Releases und Modifikationen, bis anderweitig in neuen Ausgaben angegeben.
Konsultieren Sie die Osys Software AG für aktuelle Informationen über dieses Produkt.

Für Bestellungen, Kommentare und Anregungen betreffend dieser Produktliteratur wenden Sie sich bitte an die folgende Adresse:

Osys Software AG
Dammstrasse 19, CH-6300 Zug/Schweiz
E-Mail qpac@osys.ch

Spezielles

Die unten aufgeführten Ausdrücke, die in dieser Publikation verwendet werden, sind geschützte Warenzeichen und Schutzmarken der folgenden Unternehmen:

Osys Software AG, Schweiz:
Osys-QPAC

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

Inhaltsverzeichnis

Abbildungen

Kapitel 1. Einführungsteil

Systemüberblick

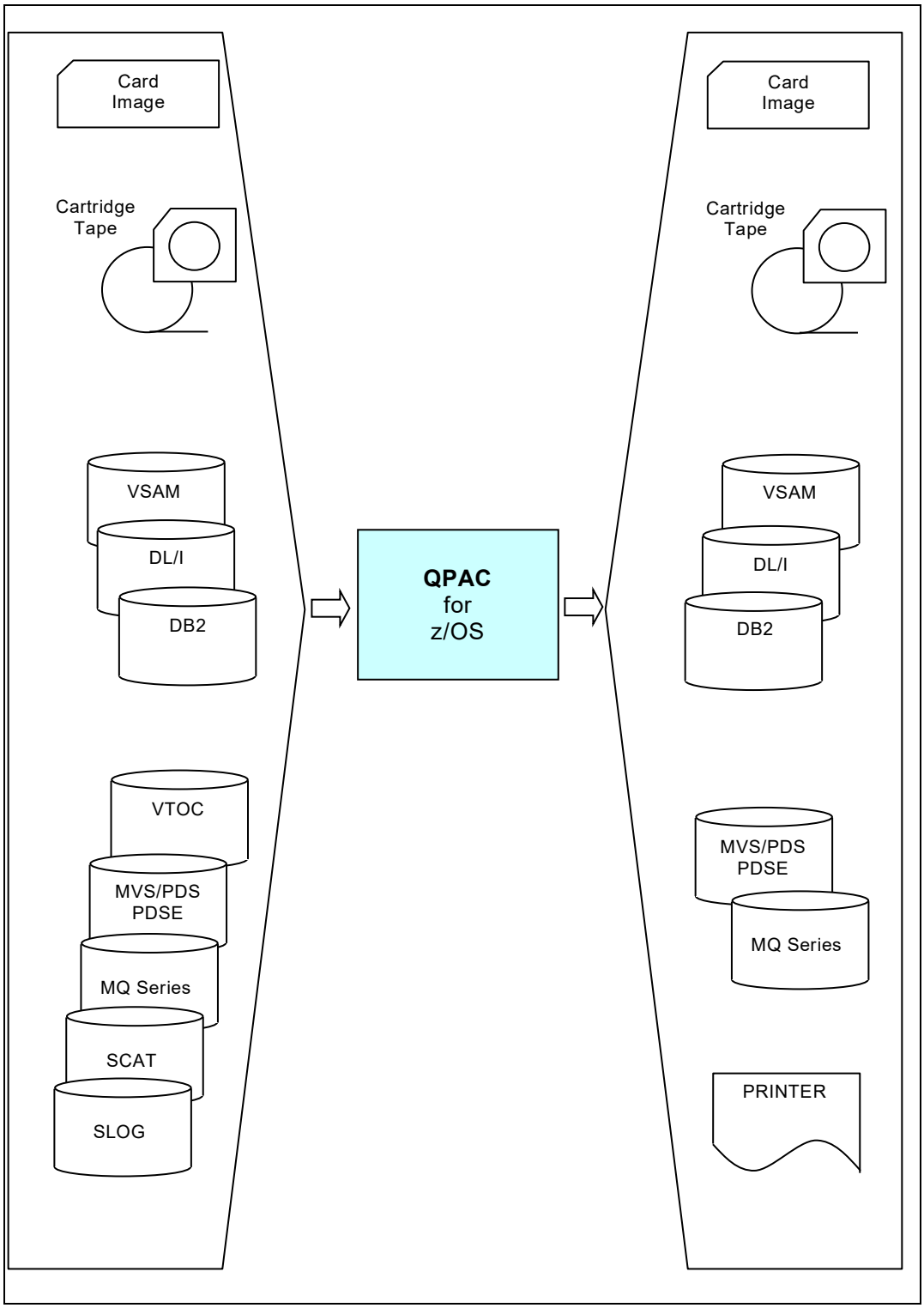


Abb. 1: QPAC-Batch Systemübersicht

Verwendungszweck von QPAC

QPAC ist ein Multi-Function Programmiersystem der 4. Generation und gestattet auf leicht anwendbare Art, irgendwelche Aufgaben im Zusammenhang mit den herkömmlich bekannten Fileorganisationen zu lösen, in einer logischen programmierbaren Form.

Insbesondere für folgende Aufgaben ist QPAC speziell (aber nicht ausschliesslich) geeignet:

- alle Arten von ad-hoc Aufgaben
- Reports erstellen
- Files erstellen, lesen, drucken
- Files analysieren, reorganisieren, modifizieren, korrigieren
- auf der Basis logischer Entscheide Record Bearbeitungen irgendwelcher Art vornehmen
- irgendwelche Verarbeitungen **ohne** File Hintergrund vornehmen.

Kurz gesagt: Es können die Probleme nach logischen Gesichtspunkten ohne aufwendige File Definitionen und ohne grosse Spezialkenntnisse frei programmiert werden.

Möglichkeiten-Spektrum

Alle IBM Standard-File Organisationen sind bearbeitbar:

- alle SAM-Organisationen auf Disk/Tape Input und Output mit fixer oder variabler Recordlänge, blockiert oder unblockiert
- alle VSAM-Organisationen Input und Output
- DL/I Datenbanken (DL/I Support Feature)
- DB2 Datenbanken (DB2 Support Feature)
- z/OS PDS/PDSE Systemlibraries
- z/OS System Catalog
- z/OS System Logger
- VTOC
- MQSeries
- ISPF/PDF-TSO
- WEB CICS Support

Das QPAC Programm

Das Programmier-System besteht aus mehreren Load-Modul Teilen.

Es wird unter dem Namen **QPAC** geführt und auch aufgerufen. Es belegt minimal eine Mainstorage Region von 1MB. Zu dieser Grösse kommen zusätzlich pro Filedefinition jeweils ca. 4K, plus deren I/O-Bereiche hinzu. Ebenfalls wird im freien Mainstoragebereich der gesamte Verarbeitungsbefehlsatz, der sich aus der Umsetzung der QPAC-Definitionen ergibt, aufgebaut.

Der Umfang der QPAC-Definitionen hängt folglich weitestgehend von der Mainstoragegrösse ab.
Wenn nicht mit Symbolen gearbeitet wird, ist die direkt in den I/O Areas adressierbare logische Recordlänge maximal 4096 Bytes. Der Record selber kann grösser sein. Die Blocklänge ist abhängig vom jeweiligen Device.

Unmittelbar nach dem Submit werden die QPAC Source Statements von QPACIN oder über das im EXEC-PARM definierte Eingabemedium eingelesen und danach vom QPAC Translator auf Syntax geprüft.
Werden keine Syntaxfehler gefunden, wird ein einmalig durchführbarer Maschinencode generiert und ausgeführt.

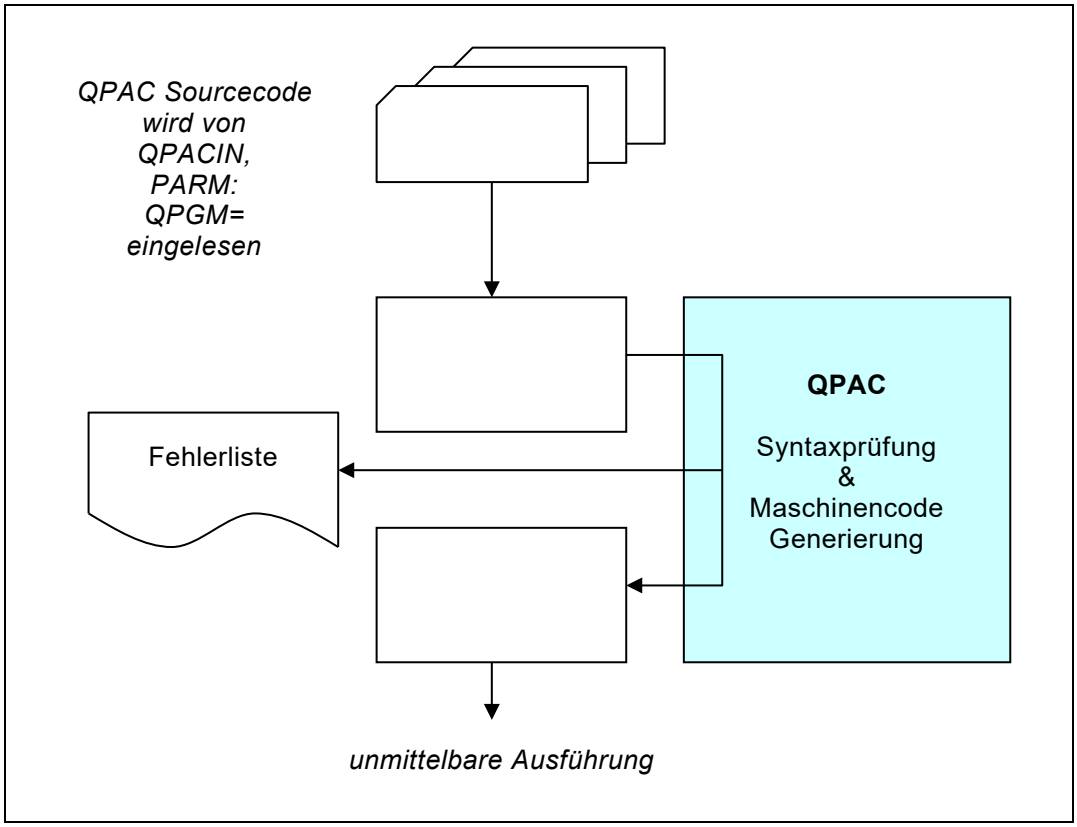


Abb. 2: Syntaxprüfung und Generierung des Maschinencodes

Job Control

Die QPAC Definitionen werden über das DD-Statement QPACIN (//QPACIN DD ∗) eingelesen.

Die Auflistung der eingelesenen Definitionen und eventuelle Fehlermeldungen erfolgen über das DD-Statement QPACLIST (//QPACLIST DD SYSOUT=A).

Die DD-Namen der zu verarbeitenden Input- bzw. Outputfiles sind im Normalfall die Fileidentifikationen selbst (//IPF DD .., //OPF DD .., //IPF1 ..etc.). Es können jedoch explizit in der Filedefinition andere Namen zugeordnet werden, z.B. `IPF=*FILEIN,SQ` (//FILEIN DD .... DCB=DSORG=PS).

Mit `PARM`-Options können Konsol-Loginformationen unterdrückt bzw. auf dem Printer über QPACLIST ausgegeben werden.

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

Abb. 3: QPAC PARM Option

Eine eventuelle Fehlermeldung wird in jedem Falle über die Konsole ausgegeben und kann nicht unterdrückt werden. Die Start- und die End-Message der Konsolausgabe lässt sich mit der Option `NOLOGTIT` unterdrücken.
Siehe hierzu unter „Das PARM Steuerstatement“.

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

Abb. 4: Beispiel JCL für z/OS

Die QPAC-Definitionen können auch direkt aus einem PDS eingelesen werden. Der DD-Name ist in diesem Falle QPACPGM. Der Membername wird über das EXEC-PARM definiert.

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

Abb. 5: Beispiel JCL für z/OS mit PARM QPGM=

Format der Definitionsstatements

Die QPAC-Definitionen können in einem absolut freien Format auf den Definitionsstatements plaziert werden, innerhalb der Kolonnen 1 bis 71. Die Kolonne 72 ist leer zu lassen. In 73 - 80 kann eine Identifikation bzw. eine Sequenznummer angegeben werden, die jedoch nicht weiter beachtet wird.

Eine Einzeldefinition muss immer gesamthaft auf einer Zeile stehen. Zwischen Einzeldefinitionen dürfen beliebig viele Leerstellen vorhanden sein. Innerhalb der Einzeldefinition gibt es logisch keine Blankstellen.

Ein Blank bedeutet Ende der Einzeldefinition.

Die Anzahl der Definitionsstatements ist nicht beschränkt. Ihr Inhalt wird in der vorhandenen prozeduralen Folge (Left - Right - Top - Down) abgearbeitet und für die Ausführung in ein Maschinenprogramm umgesetzt.

Kommentar

Ein Sternzeichen innerhalb eines Definitionsstatements sagt aus, dass die rechts davon liegenden Angaben als Kommentar zu verstehen sind.

An den Stellen, an denen das Sternzeichen mit dem Operationszeichen für Multiplikation (∗) verwechselt werden kann, gilt als Kommentar Stern-Nonblank (z.B. ∗.). **Ein Doppelstern (∗∗) hingegen ist nicht gestattet.**

Ein Sternzeichen in Kolonne 1 des Statements gilt immer als Kommentar.

Darstellung des QPAC-Listings

Mit einigen Steuerangaben kann das QPAC-Listing gesteuert werden. Diese beginnen immer ab Kolonne 1 mit einem Sternzeichen:

`*SK`	(Skip) Seitenwechsel
`*SP`	(Space) 1 Leerzeile
`*SP`*n*	(Space) *n* Leerzeilen
`*POFF`	(Print off) Liste unterdrücken
`*PON`	(Print on) Liste aktivieren

Abb. 6: Steuerzeichen für das QPAC Listing

Das PARM Steuerstatement

Als erste QPAC Definition kann ein Statement definiert werden, mit dessen Hilfe verschiedene Options zusätzlich und teilweise unabhängig einer äquivalenten Job-Control Definition angegeben werden können.

Dieses Statement hat ein festes Format, d.h. Kol.1-5 müssen fix **PARM=** enthalten. Anschliessend können in wahlweiser Folge, jeweils durch Komma getrennt, die Options definiert werden.

Dieses Statement ist z.B. im Zusammenhang mit DB-Anwendungen in z/OS Systemen unerlässlich, da dort keine EXEC-Parameter Angaben möglich sind.

Format

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

Die folgenden PARM Options stehen zur Verfügung:

CALL=SUB MAIN	COBOL oder PL/I Programme werden intern via LE (Language Environment) aufgerufen. Dabei gibt es zwei Initialisierungsfunktionen SUB Routine und MAIN Programm. Per default wird SUB angenommen. Die Unterschiede sind beim CALL Command detaillierter beschrieben.
COPYL | NOCOPYL	Auflisten der Copy-Books.
DUMP	Im Falle eines abnormalen Endes wird zusätzlich ein Systemdump ausgegeben (nötig bei unerklärlichen Situationen).
EPARM='....'	Systemparameter Übergabe. Die in Apostroph definierten Angaben können als Parameter einem Programm, das durch QPAC mit CALL-'*Programm*',*parm* abgerufen wird, weitergegeben, oder mit dem reservierten Symbol EPARM in den internen Workbereich übernommen werden. Das reservierte Symbol EPARM wird in der Länge des definierten Wertes angelegt. Im Feld EPARML steht zusätzlich die Länge des Wertes.
HSPACE=*nn*M	Grösse des Hiperspace. Dieser Bereich wird in Megabytes definiert. Die maximale Grösse wird durch die Region Size bestimmt. Für die Adressierung stehen implizite Positionssymbole HPOS*nnnn* zur Verfügung. Ebenso können Indexregister oder BASED Strukturen für die Adressierung verwendet werden. Individuelle Feldsymbole können diesem Bereich zugeordnet werden, indem der Buchstabe„H“ zwischen der Positionsangabe und dem Gleichheitszeichen eingefügt wird: z.B. 100H=FELDSYMBOL,CL20.
LCT=*nn* 60 (z/OS)	Linecounter (Seitenlänge). Überschreibt temporär den durch die Job-Control

gesetzten Wert.

LIST | NOLIST	Drucken oder Unterdrücken eines QPAC Listings. Überschreibt temporär den durch die Job-Control gesetzten Wert.
LISTL=*nnn*	Zeilenbreite beim Systemprinter. Der über PUTLST (QPACLIST) ausgegebene Printeroutput (z.B. ab WPOS5200 des internen Workbereiches) besitzt standardmässig eine Zeilenbreite von 121 Positionen (inkl. Controlcharacter). Mit dieser PARM Definition kann die Zeilenbreite bis zu 250 Bytes vergrössert werden. Dies ist im Zusammenhang mit Laserprintern von Bedeutung.
LOG | NOLOG	Drucken oder Unterdrücken von Log-Informationen. Überschreibt temporär den durch die Job-Control gesetzten Wert.
LOGTIT |NOLOGTIT	Start- und die Endmessages über das System-Log werden unterdrückt oder angezeigt.
PLIST | NOPLIST | NOPLIST=SAVE	Die Programm-Statements werden aufgelistet oder unterdrückt. Dadurch kann beispielsweise nur die Statistik ausgegeben werden. NOPLIST forciert NOXREF. Mit NOPLIST=SAVE werden die Programm Statements nur im Fehlerfalle aufgelistet. Da diese Programm Statements intern oberhalb der 2 GB Grenze zwischengespeichert werden, muss eine MEMLIMIT Grösse von 1 MB definiert werden.
QMOD=*loadmodul*	Loadmodulname des QPAC Programmcodes. Dadurch wird kein QPACIN benötigt.
QPGM=*member*	PDS-Member, das direkt über das DD-Statement //QPACPGM eingelesen wird. Die QPACPGM Datasets mit Recordlänge 80 können konkateniert werden. Ein //QPACIN ist hierbei nicht mehr notwendig.
SYNTAX	Es wird nur eine Syntaxprüfung der QPAC-Definitionen gewünscht. Es erfolgt keine Ausführung.
STRUCT	Es wird nur ein Strukturtest durchgeführt. Es erfolgt keine Ausführung.
TRACE=ON OFF	Wird TRACE=ON definiert, gibt QPAC bei einem abnormalen Ende die zuletzt durchlaufenen Statement-Folgen aus.
WORK=*nnnnn* WORK=*nn*M	Die Grösse des internen Workbereiches kann mit dieser Definition erweitert werden. Der Wert darf nicht kleiner als 12288 sein, die obere Limite ist durch den Main-Storage begrenzt, maximal aber 16 MB. Die definierte Grösse ist die Anzahl Bytes und bestimmt die Positionsadressen des internen Workbereiches. WORK=80000 ergibt Positionsadressen für den Workbereich von 1 - 80000. Für die Adressierung stehen implizite Positionssymbole WPOS*nnnn* zur Verfügung. Ebenso können Indexregister oder BASED Strukturen für die Adressierung verwendet werden. Individuelle Feldsymbole können diesem Bereich zugeordnet werden, indem der Buchstabe „W“ zwischen

	der Positionsangabe und dem Gleichheitszeichen eingefügt wird: z.B. `100W=FELDSYMBOL,CL20.` Die Positionen 1-4999 sind nur mit dem Buchstaben „`W`“ adressierbar. `WORK=` Definitionen mit einer Grösse bis 1 MB werden unterhalb der 16 MB Linie angelegt. Grössere Workbereiche kommen oberhalb zu liegen.
`XREF | NOXREF | FXREF`	Cross Reference Liste ja, nein oder „full“ (auch die nicht angesprochenen Symbole werden aufgelistet).
`NOLSR | LSR`	Defaultmässig soll bei VSAM Filedefinitionen der Operand `LSR` resp. `NOLSR` (default) angenommen werden.
`PLAN=`*`qpacplan`*	Siehe unter Kapitel 12: DB2 Feature
`DB2ID=`*`sysid`*	Siehe unter Kapitel 12: DB2 Feature
`GROUPID=`*`xx`*	Ein bis zu zweistelliger Begriff im EBCDIC Format (d.h. auch multipunch) kann definiert werden. Blank bzw. Komma gelten als Delimiter.
`PASSWORD=`*`xxxxxx`*	Ein bis zu sechsstelliger Begriff im EBCDIC Format kann definiert werden. Blank bzw. Komma gelten als Delimiter.
`USERID=`*`xxxx`*	Ein bis zu vierstelliger Begriff im EBCDIC Format kann definiert werden. Blank bzw. Komma gelten als Delimiter.
`USERID=`*`xxxx`*	Ein bis zu vierstelliger Begriff im EBCDIC Format kann definiert werden. Blank bzw. Komma gelten als Delimiter.
`STAB | NOSTAB`	Set Abend Option Damit werden Abends, welche nicht durch einen P-Check hervorgerufen worden sind, abgefangen und dokumentiert über QPACLIST ausgegeben.

PARM Option User Defaults

`PARM` Options, welche standardmässig verwendet werden sollen, können als Default in einem Load Module/Phase mit dem Namen `QPACBOPT` abgespeichert werden.

QPAC prüft und verarbeitet `PARM` Options in der folgenden Reihenfolge:

1. `PARM` Options im Load Module/Phase `QPACBOPT` sofern vorhanden
2. `PARM` Options aus dem EXEC-Parm
3. `PARM` Options am Anfang des Programm Codes

Das Load Module/Phase hat das nachfolgend aufgeführte Assembler Format:

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

Abb. 7: Format PARM Defaults Load Module QPACBOPT

COPY Statement (IBM z/OS)

Bei der z/OS-Version wird die Source Statement Library über das DD-Statement QPACCOPY angesprochen.

membername ist ein 1-8 stelliger Name.

Im aufgerufenen Member können alle Definitionsarten vorhanden sein, ausgenommen weitere `COPYs`.

```
COPY-membername
```

Abb. 8: Format (z/OS) COPY Statement

Loadmodul-Code Modus

Diese Erweiterung erlaubt dem Anwender, seine QPAC Definitionen als Loadmodul zu laden. Dadurch können z.B. sich nicht ändernde QPAC Programme ohne QPACIN-Definition im z/OS durchgeführt werden.

Die ersten 2 Stellen des Namens müssen 'QP' enthalten.

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

Abb. 9: Abruf unter z/OS

Der Loadmodul-Code beinhaltet selbst ausschliesslich nur QPAC Definitionen. Er ist entsprechend als Konstantencode umgewandelt und gelinkt worden, wie nachfolgend dargestellt ist:

```
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
/*
```

Abb. 10: QPAC Loadmodul-Code

Es ist möglich, als erste Angaben im QPAC-Code auch `PARM` Options zu definieren.

Datenschutz

QPAC stellt dem Benutzer einen USER-EXIT zur Verfügung, in dem er nach seinen eigenen Bestimmungen die **Zugriffserlaubnis** zu seinen schützenswerten Datenbeständen festlegen kann. Beim USER-EXIT handelt es sich um die Phase bzw. das Loadmodul mit dem Namen **QPACUSER**. In diesen EXIT verzweigt QPAC bei einer Filedefinition für Disk bzw. Tape mit Label und übergibt den File-ID zur Prüfung. Zusätzlich können über `PARM`-Options bis zu drei Kontrollangaben definiert werden, die ein dreistufiges Schutzschlüssel- bzw. Passwortkonzept zulassen:

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

Die Handhabung der Schutzschlüssel-Informationen, die Codierung der Prüfungen der Fileidentifikationen (Filenamen im DLBL-Statement bzw. Datasetnamen im DD-Statement), ist vollständig Angelegenheit des Benutzers; QPAC stellt ihm nur die Informationen, sofern vorhanden, zur Verfügung.

Der EXIT ist nach den offiziellen Linkage-Konventionen zu codieren, d.h. nach dem Prinzip von CALL-SAVE-RETURN.

Als RETURN Code (R15) muss dem QPAC der Wert „0“ zurückgegeben werden, wenn der File zugriffsberechtigt ist, bzw. der Wert „4“ wenn er es nicht ist.

```
QPACUSER  START 0
          USING *,15
          USING DEVDSECT,2
*
          DC    XL256'00'           MUSS X'00'SEIN
*
ENTRY1    B     GOENTRY1            POSITION X'100'
          B     *                   RESERVIERT
          B     *                   RESERVIERT
          B     *                   RESERVIERT
*
GOENTRY1  SAVE  (14,12)
          L     2,0(1)              LADEN DSECT ADRESSE
          ST    13,SAVEAREA+4
          . (USER CODIERUNG)
*
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               KONTROLLFELDER
DEVUSGID  DS    CL2                 —GROUPID  OR X'00'
DEVUSUID  DS    CL4                 —USERID   OR X'00'
DEVUSPSW  DS    CL6                 —PASSWORD OR X'00'
          DS    CL4                 RESERVIERT
DEVUSAR2  DS    CL3                 FILE ID DER QPAC DEF
DEVUSAR3  DS    CL2                 FILE ORG DER QPAC DEF
DEVUSAR4  DS    CL3                 RESERVIERT
DEVUSAR5  DS    CL8                 DDNAME
DEVUSAR6  DS    CL44                DSN
DEVUSAR7  DS    CL24                RESERVIERT
*
          END
```

Abb. 11: Grundaufbau der Routine QPACUSER

Kapitel 2. Input/Output Definitionen

File Definitionen (Fixe Recordlängen IBM z/OS)

Grundformat der Filedefinitionen

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

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

IPF	=	Inputfile Definition implizite Form
IPF*n*	=	Inputfile Definition explizite Form
UPF	=	Updatefile Definition implizite Form
UPF*n*	=	Updatefile Definition explizite Form
OPF	=	Outputfile Definition implizite Form
OPF*n*	=	Outputfile Definition explizite Form
**DDname*		explizit definierter DD-Name; fehlt diese Angabe, wird IPF/OPF/UPF als entsprechender DD-Name angenommen
og	=	Organisations-Definition
rl	=	Recordlänge (als Positional Operand)
bl	=	Blocklänge (als Positional Operand)
opt	=	Options

Zusätzliche Parameter für Filedefinitionen (Fixe Recordlängen)

File Organisationen

og	SQ SAM	Sequenziell allgemein (Device-unabhängig). Bei dieser Definition wird nur durch die zugehörige SYS-Nummer, bzw. das DD-Statement bestimmt, welchem Speichermedium der File zugeordnet ist. Es wird kein bestimmtes Speichermedium vorausgesetzt. Diese Definition ist für eine Printerfile nicht zulässig, da SQ keine zusätzlichen Steuerungsfunktionen wie z.B. Seitenwechsel unterstützt.
	SD DISK	Sequenzielles Diskfile (Typ unabhängig). Diese Definition setzt eine Disk als Speichermedium voraus, andernfalls erfolgt eine Syntax-Error Meldung.
	MT TAPE	Sequenzielles Magnetic-Tape-File. Diese Definition setzt eine Tape-Unit als Speichermedium voraus, andernfalls erfolgt eine Syntax-Error Meldung.
	PR PRINT	Printeroutput (Typ unabhängig). Diese Definition setzt einen Printer als Ausgabemedium voraus. Sie enthält zusätzliche Steuerungsfunktionen bezüglich Zeilen- und Seitenvorschub, wie auch hinsichtlich Titelzeilen.
	CD CARD	Sequenzielles Cardfile (Typ unabhängig). Diese Definition setzt einen realen/virtuellen Kartenleser/Kartenstanzer als Speichermedium voraus.
	VS VSAM KSDS ESDS RRDS	VSAM (Virtual Storage Access Method) wird als Zugriffsmethode angenommen.
	PDS PDSE	Partitioned Data Sets (siehe Kapitel 10)
	VTOC	VTOC Format 1 Data (siehe Kapitel 10)
	SCAT	System oder User Katalog (siehe Kapitel 10)
	SLOG	System Logger (siehe Kapitel 10)

Options für Allgemeine Definitionen

opt Verschiedene Options dienen ergänzenden Angaben zu den Filedefinitionen. Es sind **keine Positional-Operanden**. Ihre Reihenfolge ist daher frei. Sie werden an den zuletzt definierten Positional-Operanden angeschlossen.

z.B.
```
IPF=TAPE,RL=120,BL=2400,WP=5001
```

`BL=`
`BLKSIZE=` Blocklänge (wird unter z/OS ignoriert).
Muss bei Output Files immer übers DD Statement definiert werden.

`RL=`
`LRECL=` Recordlänge.
Damit wird eine interne Buffergrösse definiert (Default 32k).

`CLR=C'`*x*`'`
`CLR=X'`*xx*`'`
`CLR=NO` Outputarea Clearcharacter
Die Outputarea jedes Outputfiles wird standardmässig jeweils auf X'00' gelöscht, mit Ausnahme von `CD` und `PR` Definitionen, bei welchen auf X'40' gelöscht wird.
Mit `CLR=` kann ein anderer Löschwert definiert werden.
`CLR=NO` heisst: nicht löschen.

`CLE=C'`*x*`'`
`CLE=X'`*xx*`'` Inputarea Clearcharacter zur EOF Zeit.
Die Inputarea wird jeweils bei EOF standardmässig auf X'FF' gelöscht. Mit `CLE=` kann ein anderer Löschwert definiert werden, dessen Inhalt die Area bei EOF-Status dann aufweist.

`DESC=''` File-Beschreibung
von maximal 44 Bytes, welche der File Definition zugeordnet werden kann. Sie wird bei Konsol-Meldungen und File-Statistiken mit ausgegeben.

`DSN=` Data Set Name (z/OS)
kann direkt in der File Definition angegeben werden.
Ein entsprechendes DD Statement wird dann überflüssig.

z.B.
```
IPF=SAM,DSN=TEST.FILE
```

`WP=` Workbereich Position
Mit dieser Definition wird bestimmt, dass der Record in den allgemeinen Workbereich geschrieben, bzw. aus demselben gelesen werden soll. Die Definition bezieht sich auf den Platz innerhalb des Workbereichs, der für den zu bearbeitenden Record belegt wird.
Wird `WP=` definiert, existiert für die entsprechende Filedefinition **kein dynamischer Recordbereich**. Nebst den impliziten Positionssymbolen (`IPOS`*n*`, OPOS`*n* ...) kann zur Adressierung auch `WPOS`*nnnn* verwendet werden.

`FCA=` File Communication Area
FCA definiert den Platz innerhalb des allgemeinen Workbereiches, wo sich die Informationsaustauscharea befinden soll. Die FCA dient dazu z.B. Recordlängen mitzuteilen, aber auch Keywerte, wenn beispielsweise mit der 'Set Generic Key' Funktion gearbeitet werden soll.
Ist die FCA nicht definiert, wird sie defaultmässig dynamisch angelegt und kann nur über die zugeordneten Symbolnamen angesprochen werden.
Die FCA hat eine Länge von 256 Bytes.

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

Katalogisierte COBOL- und PL/I-Recordstrukturen können mit dieser Option aus einer Sourcelibrary geladen und einer Filedefinition zugeordnet werden.
Die Feldnamen werden zu QPAC Symbolnamen konvertiert (- Zeichen werden zu _ Zeichen konvertiert).
Initialwerte und Editmasken werden ignoriert.
Mehrere Strukturnamen können durch / getrennt angegeben werden. Wenn nicht alle Namen auf demselben Statement Platz finden, ist es möglich, die Definition mittels Schrägstrich-Blank (in der laufenden Zeile) auf dem Folgestatement fortzusetzen. Der nächste Strukturname wird danach auf dem folgenden Statement definiert. Beliebige führende Blankstellen sind möglich.
Wird zusätzlich `PRFX=YES` definiert, wird allen Symbolnamen die Fileidentifikation (Kurzform) vorangestellt.
Based-Definitionen werden als QPAC-Based-Strukturen umgesetzt. Der Based-Pointer muss dabei im QPAC-Programm mit der richtigen Adresse geladen werden. Im z/OS werden diese Copy Books über den PDS DD-Name QPACCOPY (//QPACCOPY DD ...) eingelesen.

Options für Tapefile Definitionen

`opt` `BWD` Tape Files mit fixer Recordlänge oder „undefined“ rückwärts lesen

Options für VSAM-File Definitionen

opt NRS — No Reset to Empty State
Standardmässig wird ein Outputfile mit dem RST Attribut behandelt, d.h. der File wird als leer betrachtet und neu erstellt.

Wenn der Outputfile (OPF) dagegen als nicht leer zu behandeln sein soll, d.h. die neuen Outputrecords werden an den bestehenden Bestand angehängt, kann NRS definiert werden.

ESD — Entry Sequenced Dataset
Standardmässig findet QPAC selbständig heraus, ob es sich um ESDS, RRDS oder KSDS handelt. Dabei tritt eine Systemmeldung ERROR X'A0' auf, wenn als File Organisation nur VSAM angegeben wurde. Diese kann jedoch ignoriert werden. Um diese Meldung zu unterdrücken, kann bei einem ESDS obige Option explizit definiert werden.

LSR
NOLSR — Local Shared Resource
VSAM soll einen oder keinen Local Shared Resource Pool verwenden.

RLS — Der VSAM File wird als Record Level Shared klassiert.

FCA= — File Communication Area
Die FCA dient dazu, zwischen Anwender und QPAC-VSAM notwendige Informationen auszutauschen. Das kann sein: Return Codes, Recordlänge, Key.

Mit FCA= wird definiert, an welcher Stelle innerhalb des allgemeinen Workbereiches sich die Informationsaustauscharea befinden soll.
Fehlt diese Definition, so wird die FCA dynamisch angelegt und kann nur über die zugeordneten Symbolnamen angesprochen werden.
Die FCA hat eine Länge von 256 Bytes.

PSW= — Passwort
Diese Option erlaubt das Definieren eines Passwortes, falls das Dataset passwortgeschützt ist. Dieses Passwort kann 1 bis 8 alphanumerische Zeichen umfassen und erscheint auf dem QPAC-Listing nicht.

RC=YES — Returncode/Feedbackcode
Der VSAM Return- und Feedbackcode wird in der FCA zurückgegeben, ohne dass im Fehlerfalle die QPAC-Verarbeitung abbricht. Bei den Random Befehlen (READ, READGE etc.) ist diese Option defaultmässig gesetzt.
Im FCA-Feld RC1 steht der binäre Returncode, im FCA-Feld RC2 steht der binäre Feedbackcode. Ist kein Fehler aufgetreten, steht im FCA-Feld RC X'0000'.
Bei EOF wird X'0804' zurückgegeben.
Nach SETGK wird immer, wenn vorhanden, X'0804' zurückgegeben.
Nach SETEK wird immer, wenn vorhanden, X'0810' zurückgegeben.

BWD — Lesen rückwärts
Das VSAM File wird rückwärts gelesen. Diese Option ist für ESDS mit fixer Recordlänge und KSDS gültig.

Options für Printerfile Definitionen

opt RL=
LRECL=
Recordlänge
Für Laserprinter kann durch die explizite Definition der Recordlänge die Zeilenbreite bis 500 Bytes vergrössert werden. Defaultmässig werden 132 Bytes angenommen.

LCT=
Linecounter
Damit kann für die entsprechende Printerfiledefinition ein vom Systemlinecounter unabhängiger Zeilenzähler definiert werden. Dieser gilt nur für den zugehörigen Printfile.

IPC
Ignore Page Control bei Printeroutput
Die automatische Seiten-Vorschubsteuerung aufgrund des Linecounters wird mit dieser Definition unterdrückt. Dadurch werden auch keine Titelzeilen auf Grund der HDR= oder TITLE Subroutine geschrieben.

z.B.

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

ASA
ASA Control Character for Printoutput.
Für die Printausgabe wird der ASA Control Characterset verwendet.

CCH
Control Character.
Die erste Position der Printzeile enthält den Control Character. Wenn zusätzlich ASA definiert ist, wird er im ASA-Format erwartet, andernfalls im Maschinen-Format.

CLASS=
Die SYSOUT CLASS kann hier innerhalb der Filedefinition angegeben werden. Ein entsprechendes DD SYSOUT Statement wird dann überflüssig.

z.B.

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

File Definitionen (Variable Recordlängen)

Wir unterscheiden hier zwischen drei Filedefinitionsarten:

- Standard Variabel Modus blockiert/unblockiert
- Variabel Spanned Modus blockiert/unblockiert
- Undefinierter Modus nur unblockiert

Grundformat der Filedefinitionen

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

Als Organisations-Definition können die sequentiellen Files auf Magnetic-Tape oder Disk definiert werden:

SQ-*Typ* SAM-*Typ*	Sequentiell, Device unabhängig
SD-*Typ* DISK-*Typ*	Sequentiell, Disk
MT-*Typ* TAPE-*Typ*	Sequentiell, Tape
PR-*Typ* PRINT-*Typ*	Printer (nur *og*-VAR möglich)
PDS-*Typ*	Partitioned Data Set (z/OS)

Die Unterscheidung der drei Organisationsarten erfolgt durch eine Zusatzdefinition:

og-VAR	Standard Variabel Modus. Der Record ist nie grösser als der Block.
og-SPN	Variabel Spanned Modus. Der logische Record kann grösser sein als der Block, sich über mehr als einen Block erstrecken. Gilt nicht für PDS.
og-UND	Undefinierter Modus. Der physische Record wird gelesen und verfügbar gemacht.

Zusätzliche Parameter für File Definitionen (Variable Längen)

Variable Recordlängen und Blocklängen

rl

Recordlänge:
Darunter ist die grösstmögliche logische Recordlänge zu verstehen. Bei VAR und bei SPN beinhalten die ersten 4 Bytes des Records stets das Recordbeschreibungswort, gemäss Standard-Konvention:

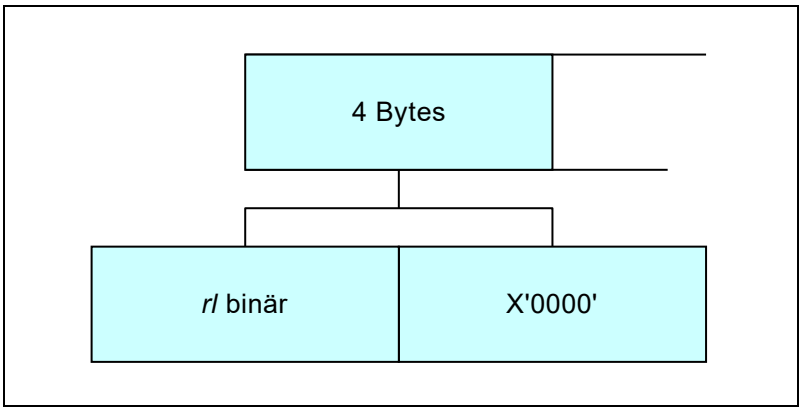


Abb. 12: Die ersten 4 Bytes: Das Recordbeschreibungswort

Die ersten 2 Bytes sind die Recordlänge in binärer Form und die zweiten zwei Bytes werden vom IOCS benötigt.
Die Recordlängenangabe muss diese 4 Bytes beinhalten.

bl

Blocklänge:
Darunter ist die grösstmögliche Blocklänge (physischer Record) zu verstehen. Im UNDefinierten Modus wird die Blocklänge ignoriert.
Bei SPN-Output ist es die Grösse der zu schreibenden physischen Records.

Ist bei VAR die Blocklänge gleich oder nicht mehr als maximal 4 Bytes grösser als die Recordlänge, wird sie immer auf 4 Bytes grösser der Recordlänge aufgerundet. In diesem Falle gilt der File als unblockiert. SPN kennt keine Unterscheidung zwischen blockiert oder unblockiert. UNDefiniert kennt nur unblockiert.

Options mit Spezieller Bedeutung für Records mit Undefinierter Länge

opt

Zu den unter fixen Recordlängen beschriebenen Options ist im Zusammenhang mit UNDefiniertem Modus die FCA (File Communication Area) noch speziell von Bedeutung.

FCA=

Die FCA dient dazu, zwischen Anwender und QPAC die jeweilige aktuelle Recordlänge auszutauschen. Mit FCA= wird definiert, an welcher Stelle innerhalb des allgemeinen QPAC Workbereiches sich die Informationsaustauscharea für die zugehörige Filedefinition befinden soll. **Fehlt diese Definition, so wird die FCA dynamisch angelegt und kann nur über die zugeordneten Symbolnamen angesprochen werden.**

z.B.

```
OPF=SQ-UND,RL=4096,FCA=8000
```

Bei Output muss vor dem Schreiben eines Records ins Feld ..LENG auf Displacement 12 der FCA (4 Bytes) die jeweilige Recordlänge in binärer Form gegeben werden.
Bei Input steht im Feld ..LENG die jeweilige Recordlänge in binärer Form des eben zur Verfügung gestellten Records.
(Für VAR und SPN wird **keine** FCA benötigt, da hier die Recordlänge Bestandteil des logischen Records selbst ist).

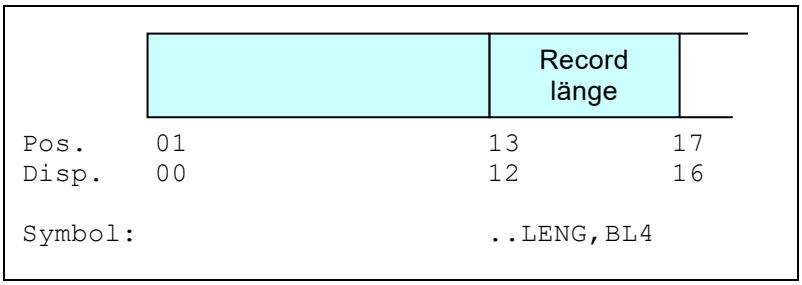


Abb. 13: File Communication Area (FCA)

Dem Längenfeld in der FCA ist das Symbol ..LENG zugeordnet.

Eine Updatefile-Definition entspricht syntaktisch der Kombination einer Input- und einer Output- Definition. Eine File-ID Nummer, die in diesem Zusammenhang vergeben wird, kann folglich sonst bei keiner anderen Filedefinition mehr vorkommen.

Ein DUMMY DD-Statement in der z/OS Job-Control wird von QPAC akzeptiert. Es ist jedoch auch möglich, in der QPAC Definition ein DUMMY zu definieren:

z.B.

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

Dies bewirkt, dass der logische Steuerungsmechanismus normal aufgebaut wird, wie wenn ein physikalischer File vorhanden wäre. Das `IPF=DUMMY` wirkt sich jedoch nicht gleich aus wie in der z/OS Job-Control; `IPF=DUMMY` ergibt nie von sich aus EOF-Status. EOF-Status kann erreicht werden, indem ein zugehöriger `LIPF=` definiert, oder ein `CLOSE-I` gegeben wird. Die adressierte Inputarea ist danach auf Basis-Wert X'FF' bzw. `CLE=` gelöscht.

Bei z/OS-Installationen können Recordlänge und Blocklänge in der QPAC-Filedefinition angegeben werden, die Blocklänge wird jedoch ignoriert. Die Recordlänge wird für die Grösse des internen Buffers benötigt. Job-Control Angaben sind für den Dataset massgebend. Sind an beiden Stellen Recordlängen definiert, so darf diejenige in der Filedefinition nicht kleiner sein als diejenige in der Job-Control, da aus der Filedefinitionsangabe bereits ein interner Input/Output Bereich aufgebaut wird. Fehlt in der Filedefinition eine Recordlänge, wird ein Bereich von 32K reserviert.

Die Outputarea wird nach jedem `PUT` standardmässig gelöscht, unabhängig davon, ob der Bereich dynamisch zugeordnet oder mit `WP=` in den Workbereich gelegt wurde. Das Löschen kann jedoch mit der Option `CLR=NO` ausgeschaltet werden.

Allgemein zu beachten bei VSAM

1. Bei der Filedefinition wird keine Unterscheidung gemacht zwischen ESDS bzw. KSDS, RRDS. Diese Information, wie auch eine eventuelle Keylänge und Keyposition, beschafft sich QPAC selbständig aus der Cluster-Definition im VSAM-Katalog.

2. Es ist möglich, VSAM-Files mit variablen Recordlängen zu bearbeiten. Die Kommunikation darüber erfolgt über die FCA (File Communication Area), deren Platzierung als Option bei der Filedefinition erfolgt oder dynamisch angelegt wird, wenn deren Definition fehlt:

z.B.
```
IPF=VS,FCA=9000
OPF=VSAM
```

   - nach jedem `GET` Befehl steht in den 4 Bytes auf Displacement 12 der FCA die aktuelle logische Recordlänge in binärer Form des eben zur Verfügung gestellten Satzes. **Bei Update darf diese Länge nicht verändert werden**.
     Dem Längenfeld in der FCA ist das Symbol `..LENG` zugeordnet.

   - vor einem `PUT` Befehl für Outputfiles kann in die 4 Bytes auf Displacement 12 der FCA die aktuelle logische Recordlänge des zu schreibenden Satzes in binärer Form platziert werden.
     Wenn die FCA binär Null enthält, wird die maximale Länge aus der Clusterdefinition genommen.
     Dem Längenfeld in der FCA ist das Symbol `..LENG` zugeordnet.

3. Die *`rl`* Definition in der Filedefinition wird nur zur internen Aufbereitung eines Bufferbereiches benötigt. Sofern die Storage-Partition gross genug ist, kann darauf verzichtet werden. Es wird dann ein Bereich von 32760 Bytes angenommen. Am Ende der Verarbeitung wird durch QPAC jedoch die grösste festgestellte *`rl`* des Files mitgeteilt.

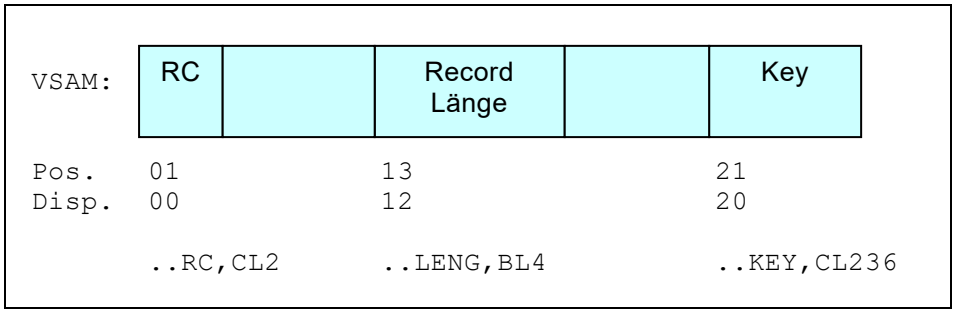


Abb. 14: File Communication Area (FCA)

Allgemeines zu den Variablen Recordlängen

1. Updating ist bei VAR, SPN und UNDefined Diskfiles unterstützt. Die jeweilige Recordlänge darf dabei nicht verändert werden.

2. Verarbeitungsmässig ist kein Unterschied zwischen VAR und SPN. Der einzige Unterschied ist in der Filedefinition zu finden, der eine andere physikalische Speicherungsform zugrunde liegt.

3. In einem z/OS Environment wird das Recordformat 'Variabel', 'Spanned' bzw. 'Undefined' von der Job-Control her akzeptiert und braucht nicht zusätzlich in der Filedefinition angegeben zu werden. Ist sie in der Filedefinition angegeben, gilt diese Definition und wird nicht überschrieben.
   Defaultangabe ist 'fix'.

4. SPN 'Spanned' ist für z/OS Partitioned Data Sets ungültig.

Dynamische Filezuordnung bei z/OS

In der z/OS Umgebung unterstützt QPAC auch die Möglichkeit von dynamischer Filezuordnung. Es werden dabei drei Definitions-Arten unterschieden und eine zusätzliche Befehlskategorie:

1. JCL Static Allocation
2. JCL Dynamic Allocation
3. Full Dynamic Allocation
4. Data Set only commands

Bei der **JCL Static Allocation** wird das DD-Statement dynamisch erstellt.
Der DSN dazu und der DD-Name sind als Bestandteil der Filedefinition in QPAC bereits vorgegeben.
JCL Static Allocation ist bei **impliziter und expliziter Filedefinition** möglich.

z.B.

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

Abb. 15: JCL Static Allocation

Bei der **JCL Dynamic Allocation** wird das DD-Statement dynamisch erstellt.
Die Fileorganisation und bestimmte Optionen werden bei der Filedefinition festgelegt, nicht jedoch der DSN und nach Bedarf der DD-Name.
JCL Dynamic Allocation ist nur bei **expliziter Filedefinition** zulässig.

z.B.

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

Abb. 16: JCL Dynamic Allocation

Bei der **Full Dynamic Allocation** wird im QPAC Programm nur ein Filedefinitions-Skelett festgelegt. Die Vervollständigung der Filedefinition erfolgt dynamisch während der Ausführungszeit. Dabei muss die Fileorganisation wie auch alle notwendigen Zusatzinformationen wie Recordlänge, Blockierung, Speichermedium usw. neben DSN und gegebenenfalls DD-Name angegeben werden.

z.B.

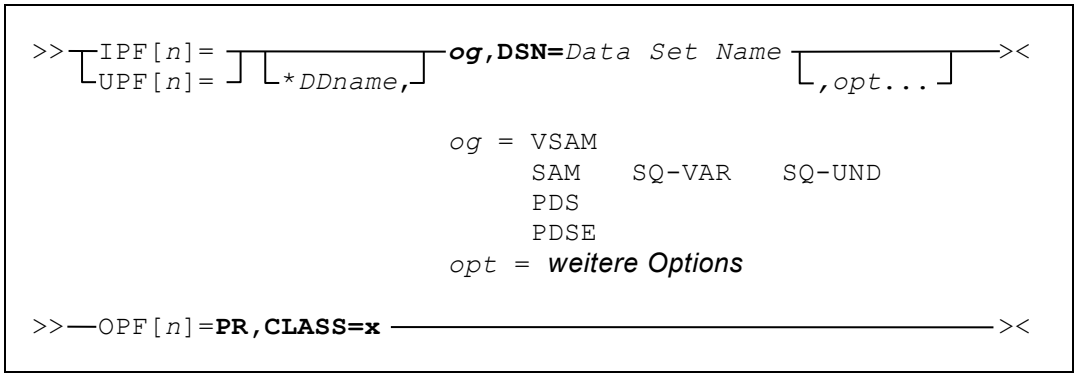
```
OPF1=DYNAMIC
```

Abb. 17: Full Dynamic Allocation

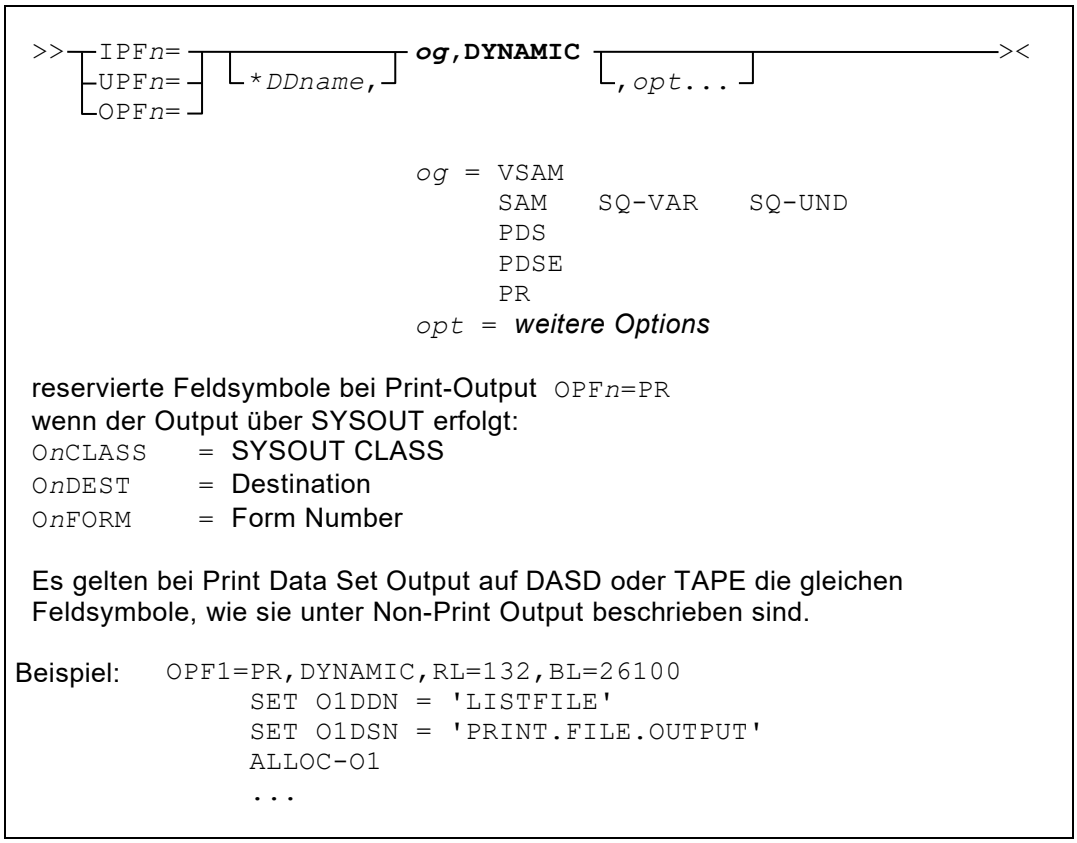
Bei Printfiles gibt es noch eine weitere Unterscheidung. Wird ein DSN definiert, erfolgt der Output auf einen Dataset auf DASD oder Tape, je nach angegebenem Medium. Default ist DASD.
Wird dagegen eine Klasse (..CLASS =) definiert, wird ein SYSOUT DD-Statement alloziert. Sind sowohl DSN und CLASS vorhanden, wird SYSOUT vorgezogen.

Data Set only Commands sind eine Befehlskategorie, mit denen man sich über einen Data Set orientieren kann bevor die eigentliche Allocation vorgenommen wird. Damit verbunden sind auch eine Gruppe von reservierten Feldnamen, die alle mit dem Präfix `ANY..` beginnen.

Grundformat der Filedefinition bei JCL Static Allocation



Grundformat der Filedefinition bei JCL Dynamic Allocation



reservierte Feldsymbole bei Print-Output `OPFn=PR`
wenn der Output über SYSOUT erfolgt:

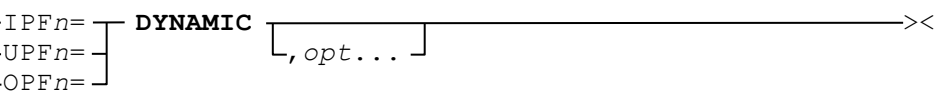
`OnCLASS` = SYSOUT CLASS
`OnDEST` = Destination
`OnFORM` = Form Number

Es gelten bei Print Data Set Output auf DASD oder TAPE die gleichen Feldsymbole, wie sie unter Non-Print Output beschrieben sind.

Beispiel:

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

Grundformat der Filedefinition bei Full Dynamic Allocation



ins reservierte Feld `..DSORG` muss die File Organisation abgefüllt werden:
`PS` oder `SQ`, `PO`, `PE` oder `VS`

reservierte Feldsymbole bei Full Dynamic:
`..DSORG` = Data Set Organisation:
`PS` = SAM, `PO` = PDS, `PE` = PDSE, `VS` = VSAM
`..VOLID` = Volume Id: z.B. für Tape, 'SCRTCH' gilt für `OPFn=`

Full Dynamic Allocation für Data Sets, deren Organisation noch nicht vorbestimmt ist, kann nur bei expliziten Filedefinitionen verwendet werden.

Die definitive Organisation wird beim `ALLOC-..` Command zugeordnet.
Dazu müssen vorausgehend die in den nachfolgenden Beispielen aufgeführten reservierten Felder abgefüllt werden.

Beispiele:

```
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
```

Zusätzliche Befehle für JCL Dynamic/Full Dynamic Allocation

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

Der `ALLOC` Befehl ist **vor** dem `OPEN` durchzuführen. Der `UNALLOC` Befehl ist nach dem `CLOSE` durchzuführen. Beide Commands verwenden den SVC 99.
Bei Fehlern wird der SVC 99 Return Code zurückgegeben und kann abgefragt werden, sofern `RC=YES` definiert wurde. Andernfalls wird abgebrochen.

z.B.

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

Zusätzliche reservierte Feldnamen für JCL Dynamic/Full Dynamic Allocation

reservierte Feldsymbole bei **IPF**:

I*n*DSN	=	'Data Set Name'
I*n*DDN	=	'DDname' Default ist 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	=	Membername, auch im Generic Format möglich (nur bei PDS)
I*n*MEMNM	=	Membername, voller Name aus PDS als SAM File

reservierte Feldsymbole bei **UPF**:

U*n*DSN	=	'Data Set Name'
U*n*DDN	=	'DDname' Default ist 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

reservierte Feldsymbole bei **OPF** (nicht für Print Output):

O*n*DSN	=	'Data Set Name'
O*n*DDN	=	'DDname' Default ist OPF*n*
O*n*RL	=	Record Länge
O*n*BL	=	Block Länge
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 oder *nnn*
O*n*SECSP	=	Secondary Space: 3 oder *nnn*
O*n*DIRBL	=	Directory Blocks für PDS: 32 oder *nnn*
O*n*RLSE	=	Release Space: 'N' = NO oder 'Y' = YES
O*n*UNIT	=	Speicher Medium: 'SYSALLDA' oder '*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

Wird Dynamic Allocation für Tape-Output verwendet, muss ein DD-Statement vorhanden sein, durch welches die Tape-Units reserviert werden. Beim DD-Statement müssen der UNIT Parameter, der DISP Parameter und evtl. der VOL Parameter vorhanden sein.

z.B.

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

Data Set Only Commands

Nachfolgende Befehle beziehen sich auf die Behandlung von Data Sets, ohne dass eine zugehörige Filedefinition notwendig ist. Damit verbunden ist eine Gruppe von reservierten Feldnamen implementiert, die alle mit dem Prefix ANY.. beginnen. Diese vordefinierten Felder sind für den Informationsaustausch notwendig. Mit dieser Befehlskategorie kann man sich über einen Data Set orientieren bevor die eigentliche Allocation vorgenommen wird, beispielsweise über die Frage, ob der Data Set schon vorhanden ist und vorgängig noch gelöscht werden muss.

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

Neutrale Commands

INQDSN-ANY	Inquire Data Set

Mit diesem Command kann ein Data Set abgefragt werden.
Im Feld ANYDSN wird vorausgehend der Name des Data Sets angegeben.
Als Ergebnis werden die Felder ANYVOLID, ANYRECFM, ANYDSORG, ANYBL, ANYRL, ANYUNIT, ANYLABEL, ANYPRISP, ANYSPTYP zurückgegeben, sofern der Return Code ANYRC X'0000' enthält.
Bei Fehler steht im ANYRC Feld der Return Code (siehe nachfolgende Liste)

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

Mit diesem Command kann ein Data Set via DD-Name abgefragt werden.
Im Feld ANYDDN wird vorausgehend der Name des DD-Statements angegeben.
Als Ergebnis werden die Felder ANYDSN, ANYVOLID, ANYRECFM, ANYDSORG, ANYBL, ANYRL, ANYUNIT, ANYLABEL, ... zurückgegeben, sofern der Return Code ANYRC X'0000' enthält.
Bei Fehler steht im ANYRC Feld der Return Code (siehe nachfolgende Liste)

UNCDSN-ANY	Uncatalog Data Set

Mit diesem Command kann ein Data Set 'uncataloged' werden. In der VTOC bleibt der File bestehen. Ins Feld `ANYDSN` wird vorausgehend der Data Set Name gestellt. Bei SMS kontrollierten Data Sets ist dieser Command wirkungslos und der Return Code ist 0.
Bei Fehler steht im Feld `ANYRC` der Return Code (siehe nachfolgende Liste)

DELDSN-ANY	Delete Data Set

Mit diesem Command kann ein Data Set aus dem Katalog entfernt wie auch aus der VTOC gelöscht werden. Tape Data Sets werden nur aus dem Katalog entfernt.
Ins Feld `ANYDSN` wird vorausgehend der Data Set Name gestellt.
Bei Fehler steht im Feld `ANYRC` der Return Code (siehe nachfolgende Liste)

ANYRC oder ..RC Return Codes

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

Aus einem PDS File kann auch dynamisch eine selektive Memberauswahl definiert werden. Hierfür gibt es zwei Möglichkeiten:

1. Ein einzelnes Member wird als SAM File gelesen, nach dem JCL Prinzip DSN=Fileid(Member), z.B. `SET I1MEMNM = 'MEMNAME'`

2. Eine generische Memberauswahl kann definiert werden, die aus dem PDS in sequenzieller Reihenfolge gelesen wird, z.B. `SET I1MN = 'ABC*'`
   Alle Members, die mit ABC beginnen, werden ausgewählt. Die Regel für die Generic-Definition ist die gleiche, wie sie bei der statischen PDS Filedefinition beschrieben ist.

Kapitel 3. Input/Output Instruktionen

Instruktionen Übersicht

Systembezogene Instruktionen

GETIN	Lesen vom System Reader
PUTPCH	Punchen auf den System Puncher
PUTLST	Schreiben auf den System Printer

Abb. 18:Übersicht der systembezogenen Instruktionen

Filebezogene Instruktionen

OPEN	-I	Öffnen Dataset
	-O	
	-U	
CLOSE	-I	Schliessen Dataset
	-O	
	-U	

Abb. 19: Übersicht der filebezogenen Instruktionen

I/O Instruktionen für Sequentielle Verarbeitung

GET	-I	Lesen nächster Record
	-U	
PUT	-O	Schreiben neuer Record
	-U	Zurückschreiben Record
PUTA	-U	Einfügen Record
PUTD	-U	Löschen Record
SETGK	-I	Aufsetzen mit Key 'greater' oder 'equal'
	-U	
SETEK	-I	Aufsetzen mit Key 'equal'
	-U	

Abb. 20: Übersicht der I/O Instruktionen für sequentielle Verarbeitung

Random Instruktionen für VSAM Dateien

READ	-I*n* -U*n*	Lesen direkt mit Key gleich oder Relativer Record Nummer
READGE RDGE	-I*n* -U*n*	Lesen direkt mit Key grösser oder gleich oder Relativer Record Nummer (Bereich ..RRN+100)
READUP RDUP	-U*n*	Lesen direkt für Update mit Key gleich oder Relativer Record Nummer
REWRITE RWRT	-U*n*	Zurückschreiben des mit READUP gelesenen Records
INSERT ISRT	-U*n*	Einfügen eines neuen Records
DELETE DLET	-U*n*	Löschen eines Records

Abb. 21: Übersicht der Random Instruktionen für VSAM

Grundformat

OPERATION-*Fileidentifikation*

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

Abb. 22: Beispiele I/O Instruktionen

Die Fileidentifikation kann weggelassen werden, wenn es sich um die '**implizite Verarbeitungslogik**' handelt, d.h. bei der Fileidentifikation keine Nummer vorhanden ist (siehe: Interne Verarbeitungslogik und Steuerung).
Die Fileidentifikation kann in Kurzform definiert werden (GET-I7).

Filebezogene Instruktionen

Die OPEN Instruktion

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

Ein OPEN kann jederzeit gegeben werden, ist jedoch für das erstmalige Eröffnen **nicht unbedingt erforderlich**. Ein GET-Befehl eröffnet den angesprochenen File automatisch; der GET-Befehl jedoch nur, sofern der File bisher noch nie eröffnet war. Ein wiederholtes Eröffnen eines Files kann nur durch Verwendung der OPEN-Funktion vorgenommen werden.

Der OPEN-Funktion muss immer eine Fileidentifikation beigefügt werden.

Die CLOSE Instruktion

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

Ein CLOSE kann jederzeit gegeben werden, ist jedoch **nicht unbedingt erforderlich**, da Inputfiles bei EOF-Kondition und Outputfiles spätestens bei Verarbeitungsende automatisch geschlossen werden.

Ein mehrmaliges Lesen eines Inputfiles ist jedoch jederzeit mit CLOSE-I OPEN-I möglich, auch wenn der File noch nicht EOF-Status aufweist.

Der CLOSE-Funktion muss immer eine Fileidentifikation beigefügt werden.

ALLOC / UNALLOC für Dynamische Filezuordnung (z/OS)

```
ALLOC   -In          Allocate Data Set
        -Un
        -On
```

```
UNALLOC   -In        Unallocate Data Set
          -Un
          -On
```

Vor der ALLOC Instruktion müssen die zugehörigen Werte in die vorgesehenen reservierten Feldsymbole abgefüllt werden: bei IPF mindestens I*n*DSN.

```
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
```

Abb. 23: Beispiel einer Befehlsfolge für dynamische Filezuordnung

I/O Instruktionen für Sequentielle Verarbeitung

Die GET Instruktion

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

Mit GET wird ein logischer Inputfile-Record zur Verfügung gestellt. Alle Inputadressen in nachfolgenden Verarbeitungsinstruktionen beziehen sich nachfolgend automatisch auf den durch den GET-Befehl gelesenen Record-Bereich, jedoch nur in hierarchischer Ordnung.

Bezieht sich der GET-Befehl auf einen Updatefile, beziehen sich nachfolgend auch die Outputadressen auf den gleichen Bereich.

Dem GET-Befehl kann unmittelbar ein EOF-Steuerblock AT-EOF folgen, der bei End of File durchlaufen wird. Fehlt der EOF-Steuerblock, wird bei EOF-Status der ganze GET-Block übersprungen. Ist der EOF-Steuerblock vorhanden, wird er bei EOF-Status durchlaufen, und am Ende des Blockes erfolgt kein automatischer GET-Block Sprung. Die Verarbeitung geht nach der Definition ATEND weiter.

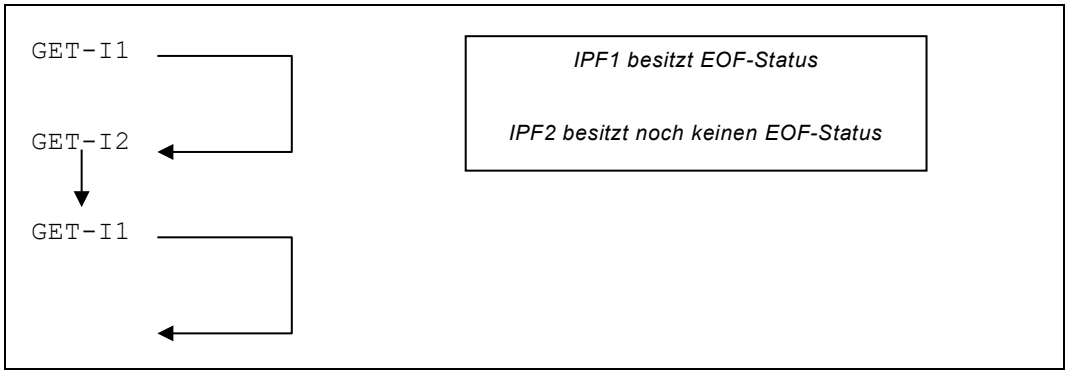


Abb. 24: Beispiel GET Befehl ohne EOF-Block

Merke:
Ein GET-Befehl auf hierarchischem Level 0 kennzeichnet die zugehörige Filedefinition als **leitenden Bestand**.

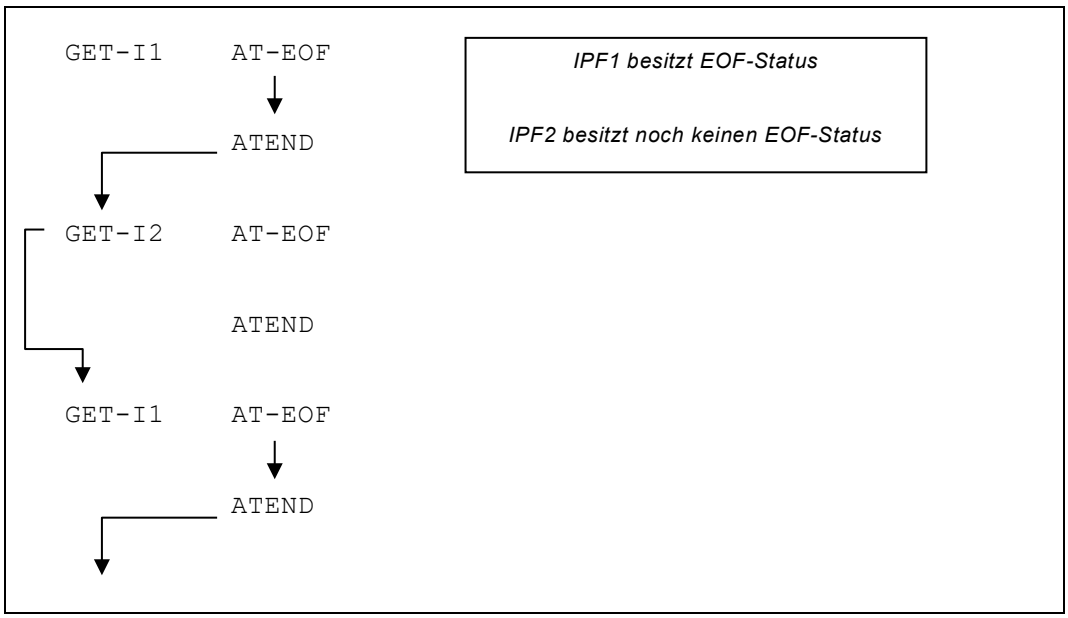


Abb. 25: Beispiel GET Befehl mit EOF-Block

Die PUT Instruktion

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

Mit `PUT` wird ein logischer Record auf den Ausgabefile geschrieben bzw. auf einen 'Updatefile' zurückgebracht.

Nach Ausführung des Befehls wird der Ausgabebereich, **nicht jedoch ein Updatebereich**, automatisch gelöscht (Basislöschung = X'00', X'40' bei Card- und Printeroutput, bzw. `CLR=X'..'`).

Für VSAM KSDS, DB2 und DL/I Datenbanken sind zwei zusätzliche I/O Befehle unterstützt.
Die Anwendung bei DB2 ist im DB2 Support Feature beschrieben, die Anwendung bei DL/I im DL/I Support Feature.
Nachfolgende Angaben beziehen sich auf VSAM Datasets.

Die PUTA (Put Addition) Instruktion

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

Dieser Befehl kann nur bei **Update Files** verwendet werden, nicht aber bei Filedefinitionen, die als Input-only (IPF=) oder Output-only (OPF=) gelten.

Der Befehl bewirkt das Hinzufügen des Records, der sich zu diesem Zeitpunkt in der I/O-Area befindet. In die FCA kann nach Bedarf die Recordlänge gesetzt werden. Ist die Länge in der FCA null, wird die Maximallänge der Cluster-Definition angenommen. Bei RRDS Files muss ins Feld ..RRN auch die relative Recordnummer gesetzt werden.

PUTA arbeitet über den RPL für sequentielle Verarbeitung. Die Positionierung wird gesteuert.

Die PUTD (Put Delete) Instruktion

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

Dieser Befehl kann nur bei **Update Files** verwendet werden, nicht aber bei Filedefinitionen, die als Input-only (IPF=) oder Output-only (OPF=) gelten.

Der Befehl bewirkt das Löschen des zuletzt durch GET gelesenen Records.

Die FCA wird von diesem Befehl nicht benötigt.

Die SETGK und die SETEK Instruktionen

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

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

SETGK ist eine ergänzende Input/Output Operation, die es erlaubt, bei Files, die auf der Basis von Keys aufgebaut sind, auf beliebige Generic-Keys aufzusetzen, um an dieser Stelle die Verarbeitungssequenz fortzusetzen.

Generic Key bedeutet: **gleicher oder nächst höherer Key** als derjenige, mit dem aufgesetzt wird.

SETEK ist eine ergänzende Input/Output Operation, die es erlaubt, bei Files, die auf der Basis von Keys aufgebaut sind, auf einen bestimmten Key aufzusetzen, um an dieser Stelle die Verarbeitungssequenz fortzusetzen.

'Key equal to' bedeutet: der Record mit dem **gleichen Key** muss vorhanden sein, sonst gibt es eine 'Not Found' Condition.

Vor Ausführung der SETGK bzw. der SETEK Funktion muss der Key-Wert in die FCA gegeben werden und zwar ab Offset 20 (Position 21). Die Positionen 1-20 der FCA sind je nach Organisation unterschiedlich verwendet.

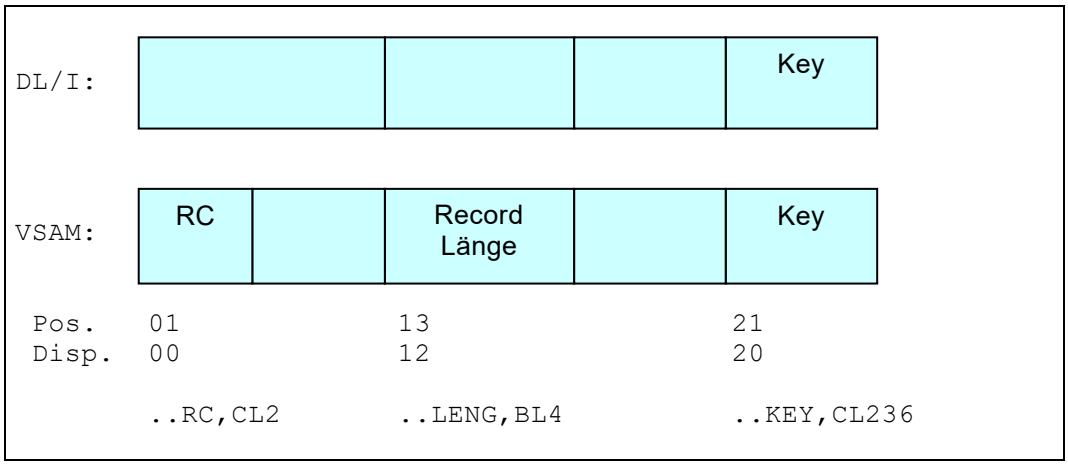


Abb. 26: File Communication Area (FCA)

Der SETGK/SETEK Befehl beinhaltet einen OPEN, sofern der Filezustand „closed“ ist.

Random Instruktionen für VSAM Dateien

Die READ Instruktion

```
READ-In
     -Un
```

Lesen direkt mit Key gleich oder Relativer Record Nummer.

a) VSAM KSDS
Der Record mit dem im FCA Feld ..KEY spezifizierten Key wird gelesen.
Im FCA Feld ..RC wird der Original VSAM Return- und Feedbackcode retourniert:
Position 1 (..RC1) enthält den Returncode X'00', X'04', X'08' etc.
Position 2 (..RC2) enthält den Feedbackcode X'..'.

```
SET I1KEY = '123456'
READ-I1
IF I1RC = X'0810' THEN ...    (kein Record gefunden)
```

Abb. 27: Abfrage des Returncodes nach einer READ Instruktion

b) VSAM RRDS
Der Record mit der im FCA Feld ..RRN spezifizierten Relativen Record Nummer wird gelesen.
Im FCA Feld ..RC wird der Original VSAM Return- und Feedbackcode retourniert:
Position 1 (..RC1) enthält den Returncode X'00', X'04', X'08' etc.
Position 2 (..RC2) enthält den Feedbackcode X'..'.

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

Abb. 28: Direktzugriff auf einen VSAM RRDS mit READ

Die READGE Instruktion

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

Lesen direkt mit Key grösser oder gleich oder mit Relativer Record Nummer (Bereich ..RRN+100).

a) VSAM KSDS
Der Record mit dem im FCA Feld `..KEY` spezifizierten Key oder der nächste Record in Sequenz wird gelesen.
Im FCA Feld `..RC` wird der Original VSAM Return- und Feedbackcode retourniert:
Position 1 (`..RC1`) enthält den Returncode X'00', X'04', X'08' etc.
Position 2 (`..RC2`) enthält den Feedbackcode X'..'.

```
SET I1KEY = '123456'
READGE-I1
IF I1RC = X'0810' THEN ...    (kein Record gefunden)
```

Abb. 29: Abfrage des Returncodes nach einer READGE Instruktion

b) VSAM RRDS
Der Record mit der im FCA Feld `..RRN` spezifizierten Relativen Record Nummer oder der nächste in Sequenz innerhalb der nächsten 100 Slots wird gelesen.
Im FCA Feld `..RC` wird der Original VSAM Return- und Feedbackcode retourniert:
Position 1 (`..RC1`) enthält den Returncode X'00', X'04', X'08' etc.
Position 2 (`..RC2`) enthält den Feedbackcode X'..'.

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

Abb. 30: Direktzugriff auf einen VSAM RRDS mit READGE

Die READUP Instruktion

```
READUP-Un
RDUP
```

Lesen für Update direkt mit Key gleich oder Relativer Record Nummer.

a) VSAM KSDS
Diese Funktion ist dieselbe, wie unter `READ` beschrieben, aber die File Definition muss `UPFn` sein.

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

Abb. 31: Read for Update mit der READUP Instruktion

b) VSAM RRDS
Diese Funktion ist dieselbe, wie unter READ beschrieben, aber die File Definition muss UPF*n* sein.

Die REWRITE Instruktion

```
REWRITE-Un
RWRT
```

Zurückschreiben des Records, der vorher mit READUP gelesen wurde.

a) VSAM KSDS
Der Record in der Record Area wird zurückgeschrieben (Updated). Im FCA Feld ..RC wird der Original VSAM Return- und Feedbackcode retourniert:
Position 1 (..RC1) enthält den Returncode X'00', X'04', X'08' etc.
Position 2 (..RC2) enthält den Feedbackcode X'..'.

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

REWRITE-U1
IF U1RC1 <> X'00' THEN ...    (irgend ein Fehler)
```

Abb. 32: Update eines mit READUP gelesenen Records mit REWRITE

b) VSAM RRDS
Der Record in der Record Area wird zurückgeschrieben (Updated). Im FCA Feld ..RC wird der Original VSAM Return- und Feedbackcode retourniert:
Position 1 (..RC1) enthält den Returncode X'00', X'04', X'08' etc.
Position 2 (..RC2) enthält den Feedbackcode X'..'.

Die INSERT Instruktion

```
INSERT-Un
ISRT
```

Einfügen eines Records.

a) VSAM KSDS:
Der Record in der Recordarea wird in den Dataset eingefügt (inserted). Der Key des Recordinhaltes ist für die Positionierung massgebend. Im FCA Feld ..RC wird der Original VSAM Return- und Feedbackcode retourniert:
Position 1 (..RC1) enthält den Returncode X'00', X'04', X'08' etc.
Position 2 (..RC2) enthält den Feedbackcode X'..'.

b) VSAM RRDS:
Der Record in der Record Area wird in den leeren Slot eingefügt (inserted). Die Relative Recordnummer (RRN) muss vorgängig im Feld

..RRN gespeichert werden. Im FCA Feld ..RC wird der Original VSAM Return- und Feedbackcode retourniert:
Position 1 (..RC1) enthält den Returncode X'00', X'04', X'08' etc.
Position 2 (..RC2) enthält den Feedbackcode X'..'.

```
SET U1RRN = 5
INSERT-U1
IF U1RC1 <> X'00' THEN ...    (irgend ein Fehler)
```

Abb. 33: Einfügen eines Records mit INSERT

Die DELETE Instruktion

DELETE-U*n*
DLET

Löschen eines Records.

a) VSAM KSDS
Der Record, dessen Key im Feld ..KEY gespeichert ist, wird gelöscht. Im FCA Feld ..RC wird der Original VSAM Return- und Feedbackcode retourniert:
Position 1 (..RC1) enthält den Returncode X'00', X'04', X'08' etc.
Position 2 (..RC2) enthält den Feedbackcode X'..'.

```
SET U1KEY = '123456'
DELETE-U1
IF U1RC1 <> X'00' THEN ...    (irgend ein Fehler)
```

Abb. 34: Löschen eines Records mit DELETE

b) VSAM RRDS
Der Record, dessen Relative Recordnummer (RRN) im Feld ..RRN steht, wird gelöscht. Im FCA Feld ..RC wird der Original VSAM Return- und Feedbackcode retourniert:
Position 1 (..RC1) enthält den Returncode X'00', X'04', X'08' etc.
Position 2 (..RC2) enthält den Feedbackcode X'..'.

```
SET U1RRN = 5
DELETE-U1
IF U1RC1 <> X'00' THEN ...    (irgend ein Fehler)
```

Abb. 35: Löschen eines Records mit DELETE

Printerfile bezogene Instruktionen

Für Printerfiles können neben dem `PUT`-Befehl auch nachfolgende Ausgabebefehle verwendet werden:

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	

Abb. 36: Instruktionsübersicht für Printerfiles

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

Abb. 37: Instruktionsübersicht für Printerfiles

Durch die Immediate-Operationen wird der Ausgabebereich nicht gelöscht.

PDS bezogene Instruktionen (z/OS)

Für z/OS Partitioned Datasets steht der nachfolgende Spezialbefehl zur Verfügung für die Verbuchung von Membernamen in der PDS-Directory.

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

Für das zuletzt erstellte Member wird der Name aus dem FCA-Feld `..MEMNM` genommen und in der PDS-Directory eingetragen.
Steht im FCA Feld `..STOWID` ein 'A', darf das Member noch nicht vorhanden sein.
Steht darin ein 'R', wird ein allenfalls bestehendes Member ersetzt.
Steht darin kein ungültiger Code, so wird 'A' angenommen.

Zusätzlich kann für die Nachführung der Statistikdaten im Directory Record die Version-Modifikation und/oder die User-Id vorgegeben werden. Dazu dienen die reservierten Felder `..STOWVV`, `....STOWMM` und `....STOWUSER`.

Systembezogene Instruktionen

Die GETIN Instruktion

```
GETIN
```

Dieser Lesebefehl liest den Instreamfile (QPACIN).

Datenrecords, die dem END Statement des QPAC folgen, werden damit ohne zusätzliche Filedefinition direkt in die **Working-Storage Positionen 5001 - 5080** gelesen.
EOF wird dadurch angezeigt, dass die Inputarea auf X'FF' High-Value gesetzt wird.

Die PUTLST Instruktion

```
PUTLST
PUTLST-'Literal'
PUTLST-Symbolname[,l]
PUTLST-'Literal',Symbolname[,l],'Literal'...
```

Mit diesem Schreibbefehl kann ohne zusätzliche Filedefinition auf den System-Printfile (QPACLIST) geschrieben werden.

Die **Working-Storage Positionen 5201 - 5320** enthalten den zu schreibenden Zeilentext, sofern kein Literal oder Symbolname angegeben wurde. Auf Position 5200 kann ein ASA Control-Character gesetzt werden (Default ist blank).
Beim ersten PUTLST ab Working-Storage Position 5201-5320 wird intern in jedem Falle ein Seitenwechsel vorgenommen. Die Outputarea wird nicht automatisch gelöscht.
Die Zeilenbreite wird mit 121 Stellen angenommen (inklusive Control-Character). Mit der PARM Option LISTL= (Listlength) kann die Zeilenbreite auf bis zu 250 Positionen vergrössert werden.

Es können auch mehrere Operanden, durch Komma getrennt, definiert werden, Literale oder Feldsymbole. Ihre inhaltliche Gesamtlänge darf aber 120 Bytes nicht überschreiten.

Die PUTPCH Instruktion

```
PUTPCH
```

Mit diesem Punchbefehl kann ohne zusätzliche Filedefinition auf den System-Punchfile (QPACPUN) gestanzt werden.

Die **Working-Storage Positionen 5101 - 5180** enthalten den zu stanzenden Record.
Der Stanzbereich wird nicht automatisch gelöscht.

Titelzeilen für Printerfiles

Die HEADER Definition (Statische Titelzeilen)

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

Mit einem HDR-Paar wird eine Titelzeile definiert.
Das erste HDR-Statement definiert die Printpositionen 1 - 66, das zweite HDR-Statement definiert die Printpositionen 67 - 132.

Die Reihenfolge der HDR-Statements ist bestimmend für die Reihenfolge der Titelzeilen.

HDR-Statements mit Identifikationsangabe (HDR-O1) ermöglichen nur die Printpositionen von 1 - 65 bzw. 68 - 131 abzudecken.

Es ist zu beachten, dass anstelle dieser statischen Überschriftenzeilen die **TITLE-Routine** angewendet werden kann. Werden gleichzeitig HDR und TITLE definiert, wird HDR unwirksam.

Es ist zu beachten, dass wenn in der File Definition der Parameter IPC angegeben wird, keine Titelzeilen produziert werden.

Die TITLE Definition (Dynamische Titelzeilen)

Neben der Möglichkeit, statische Titelzeilen mit dem HDR-Statement zu definieren, besteht auch die Möglichkeit, dynamische Titelzeilen zu definieren, deren Angaben unmittelbar während der Verarbeitung verändert werden können.

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

Es handelt sich hier um einen Subroutine Strukturblock, der **automatisch zur 'Title-Time'** durchlaufen wird.

'Title-Time' ist gegeben:

- beim Schreiben der ersten Printzeile
- bei Erreichen des Linecounters (Seitenende)
- bei Verwendung von SK1 bzw. WASK1

Das Schlüsselwort TITLE (mit Fileidentifikation bei expliziter Verarbeitungslogik) definiert den Beginn des Routinestrukturblocks und TITLEND das Strukturblockende.

Die TITLE Definition enthält implizit den Seitenwechsel und die Zuordnung auf die Printeroutputarea. Die Printeroutputarea wird in gelöschtem Zustand zur Verfügung gestellt. Die implizite Zuordnung einer Inputarea ist zur 'Title-Time' nicht eindeutig vorbestimmt.

Die TITLEND Definition enthält bei **impliziter Verarbeitungslogik** (OPF=) einen Writebefehl (W). Es ist jedoch möglich, explizit beliebige **weitere Writeoperationen innerhalb des Routineblocks** zu definieren, falls mehrere Titelzeilen geschrieben werden sollen. Nach TITLEND entspricht die semantische Instruktions-Adresszuordnung wieder der Situation vor TITLE.

Der TITLE-Routinestrukturblock kann irgendwo innerhalb der Verarbeitungs-definitionen, jedoch nach der zugehörigen Printerfiledefinition, stehen.

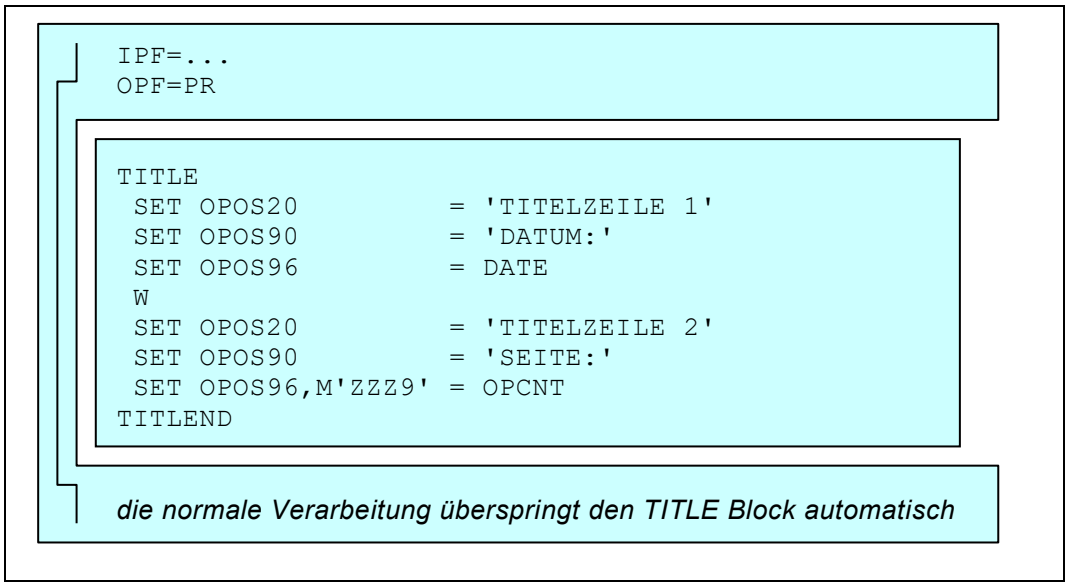


Abb. 38: Die dynamische TITLE Routine

☞ Es ist zu beachten, dass wenn in der File Definition der Parameter IPC angegeben wird, keine Titelzeilen produziert werden.

Verarbeitungslimitierungs Definitionen

Grundformat

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

LIDB[n]=  (bei Datenbanken)
LUDB[n]=
LODB[n]=
```

Damit wird der File bezüglich der zu verarbeitenden Records limitiert.

s	definiert den ersten zu berücksichtigenden Record (Startrecord)
e	definiert den letzten zu berücksichtigenden Record (Endrecord)

```
LIPF=2-50          ab 2. Record bis und mit 50. Record verarbeiten;
                         51. Record forciert EOF-Kondition
```

Abb. 39: Verarbeitungslimitierung Grundformat

Es können beliebige *von-bis* Gruppen aneinandergereiht definiert werden. Es ist auch möglich, die *von-bis* Gruppen auf mehrere Statements aufzuteilen:

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

Abb. 40: Verarbeitungslimitierung (mehrere von-bis Gruppen)

Es muss darauf geachtet werden, dass die *von-bis* Gruppen in **aufsteigender Reihenfolge** definiert werden.

`EOP` bei der Outputlimitierung bedeutet 'End of Processing'. Damit kann veranlasst werden, dass die Verarbeitung bei Erreichen der Outputlimite beendet wird. Sind mehrere `EOP` Definitionen vorhanden, müssen alle Outputlimiten erreicht worden sein, damit terminiert wird.

```
LOPF=50-100,EOP          Schreiben Records 50-100;
                         danach Verarbeitungsende
```

Abb. 41: Verarbeitungsende bei Erreichen der Limite

Spezialformate

`LIPF[n]=-e`	(*s*=1 wird angenommen)
`LIPF[n]=s-`	(*e*=EOF wird angenommen)

`LIPF1=-100`	nur Records 1 bis 100 werden gelesen
`LUPF3=10-`	nur Records von 10 bis EOF für Update lesen

Abb. 42: Verarbeitungslimitierung Spezialformat

Operator Kommunikations Instruktionen

Die WTO Instruktion (Write to Operator with No Response)

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

Der Inhalt des Sendefeldes in der Länge von *length* wird über die Konsole ausgegeben.
Als Sendefeld kann auch eine Character-Konstante definiert werden.

```
WTO-WPOS5000,10

WTO-'MELDUNG AN OPERATOR'
```

Abb. 43: Operatorkommunikation - Konsolausgabe

Die WTOR Instruktion (Write to Operator with Response)

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

Der Inhalt des Sendefeldes wird in der Länge *length* auf die Konsole geschrieben.
Im Empfangsfeld steht die Antwort des Operators.

Synchronisations Instruktionen

```
ENQ-Feldname    [,STEP]
ENQ-'Literal'   [,SYSTEM]
                [,SYSTEMS]
```

Die Anwendung des ENQ (Enqueue) Commands entspricht im Wesentlichen der ENQ Macro, wie sie in der IBM Literatur beschrieben ist.

Der Inhalt des zugeordneten Feldes bzw. des direkt definierten Literals teilt sich in zwei Teile auf. Die ersten 8 Bytes bilden den Major-Teil bzw. Queue Name, der anschliessende Teil bildet den Minor-Teil bzw. Resource Name.
Die maximale Länge des Literals bzw. des Feldes ist 128 Bytes.
STEP, SYSTEM oder SYSTEMS bilden den "Scope", wie er in der IBM Literatur beschrieben ist.
Scope SYSTEMS gilt als Default, wenn keine Angabe gemacht wird.

```
DEQ-Feldname    [,STEP]
DEQ-'Literal'   [,SYSTEM]
                [,SYSTEMS]
```

Mit dem DEQ Command wird eine ENQed Resource wieder freigegeben.

Kapitel 4. Statische Programmgliederung

Automatische Verarbeitungssteuerung

NORMAL	(optional)
LAST	(optional)
END	(notwendig)

Mit diesen drei Schlüsselwörtern wird die Verarbeitung in ihre Hauptsteuerteile gegliedert.

Die END Anweisung

Das **END** Statement ist eine Steueranweisung an den QPAC Assembler.

Damit wird das physische Ende aller Definitionen angegeben. Informationen, die dem END auf dem gleichen Statement folgen, werden ignoriert.
Nachfolgende Statements gelten als Daten oder wieder als Job Control.
Ist nur das END Statement definiert, besteht das ganze QPAC Programm aus einem einzigen Hauptverarbeitungsteil:

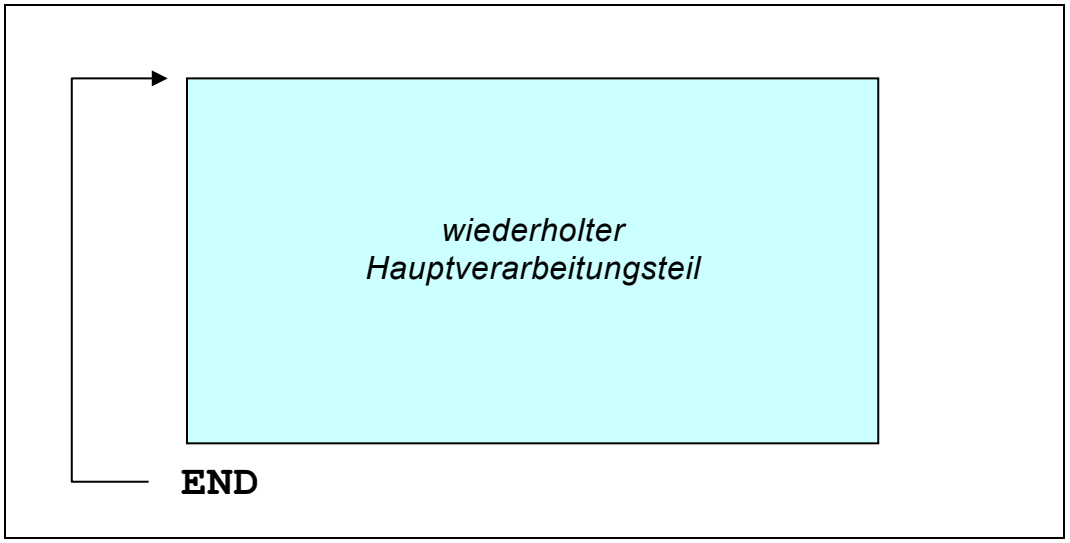


Abb. 44: Die END Anweisung ist das physische Ende aller Definitionen

Die END Anweisung kann irgendwo in den Kol. 1 - 71 stehen:

z.B.

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

Die NORMAL Anweisung

Wird zusätzlich zum `END` ein **`NORMAL`** Statement definiert, wird der davorliegende Programmteil zu einem **einmalig durchlaufenen Vorverarbeitungsteil**. Der wiederholt durchlaufene Teil folgt dem `NORMAL` Statement:

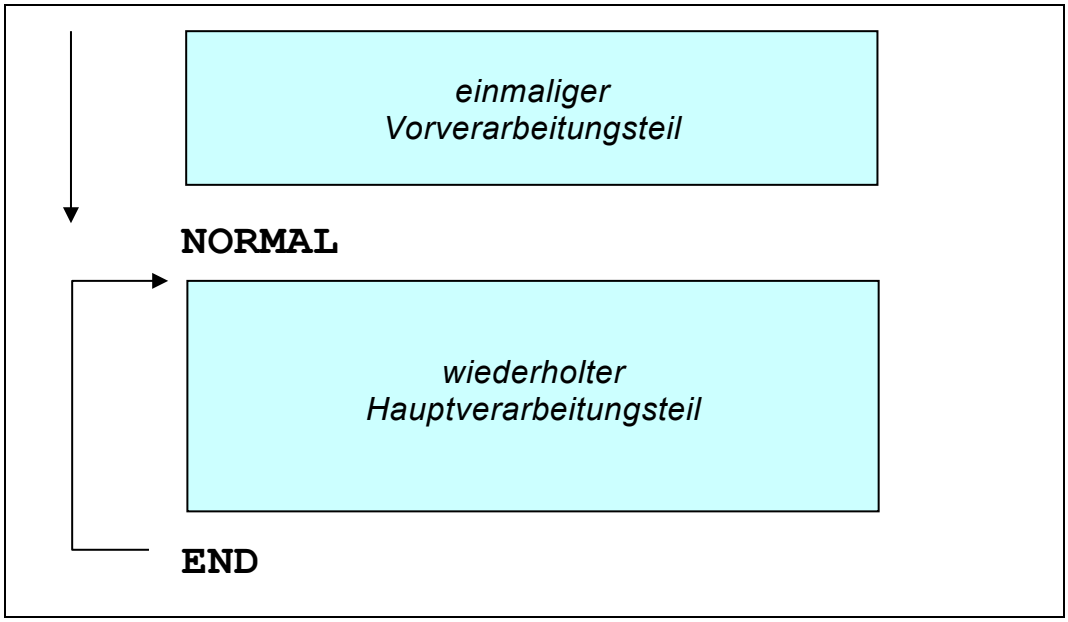


Abb. 45: Vorverarbeitung mit der NORMAL Anweisung

Die LAST Anweisung

Um eine **Endverarbeitungsroutine** durchzuführen, wenn das Verarbeitungsende erreicht worden ist (normalerweise EOF), kann das **`LAST`** Statement definiert werden. Alle Instruktionen die dem Statement `LAST` folgen, werden nur **einmal bei Verarbeitungsende durchlaufen:**

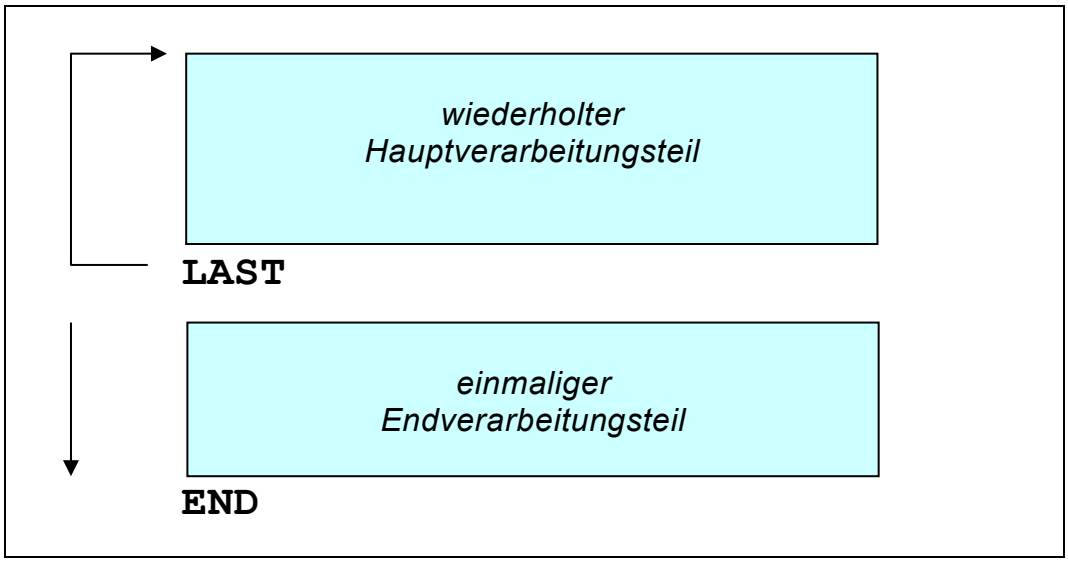


Abb. 46: Endverarbeitung mit der LAST Anweisung

Die FIRST Anweisung

Es ist möglich, mit der **FIRST** Anweisung mehrere Verarbeitungssequenzen zu definieren:

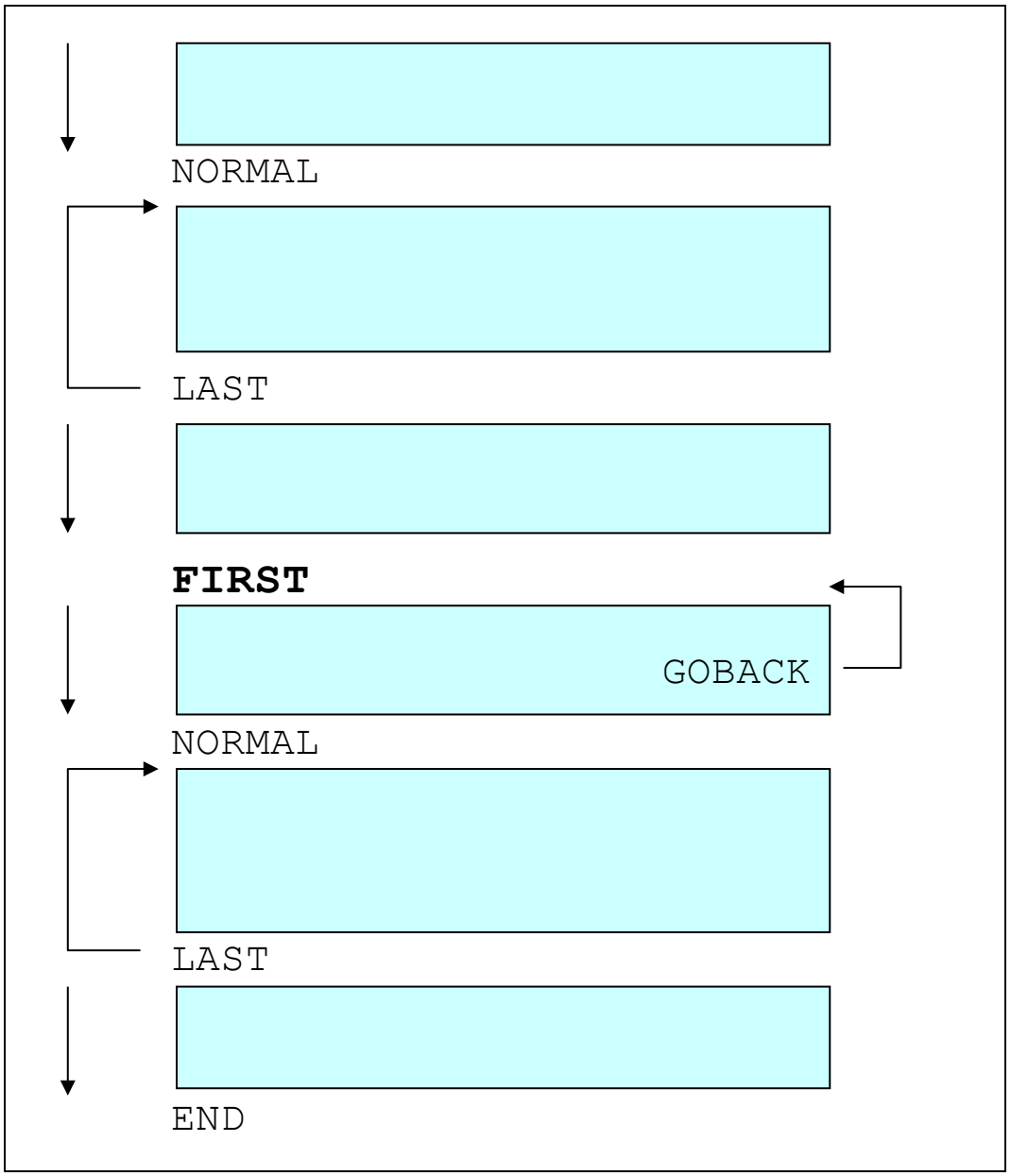


Abb. 47: Die FIRST Anweisung erlaubt mehrere Verarbeitungszyklen

Alle diese Schlüsselwörter müssen auf Level 0 stehen. d.h. alle IF- , SUB- bzw. DO-Blöcke müssen mit der eigenen *xx*END Definition abgeschlossen sein. Ist dies nicht der Fall und es werden zum END-Zeitpunkt noch offene Strukturblöcke festgestellt, wird ein Fehler gemeldet:

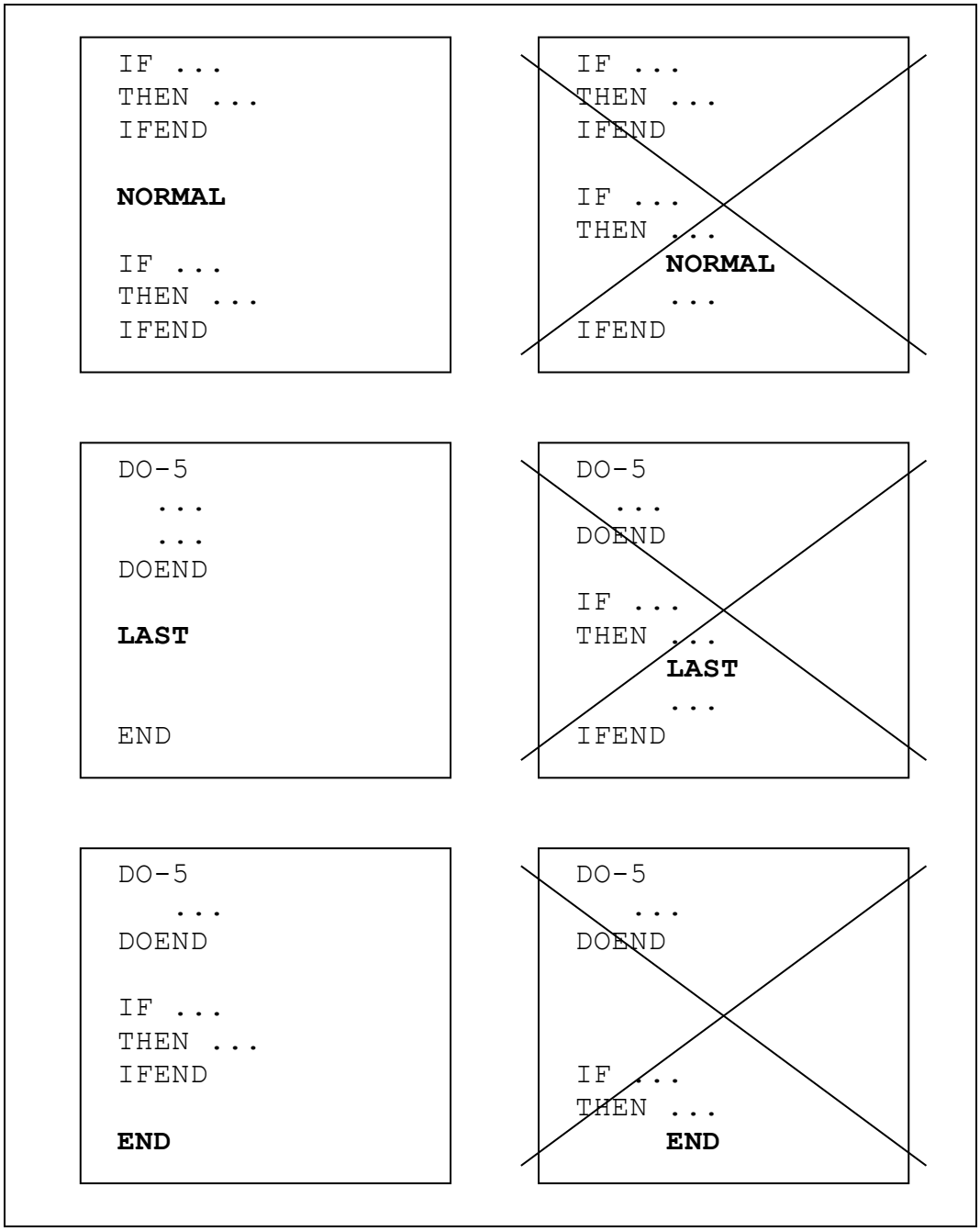


Abb. 48: Hierarchischer Level 0

Zusammenspiel von NORMAL, LAST, END bei impliziter Steuerung

Wenn IPF oder UPF als implizite Filedefinitionen vorhanden sind, enthält das NORMAL Statement einen GET Befehl.

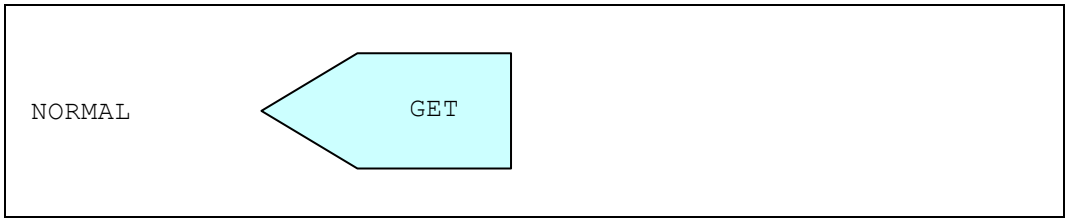


Abb. 49: Generierter GET Befehl bei NORMAL mit impliziter Steuerung

Wenn OPF oder UPF als implizite Filedefinitionen vorhanden sind, enthält das LAST Statement, oder wenn LAST nicht definiert worden ist, das END Statement einen PUT Befehl.

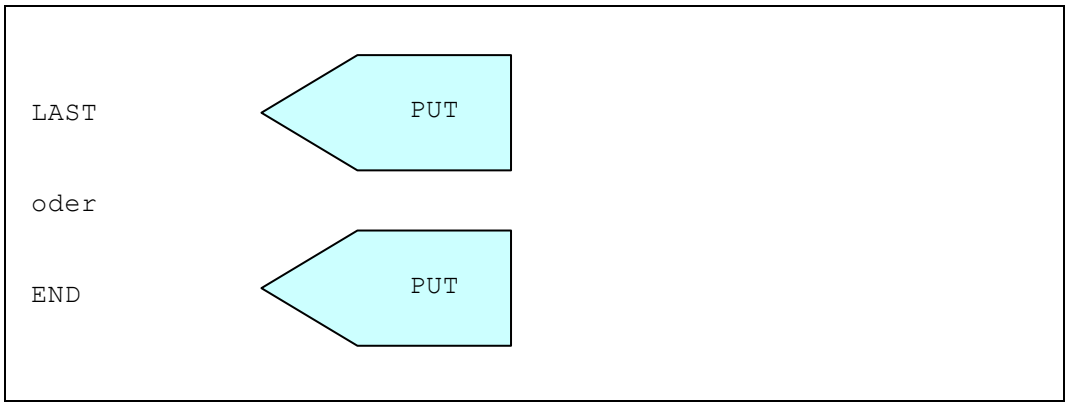


Abb. 50: Generierter PUT Befehl bei LAST oder END mit impliziter Steuerung

Detailliertere Erläuterungen folgen im Kapitel „Interne Verarbeitungslogik und Steuerung“.

Die GOSTART Anweisung

Mit dieser Schlüsselwortanweisung wird an den unmittelbaren Anfang verzweigt. Der momentane Zustand des Workbereiches wird nicht verändert.

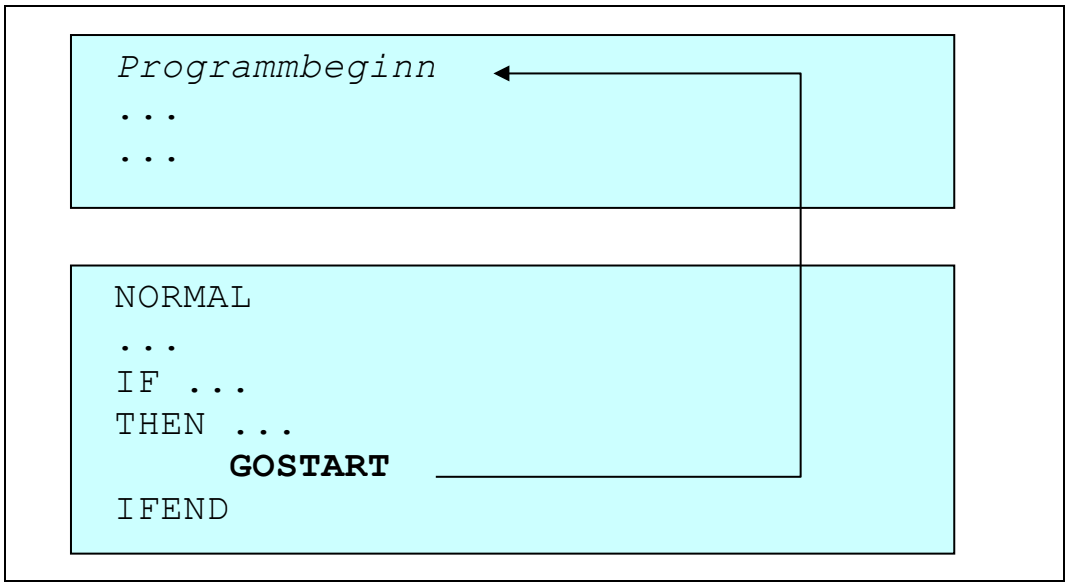


Abb. 51: Die GOSTART Anweisung

Die GOBACK Anweisung

Mit dieser Schlüsselwortanweisung wird das Zurückkehren an den Anfang der Verarbeitungsinstruktionen (Anfang der Hauptverarbeitung) befohlen. Wurde kein `NORMAL` Statement definiert, ist der Verarbeitungsanfang der Rückkehrpunkt. Wurde `NORMAL` definiert, so wird dieses zum `GOBACK` Rückkehrpunkt.

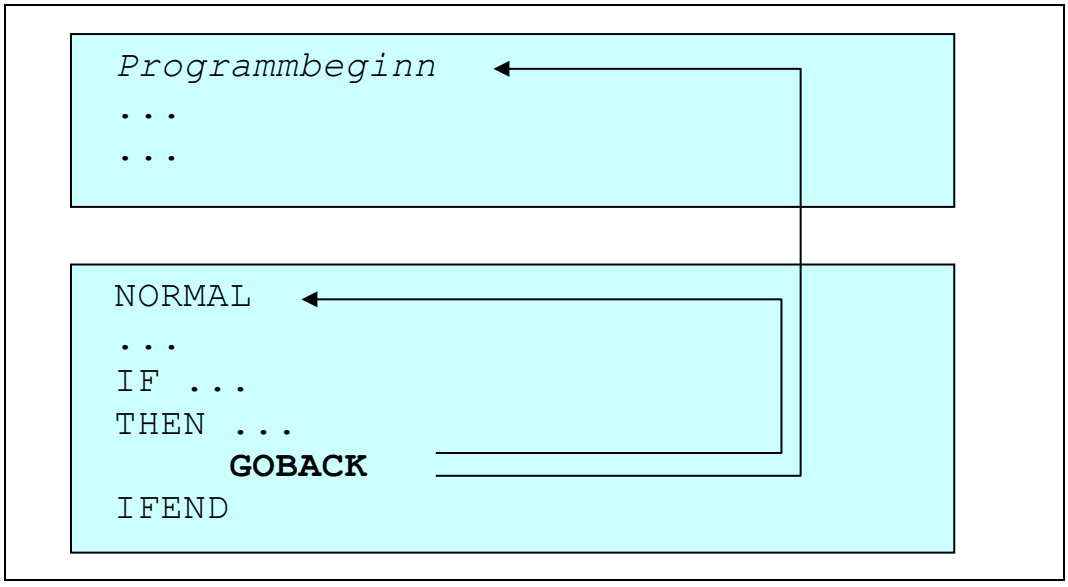


Abb. 52: Die GOBACK Anweisung

Die GOLAST Anweisung

Mit dieser Schlüsselwortanweisung wird direkt **hinter das nächstfolgende** `LAST`, oder wenn `LAST` nicht vorhanden ist, zum `END` verzweigt.

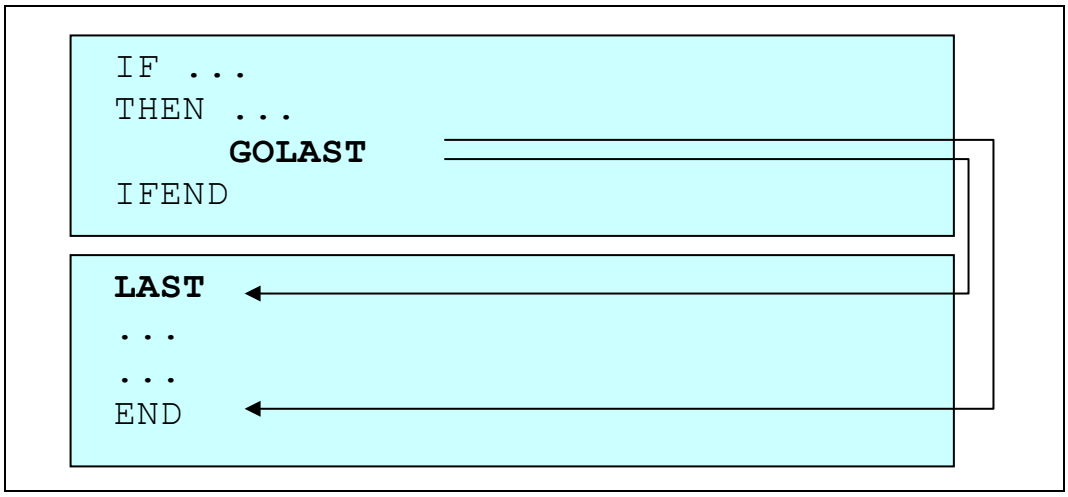


Abb. 53: Die GOLAST Anweisung

Die GO TO Anweisung

Dieser allgemein gültige Sprungbefehl kann dazu verwendet werden **vom Anwender definierte Labels** anzuspringen. Es ist dabei zu beachten, dass sich das Label selbst auf der hierarchischen Stufe 0 befinden muss. Das Label ist ein alphanumerisches Symbol von maximal 8 Stellen, beginnend mit einem Buchstaben. Abgeschlossen wird die Labeldefinition mit einem Doppelpunkt.

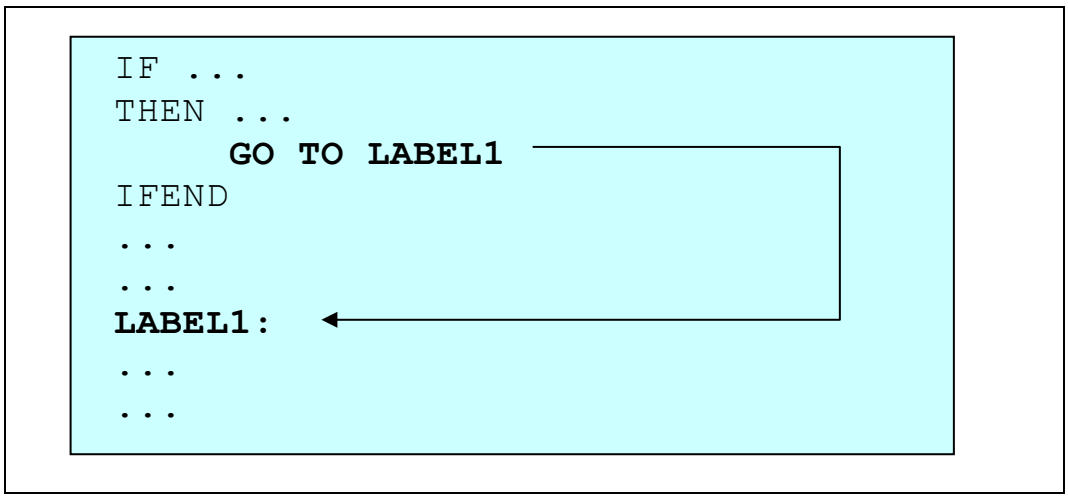


Abb. 54: Die GO TO Anweisung

Die GODUMP Anweisung

Mit dieser Anweisung wird die Verarbeitung an dieser Stelle mit einem **Dump** abgebrochen.

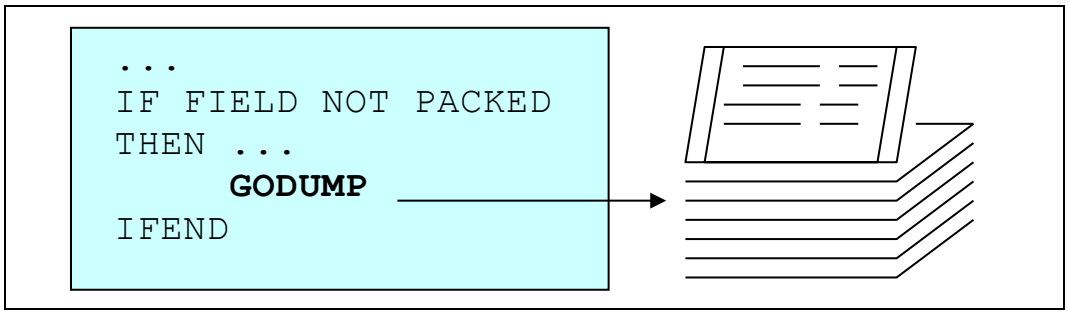


Abb. 55: Die GODUMP Anweisung

Die GOABEND [,nn] Anweisung

Mit dieser Schlüsselwortanweisung wird die Verarbeitung sofort abgebrochen (CANCEL). Es kann ein **Abendcode von 1-4095** definiert werden. Default ist 12.

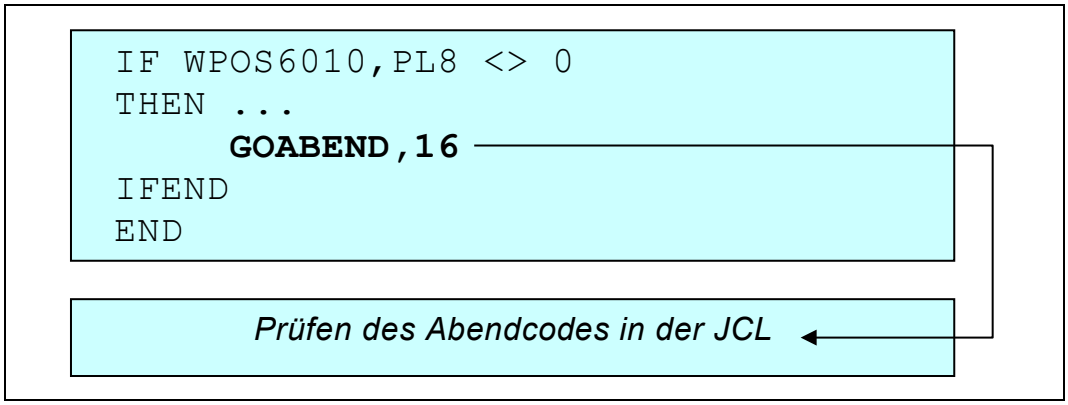


Abb. 56: Die GOABEND Anweisung

Die GOEND [,nn] Anweisung

Mit dieser Schlüsselwortanweisung wird direkt zum `END` verzweigt und damit die **Verarbeitung beendet**. Es kann ein **Returncode (Conditioncode) von 1-4095 definiert werden.**

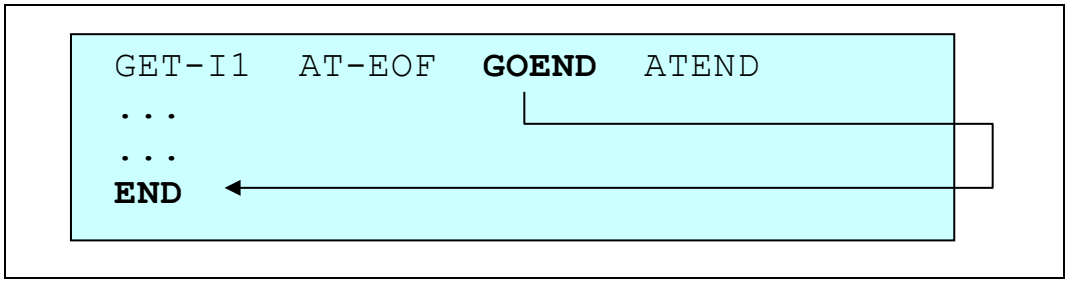


Abb. 57: Die GOEND Anweisung

Kapitel 5. Interne Verarbeitungslogik und Steuerung

Implizite Verarbeitungslogik

Bei der **Impliziten Verarbeitungslogik** werden verschiedene Prozesse wie `OPEN`, `CLOSE`, `GET`, `PUT` (oder `W` für Printer) automatisch generiert. Der Anwender muss nur die Dateninstruktionen und die Filedefinitionen definieren.

QPAC arbeitet mit impliziter Verarbeitungslogik, wenn den Fileidentifikationen **keine Nummer** beigefügt wird:

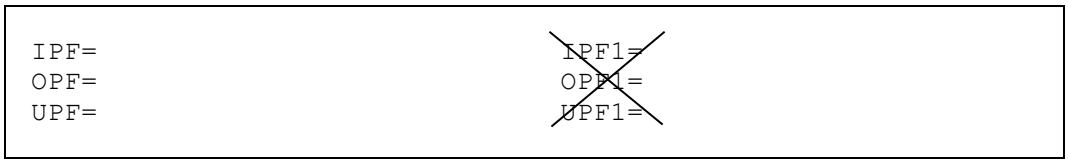


Abb. 58: Implizite Verarbeitungslogik mit Fileidentifikationen ohne Nummern

Daraus ergibt sich, dass normalerweise nur **je ein Inputfile und/oder je ein Outputfile, bzw. ein Updatefile** definiert werden.

Die implizite Verarbeitungslogik ist dazu gedacht, 'Singlefile' Operationen ohne Berücksichtigung von Input-/Outputbefehlen auf einfachste Art durchführen zu können. Es steht dem Anwender jedoch frei, Input-/Outputbefehle ausserhalb der feststehenden Logik noch zusätzlich zu verwenden:

- Unter impliziter Kontrolle enthält die **`IPF`** oder **`UPF`** Definition den Lesebefehl **`GET`**
- Ist eine `OPF` oder `UPF` Definition vorhanden, so enthält das **`END`** Statement einen **`PUT`** Befehl
- Ist **`NORMAL`** definiert, enthält es den impliziten **`GET`** Befehl, welcher dabei von der vorstehenden `IPF` oder `UPF` Definition entfernt wird.
- Ist **`LAST`** definiert, enthält es den **`PUT`** Befehl, welcher dabei vom nachstehenden `END` Statement entfernt wird.
- Inputrecords **vor einem `NORMAL`** müssen explizit mit einem **`GET`** gelesen werden.
- Outputrecords **nach einem `LAST`** müssen explizit mit **`PUT`** bzw. **`W`** geschrieben werden.
- Die automatische Steuerung für Printerfiles beinhaltet Seitenwechsel mit Druck von Titelzeilen, sowie Zeilenprint und Zeilenvorschub.

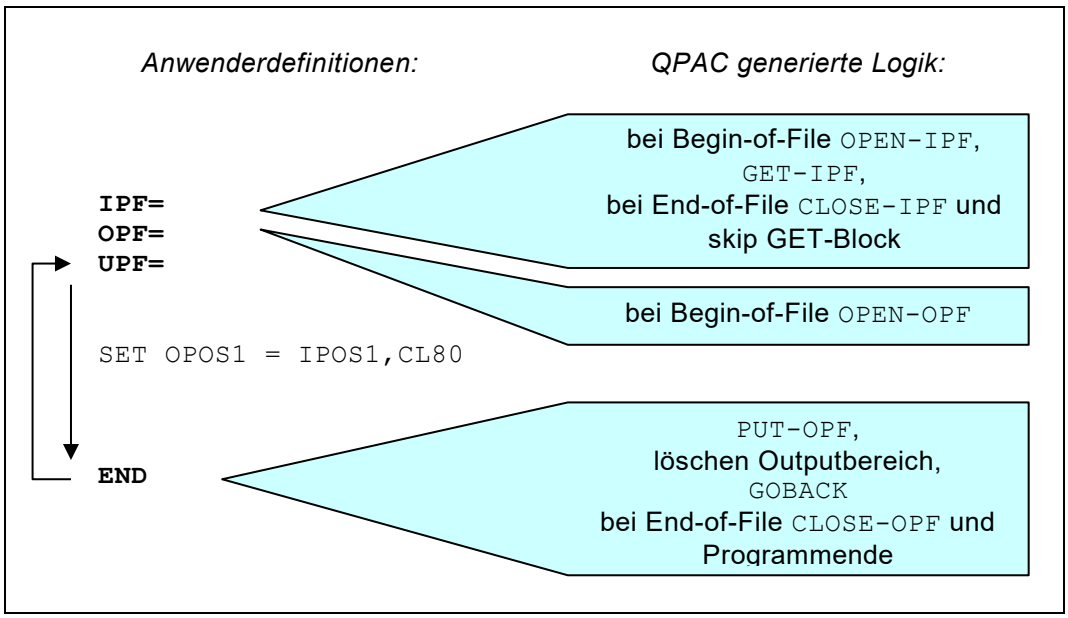


Abb. 59: QPAC-Logik bei Impliziter Verarbeitung

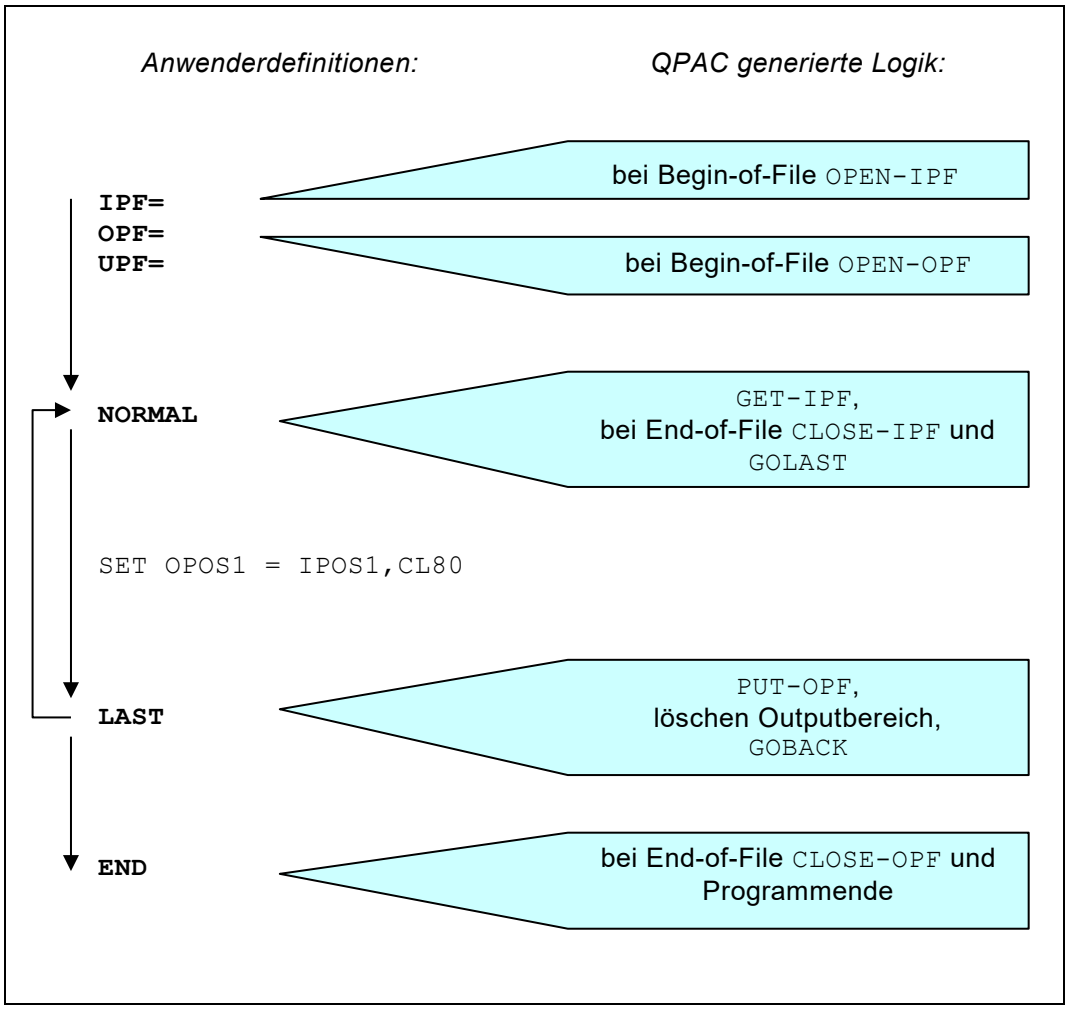


Abb. 60: QPAC-Logik bei Impliziter Verarbeitung mit NORMAL und LAST

Explizite Verarbeitungslogik

Bei der **Expliziten Verarbeitungslogik** müssen die Input-/Outputbefehle zusätzlich definiert werden.

Jegliche I/O-Automatik mit der Ausnahme von `OPEN` und `CLOSE` und Printfile Seitenwechsel sind in der expliziten Verarbeitungslogik ausgeschlossen.

QPAC arbeitet mit expliziter Verarbeitungslogik, wenn den Fileidentifikationen **Nummern** (von 1 bis 99) beigefügt werden:

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

Abb. 61: Explizite Verarbeitungslogik mit Fileidentifikationen mit Nummern

`UPF` schliesst `IPF` und `OPF` mit gleichen Nummern aus.

Die explizite Verarbeitungslogik gestattet die Bearbeitung von **mehreren Inputfiles und/oder mehreren Outputfiles** gleichzeitig. Die notwendigen zugehörigen Input-/Outputbefehle besitzen die referenzierende Fileidentifikation (mit Nummer) als Bestandteil der Befehlsform.

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

Abb. 62: I/O Instruktionen bei Expliziter Verarbeitungslogik

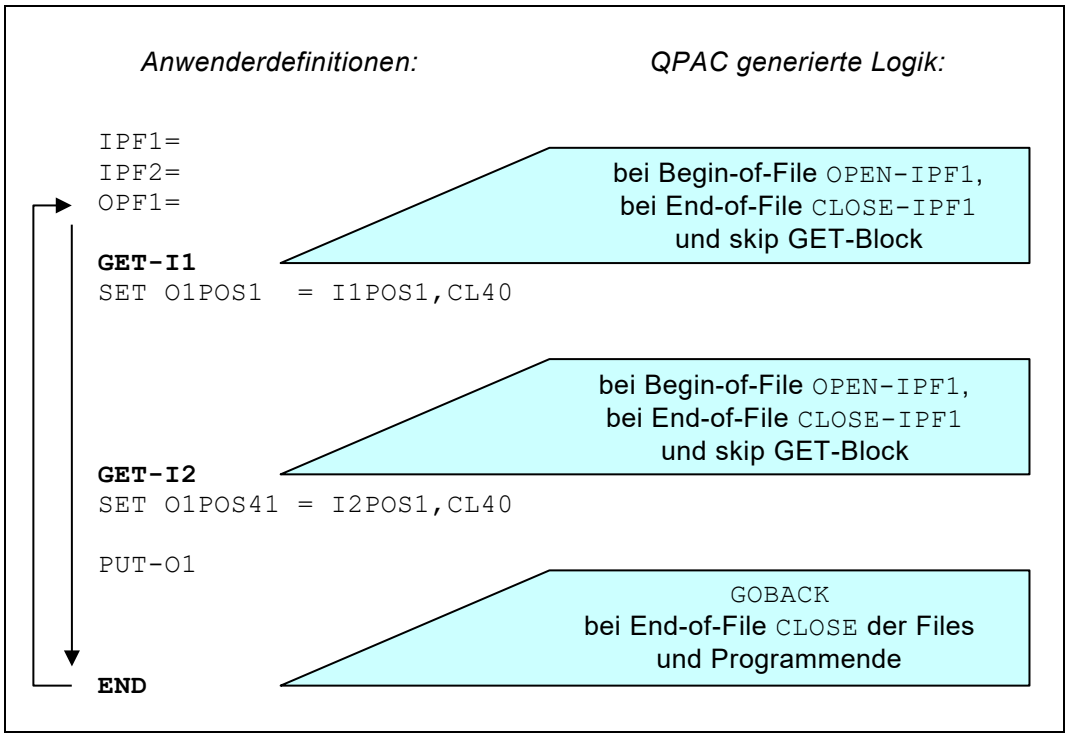


Abb. 63: QPAC generierte Logik bei Expliziter Verarbeitung

Die folgenden Regeln gelten, wenn mehrere Inputfiles verwendet werden:

- Alle Verarbeitungsdefinitionen, die einem GET-Befehl folgen, bilden einen sogenannten **GET-Block**.

  Tritt bei einem GET-Befehl EOF (End of File) auf, wird der gesamte GET-Block übersprungen.

  Wird ein GET-Befehl in der Verarbeitung durchgeführt, der einen File anspricht, bei dem schon früher EOF aufgetreten war, wird der GET-Block ebenfalls übersprungen.

  Die Verarbeitung gilt dann als beendet, wenn alle **leitenden Inputfiles** den Status EOF erreicht haben.

  Folgt einem GET-Befehl unmittelbar ein AT-EOF, so wird bei EOF-Status der EOF-Block, welcher mit ATEND abgeschlossen wird, durchlaufen.
  Der GET-Block wird in diesem Falle nicht übersprungen.

- Inputfiles gelten dann als **leitend**, wenn zugehörige GET-Befehle auf der hierarchischen Stufe 0 vorkommen, d.h. nicht ausschliesslich innerhalb von IF- bzw. DO-Blöcken oder SUBroutinen vorhanden sind.

  Sind für einen Inputfile zugehörige GET-Befehle nur innerhalb von IF- bzw. DO-Blöcken oder SUBroutinen vorhanden, d.h. in einer hierarchischen Stufe grösser als 0, gilt dieser Inputfile als **bedingungsabhängig** und hat für die Verarbeitungsbeendigung keinen steuernden Einfluss.

- GET-Blöcke werden durch NORMAL, LAST, END, durch einen neuen GET-Befehl, durch ein ELSE, IFEND oder auch durch ein DOEND oder SUBEND abgeschlossen.

  Dies bedeutet, dass der Blocksprung vom GET-Befehl mit EOF-Status zum jeweiligen Begrenzungspunkt erfolgt:
  Im NORMAL-Teil zum nächsten GET bzw. LAST bzw. END, im LAST-Teil zum nächsten GET bzw. END, in einem IF-Satz zum nächsten GET bzw. IFEND, in einer DO-Schlaufe zum nächsten GET bzw. DOEND usw.

- Sofern mit dem Instruktionsformat des alten und inzwischen überholten **QPAC-Basisteils** gearbeitet wird, werden in einem GET-Block die **Inputadressen implizit dem durch den GET-Befehl adressierten Inputfile zugeordnet**.

  Es ist zu beachten, dass dies nicht der gleiche File sein muss, wenn ein GET-Befehl in einem IF-Satz vorhanden ist, wie nach dem zugehörigen IFEND.
  Siehe hierzu Anhang A: Basis Instruktionsformate.

  Wird hingegen mit Symbolen - impliziten oder expliziten - gearbeitet, wird diese Regel bedeutungslos.

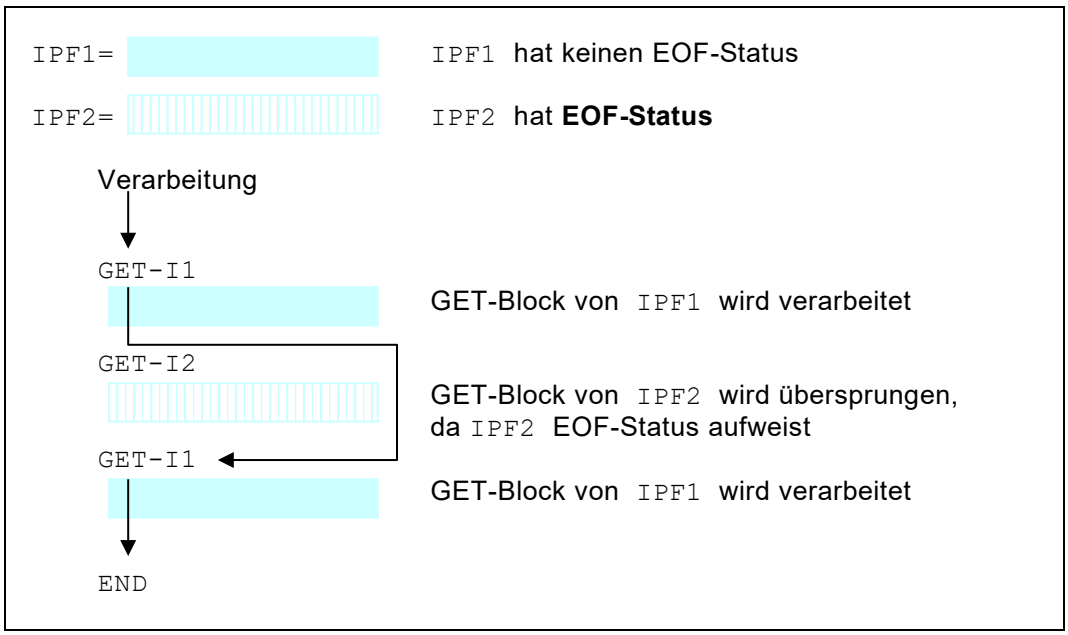


Abb. 64: EOF-Steuerung ohne AT-EOF Definition

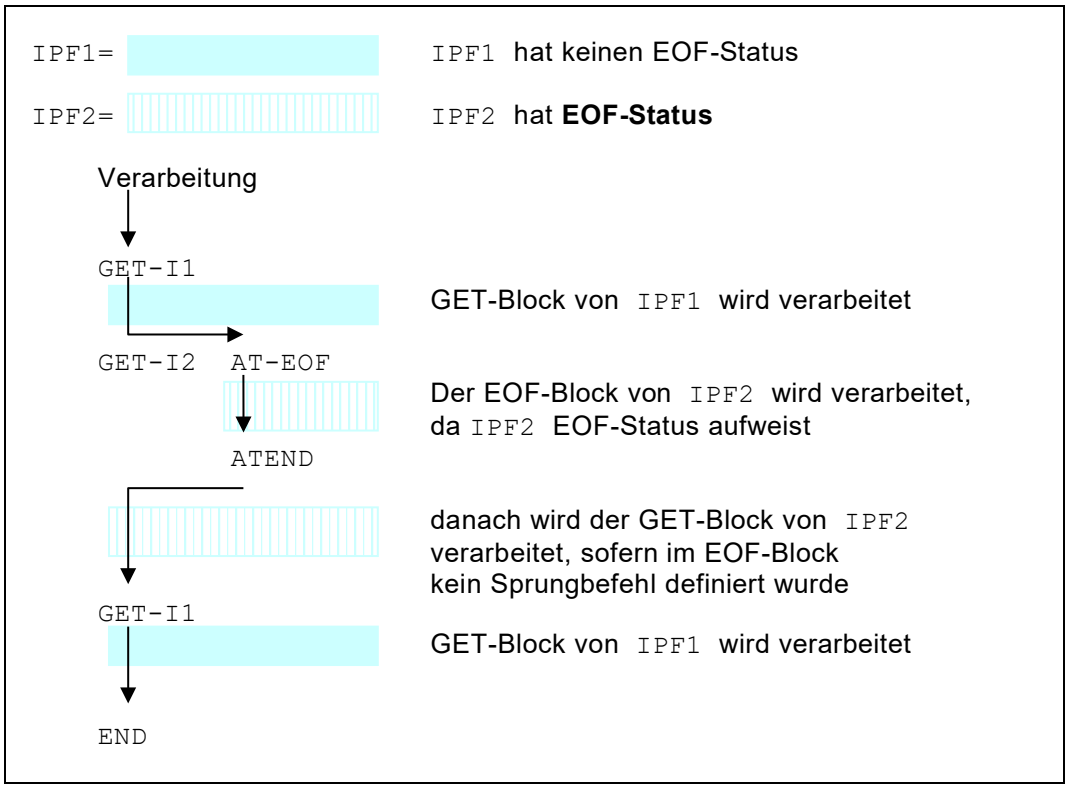


Abb. 65: EOF-Steuerung mit AT-EOF Definition

Kapitel 6. Felddefinitionen und Symbolzuordnungen

Übersicht und Hinweise

Den einzelnen Feldern in den Recordstrukturen der Datasets können Symbole zugeordnet werden. Zusätzlich können im internen **Working Storage** sowie im **Hiperspace** beliebige Felder mit Symbolen versehen werden, die in erster Linie als Zwischenspeicher dienen können. Bei der Namensvergabe ist zu beachten, dass es **reservierte Symbolnamen** gibt, die durch QPAC bereits intern vordefiniert wurden. Den verwendeten Symbolen wird die Feldlänge und das Feldformat beigefügt. QPAC verwendet diese Informationen, um bei den Instruktionen automatisch die notwendigen Feldkonversionen vornehmen zu können.

Der Interne Workbereich (Unterhalb der 16 MB Linie)

Ein interner Workbereich von 32767 Bytes steht dem Anwender per Default zur Benutzung zur Verfügung.
Auf diesen Bereich wird mit den Adressen `WPOS1 - WPOS32767` zugegriffen. Diese Adresswerte werden in diesem Manual mit *`wadr`* bezeichnet. Dieser Bereich kann über `PARM=WORK=` bis maximal 16 MB vergrössert werden. Ein `WORK` Bereich grösser als 1 MB wird oberhalb der 16 MB Grenze angelegt.

Zur QPAC-**Initialisierungszeit** ist der Workbereich folgendermassen vorformatiert:

1	-	4999	*nicht vorbestimmt*
5000	-	5999	Low Value X'00'
6000	-	6999	alle 10 Bytes wie folgt: X'000000000000000C4040
7000	-	7999	Blank X'40'
8000	-	8999	High Value X'FF'
9000	-	9999	Low Value X'00'
10000	-	32767	*nicht vorbestimmt*

Abb. 66: Der vorformatierte interne Workbereich

Der Interne Hiperspace (Oberhalb der 16 MB Linie)

Auf diesen Bereich wird mit den Adressen `HPOS1` – `HPOS`*`nnnnn`* zugegriffen.

Der Externe Workbereich (ausserhalb des QPAC-Programms)

Auf diesen Bereich wird mit den Adressen `XPOS1` – `XPOS`*`nnnnn`* zugegriffen.

Die Beschreibung dieses externen Workbereiches ist unter dem Kapitel 9. Subroutinen und Externe Programme zu finden.

Die Implizite Symbolzuordnung

QPAC kennt einerseits das **Positionssymbol** und andererseits **reservierte Symbolnamen**, denen intern eine feste Bedeutung zugeordnet ist.

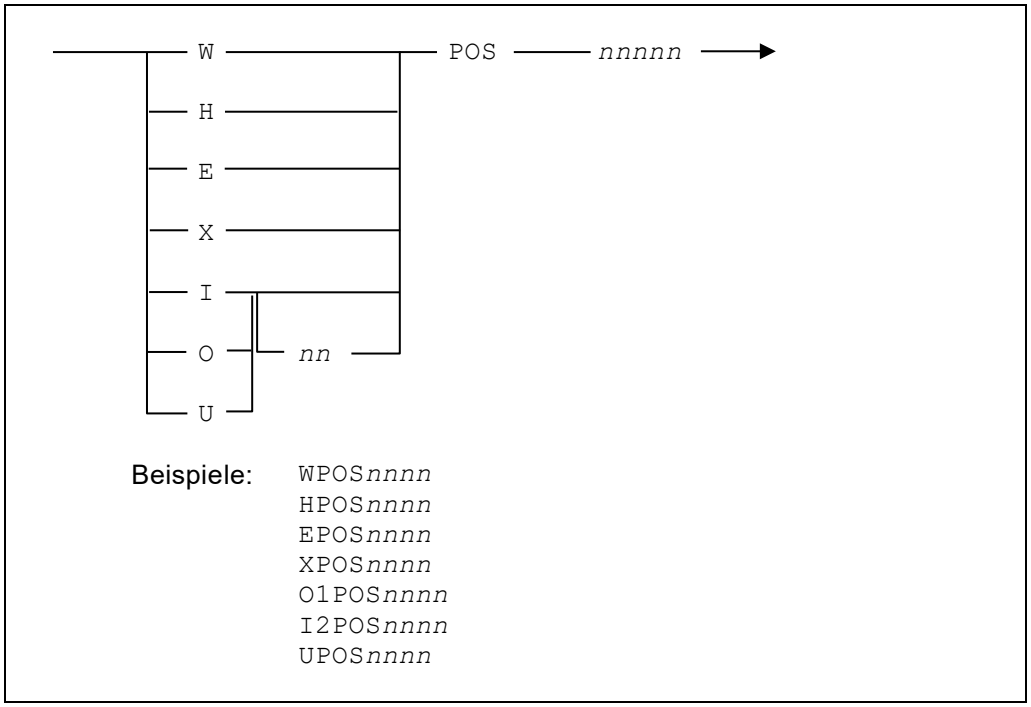


Abb. 67: I/O Instruktionen bei Expliziter Verarbeitungslogik

Implizite Positionssymbole sind nichts anderes als direkte Positionsadressen in Symbolform. Dabei unterscheiden wir zwischen Positionsadressen für den internen Workbereich, den Hiperspace, den Externen Bereich und solchen für die I/O-Bereiche.

Positionsadressen z.B. für den internen Workbereich beginnen mit `WPOS` und der anschliessenden Positionsangabe.

Positionsadressen für bestimmte Filebereiche beginnen mit der **Kurzform** der Fileidentifikation vor dem Wort `POS`, gefolgt von der Positionsangabe.

```
OPOS11          (Position 11 in der Recordarea OPF=)
UPOS1           (Position 1 in der Recordarea UPF=)
WPOS6000        (Position 6000 im internen Workbereich)
HPOS300         (Position 300 im Hiperspace)
I1POS50         (Position 50 in der Recordarea IPF1=)
O2POS30         (Position 30 in der Recordarea OPF2=)
```

Abb. 68: Implizite Positionssymbole

Jeder symbolischen Positionsadresse kann unmittelbar an verwendeter Stelle bei Konversionsinstruktionen auch das Feldformat mit Länge beigefügt werden, durch Komma getrennt.

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

Abb. 69: Explizite Angabe von Feldformaten

Die Explizite Symbolzuordnung

Hier unterscheiden wir zwischen Felddefinitionen, die einem bestimmten Bereich zugeordnet werden, (einer I/O-Area, dem internen Workbereich, dem Hiperspace oder dem Externen Bereich) oder solchen, die als Single Fields bzw. Literale gelten. Single Fields sind keinem bestimmtem Bereich, weder einer I/O Area, dem internen Workbereich noch dem Hiperspace zugeordnet, sie stehen für sich allein und werden irgendwo im dynamischen Hauptspeicher abgelegt. Single fields dienen im Allgemeinen als einfache Zwischenspeicher oder dergleichen.

Definitionen für einen bestimmten Bereich werden mit einer Positionsadresse von 1 bis *nnnnn* (Bereichsende) versehen. Single Fields und Literale werden mit einer Positionsadresse 0 versehen. Ausserdem wird (ausser bei I/O- Bereichen) nach der Positionsadresse ein Identifikator gesetzt, welcher aussagt, zu welchem Bereich die Zuordnung gilt, oder ob es sich um ein Single Field oder ein Literal handelt.

*nnnn***W**=*Symbol*	gilt für den internen Workbereich
*nnnn***H**=*Symbol*	gilt für den Hiperspace
*nnnn***E***Symbol*	gilt für den EXCI Communication Bereich
nnnn=*Symbol*	gilt für den I/O-Bereich der vorausgehenden Filedefinition
0S=*Symbol,Format-Länge*	gilt als Single Field
0L=*Symbol,Wert*	gilt als Literal

Abb. 70: Explizite Symboltypen

Grundformat der Expliziten Symbolzuordnung für Single Fields und Literale

0L=*Symbol,Format-Länge,Wert*	Literale
0S=*Symbol,* — *Format-Länge[,Wert]*	Single Field
' '	Character Literal
C' '	
CL*n*' '	
V*n*' '	Variabel Character
VL*n*' '	
X' '	Hexadezimal Literal
XL*n*' '	
Z'*nnnnn*'	Zoned Dezimal Literal
ZL*n*'*nnn*'	
P'*nnnnn*'	Packed Dezimal Literal
PL*n*'*nnn*'	
BL*n*'*nnn*'	Binär Literal
B'........'	Bit Literal
B'........'	ist ein Spezialformat als Bit-Literal, wenn kein Längenattribut angegeben wird. Es hat eine fixe Länge von 8 Bit (1 Byte) und kann nur 0 oder 1 enthalten.

Abb. 71: Grundformat explizite Symbolzuordnung für Single Fields und Literale

Grundformat der Expliziten Symbolzuordnung für Bereiche

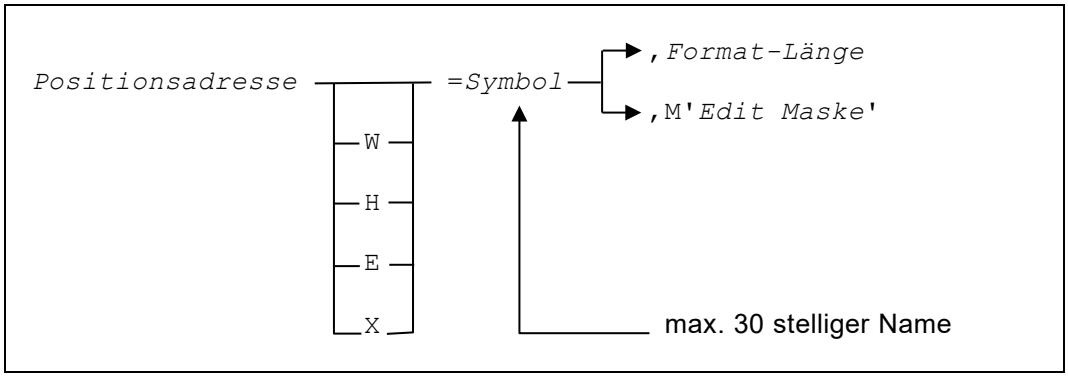


Abb. 72: Grundformat explizite Symbolzuordnung für Bereiche

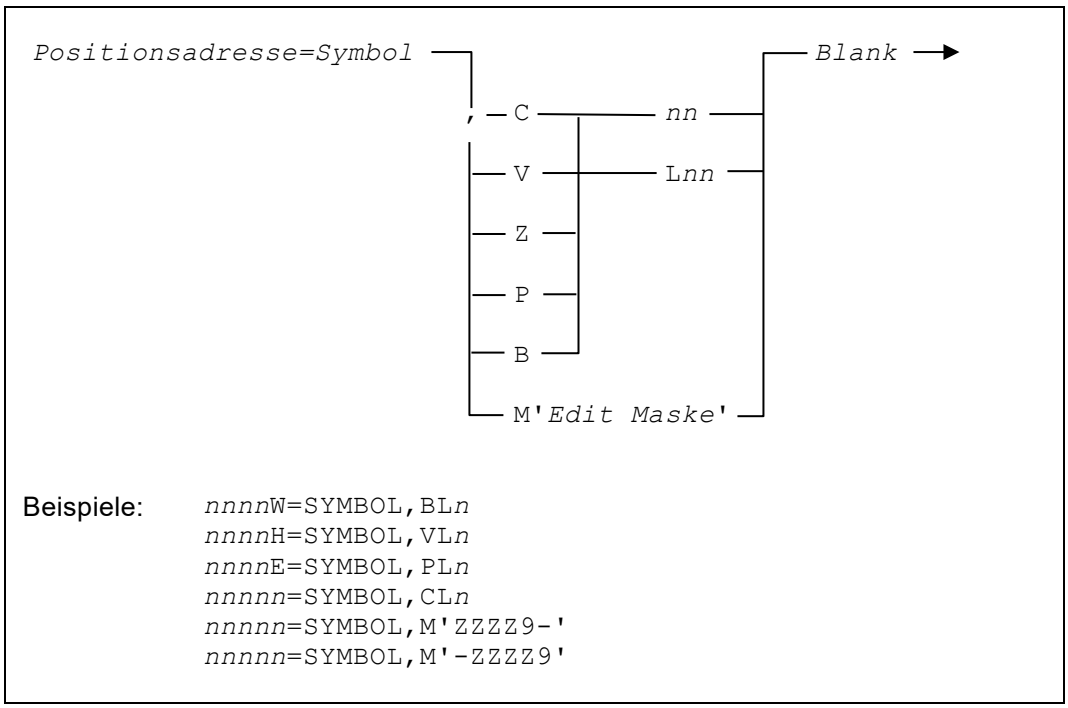


Abb. 73: Diagramm explizite Symbole für Bereiche

Links des Gleichheitszeichens wird die Positionsadresse (eventuell ergänzt mit `W` für den internen Workbereich, `H` für den Hiperspace oder `E` für den externen EXCI Bereich) definiert, der das Symbol, das rechts des Gleichheitszeichens steht, zugeordnet werden soll. Links und rechts des Gleichheitszeichens sind **keine** Blanks zulässig.
Nach dem Symbol kann wahlweise eine Länge, kombiniert mit dem Feldformat, definiert werden. Die Länge ist in **Bytes** zu verstehen.

Es sind fünf Feldformate unterstützt:

, CL*n*	oder	, C*n*	Character	(1 – 9999999)
, VL*n*	oder	, V*n*	Variabel Character	(2 – 32767)
, ZL*n*	oder	, Z*n*	Zoned decimal	(1 – 20)
, PL*n*	oder	, P*n*	gepackt	(1 – 16)
, BL*n*	oder	, B*n*	binär	(1 – 8)

Abb. 74: Feldformate für Konversionsinstruktionen

Zu beachten bei Variabel Character Feldern (`VLn`):
Bei Variabel Character Feldern wird automatisch ein 2-Bytes Binärfeld vor das eigentliche Feld platziert. Diesem Längenfeld wird der gleiche Symbolname mit einem angehängten Nummernzeichen (#) zugeteilt. V entspricht sonst dem Character Attribut und kann nicht als Literalfeld definiert werden.

```
    1=RECORDTYPE,CL1
 5010=ARTICLE,CL30          (W wird angenommen, wenn der Wert
                             ausserhalb der I/O Areagrösse liegt)
5010W=ARTICLE,CL30
 7100=QUANTITY,P5           (W wird angenommen)

   1H=HIPERFIELD,CL5        (Hiperspace)
   1E=EXCIFIELD,BL2         (CICS EXCI Kommunikation)
   1X=EXTERNALFIELD,CL100   (QPAC als Subroutine)
```

Abb. 75: Beispiele von expliziten Symbolen mit Feldformaten

Anstelle eines Feldformates kann auch eine Edit Maske für numerische Werte definiert werden. Die Editmaske bestimmt zugleich die Länge. Ziffernstellen werden mit **9** definiert. Führende Stellen, bei denen Nullenunterdrückung gewünscht wird, werden mit **Z** anstelle 9 gekennzeichnet.
Die Maske kann mit einem – oder + Zeichen abgeschlossen werden.
Ein **+** Zeichen bleibt bei positivem Wert erhalten, wird jedoch bei negativem Wert durch ein – Zeichen ersetzt.
Ein **–** Zeichen wird bei positivem Wert auf Blank gesetzt und bleibt bei negativem Wert erhalten.
Ein **–** Zeichen kann auch links in der Maske angegeben werden. Dann wird es zum gleitenden Minuszeichen.
Die ganze Maske steht zwischen Apostrophs und einem führenden Attribut `M`:

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

Abb. 76: Numerische explizite Symbole mit Edit Masken

Symbolnamen können alphanumerisch sein. Sie müssen mit einem Alphazeichen beginnen und können die Spezialzeichen $ # @ _ enthalten.

Es ist zu beachten, dass **reservierte Symbolnamen** existieren.

Vereinfachtes Format der Expliziten Symbolzuordnung

Eine vereinfachte Definitionsform von Feldstrukturen kann dadurch erreicht werden, wenn hinter einer Positionsangabe Felder definiert werden, die ohne Positionsangabe direkt mit einem führenden Gleichheitszeichen an das vorhergehende Feld angeschlossen werden.
Das Pseudosymbol `FILLER` kann als Platzhalter verwendet werden.

```
1=RECORDTYP,CL1
 =ARTIKEL,CL30
 =MENGE,PL3
 =PREIS,PL5
 =FILLER,CL3
 =LAGERORT,CL1
```

Abb. 77: Vereinfachtes Format der Expliziten Symbolzuordnung

Redefines bei Strukturen

Innerhalb von Strukturen kann ein einzelnes Feld redefiniert werden, wenn vor dem Gleichheitszeichen ohne Positionsangabe ein `R` angegeben wird (`R=`).
Der Redefine (`R=`) startet immer beim Anfang des vorausliegenden Feldes.

Workarea Struktur:

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

Abb. 78: Redefinieren von Workarea Strukturen

Single Field Struktur:

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

0S=FIELDZ,CL5
```

Abb. 79: Redefinieren von Single Field Strukturen

Literal Struktur:

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

Abb. 80: Redefinieren von Literal Strukturen

Based Struktur:

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

Abb. 81: Redefinieren von Based Strukturen

I/O Struktur:

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

Abb. 82: Redefinieren von I/O Strukturen

Explizite Symbole für Filedefinitionen

Die Zuordnung der Positionsadressen an die zugehörige Recordarea der entsprechenden Filedefinition unterliegt nachfolgend aufgeführter Regel:

a) Nach einer Filedefinition werden die Symbole dieser Eingabe- / Ausgabe Area zugeordnet:

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

Abb. 83: Filеareazuordnung bei Expliziten Symbolen

b) Nach einer Input-Operation werden die Symbolpositionen der Area des Files, das durch die Operation angesprochen wird, zugeordnet.

c) Die Zuordnung von Symbolen an den internen Working Storage Bereich kann grundsätzlich überall erfolgen, da diese Zuordnungskriterien von keinen Filedefinitionen abhängig sind. Sie unterliegen deshalb auch keiner Strukturblock-Hierarchie. Positionsadressen ab 5000 werden nur dem internen Workbereich zugeordnet, sofern der I/O-Bereich der vorausliegenden Filedefinition kleiner ist.

Die Zuordnungsregel richtet sich ebenfalls nach der hierarchischen Ordnung, d.h. nach einem logischen Strukturblockende gilt die Zuordnung, wie sie vor dem Strukturblockbeginn bestand.

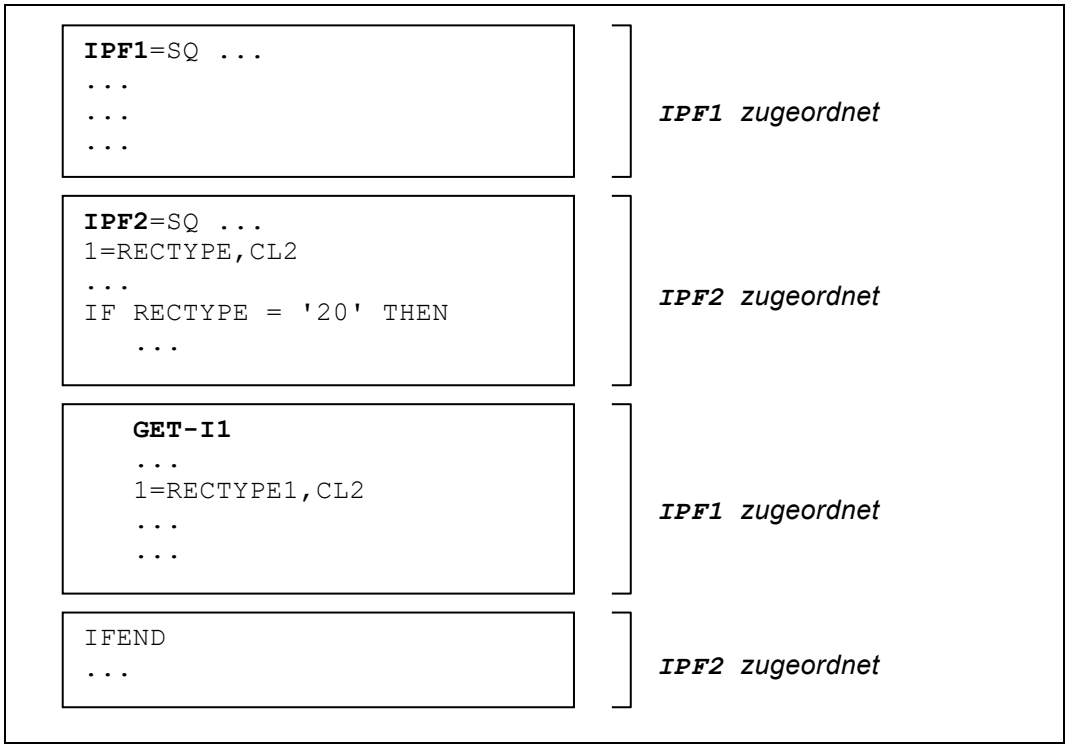


Abb. 84: Hierarchische Ordnung der Symbolzuordnung

Explizite COBOL und PL/I Recordstruktur-Zuordnung

Katalogisierte COBOL und PL/I Recordstrukturen können mit dieser Symboldefinition aus einer Sourcelibrary geladen und einer Filedefinition oder einem Workbereich zugeordnet werden. Die Feldnamen werden zu QPAC Symbolnamen konvertiert (- Zeichen werden zu _ Zeichen konvertiert).
Initialwerte und Editmasken werden ignoriert.
Based-Definitionen innerhalb von PL/I Copy Books sind mit Einschränkungen unterstützt.

```
Positionsadresse=COBREC=bookname
Positionsadresse=PLIREC=bookname
```

Abb. 85: Diagramm explizite COBOL und PL/I Recordstruktur-Zuordnung

Positionsadresse kann eine Record- oder eine Workareaposition (*nn*W=) sein.
bookname ist der Name einer katalogisierten COBOL oder PL/I Recordstruktur.
COBREC= sowie PLIREC= sind Schlüsselwörter.

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

7001W=COBREC=F4200RD
```

Abb. 86: Importieren einer bestehenden COBOL Recordstruktur

Unter z/OS werden diese Copy Books über den PDS DD-Namen //QPACCOPY eingelesen.

Zu beachten bei PL/I Books:
Bei BASE(ADDR(SYMBOL)) wird dem Symbol, als PTR Feld definiert, zusätzlich ein #-Zeichen angehängt.

BASED Strukturen

In QPAC können Based-Strukturen definiert werden, mit denen auf einfache Weise Tabellenverarbeitungen vorgenommen werden können, ohne dass die einzelnen Tabellenelemente mit Hilfe von Indexregistern indiziert werden müssen. In der Assembler-Sprache entspricht diese Art den sogenannten DSECTs (Dummy Sections).

Einer Based-Struktur muss vorangestellt ein Pointer-Feld definiert werden, dessen Inhalt jeweils die aktuelle Speicheradresse sein muss. Die Felder der nachfolgenden Struktur werden während der Ausführung in Abhängigkeit des Pointers adressiert. Der Pointerinhalt kann durch Addition oder Subtraktion von Werten verändert werden.

Initialisiert wird der Pointer-Inhalt durch einen Direktwert oder durch die Adresse eines anderen Feldes. Für den letzteren Fall gibt es eine `SET` Spezialwert Instruktion `=ADR`, die besagt, dass die Adresse des Sendefeldes in den Pointer, der als Empfangsfeld definiert wurde, geladen werden soll.

```
0B=POINTER,PTR
1B=ADRESSFELD1,BL4
  =FELD2,PL3
8B=FELD3,CL5
...
```

Abb. 87: Grundformat der Based-Struktur

Zwischen Positionsadresse und Gleichheitszeichen (=) wird der Identifikator "`B`" gesetzt. Dieser besagt, dass es sich um eine Based-Struktur handelt.

Der Kopf einer Based-Struktur wird durch das zugehörige Pointerfeld definiert. Intern entspricht das Pointerfeld einem 4 Byte Binärfeld.

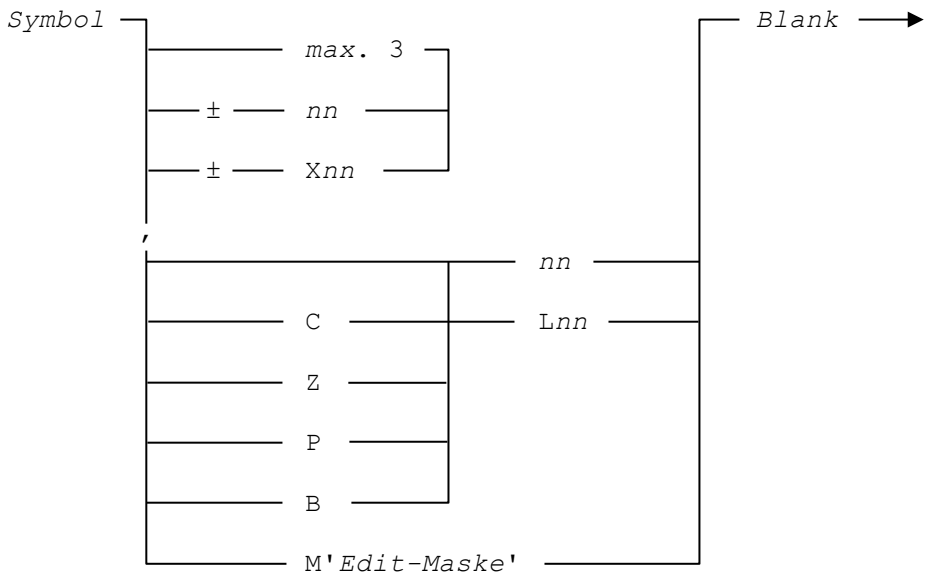
```
SET POINTER =ADR WPOS10001

oder

SET POINTER = nnnnn
```

Abb. 88: Laden des Pointerfeldes

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

Abb. 89: Verschieben einer Struktur

Die Symbolische Indizierte Adressierung



Beispiele:		
	Symbol,Formatlänge	MENGE,PL5
	Symbol±X*n*	MENGE+X1-X2
	Symbol±*n*	MENGE+5
	Symbol±X*n,fl*	MENGE+X1,PL4
	Symbol±*n,fl*	MENGE+5,PL5
	Symbol±*n*±X*n*	MENGE+5+X1
	Symbol±*n*±X*n,fl*	MENGE-5+X1,BL1

Abb. 90: Diagramm symbolische indizierte Adressierung

Eine Adressmodifikation kann durch Definition eines absoluten Wertes oder durch ein Indexregister für die dynamische Form vorgenommen werden.
Indizierte Adressierung erfolgt, wenn ein **Indexregister** an ein Feldsymbol mit einem Plus- oder Minuszeichen **angehängt** wird.
Maximal 3 Indexregister können angehängt werden.

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

Abb. 91: Indizierte Adressierung durch Anhängen von Indexregistern

Eine implizite Formatlängen-Zuordnung kann temporär durch eine explizite Definition überschrieben werden.

```
7001=DATUMS_TABELLE,CL72                *. 12 MONATE
7001=EINGANGS_DATUM,ZL6                 *. YYMMDD
X1=0                                    *. INIT OFFSET X1
DO-12
   SET EINGANGS_DATUM+X1,ZL2 = 92       *. INIT JAHR
   X1+6                                 *. NAECHSTES YY
DOEND
```

Abb. 92: Explizite Längenangabe überschreibt implizite Längendefinition

Reservierte Feldsymbole

Reservierte Symbolnamen sind interne Felder oder Register, die dem Anwender zur Verfügung gestellt werden, ohne dass sie erst explizit definiert werden müssen.

Nachfolgende Symbole entsprechen einer vorgegebenen Bedeutung in Bezug auf Format und Länge und sind reservierte Namen innerhalb von QPAC-Batch.

`..` Einige Feldsymbole enthalten die entsprechende Fileidentifikation in ihrem Symbolnamen, durch zwei Punkte dargestellt (z.B. `..KEY`). Ersetzen Sie diese durch eine gültige Fileidentifikation in der Kurzform.

Reservierte Feldsymbole gruppiert

Symbolname	Beschreibung		Formatlänge
X*n*	Indexregister (*n* = 1-99)		BL4
ACCU*n*	Rechenfelder (*n* = 0-99) sind fest den Workbereichpositionen 6000-6990 zugeordnet		PL8
FILLER	Platzhalter für Explizite Symbolzuordnungen		
GPOS*nnnn*	Globale Workbereich Position (*nnnn* = 1-32767)		
WPOS*nnnn*	Interne Workbereich Position (1-*nnnn*)		
HPOS*nnnn*	Interne Hiperspace Position (1-*nnnn*)		
EPOS*nnnn*	EXCI Communication Bereich Position (1 - *nnnn*)		
XPOS*nnnn*	Externer Bereich Position (1 - *nnnn*)		
I[*n*]POS*nnnn*	Recordarea Position *nnnn* von IPF[*n*]		
O[*n*]POS*nnnn*	Recordarea Position *nnnn* von OPF[*n*]		
U[*n*]POS*nnnn*	Recordarea Position *nnnn* von UPF[*n*]		
D*n*POS*nnnn*	Recordarea Position *nnnn* von DS*n*, DB*n*		
T*n*POS*nnnn*	Recordarea Position *nnnn* von TD*n*, TS*n*		
S*n*POS*nnn*	Recordarea Position *nnn* von SPO*n*		
DIVREM	Division Remainder Feld		PL8
DB2COMMIT	DB2 Auto Commit Counter		PL8
USDATE	Systemdatum US Format	'MM/DD/CCYY'	CL10
EDATE	Systemdatum EURO Format	'DD.MM.CCYY'	CL10
ISODATE	Systemdatum ISO Standard	'CCYY-MM-DD'	CL10
DATE	Systemdatum	'DD.MM.YY'	CL8
DAY	Tag des Systemdatums	DD	ZL2
MONTH	Monat des Systemdatums	MM	ZL2
YEAR	Jahr des Systemdatums	YY	ZL2
TIME	QPAC Startzeit	'HH:MM:SS'	CL8
HOUR	Stunde der Startzeit	HH	ZL2
MINUTE	Minute der Startzeit	MM	ZL2
SECOND	Sekunde der Startzeit	SS	ZL2
CDATE	aktuelles laufendes Datum	DDMMYY	PL4
CTIME	aktuelle laufende Zeit	HHMMSS	PL4
CDTIME	laufendes Datum & Zeit (Achtung: nur im Basis-Instruktionsformat gültig)	DDMMYYHHMMSS	2xPL4

Symbolname, Beschreibung, Formatlänge			
CCDATE	aktuelles Jahrhundert-Datum	CCYYMMDD	PL5
CCYEAR	aktuelles Jahrhundert-Jahr	CCYY	ZL4
RC	Returncode	Spezialregister	BL4
FC	Functioncode	Spezialregister	BL4
IV	Intervallwert	Spezialregister	BL4
EPARM	Externer Parameterbereich	max.	CL100
EPARML	Länge des externen Parameterwertes		BL2
FUNCMSG	Function Return Message		CL72
USERID	QPAC-Online User Identifikation		CL16
PRGNAME	QPAC-Online Programmname		CL16
MAPNAME	QPAC-Online aktiver Mapname		CL16
APPLID	VTAM CICS Applikations Identifikation		CL8
NETNAME	VTAM Terminal Netzname		CL8
CUSERID	CICS User Identifikation		CL8
CICSID	CICS System Identifikation		CL4
TERMID	CICS Terminal Identifikation		CL4
OPERID	CICS Terminal Operator Identifikation		CL3
CURSOR	Cursorposition nach GET MAP		BL4
MAXROW	Maximale Zeilen pro Bildschirm		BL2
MAXCOL	Maximale Spalten pro Bildschirm		BL2
JOBNAME	JCL Jobname		CL8
STEPNAME	z/OS JCL Stepname		CL8
JOBNUM	z/OS JCL Jobnummer		ZL5
JOBACTELNO	z/OS Job Accounting Element No		BL4
JOBACTINFO	z/OS Job Accounting Info		CL144
JOBCLASS	z/OS Job Klasse vom JOB Statement CLASS=		CL1
JOBCLASSLG	z/OS lange Job Klasse vom JOB Statement CLASS= oder //*MAIN CLASS=		CL1
JOBPROGRNM	z/OS Programmer Name vom JOB Statement		CL20
JOBSTIME	z/OS Job Start Time hhmmss		PL4
SYSNAME	z/OS System Name / SYSID		CL8
RACFUSER	z/OS RACF User Id		CL8
CALDRDATE	CALENDAR Datum Format CCYYMMDD		ZL6
CALDRWKNR	CALENDAR Wochennummer (01-53)		ZL2
CALDRWKDY	CALENDAR Wochentag (1-7)		ZL1
CALDRWARN	CALENDAR Warnung für Tagesanpassung Ende Monat		CL1
CALDRTXMT	CALENDAR Monatsname (January – December)		CL10
CALDRTXWD	CALENDAR Tagesname (Monday – Sunday)		CL10
C.FCNT	EXECSQL Fetched Counter		PL8
..GCNT	GET sequentieller Lesezähler		PL8

Symbolname, Beschreibung, Formatlänge		
..RCNT	READ Random Lesezähler MAP Receivezähler	PL8
..PCNT	PUT Recordzähler Seitenzähler (Printfiles)	PL8 PL4
..LCNT	Zeilenzähler pro Seite (Printfiles)	PL2
..MAXLCNT	Maximale Anzahl Zeilen pro Seite (Printfiles)	PL2
..UCNT	Updated Recordzähler UPF File	PL8
..ACNT	Added Recordzähler UPF File	PL8
..DCNT	Deleted Recordzähler UPF File	PL8
..SCNT	MAP Sendezähler	PL8
..FN	Filename	CL8
..FORM	Printer FORM Name	CL8
..FT	VSAM Filetyp KS,ES,RR	CL2
..RL	definierte maximale Recordlänge	BL2
..BL	definierte maximale Blocklänge	BL2
..KL	definierte Keylänge	BL2
..KP	definierte Keyposition	BL2
..DSN	Data Set Name	CL44
..DDN	DD Name DD Name für dynamische Zuordnung	CL8
..DBN	FCA DL/I Datenbankname DB Name für dynamische Zuordnung	CL8
..PSB	FCA DL/I PSB Name PSB Name für dynamische Zuordnung	CL8
..SEGNM	FCA DL/I Segmentname	CL8
..LEV	FCA DL/I Segmentlevel	ZL2
..SSA*n*	FCA DL/I SSA Felder (*n* = 1 - 8)	CL256
..SSAN	FCA DL/I Anzahl verwendeter SSA Felder	BL1
..SEGLENG	FCA DL/I Segmentlänge (ohne DBCTL)	BL4
..KFBAREA	FCA DL/I key feedback area	CL256
..KFBALENG	FCA DL/I key feedback area length	BL4
..PCB	PCB Nummer für dynamische Zuordnung	BL4
..RTN	Rootsegmentname für dynamische Zuordnung	CL8
..KFN	Keyfeldname für dynamische Zuordnung	CL8
..KEY	FCA Keyfeld	CL236
..RBA	FCA Relative Byte Adresse	BL4
..RRN	FCA Relative Record Nummer	BL4
..LENG	FCA Recordlänge	BL4
..RC	FCA Returncode (VSAM, DL/I, Queues, ...)	CL2
..RC1	FCA Returncode Position 1	CL1
..RC2	FCA return code position 2	CL1
..SQLCODE	FCA SQL Returncode	BL4
..WHERE	*für internen Gebrauch (DB2 Support)*	
..ROWLENG	FCA SQL Row Länge	BL4
..TBN	FCA SQL Tabellenname	CL18
..QNM	FCA Queue Name	CL8
..ITEM	FCA Queue Item	BL2

Symbolname, Beschreibung, Formatlänge

..MEMDIRVV	FCA Version aus Statistikrecord		PDS	BL1
..MEMDIRMM	FCA Modification aus Statistikrecord		PDS	BL1
..MEMDIRCRDT	FCA Creation Date aus Statistikrecord		PDS	PL5
..MEMDIRCHDT	FCA Changed Date aus Statistikrecord		PDS	PL5
..MEMDIRTIME	FCA Last Changed Time aus Stat.record		PDS	PL4
..MEMDIRSIZE	FCA Anzahl Records im Member		PDS	BL2
..MEMDIRINIT	FCA Anzahl Initial Records		PDS	BL2
..MEMDIRUSER	FCA User Id		PDS	CL7
..MEMNM	FCA Membername	PDS		CL8
..STOWID	FCA STOW Identifikation	PDS		CL1
..STOWVV	FCA STOW Statistik Version		PDS	BL1
..STOWMM	FCA STOW Statistik Modification		PDS	BL1
..STOWUSER	FCA STOW Statistik User Id		PDS	CL7

..DSN	Data Set Name für dyn. Zuordnung ALLOC		CL44
..DDN	DD Name für ALLOC		CL8
..SDISP	Status Disp für ALLOC:	DISP=SHR	CL1
..NDISP	Normal Disp für ALLOC:	DISP=(,CATLG...	CL1
..CDISP	Cancel Disp für ALLOC:	DISP=(...,...,DELETE	CL1
..TYPSP	Type of Space für ALLOC:	SPACE=(CYL,...	CL1
..PRISP	Primary Space für ALLOC:	SPACE=(...,(*nn*,...	BL2
..SECSP	Sec. Space für ALLOC:	SPACE=(...,(...,*nn*	BL2
..DIRBL	Directory Blks für ALLOC:	SPACE=(...,(...,...,*nn*	BL2
..RLSE	Release Space für ALLOC:	Yes oder No	CL1
..UNIT	Unit Name für ALLOC:	SYSDA	CL8
..DIR	Option DIR für PDS für ALLOC 'D'		CL1
..BWD	Option BWD für VSAM für ALLOC 'B'		CL1
..DSORG	data set organization for ALLOC:		CL2
..RECFM	record format for ALLOC:		CL3
..VOLID	volume serial ident for ALLOC:		CL6
..LABEL	tape label NL or SL for ALLOC:		CL2
..DCLAS	Data Class Name für ALLOC:	nur SMS	CL8
..MCLAS	Management Class für ALLOC:	nur SMS	CL8
..SCLAS	Storage Class für ALLOC:	nur SMS	CL8
..MN	Membername für generische Selektion ALLOC		CL8

..SCATNM	Selected Catalog Name	CL44
..SDSNM	Selected Generic Dataset Name	CL44

W.AT	WEB AT Position Symbol	CL16
W.BOOKMARK	WEB Bookmark Symbol	CL16
W.CADDRLENGTH	WEB Client Address Length	BL4
W.CLIENTADDR	WEB Client Address	CL*n*
W.CLIENTCODEPAGE	WEB Client Codepage Value	CL40
W.CLIENTNAME	WEB Client Name	CL*n*

Symbolname, Beschreibung, Formatlänge		
W.CNAMELENGTH	WEB Client Name Length	BL4
W.DATAONLY	WEB Data Only Flag	CL1
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 Codepage 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
Q.BUFFLENG	MQSeries maximale Bufferlänge	BL4
Q.CHARATTAREA	MQSeries Character Attribute Area	CL256
Q.CHARATTLENG	MQSeries Character Attribute Length	BL4
Q.CLOSEOPT	MQSeries Close Options	BL4
Q.CMDTEXT	MQSeries Command Text	CL10
Q.COMPCODE	MQSeries Completion Code	BL4
Q.CORRELID	MQSeries Correlation Id	CL24
Q.DATALENG	MQSeries current Message Länge	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.INTATTn	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

Symbolname, Beschreibung, Formatlänge		
Q.SELECTOR*n*	MQSeries Selector 1-16	BL4

Reservierte Feldsymbole in Alphabetischer Reihenfolge

Symbolname	Beschreibung		Formatlänge
ACCU*n*	Rechenfelder (*n* = 0-99) sind fest den Workbereichpositionen 6000-6990 zugeordnet		PL8
..ACNT	Added Recordzähler UPF File		PL8
APPLID	VTAM CICS Applikations Identifikation		CL8
..BL	definierte maximale Blocklänge		BL2
..BWD	Option BWD für VSAM für ALLOC 'B'		CL1
CALDRDATE	CALENDAR Datum Format CCYYMMDD		ZL6
CALDRTXMT	CALENDAR Monatsname (January – December)		CL10
CALDRTXWD	CALENDAR Tagesname (Monday – Sunday)		CL10
CALDRWARN	CALENDAR Warnung für Tagesanpassung Ende Monat		CL1
CALDRWKDY	CALENDAR Wochentag (1-7)		ZL1
CALDRWKNR	CALENDAR Wochennummer (01-53)		ZL2
CCDATE	aktuelles Jahrhundert-Datum	CCYYMMDD	PL5
CCYEAR	aktuelles Jahrhundert-Jahr	CCYY	ZL4
CDATE	aktuelles laufendes Datum	DDMMYY	PL4
..CDISP	Cancel Disp für ALLOC:	DISP=(...,...,DELETE	CL1
CDTIME	laufendes Datum & Zeit (Achtung: nur im Basis-Instruktionsformat gültig)	DDMMYYHHMMSS	2xPL4
C.FCNT	EXECSQL Fetched Counter		PL8
CICSID	CICS System Identifikation		CL4
CTIME	aktuelle laufende Zeit	HHMMSS	PL4
CURSOR	Cursorposition nach GET MAP		BL4
CUSERID	CICS User Identifikation		CL8
DATE	Systemdatum	'DD.MM.YY'	CL8
DAY	Tag des Systemdatums	DD	ZL2
DB2COMMIT	DB2 Auto Commit Counter		PL8
..DBN	FCA DL/I Datenbankname DB Name für dynamische Zuordnung		CL8
..DCLAS	Data Class Name für ALLOC:	nur SMS	CL8
..DCNT	Deleted Recordzähler UPF File		PL8
..DDN	DD Name DD Name für dynamische Zuordnung		CL8
..DDN	DD Name für ALLOC		CL8
..DSN	Data Set Name		CL44
..DSN	Data Set Name für dyn. Zuordnung ALLOC		CL44
..DSORG	data set organization for ALLOC:		CL2
..DIR	Option DIR für PDS für ALLOC 'D'		CL1
..DIRBL	Directory Blks für ALLOC:	SPACE=(...,(...,...,*nn*	BL2
DIVREM	Division Remainder Feld		PL8
D*n*POS*nnnn*	Recordarea Position *nnnn* von DS*n*, DB*n*		
EDATE	Systemdatum EURO Format	'DD.MM.CCYY'	CL10
EPARM	Externer Parameterbereich	max.	CL100
EPARML	Länge des externen Parameterwertes		BL2

Symbolname, Beschreibung, Formatlänge				
EPOS*nnnn*	EXCI Communication Bereich Position (1 - *nnnn*)			
FC	Functioncode	Spezialregister		BL4
FILLER	Platzhalter für Explizite Symbolzuordnungen			
..FN	Filename			CL8
..FORM	Printer FORM Name			CL8
..FT	VSAM Filetyp KS,ES,RR			CL2
FUNCMSG	Function Return Message			CL72
..GCNT	GET sequentieller Lesezähler			PL8
GPOS*nnnn*	Globale Workbereich Position (*nnnn* = 1-4096)			
HOUR	Stunde der Startzeit	HH		ZL2
HPOS*nnnn*	Interne Hiperspace Position (1-*nnnn*)			
ISODATE	Systemdatum ISO Stadard	'CCYY-MM-DD'		CL10
..ITEM	FCA Queue Item			BL2
IV	Intervallwert	Spezialregister		BL4
I[*n*]POS*nnnn*	Recordarea Position *nnnn* von IPF[*n*]			
JOBACTELNO	z/OS Job Accounting Element No			BL4
JOBACTINFO	z/OS Job Accounting Info			CL144
JOBCLASS	z/OS Job Klasse vom JOB Statement CLASS=			CL1
JOBCLASSLG	z/OS lange Job Klasse vom JOB Statement CLASS= oder //*MAIN CLASS=			CL1
JOBNAME	JCL Jobname			CL8
JOBNUM	z/OS JCL Jobnummer			ZL5
JOBPROGRNM	z/OS Programmer Name vom JOB Statement			CL20
JOBSTIME	z/OS Job Start Time hhmmss			PL4
..KEY	FCA Keyfeld			CL236
..KFBALENG	FCA DL/I key feedback area length			BL4
..KFBAREA	FCA DL/I key feedback area			CL256
..KFN	Keyfeldname für dynamische Zuordnung			CL8
..KL	definierte Keylänge			BL2
..KP	definierte Keyposition			BL2
..LABEL	tape label NL or SL for ALLOC:			CL2
..LCNT	Zeilenzähler pro Seite (Printfiles)			PL2
..LENG	FCA Recordlänge (bei LIBR BL2)			BL4
..LENG	FCA Recordlänge	LIBR		BL2
..LEV	FCA DL/I Segmentlevel			ZL2
MAPNAME	QPAC-Online aktiver Mapname			CL16
MAXCOL	Maximale Spalten pro Bildschirm			BL2
..MAXLCNT	Maximale Anzahl Zeilen pro Seite (Printfiles)			PL2
MAXROW	Maximale Zeilen pro Bildschirm			BL2
..MCLAS	Management Class für ALLOC: nur SMS			CL8
..MEMDIRCHDT	FCA Changed Date aus Statistikrecord		PDS	PL5
..MEMDIRCRDT	FCA Creation Date aus Statistikrecord		PDS	PL5
..MEMDIRINIT	FCA Anzahl Initial Records		PDS	BL2
..MEMDIRMM	FCA Modification aus Statistikrecord		PDS	BL1
..MEMDIRSIZE	FCA Anzahl Records im Member		PDS	BL2
..MEMDIRTIME	FCA Last Changed Time aus Stat.record		PDS	PL4

Symbolname, Beschreibung, Formatlänge				
..MEMDIRUSER	FCA User Id		PDS	CL7
..MEMDIRVV	FCA Version aus Statistikrecord		PDS	BL1
..MEMNM	FCA Membername	PDS		CL8
MINUTE	Minute der Startzeit	MM		ZL2
..MN	Membername für generische Selektion ALLOC			CL8
MONTH	Monat des Systemdatums	MM		ZL2
..NDISP	Normal Disp für ALLOC:	DISP=(,CATLG...		CL1
NETNAME	VTAM Terminal Netzname			CL8
OPERID	CICS Terminal Operator Identifikation			CL3
O[*n*]POS*nnnn*	Recordarea Position *nnnn* von OPF[*n*]			
..PCB	PCB Nummer für dynamische Zuordnung			BL4
..PCNT	PUT Recordzähler			PL8
	Seitenzähler (Printfiles)			PL4
PRGNAME	QPAC-Online Programmname			CL16
..PRISP	Primary Space für ALLOC:	SPACE=(...,(*nn*,...		BL2
..PSB	FCA DL/I PSB Name			CL8
	PSB Name für dynamische Zuordnung			
..QNM	FCA Queue Name			CL8
Q.BUFFLENG	MQSeries maximale Bufferlänge			BL4
Q.CHARATTAREA	MQSeries Character Attribute Area			CL256
Q.CHARATTLENG	MQSeries Character Attribute Length			BL4
Q.CLOSEOPT	MQSeries Close Options			BL4
Q.CMDTEXT	MQSeries Command Text			CL10
Q.COMPCODE	MQSeries Completion Code			BL4
Q.CORRELID	MQSeries Correlation Id			CL24
Q.DATALENG	MQSeries current Message Länge			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
Q.SELECTOR*n*	MQSeries Selector 1-16			BL4
RACFUSER	z/OS RACF User Id			CL8
..RBA	FCA Relative Byte Adresse			BL4
RC	Returncode	Spezialregister		BL4
..RC	FCA Returncode (VSAM, DL/I, Queues, ...)			CL2
..RC1	FCA Returncode Position 1			CL1
..RC2	FCA return code position 2			CL1

Symbolname, Beschreibung, Formatlänge			
..RCNT	READ Random Lesezähler MAP Receivezähler		PL8
..RECFM	record format for ALLOC:		CL3
..RL	definierte maximale Recordlänge		BL2
..RLSE	Release Space für ALLOC:	Yes oder No	CL1
..ROWLENG	FCA SQL Row Länge		BL4
..RRN	FCA Relative Record Nummer		BL4
..RTN	Rootsegmentname für dynamische Zuordnung		CL8
..SCATNM	Selected Catalog Name		CL44
..SCLAS	Storage Class für ALLOC:	nur SMS	CL8
..SCNT	MAP Sendezähler		PL8
..SDISP	Status Disp für ALLOC:	DISP=SHR	CL1
..SDSNM	Selected Generic Dataset Name		CL44
SECOND	Sekunde der Startzeit	SS	ZL2
..SECSP	Sec. Space für ALLOC:	SPACE=(...,(...,*nn*	BL2
..SEGLENG	FCA DL/I Segmentlänge (ohne MPB/DBCTL)		BL4
..SEGNM	FCA DL/I Segmentname		CL8
..SQLCODE	FCA SQL Returncode		BL4
..SSAN	FCA DL/I Anzahl verwendeter SSA Felder		BL1
..SSA*n*	FCA DL/I SSA Felder (*n* = 1 - 8)		CL256
STEPNAME	z/OS JCL Stepname		CL8
..STOWID	FCA STOW Identifikation PDS		CL1
..STOWMM	FCA STOW Statistik Modification	PDS	BL1
..STOWUSER	FCA STOW Statistik User Id	PDS	CL7
..STOWVV	FCA STOW Statistik Version	PDS	BL1
SYSNAME	z/OS System Name / SYSID		CL8
S*n*POS*nnn*	Recordarea Position *nnn* von SPO*n*		
..TBN	FCA SQL Tabellenname		CL18
TERMID	CICS Terminal Identifikation		CL4
TIME	QPAC Startzeit	'HH:MM:SS'	CL8
..TYPSP	Type of Space für ALLOC:	SPACE=(CYL,...	CL1
T*n*POS*nnnn*	Recordarea Position *nnnn* von TD*n*, TS*n*		
..UCNT	Updated Recordzähler UPF File		PL8
..UNIT	Unit Name für ALLOC:	SYSDA	CL8
USDATE	Systemdatum US Format	'MM/DD/CCYY'	CL10
USERID	QPAC-Online User Identifikation		CL16
U[*n*]POS*nnnn*	Recordarea Position *nnnn* von UPF[*n*]		
..VOLID	volume serial ident for ALLOC:		CL6
..WHERE	*für internen Gebrauch (DB2 Support)*		
WPOS*nnnn*	Interne Workbereich Position (1-*nnnn*)		
W.AT	WEB AT Position Symbol		CL16
W.BOOKMARK	WEB Bookmark Symbol		CL16
W.CADDRLENGTH	WEB Client Address Length		BL4
W.CLIENTADDR	WEB Client Address		CL*n*
W.CLIENTCODEPAGE	WEB Client Codepage Value		CL40
W.CLIENTNAME	WEB Client Name		CL*n*

Symbolname	Beschreibung		Formatlänge
W.CNAMELENGTH	WEB Client Name Length		BL4
W.DATAONLY	WEB Data Only Flag		CL1
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 Codepage 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*	Externer Bereich Position (1 - *nnnn*)		
X*n*	Indexregister (*n* = 1-99)		BL4
YEAR	Jahr des Systemdatums	YY	ZL2

Ergänzende Informationen zu reservierten Feldsymbolen

JOBACTELNO — Enthält die Anzahl Elemente, die im JOB Statement definiert und im Feld JOBACTINFO zur Verfügung gestellt werden.

JOBACTINFO — Enthält die Job Accounting Informationen aus dem JOB Statement. Das Feld ist in 9 Unterfelder zu je 16 Bytes eingeteilt, in welchen die Informationen aus dem JOB Statement linksbündig abgespeichert sind.

SYSNAME — Enthält den z/OS Systemnamen, der auch unter der Bezeichnung SYSID bekannt ist.

Cross Reference von Symbolen

Sofern über die `PARM` Option `XREF` die Cross Reference Liste gewünscht wird, ist daraus ersichtlich, wie und wo ein Symbol von QPAC behandelt wurde. Dies kann im Zweifelsfalle Aufschluss geben über Format und Länge, sowie über die Zuordnungsregelung.

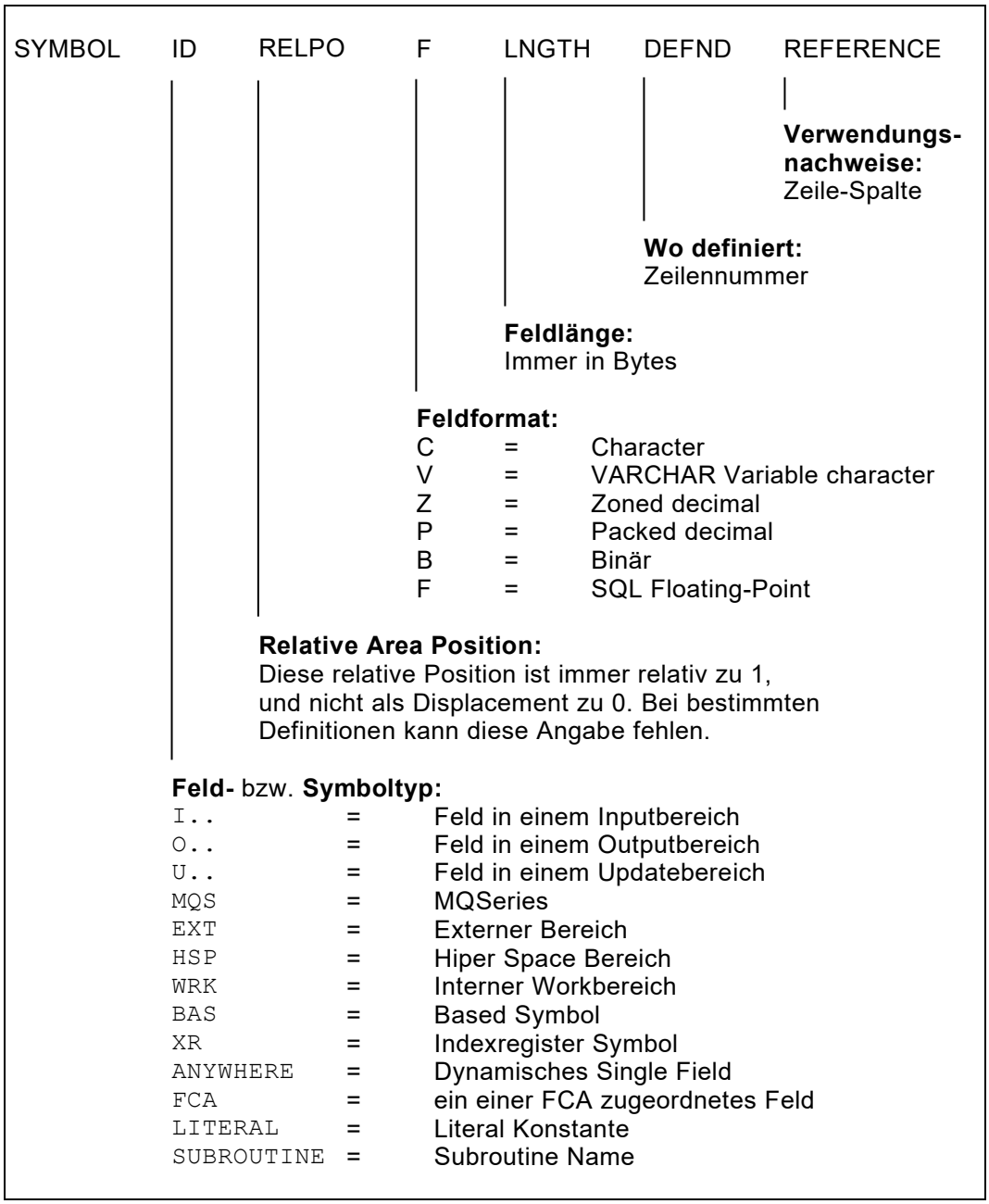


Abb. 93: Diagramm Symbol Cross Reference

```
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
```

Abb. 94: Auszug aus einer Symbol Cross Reference Liste

Kapitel 7. Die Verarbeitungsbefehle

Übersicht und Hinweise

QPAC-Batch beinhaltet einen kompletten „High-Level-Format“ Instruktionssatz, der es ermöglicht, die Datenkonversion aufgrund der Feldformate automatisch vorzunehmen. Dabei unterscheiden wir zwischen logischen und arithmetischen Feldformaten, die normalerweise nicht gemischt werden dürfen:

`C` = Character	logisches Format
`P` = Packed	arithmetisches Format
`Z` = Zoned	arithmetisches Format
`B` = Binär	arithmetisches Format
`Editmaske`	Empfangsfeld von arithmetischem Sendefeld

Abb. 95: Übersicht Feldformate

Das Feldformat, das bei der Felddefinition bestimmt wurde (Character oder Zoned), kann nicht dynamisch verändert werden.

Die High-Level-Format Instruktion SET

Grundformat

```
SET Empfangsfeld  =  Sendefeld-1  [  op  Sendefeld-2
                     Literal         op  Sendefeld-3
                                     op  Literal  ...  ]
```

op kann sein:	+	für Addieren
	-	für Subtrahieren
	*	für Multiplizieren
	/	für Dividieren
	%	für Modulo
	|	für konkatenieren

Literal kann sein:	*nnn*	bei arithmetischen Ausdrücken
	-nnn	negativer arithmetischer Ausdruck
	'..'	Character Konstante, logischer Ausdruck
	X'..'	Hexadezimale Konstante, logischer Ausdruck
	B'........'	Bit Konstante, 8 Bits 0 oder 1, logischer Ausdruck
	SPACE	Figurativ-Ausdruck
	BLANK	Figurativ-Ausdruck
	LOWVAL	Figurativ-Ausdruck
	HIGHVAL	Figurativ-Ausdruck

Ein Character-Literal kann bis zu 2048 Bytes, ein Hexadezimal-Literal bis 1024 Bytes lang sein. Sollte die Definition länger als eine Statement-Zeile sein, ist jeweils pro Statement mit "Apostroph - Schrägstrich - Blank" abzuschliessen und auf der Folgezeile mit Apostroph weiterzufahren.

```
CHARKONST = 'ABCDEFGH'/          *. KOMMENTAR
            'IJKLMNOP'/          *. KOMMENTAR
            'QRSTUVW'/           *. KOMMENTAR
            'XYZ'

HEXKONST = X'0102030405'/        *. KOMMENTAR
           '060708090A'/         *. KOMMENTAR
           '0B0C0D0E0F'          *. KOMMENTAR
```

Abb. 96: Literale über mehrere Zeilen

Als erweiterte Definitionsform kann dem einzelnen Literal String eine **explizite Längenangabe** vorangestellt werden:

z.B. `CL80'ABCDEF'`
`XL40'01020304'`

Ist der nachfolgende String kleiner als die Längenangabe, wird rechtsbündig mit Blanks resp. für Hexadezimal-Literals mit Low-Value aufgefüllt.
Werden Fortsetzungszeilen definiert, können diese mit oder ohne Längenangaben gemischt vorkommen. Eine einzelne Längenangabe wirkt sich in einem solchen Falle immer nur auf die laufende Zeile aus:

```
z.B.   0L=CL80'ABCDE'/
              '12345'/
              '67890'/
         CL70'VWXYZ'
```

Obiges Beispiel ergibt eine Gesamtlänge von 160 Bytes.
Zwischen der Zahl "0" und dem Buchstaben "V" sind 65 Blanks, ab dem Buchstaben "Z" sind 65 Blanks.

Spezialformate

SET *Empfangsfeld*	=CX	*Sendefeld*	(char -> hex)
	=XC		(hex -> char)
	=TR		(Translate)
	=MN		(Move Numeric)
	=MZ		(Move Zone)
	=MO		(Move with offset)
	=AND		(Boolsches AND)
	=OR		(Boolsches OR)
	=XOR		(Boolsches Exclusive OR)
	=ADR		(Feldadresse laden)
	=C'.'		(Padding Character)
	=X'..'		(Padding Hexwert)
	=CTS		(Convert Timestamp)
	=*Editmaske*		(vordefinierte Editmasken)

```
SET Empfangsfeld,CLXn = Sendefeld        (variable Feldlängen)
SET Empfangsfeld      = Sendefeld,CLXn
SET Empfangsfeld,CLXn = Sendefeld,CLXn
```

Es wird zwischen **logischen** und **arithmetischen** Operationscodes unterschieden. In einer Operandenfolge dürfen diese **nicht gemischt** vorkommen.

Klammerausdrücke sind beim Grundformat mit arithmetischer Operandenfolge unterstützt.

Die High-Level-Format Instruktion beginnt mit dem Schlüsselwort `SET` und wird als Ergibtformat-Instruktion bezeichnet. Bei der Feldbehandlung werden deren Formatattribute berücksichtigt, d.h. unterschiedliche Formate werden automatisch ins richtige Format konvertiert.

Die einzelnen logischen Glieder der SET Instruktion werden durch Blanks voneinander getrennt definiert, d.h. Blank ist jeweils der Delimiter einer logischen Einheit. Eine einzelne Instruktion kann ein Empfangsfeld und mehrere Sendefelder bzw. Operanden beinhalten.

Das Empfangsfeld wird durch das Ergibtzeichen = vom Sendefeld getrennt.

Die Operandenfolge ist durch das Vorhandensein eines Operationscodes bestimmt. Das Fehlen desselben wird als Instruktionsende interpretiert. Mindestens ein Operand muss jedoch pro Instruktion vorhanden sein.

Empfangsfeld = Sendefeld

Folgt bei arithmetischen Ausdrücken anstelle des ersten Operanden gleich ein Operationszeichen nach der Ergibtdefinition, wird das Empfangsfeld als erster

Operand angenommen.

Empfangsfeld = *op* Operand-2

(entspricht: Empfangsfeld = Empfangsfeld *op* Operand-2,
op kann ein +, -, *, / oder % sein)

Als Empfangsfeld kann ein Symbol definiert werden, welches den Regeln der Symbole, wie sie im vorausgegangenen Kapitel beschrieben wurden, entspricht.

Als Sendefeld bzw. Operand kann ein Symbol oder ein Literal definiert werden. Das Literal ist entweder ein Characterstring, wenn er zwischen zwei Apostrophs steht, ein hexadezimaler Wert, wenn dem führenden Apostroph ein X voransteht, oder ein arithmetischer Wert (numerisches Literal), wenn es sich um eine einfache Zahl handelt.

Bei VARCHAR variablen Characterfeldern wird normalerweise die Länge im vorangestellten Längenfeld berücksichtigt bzw. modifiziert. Dies wird jedoch aufgehoben, wenn das Feld adressmodifiziert bzw. mit einer expliziten Format-Längen Angabe definiert wird.
Ist ein variables Characterfeld als Empfangsfeld definiert, wird die Gesamtlänge des Sendeteils im führenden Längenfeld gespeichert, sofern sie nicht grösser ist als die maximale Länge des variablen Characterfeldes. Im letzteren Fall wird die maximale Feldlänge gespeichert und ein eventueller Rest des Sendeteils ignoriert. Ist der Sendeteil kleiner als die maximale Empfangsfeldlänge wird der Rest mit Blank bzw. mit einem explizit definierten Padding-Wert aufgefüllt.

Variable Characterfelder können kombiniert mit Literalen oder mit Indexregistern variabel definierten Feldern konkateniert werden.

Die SET Übertragungs-Instruktion

```
SET Empfangsfeld  =  Sendefeld

SET Empfangsfeld  =  Sendefeld-1  |  Sendefeld-2  ...
```

Die Übertragungsinstruktion gilt als logische Instruktion.

Logischer Operationscode:

| Senkrecht-Strich = Konkatenieren

Der Operationscode muss von den Operanden durch mindestens eine Blankstelle getrennt sein.

Die Kategorie der logischen Operationscodes kann das Feldformat C kombiniert mit Character- oder Hexa-Literalen behandeln.

```
SET STRING = FELD_A
SET STRING = 'ABCDEFG'

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

Abb. 97: Übertragen und Konkatenieren mit der SET Übertragungsinstruktion

Das Empfangsfeld einer logischen 'Ergibt-Instruktion' kann eine unterschiedliche Länge als das Sendefeld bzw. die Summe der konkatenierten Operanden haben.

Ist das Empfangsfeld zu kurz, wird der Rest der Sendeseite ignoriert. Ist das Empfangsfeld länger als die Summe der Sendeseite, wird der verbleibende Rest mit Blanks aufgefüllt, sofern kein Padding Wert angegeben wurde, andernfalls mit dem definierten Character oder Hexwert.

Dieser Grundregel steht eine Ausnahme gegenüber:

a) Ist das Empfangsfeld **adressmodifiziert und keine explizite Länge zusätzlich definiert**, wird die implizite Feldlänge aufgehoben und durch die Länge des **Sendefeldes** bzw. die Gesamtlänge aller Sendefelder ersetzt, sofern diese kleiner ist als die implizite Empfangsfeldlänge.

b) Ist das Empfangsfeld adressmodifiziert und eine explizite Feldlänge ist zusätzlich definiert, gilt letztere als Empfangsfeldlänge:

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

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

Abb. 98: SET Übertragungsinstruktion mit adressmodifiziertem Empfangsfeld

c) Ist das Sendefeld ein Figurativ-Ausdruck, z.B. SPACE, und einziger Operand (nicht konkateniert), wird das ganze Empfangsfeld auf diesen Wert gesetzt:

```
SET FELD = LOWVAL
```

Abb. 99: SET Übertragungsinstruktion mit Figurativ-Ausdruck

Die SET Arithmetik-Instruktion

```
SET Empfangsfeld  =  Sendefeld-1  op  Sendefeld-2
```

Es wird zwischen logischen und arithmetischen Operationscodes unterschieden. In einer Operandenfolge dürfen diese nicht gemischt vorkommen.

arithmetische Operationscodes:

- **+** **Addieren**
- **-** **Subtrahieren**
- ***** **Multiplizieren**
- **/** **Dividieren** (ein Restwert steht im Feld `DIVREM`)
- **%** **Modulo**

Der Operationscode muss von den Operanden durch mindestens eine Blankstelle getrennt sein.

Die Kategorie der arithmetischen Operationscodes kann die Feldformate `P`, `Z`, `B` in beliebiger Reihenfolge gemischt behandeln, kombiniert mit numerischen Literalen.

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

Abb. 100: Kombination von arithmetischen Feldformaten

Ein arithmetischer Ausdruck besteht aus einer Folge arithmetischer Operationen und wird nach den mathematischen Grundregeln abgearbeitet, d.h. Multiplikation, Division und Modulo werden vor Addition und Subtraktion ausgeführt.

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

Abb. 101: Auflösung nach den mathematischen Grundregeln

Ein negatives numerisches Literal kann definiert werden, wenn der Zahl ein Minuszeichen ohne Blank vorangestellt wird.

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

Abb. 102: Negative numerische Literale

Bei arithmetischen Operationen sind Klammerausdrücke unterstützt.

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

Abb. 103: Klammerausdrücke

Merke: Im Falle einer Division steht ein eventueller Rest im reservierten Feld `DIVREM` (Division Remainder) und wird von einer nachfolgenden Division jeweils wieder überschrieben.

Mit der Modulo Operation (% Zeichen) kann in einem arithmetischen Ausdruck ein bestimmter Modulo Wert ausgerechnet und im Empfangsfeld gespeichert werden.

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

Abb. 104: Modulo

Im Empfangsfeld ACCU1 steht nach Ausführung der Modulo Wert der Operation ACCU0 mod 7, im Beispiel der Restwert der ganzzahligen Division 10 / 7 = 1, Rest 3.

Die SET Übertragungs-Instruktion im Spezialformat

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

Die Spezialinstruktion **=CX** (Character nach Hexadezimal) konvertiert das Sendefeld ins hexadezimale Format und stellt das Ergebnis ins Empfangsfeld. Das Empfangsfeld muss doppelt so gross sein wie das Sendefeld.

Die Spezialinstruktion **=XC** (Hexadezimal nach Character) konvertiert das Sendefeld, dessen Inhalt in hexadezimalem Characterformat erwartet wird, ins nonhexadezimale Format und stellt das Ergebnis ins Empfangsfeld. Das Empfangsfeld ist halb so gross wie das Sendefeld.

```
SET HEXFELD,CL8  =CX CHARFELD,CL4
SET CHARFELD,CL4 =XC HEXFELD,CL8
```

Abb. 105: Character - Hexadezimal Konversion

Die Spezialinstruktion **=TR** (Translate) ist eine Translation-Funktion. Das Empfangsfeld wird translated; das Sendefeld dient dabei als 256 Bytes grosse Translatetabelle, deren Inhalt der Anwender definiert.

```
SET FELD =TR X'00010203......'
SET FELD =TR TABELLE
```

Abb. 106: Translate

Die Spezialinstruktionen **=MN** und **=MZ** ermöglichen das Übertragen nur des Numerisch-Teiles (=MN) oder nur des Zonen-Teiles (=MZ) vom Sendefeld zum Empfangsfeld. Das jeweilige gegenteilige Halbbyte des Empfangsfeldes wird dabei nicht verändert.

```
vorher:                              NULLFELD = X'F0F0F0F0'
                                     WERTFELD = X'C1A1F1F2'
SET NULLFELD =MN WERTFELD

nachher:                             NULLFELD = X'F1F1F1F2'
                                     WERTFELD = unverändert
SET WERTFELD =MZ NULLFELD

nachher:                             WERTFELD = X'F1F1F1F2'
                                     NULLFELD = unverändert
```

Abb. 107: Move Numeric und Move Zone

Die Spezialinstruktion **=MO** überträgt das Sendefeld um ein halbes Byte nach links verschoben in das Empfangsfeld. Ist das Empfangsfeld grösser als das Sendefeld, wird es linksbündig mit binär-null aufgefüllt.

```
vorher:                              DATUM X'19990112'
                                     FELD  X'000000000C'

SET FELD =MO DATUM

nachher:                             FELD  X'019990112C'
```

Abb. 108: Move with Offset

Die Spezialinstruktion **=AND** (Boolsches AND) verknüpft Empfangsfeld mit Sendefeld nach Boolscher Regel „UND“.

```
SET FELD1 =AND FELD2          1 + 1 = 1
                              0 + 1 = 0   beide 1 -> 1
                              1 + 0 = 0   andere Fälle -> 0
                              0 + 0 = 0
```

Abb. 109: Boolsche AND Verknüpfung

Die Spezialinstruktion **=OR** (Boolsches OR) verknüpft Empfangsfeld mit Sendefeld nach Boolscher Regel „ODER“.

```
SET FELD1 =OR FELD2           1 + 1 = 1
                              0 + 1 = 1   beide 0 -> 0
                              1 + 0 = 1   andere Fälle -> 1
                              0 + 0 = 0
```

Abb. 110: Boolsche OR Verknüpfung

Die Spezialinstruktion **=XOR** (Boolsches Exclusive OR) verknüpft Empfangsfeld mit Sendefeld nach Boolscher Regel „EXKLUSIV ODER“.

```
SET FELD1 =XOR FELD2          1 + 1 = 0
                              0 + 1 = 1   beide gleich -> 0
                              1 + 0 = 1   andere Fälle -> 1
                              0 + 0 = 0
```

Abb. 111: Boolsche Exclusive OR Verknüpfung

Die Spezialinstruktion **=CTS** ermöglicht auf einfache Art und Weise die Konvertierung des TOD Clock Time Stamp-Formates in ein Zoned Format und umgekehrt.
Der Time Stamp kann entweder in 8 Bytes binärer Form (TOD Clock) oder in 20 Bytes Zoned Format vorhanden sein.
Die Konvertierungsrichtung wird von den Feldern selber bestimmt.
Ist das Empfangsfeld 8 Bytes binär wird zwingend vorausgesetzt, dass das Sendefeld 20 Bytes Zoned oder Character definiert ist. Umgekehrt muss das Empfangsfeld 20 Bytes Zoned oder Character sein, wenn das Sendefeld 8 Bytes binär ist.
Eine Beschreibung des TOD Clock Formates ist im IBM Handbuch "Principles of Operation" zu finden.

```
0S=BINARYTS,BL8               Time Stamp im Binärformat (TOD Clock)
0S=ZONEDTS,ZL20               Time Stamp im Display Format
                              Kann auch als CL20 definiert sein.
                              ZonedTS Format:
                              YYYYMMDDHHMMSShtmu00

SET BINARYTS =CTS ZONEDTS     Konvertierung vom Zoned Format ins
                              Binäre Format

SET ZONEDTS =CTS BINARYTS     Konvertierung vom Binären Format ins
                              Zoned Format
```

Abb. 112: Konvertierung Timestamp

Die SET Edit-Instruktion

```
SET EMPFANGSFELD,M'Edit-Maske' = SENDEFELD

z.B. EMPFANGSFELD,M'ZZ9.999,99-'
                 ,M'-ZZ9.999,99'
```

Abb. 113: Frei wählbare Edit-Maske

Für das Empfangsfeld kann eine frei wählbare Edit-Maske definiert werden. Führende Z bedeuten Nullen Unterdrückung. Z und 9 repräsentieren Digit-Positionen.

```
SET Empfangsfeld =EDA   Sendefeld
                 =EDAZ
                 =EDAS

                 Maskentyp A = ▲▲▲▲▲▲–
```

Abb. 114: Maskentyp A

Das numerische *Sendefeld* wird ins *Empfangsfeld* gemäss Maskentyp A editiert. Diese erweiterte Editieroperation ermöglicht das Editieren ohne Interpunktionen. Ein negatives numerisches Feld wird rechtsbündig durch ein Minuszeichen (-) gekennzeichnet.

Nullen Unterdrückung ist möglich, wenn **EDAZ** oder **EDAS** definiert wird:
- EDAS ergibt Nullen Unterdrückung **exklusive** letzter Ziffernstelle
- EDAZ ergibt Nullen Unterdrückung **inklusive** letzter Ziffernstelle

z.B.

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

                    Pos.10        =  X'019376219D'
                    voll editiert =     19376219-
                    Pos.65        =           219-
```

Abb. 115: Auflösungsbeispiel Maskentyp A

```
SET Empfangsfeld =EDB Sendefeld

                    Maskentyp B = ▲▲.▲▲.▲▲.▲▲
```

Abb. 116: Maskentyp B

Das numerische *Sendefeld* wird ins *Empfangsfeld* gemäss Maskentyp B editiert. Diese erweiterte Editieroperation ermöglicht das Editieren in Gruppen von 2 dezimalen Stellen mit Punktetrennung.
Ein negatives Feld wird nicht als solches gekennzeichnet.
Nullen Unterdrückung ist nicht möglich.

z.B.

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

                    Pos.10        =  X'019376219D'
                    voll editiert =   19.37.62.19
                    Pos.65        =      37.62.19
```

Abb. 117: Auflösungsbeispiel Maskentyp B

Die Editierregeln für die folgende Editmaske sind dieselben, wie sie für EDB beschrieben sind:

```
SET Empfangsfeld =EDC Sendefeld

                    Maskentyp C = ▲▲:▲▲:▲▲:▲▲:▲▲
```

Abb. 118: Maskentyp C

Die Editierregeln für die folgenden Editmasken sind dieselben, wie sie für EDA beschrieben sind:

```
SET Empfangsfeld =EDD  Sendefeld
                 =EDDZ
                 =EDDS

                 Maskentyp D = ▲▲▲.▲▲▲.▲▲▲,▲▲▲
```

Abb. 119: Maskentyp D

```
SET Empfangsfeld =EDE  Sendefeld
                 =EDEZ
                 =EDES

                 Maskentyp E = ▲▲▲,▲▲▲,▲▲▲.▲▲–
```

Abb. 120: Maskentyp E

```
SET Empfangsfeld =EDF  Sendefeld
                 =EDFZ
                 =EDFS

                 Maskentyp F = ▲▲▲'▲▲▲'▲▲▲.▲▲–
```

Abb. 121: Maskentyp F

```
SET Empfangsfeld =EDG  Sendefeld
                 =EDGZ
                 =EDGS

                 Maskentyp G = ▲▲▲ ▲▲▲ ▲▲▲–
```

Abb. 122: Maskentyp G

```
SET Empfangsfeld =EDH  Sendefeld
                 =EDHZ
                 =EDHS

                 Maskentyp H = ▲▲▲,▲▲▲,▲▲▲–
```

Abb. 123: Maskentyp H

```
SET Empfangsfeld =EDI  Sendefeld
                 =EDIZ
                 =EDIS

                 Maskentyp I = ▲▲▲.▲▲▲.▲▲▲–
```

Abb. 124: Maskentyp I

```
SET Empfangsfeld =EDK  Sendefeld
                 =EDKZ
                 =EDKS

                 Maskentyp K = ▲▲▲▲▲▲▲▲▲▲▲,▲▲–
```

Abb. 125: Maskentyp K

Die SET Übertragungs-Instruktion für variable Feldlängen

Ebenfalls als Spezialformat gilt folgende Übertragungsinstruktion, welche es erlaubt, variable Feldlängen zu definieren.

```
SET Empfangsfeld,CLXn = Sendefeld
SET Empfangsfeld      = Sendefeld,CLXn [ | Sendefeld,CLXn ..]
SET Empfangsfeld,CLXn = Sendefeld,CLXn [ | Sendefeld,CLXn ..]
```

Explizit kann ein Indexregister definiert werden, dessen Inhalt als Feldlänge genommen wird.

Indexregister-Instruktionen und Indizierte Adressierung

Es stehen dem Anwender 99 Indexregister zur Verfügung, die für indizierte Adressierung verwendet werden können. Die Indexregister heissen:
`X1 X2 X3 X4 X5 X8 X7 X8 X9 X10.........X99`
Eine Adressindizierung erfolgt, wenn anschliessend an das Feldsymbol mit einem Plus- oder Minusoperator ein Indexregister angehängt wird.
Es können bis zu 3 Indexregister definiert werden.

Der aktuelle Inhalt des Indexregisters zur Ausführungszeit wird je nach Operator zur Basisadresse dazu oder abgerechnet.

Es ist wichtig zu beachten, dass falsch gesetzte Indexregister seitens des Empfangsfeldes überprüft werden, seitens der Sendefelder jedoch nicht. Eine Adressierung ausserhalb der Record- oder Workarea-Limiten ist dadurch möglich.

```
SET TABELLE+X1    = EINGABEFELD
SET TABELLE+X1-X2 = + 1

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

Abb. 126: SET Instruktion mit indizierter Adressierung

Indexregister sind zu Beginn mit Null initialisiert.

Ihre Inhalte können durch nachfolgend aufgeführte **Kurzform-Indexregister-Instruktionen** verändert werden (keine Multiplikation oder Division).

X*n*+*m*	Wert *m* zu Indexregister X*n* addieren
X*n*-*m*	Wert *m* von Indexregister X*n* subtrahieren
X*n*=*m*	Indexregister X*n* mit Wert *m* laden
X*n*+X*m*	Indexregister X*m* zu X*n* addieren
X*n*=X*m*	Indexregister X*n* mit Wert von X*m* laden

Abb. 127: Kurzform-Indexregister-Instruktionen

Die Indexregister können auch durch die `SET` Instruktion gleich irgendwelchen Datenfeldern behandelt werden.

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

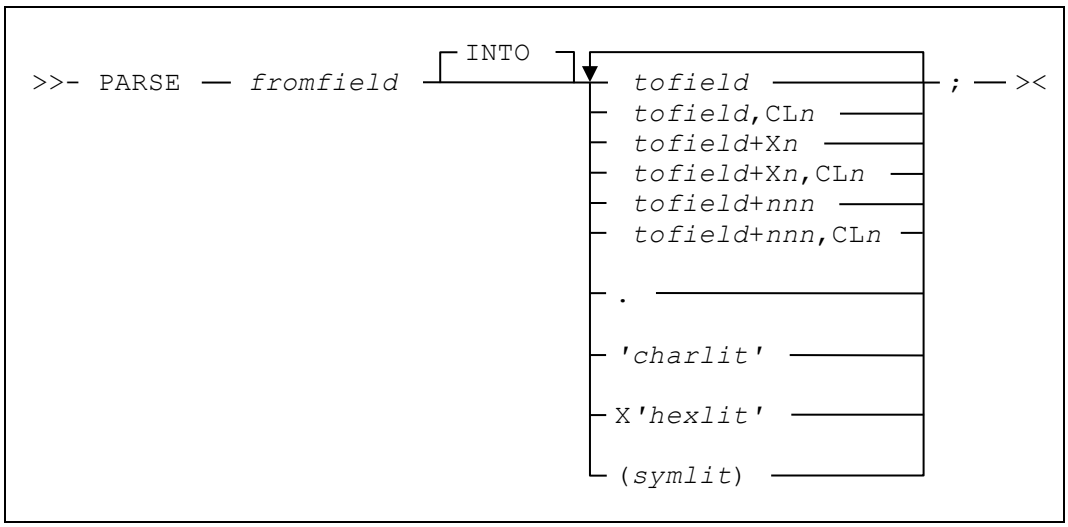
Abb. 128: SET Instruktion und Indexregister

Beim Laden von Registern aus dem Input- oder Workbereich, müssen diese Werte numerisch sein. Die Indexregister besitzen das Format `BL4` (4 Bytes binär).

```
SET X1 = ZONEDFELD,ZL2
SET X1 = BINAERFELD,BL2
```

Abb. 129: Laden von Indexregistern mit der SET Instruktion

Die PARSE Instruktion



Die PARSE Instruktion ermöglicht das Extrahieren der Teile eines Feldinhaltes in nachfolgend definierte Empfangsfelder. Dabei können die Sendeteile nach innerhalb der Instruktion definierten Delimiters verteilt werden. Fehlt ein expliziter Delimiter, wird Blank angenommen.

Zu beachten: Die Empfangsfelder müssen vorausgehend definiert sein. Die PARSE Instruktion definiert diese Felder nicht automatisch.

fromfield	Sendefeld, dessen Inhalt auf die Empfangsfelder verteilt wird.
INTO	Optionales Keyword, nur zur Dokumentation.
tofield	Empfangsfeld Je nach Feldposition in der PARSE Instruktion wird der entsprechende Teil (String) des Sendefeldes hineingespeichert. Dabei ist jedoch ein eventueller Delimiter, der dem *tofield* folgen kann, zu beachten.
tofield,CL*n* *tofield*+X*n* *tofield*+X*n*,CL*n* *tofield*+*nnn* *tofield*+*nnn*,CL*n*	Das Empfangsfeld kann nach QPAC-Regeln adressmodifiziert werden. Ist das Empfangsfeld hingegen ein VARCHAR Feld, wird das Attribut V aufgehoben und C angenommen.
.	Punkt Ein Punkt signalisiert, dass der entsprechende Teil (String) im Sendefeld übersprungen werden soll.
'*charlit*'	Character Literal als Delimiter Der vorausgehende Teil des Sendefeldes bis hin zu diesem Delimiter wird in das dem Delimiter voraus definierte Empfangsfeld gespeichert.
X'*hexlit*'	Hexadezimales Literal als Delimiter Der vorausgehende Teil des Sendefeldes bis hin zu diesem Delimiter wird in das dem Delimiter voraus definierte Empfangsfeld gespeichert.

(symlit)	Ein in Klammern definiertes Feldsymbol, dessen Inhalt als Delimiter gilt Der vorausgehende Teil des Sendefeldes bis hin zu diesem Delimiter wird in das dem Delimiter voraus definierte Empfangsfeld gespeichert.

Sind weniger Empfangsfelder definiert, als im Sendefeld Teile (Strings) vorhanden sind, wird der verbleibende Rest auch im letzten Empfangsfeld gespeichert.

Sind mehr Empfangsfelder definiert, als im Sendefeld Teile (Strings) vorhanden sind, werden die restlichen Empfangsfelder attributgerecht gelöscht.

Kapitel 8. Die Ablaufsteuerungsbefehle

QPAC-Batch kennt Condition Definitionen bei der `IF`- bzw. `DO`-Struktur. Dabei handelt es sich einerseits um Relation Conditions, wodurch zwei Operanden miteinander verglichen werden, andererseits kann es sich auch um eine blosse Abfrage einer Status Condition handeln, beispielsweise um die Abfrage einer bestimmten Fehlersituation.

Die IF THEN ELSE Instruktion

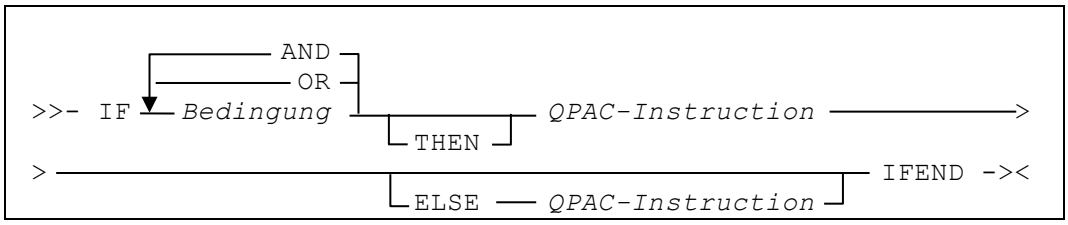


Abb. 130: Diagramm Grundformat Vergleichsoperation

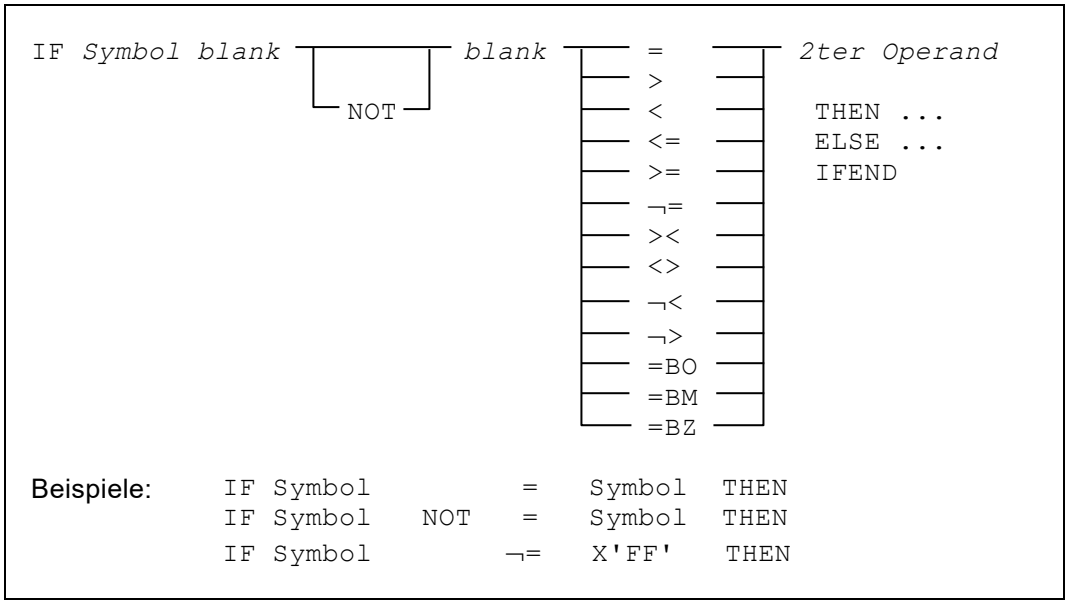


Abb. 131: Diagramm Relationsbedingung (relation condition)

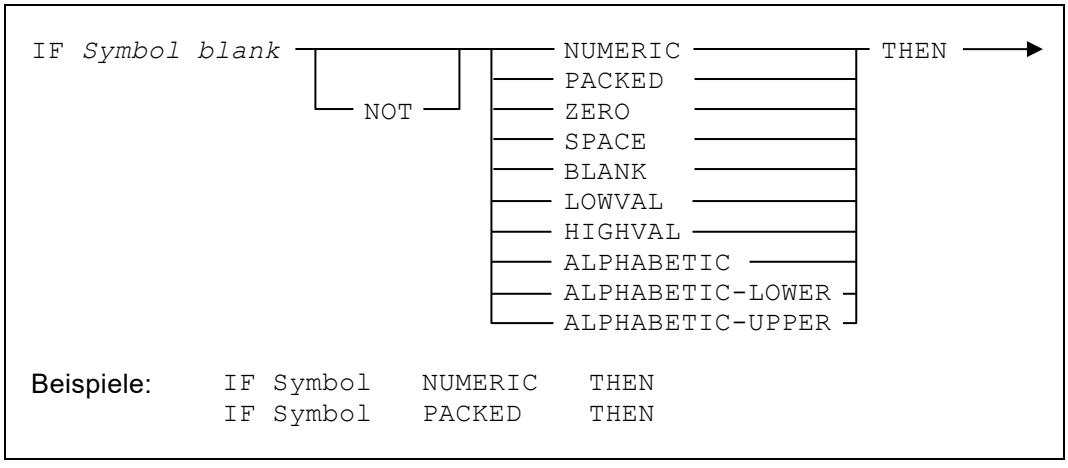


Abb. 132: Diagramm Statusbedingung (status condition)

Die einzelnen logischen Glieder einer Bedingungsdefinition werden durch Blank getrennt. Eine einzelne Bedingung besteht normalerweise aus zwei Operanden, die durch den Vergleichsoperator zueinander in Beziehung gebracht werden.
Beide Vergleichsoperanden können durch ein Symbol definiert werden, welches den Regeln der Symbole, wie sie vorgängig beschrieben wurden, entspricht.
Als *2ter Operand* kann anstelle eines Symbols auch ein Literal definiert werden. Ein Characterstring in Apostroph gilt als Characterliteral. Ein dem Apostroph vorangestelltes X bezeichnet ein Hexaliteral. Eine einfache Zahl bezeichnet ein numerisches Literal, ein dem Apostroph vorangestelltes `B` bezeichnet ein Bit Literal..
Ein negatives numerisches Literal kann definiert werden, wenn der Zahl ein Minuszeichen vorangestellt wird:

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

Abb. 133: Symbole und Literale in Vergleichsoperationen

Als Vergleichsoperatoren können nachfolgende Zeichen verwendet werden:

=	Vergleich auf gleich	<=	Vergleich auf kleiner oder gleich
>	Vergleich auf grösser als	=<	Vergleich auf kleiner oder gleich
<	Vergleich auf kleiner als	>=	Vergleich auf grösser oder gleich
¬>	Vergleich auf nicht grösser	=>	Vergleich auf grösser oder gleich
¬<	Vergleich auf nicht kleiner		
¬=	Vergleich auf nicht gleich	=BO	Test Bit Ones
<>	Vergleich auf nicht gleich	=BM	Test Bit Mixed
><	Vergleich auf nicht gleich	=BZ	Test Bit Zero

Die Negation der Vergleichsoperatoren ist auch durch das vorangehende Schlüsselwort **`NOT`** möglich:

NOT =	Vergleich auf NICHT gleich
NOT >	Vergleich auf NICHT grösser als
...	...

Logische und arithmetische Feldformate dürfen als Operandenpaar nicht gemischt werden, da durch diese Mischung keine gültige Konversion möglich ist. QPAC weist eine solche Konstellation schon zur Umsetzungszeit als Fehler zurück.

Bei variablen Characterfeldern wird zum Vergleich der Inhalt des vorangestellten Längenfeldes berücksichtigt.

Besitzen die zwei zu vergleichenden Operanden bei logischen Feldformaten unterschiedliche Länge, wird in der **Länge des kleineren Feldes** verglichen, sofern eine der beiden Längen nicht grösser als 256 Bytes ist und es sich nicht um variabel definierte Felder handelt.

Arithmetische Feldformate können untereinander beliebig gemischt vorkommen:

```
IF X1 < ACCU0
```

Abb. 134: Vergleich von verschiedenen arithmetischen Formaten

Als Vergleichsoperator kann auch eine Figurativ-Konstante definiert werden. In diesem Falle fällt der *2te Operand* weg.

Folgende Figurativ-Ausdrücke sind in dieser Kurzschreibweise unterstützt:

SPACE	X'40'
BLANK	X'40'
ZERO	X'F0'
LOWVAL	X'00'
HIGHVAL	X'FF'
PACKED	Ist das Feld inhaltlich gepackt?
NUMERIC	(Zoned Format) Ist das Feld inhaltlich rein numerisch?
ALPHABETIC	Kombination aus A bis Z, a bis z, und space?
ALPHABETIC-UPPER	Kombination aus A bis Z, und space?
ALPHABETIC-LOWER	Kombination aus a bis z, und space?
ENTERED	*(nur QPAC-Online Mapfelder)*
ERASED	*(nur QPAC-Online Mapfelder)*

Diese Figurativ-Ausdrücke können ebenfalls mit NOT negiert werden:

```
IF MENGE PACKED
IF MENGE NUMERIC

IF MENGE NOT PACKED
IF MENGE NOT NUMERIC

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

Abb. 135: Figurativ-Konstanten in Vergleichsoperationen

Vergleichsoperanden mit variabler Länge sind bei Characterfeldern möglich.

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

Abb. 136: Vergleichsoperanden mit variabler Länge

Eine Condition Definition bestehend aus zwei Operanden, die miteinander verglichen werden, wird als **Relation Condition** bezeichnet.

Solche Relation Conditions können zusätzlich logisch miteinander verknüpft werden. QPAC kennt zu diesem Zwecke die zwei **boolschen Operatoren** AND bzw. OR.

Die boolsche Verknüpfung von einfachen Relation Conditions wird auch als **Combined Condition** bezeichnet.
Es können beliebig viele Combined Conditions in beliebiger Variation mit den Operatoren AND bzw. OR definiert werden. Dabei muss jedoch die Regel beachtet werden, nach der sich die Auflösung richtet.
Diese Regel lautet: **Auf gleicher hierarchischer Stufe werden zuerst die AND Verknüpfungen aufgelöst und zwar in prozeduraler Reihenfolge**.

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

löst sich im 1. Schritt folgendermassen auf:

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

löst sich im 2. Schritt folgendermassen auf:

```
IF          c12          OR          c34          THEN
```

Abb. 137: Beispiel zur Auflösungsregel von Combined Conditions

QPAC Condition Definitionen erlauben zusätzlich die Verwendung von Klammerausdrücken. Dadurch können teilweise redundante Relations-Definitionen vermieden werden, oder es kann die Regel über die Auflösungsreihenfolge der boolschen Operatoren, durch Verschieben der hierarchischen Stufe, beeinflusst werden.

Die Verwendung von Klammern ist nichts anderes als ein erweitertes Format der Combined Conditions. Zur Definition von Combined Conditions kennen wir daher in QPAC nachfolgende Elemente:

a)	Simple-Condition	(Relation-Condition)
b)	`AND`	(boolscher Operator)
c)	`OR`	(boolscher Operator)
d)	(	Combined Structure Element)
e)	)	Combined Structure Element)

Nachfolgende Tabelle zeigt die Regel über die erlaubte Anwendungssequenz der Elemente für Combined Conditions auf:

Combined Condition Element	erlaubt als erste Definition links?	was kann diesem Element vorausgehen?	was kann diesem Element folgen?	erlaubt als letzte Definition rechts?
Simple Condition	Ja	AND OR (	AND OR)	Ja
AND OR	Nein	)	Simple Condition (	Nein
(	Ja	AND OR (	Simple Condition	Nein
)	Nein	Simple Condition)	AND OR)	Ja

Abb. 138: Regel für Combined Conditions

Klammern müssen nie definiert werden, wenn in einer Combined Condition nur entweder AND oder OR verwendet werden:

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

Klammern sind nicht nötig, wenn die Definition von Combined Conditions der Auflösungsregel entspricht, nach der zuerst die AND-Verknüpfungen aufgelöst werden:

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

Klammern können definiert werden um redundante Simple Conditions' zu vermeiden:

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

Werden Klammern definiert, müssen sie eins-zu-eins zwischen links und rechts korrespondieren:

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

Klammern können bis zu einer Tiefe von 10 Stufen definiert werden. Solange die hierarchische Strukturtiefe nicht erreicht wird, ist die Anzahl Klammernpaare nicht limitiert.

Visualisierte Darstellungsbeispiele

Bedingungssatz mit oder ohne Alternative:

Ein Bedingungssatz beginnt mit der Frage IF, gefolgt von einer Vergleichsoperation, und endet logisch für den **'wahr'-Fall** mit THEN, für den **'falsch'-Fall** mit ELSE:

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

Absolut wird das gemeinsame logische, wie auch physische Ende des THEN- und ELSE-Zweiges mit dem Schlüsselwort IFEND abgeschlossen:

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

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

Mehrere Vergleichsoperationen mit AND bzw. OR verknüpft:

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

Der gleiche Bedingungssatz mit Klammern formuliert:

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

Mehrere Bedingungssätze können bis zu 15 Stufen ineinander verschachtelt werden.

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

Abb. 139: Beispiel 1 ohne Alternativen

```
IF I1POS10 = '1' THEN ──────────────────────────────────────────────────────
|  IF I1POS20 = '2' THEN ──────────────────────────────────────────────────
|  |  IF I1POS30 = '3' THEN ──────────────────────────────────────────────
|  |  |  IF I1POS40 = '4' THEN ──────────────────────────────────────────
|  |  |  |  IF I1POS50 = '5' THEN ──────────────────────────────────
|  |  |  |  |  SET OUTPUTFELD1 = '11'
|  |  |  |  |  SET OUTPUTFELD2 = '22'
|  |  |  |  |  SET INDICATOR   = '5'
|  |  |  |  ELSE ──────────────────────────────────────────────────
|  |  |  |  |  SET INDICATOR   = '4'
|  |  |  |  IFEND ──────────────────────────────────────────────────
|  |  |  ELSE ──────────────────────────────────────────────────────
|  |  |  |  IF I1POS60 = '6' THEN ──────────────────────────────
|  |  |  |  |  SET OUTPUTFELD3 = '33'
|  |  |  |  |  SET OUTPUTFELD2 = '22'
|  |  |  |  |  SET OUTPUTFELD4 = '44'
|  |  |  |  ELSE ──────────────────────────────────────────────
|  |  |  |  |  SET INDICATOR   = '8'
|  |  |  |  IFEND ──────────────────────────────────────────────
|  |  |  IFEND ──────────────────────────────────────────────────────
|  |  ELSE ──────────────────────────────────────────────────────────
|  |  |  SET INDICATOR = '1'
|  |  IFEND ──────────────────────────────────────────────────────────
|  IFEND ──────────────────────────────────────────────────────────────
IFEND ──────────────────────────────────────────────────────────────────
```

Abb. 140: Beispiel 2 mit Alternativen

ELSEIF Case Struktur

Bedingungssatz mit mehreren kontinuierlich folgenden Alternativen

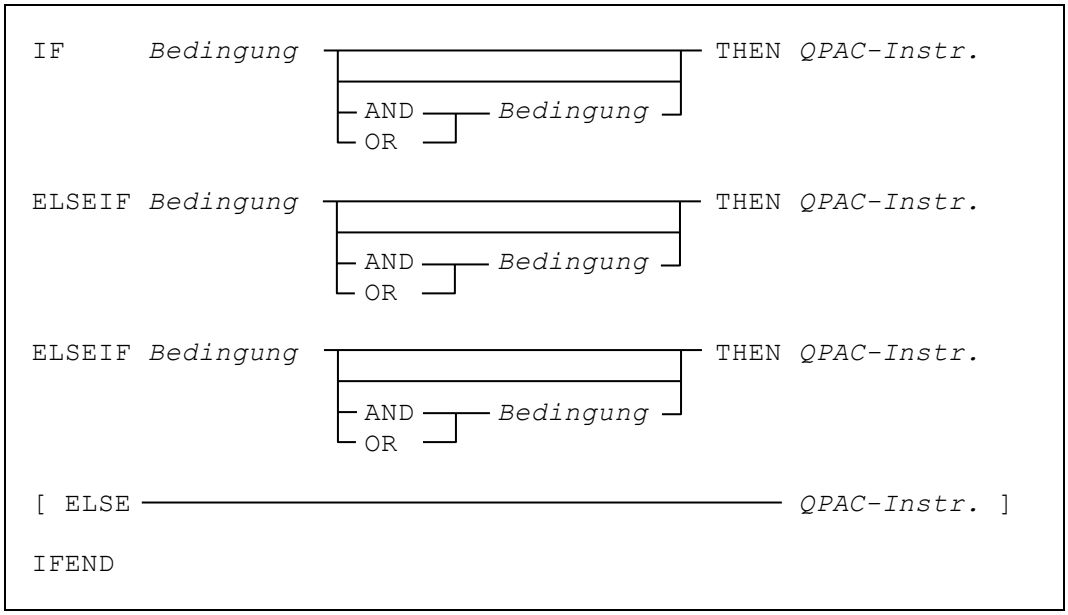


Abb. 141: Diagramm ELSEIF Case Struktur

Eine vereinfachende Form eines Bedingungssatzes mit mehreren möglichen Alternativen bildet die Anwendung des `ELSEIF` Schlüsselwortes.
Dieses ermöglicht im 'falsch'-Falle das erneute Abfragen von Bedingungen.

```
IF      I1POS1,PL4 = 1  THEN  CALL-'ROUTINE1'
ELSEIF  I1POS1,PL4 = 2  THEN  CALL-'ROUTINE2'
ELSEIF  I1POS1,PL4 = 3  THEN  CALL-'ROUTINE3'
ELSE    WTO-'DATENFEHLER - PROGRAMM BEENDET'
IFEND
```

Abb. 142: Anwendung der ELSEIF Case Struktur

Das obige Beispiel würde mit verschachtelten Standardbedingungssätzen folgendermassen aussehen:

```
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-'DATENFEHLER - PROGRAMM BEENDET'
          IFEND
     IFEND
IFEND
```

Abb. 143: Traditionelle Verschachtelung von IF-Sätzen

DO-Schleifen Instruktionen

Mit der DO Instruktion können Schleifen mit einem fixen oder bedingten Wiederholintervall spezifiziert werden.

DO-*nn*	Absolutes Wiederholintervall
DO-X*n*	Absolutes Wiederholintervall in Indexregister
DO-WHILE	Bedingtes Wiederholintervall positiv
DO-UNTIL	Bedingtes Wiederholintervall negativ
DO-FOREVER	Endlos Wiederholintervall
DOBREAK	Sprung zum DO-Anfang
DOQUIT	DO-Schleife sofort verlassen
DOEND	Schleifenende

Abb. 144: DO-Schleifen Instruktionen Übersicht

Die DO-nn Instruktion

Schleifeninstruktion mit **fixem Wiederholintervall**.

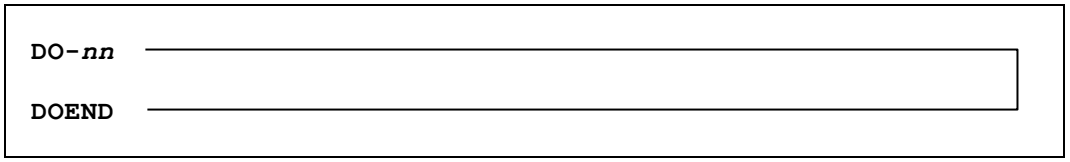


Abb. 145: Diagramm mit fixem Wiederholintervall

nn ist ein absoluter Wert, mit dem das Wiederholintervall definiert wird.
Mit DOEND wird das Schleifenende bestimmt.

Die Verschachtelung von mehreren DO-Schlaufen ist bis zu **10 Stufen** möglich.

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

Abb. 146: Diagramm mit fixem Wiederholintervall

Damit werden z.B 10 Tabellenfelder zu 8 Bytes initialisiert (siehe auch indizierte Adressierung).

Die DO-Xn Instruktion

Schleifeninstruktion mit **fix modifizierbarem Wiederholintervall**.

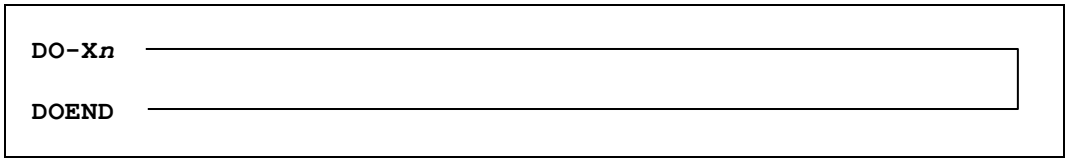


Abb. 147: Diagramm mit fix modifizierbarem Wiederholintervall

Xn ist ein Indexregister, dessen Inhalt die Wiederholhäufigkeit bestimmt. Dieses Indexregister wird vor jedem Wiederholzyklus um 1 vermindert. Es kann dadurch im DO-Block auch zur relativen Adressierung verwendet werden. Es kann ausserdem im DO-Block verändert werden, um damit den Wiederholzyklus zu beeinflussen.
Mit DOEND wird das Schleifenende bestimmt.

```
X10=80
DO-X10                  (beim ersten Durchlauf ist X10 = 79)
   SET OPOS80-X10 = IPOS1+X10,CL1
DOEND
```

Abb. 148: Anwendung mit fix modifizierbarem Wiederholintervall

Der 80-stellige Inputrecord wird umgekehrt in den Outputrecord-Bereich übertragen.

Die DO-WHILE Instruktion

Schleifeninstruktion mit **positiv bedingtem Wiederholintervall "während“**.

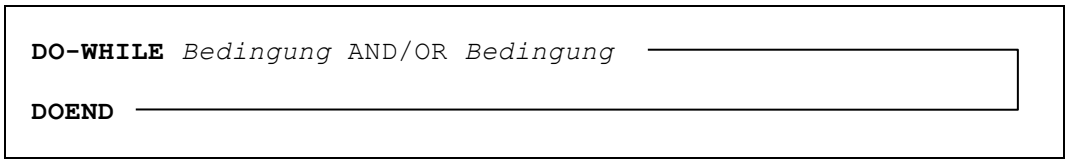


Abb. 149: Diagramm mit positiv bedingtem Wiederholintervall

Bedingung ist die Angabe der Ausführungsabhängigkeit. Solange die Bedingungen zutreffen, wird die DO-Schleife durchlaufen. Das Format und die Anwendungsform der Bedingungen entspricht genau dem des IF-Satzes, mit Ausnahme, dass hier nach der letzten Bedingung kein THEN definiert wird.
Mit DOEND wird das Schleifenende bestimmt.

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

Abb. 150: Anwendung mit positiv bedingtem Wiederholintervall

Sofern beim Record die Stellen 1-2 gleich '10' sind und die Stelle 12 gleich X'00' ist, werden die ersten vier Stellen übertragen und nachgelesen. Erfüllt ein Record diese Bedingungen nicht, wird die Schlaufe verlassen. Dies gilt auch bei EOF-Situation.

Die DO-UNTIL Instruktion

Schleifeninstruktion mit **negativ bedingtem Wiederholintervall „bis“**.

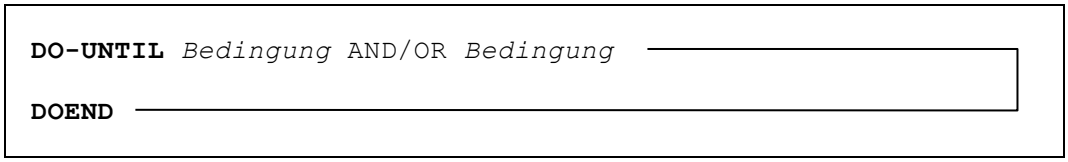


Abb. 151: Diagramm mit negativ bedingtem Wiederholintervall

Bedingung ist die Angabe der Ausführungsabhängigkeit. Solange die Bedingungen nicht zutreffen, wird die DO-Schleife durchlaufen. Das Format und die Anwendungsform der Bedingungen entspricht genau dem des IF-Satzes, mit Ausnahme, dass hier nach den Bedingungen kein THEN definiert wird.
Mit DOEND wird das Schleifenende bestimmt.

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

Abb. 152: Anwendung mit negativ bedingtem Wiederholintervall

In der Schleife wird so lange nachgelesen, bis ein Record mit der Identifikation 100 in den Kolonnen 1-3 gefunden wird, oder EOF eintritt.

Die DO-FOREVER Instruktion

Schleifeninstruktion mit **Endlos-Zyklus**.

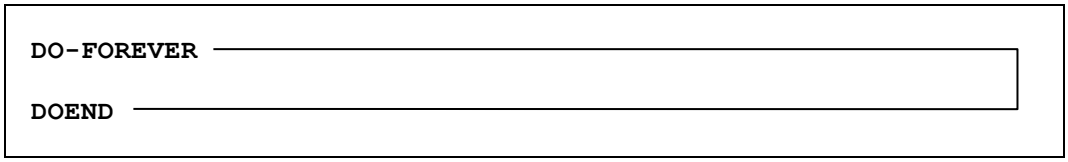


Abb. 153: Diagramm mit negativ bedingtem Wiederholintervall

Im DO-Header ist keine Bedingung bzw. ein Wert, der sich zyklisch abbaut, definiert. Die Schlaufe hört nie auf. Sie kann im Schlaufenkörper durch DOQUIT oder gegebenenfalls auch durch einen GO... Befehl verlassen werden.

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

Abb. 154: Anwendung mit Endlos-Zyklus

Ergänzende Steuerbefehle für Schleifeninstruktionen

Die DOBREAK Instruktion

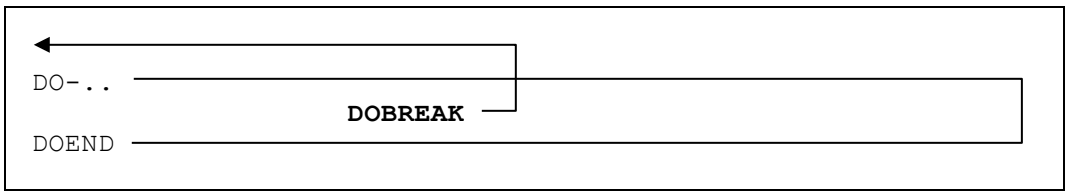


Abb. 155: Diagramm DOBREAK Instruktion

Sofortiges Beenden der prozeduralen Folge und Rücksprung zum DO-Schleifen Kopf.

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

Abb. 156: DOBREAK springt direkt zum DO-Schlaufen Kopf zurück

In obigem Beispiel wird die erste Nicht-Blankstelle von rechts her gesucht.

Die DOQUIT Instruktion

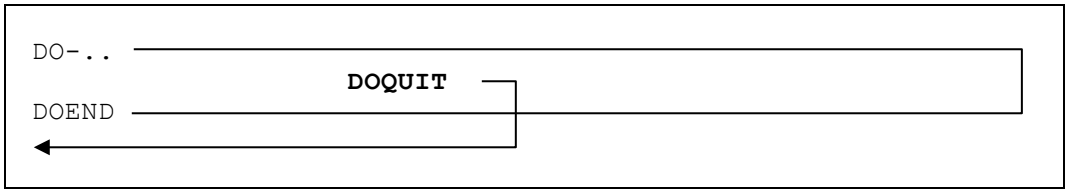


Abb. 157: Diagramm DOQUIT Instruktion

Diese Funktion verursacht das unmittelbare Verlassen der DO-Schleife.
Die Verarbeitung wird nach dem DOEND fortgesetzt.

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

Abb. 158: Die DOQUIT Instruktion verlässt die DO-Schlaufe unmittelbar

Kapitel 9. Subroutinen und Externe Programme

Interne Subroutine CSUB

Mit CSUB (Call Subroutine) kann eine für sich selbständig definierte Subroutine aufgerufen werden. Der Subroutinename kann 1 bis 30 alphanumerische Stellen umfassen. Es können beliebig viele CSUB Befehle auf die gleiche Subroutine zeigen. Ebenfalls kann der CSUB Befehl an beliebiger logisch richtiger Stelle im QPAC-Ablauf stehen.

CSUB-*Name*	=	CALL Subroutine
SUB-*Name*	=	Subroutine Header Definition
SUBEND	=	Subroutine End Definition

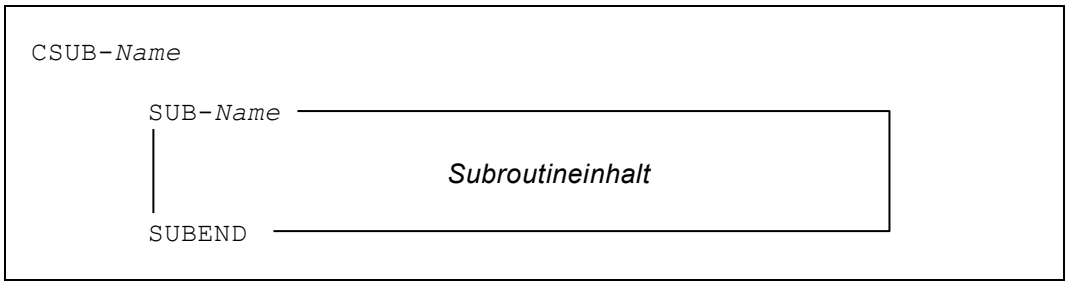


Abb. 159: Grundformat der Subroutineanwendung

SUB-*Name* definiert den Subroutinenanfang, wobei *Name* die Identifikation ist, über welche die Subroutine mittels CSUB aufgerufen wird.
Alle Verarbeitungs- und logischen Instruktionen sind möglich, inklusive Calls von anderen Subroutinen, d.h. eine Verschachtelung von Subroutinen ist möglich.

SUBEND definiert das Ende der Subroutine. SUB-*Name* und sein entsprechendes SUBEND müssen auf hierarchischer Stufe 0 stehen, d.h. sie können nicht innerhalb von IF- oder DO-Strukturblöcken stehen.

Eine Subroutine wird nur durch Aufruf mit CSUB durchlaufen. Die übrige Verarbeitungssequenz überspringt jede in der prozeduralen Folge vorhandene Subroutine.

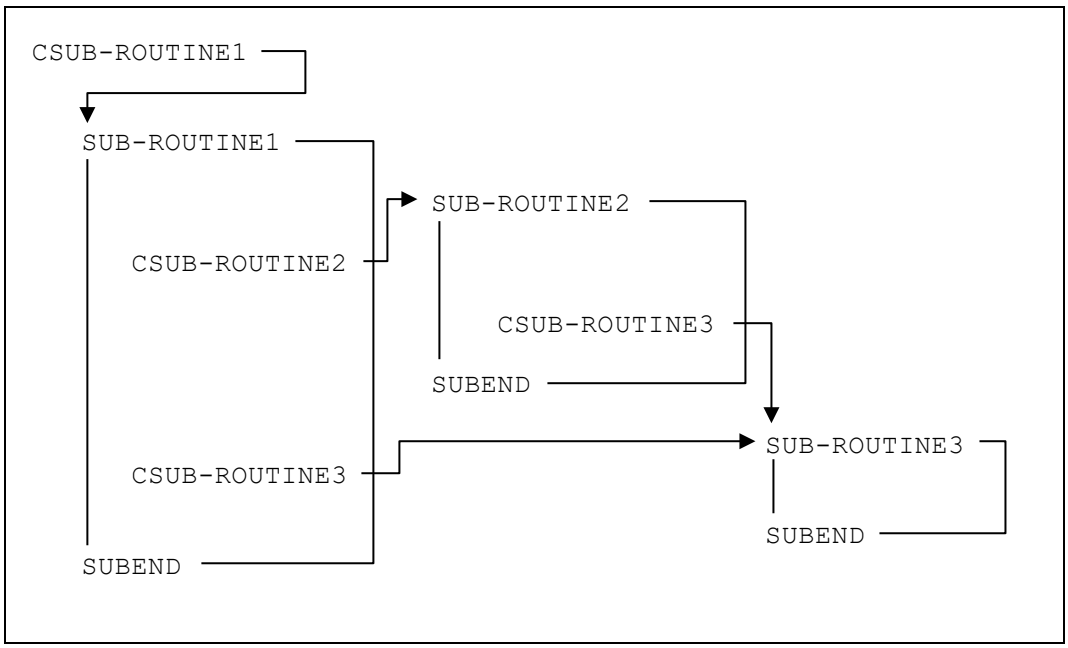
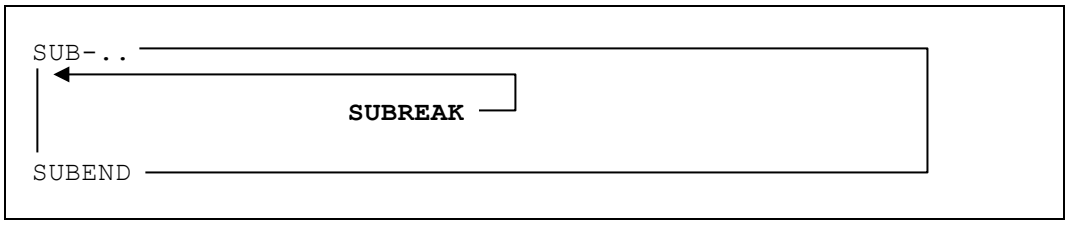


Abb. 160: Graphische Darstellung einer Subroutineverschachtelung

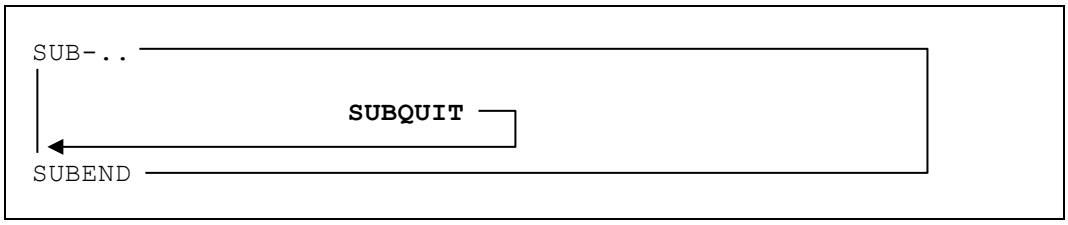
Ergänzende Steuerbefehle für Interne Subroutinen

Die SUBREAK Instruktion



Bewirkt ein sofortiges Beenden der prozeduralen Folge des Subroutineablaufes und einen unmittelbaren Rücksprung zum Subroutinenanfang.

Die SUBQUIT Instruktion



Bewirkt ein sofortiges Verlassen der Subroutine, bzw. einen unmittelbaren Sprung zum `SUBEND`.

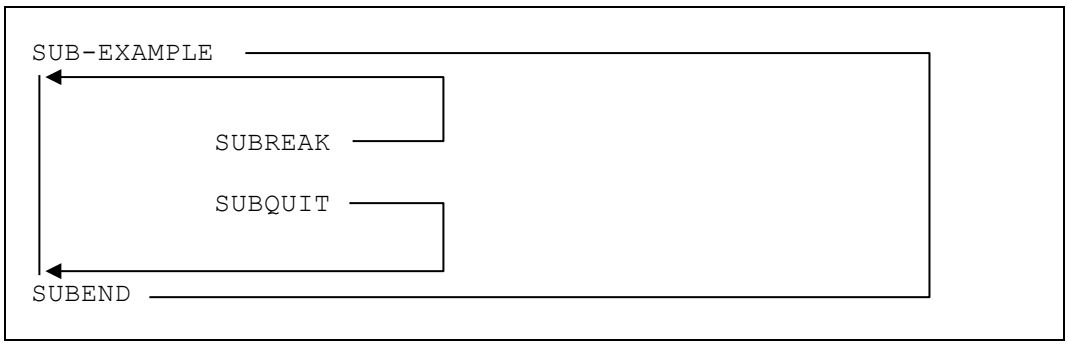


Abb. 161: Die Subroutineinstruktionen SUBREAK und SUBQUIT

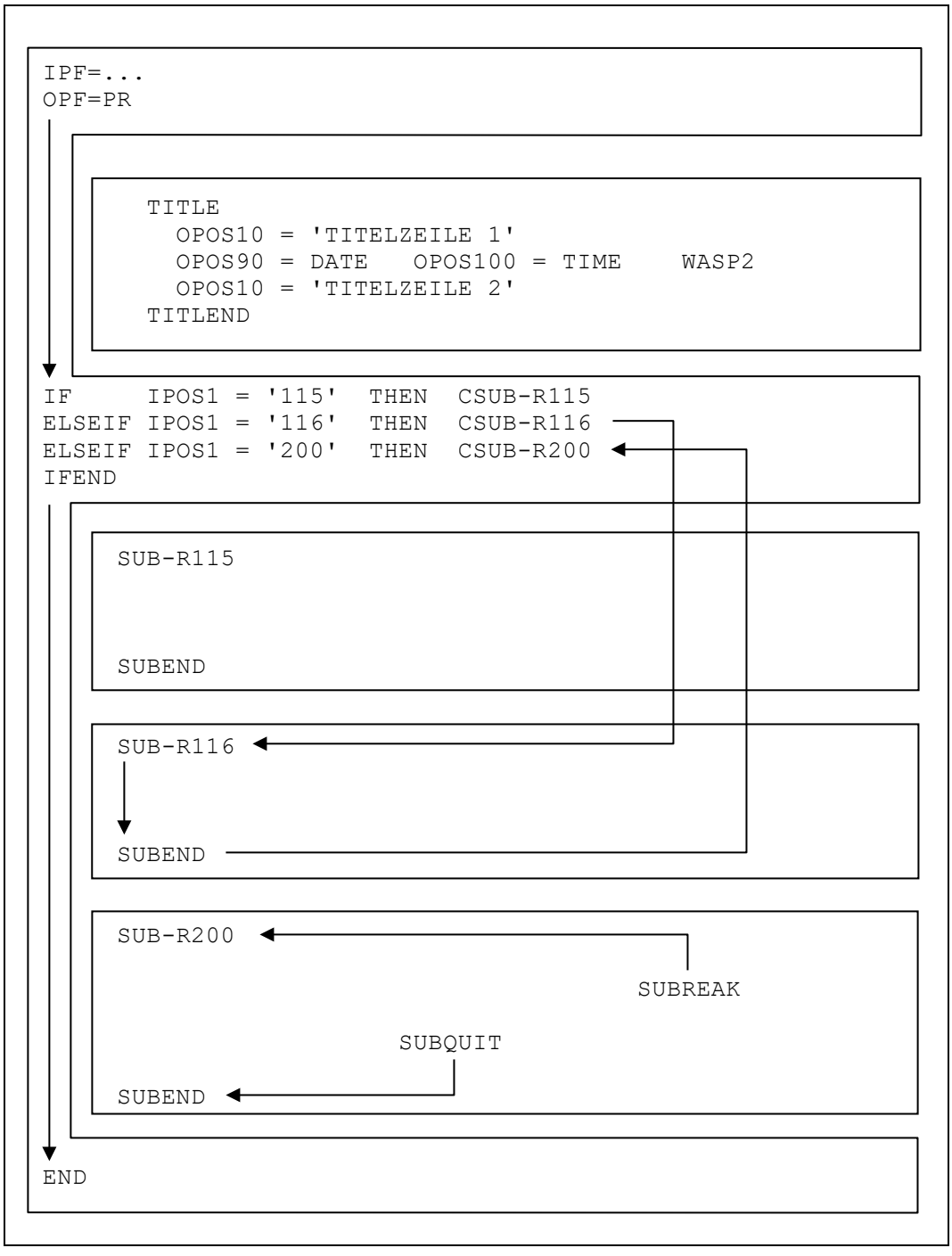


Abb. 162: Die Subroutineinstruktionen SUBREAK und SUBQUIT

Externe Subroutinen (CALL Exit Routinen)

```
CALL='Loadmodul'   [ ,Parameter,Parameter ]
```

Spezifische **externe Verarbeitungsroutinen** können mit der `CALL` Funktion an beliebigen Stellen innerhalb der QPAC-Definitionen aufgerufen werden.
Es handelt sich hierbei um QPAC unabhängige Routinen, die in einem z/OS System als Loadmodul aus der Linklib zur Verarbeitung hinzugeladen werden.

Der Aufbau entspricht der offiziellen Linkage Konvention, d.h. die Routine sollte mit der SAVE-Makro beginnen und mit RETURN enden.

Der Routine wird durch eine Parameteradresse der interne Workbereich zur Verfügung gestellt.

Beim Eintritt in die Routine enthält:	Register 15	Routine Basiswert
	Register 14	Return-Adresse
	Register 1	Parameterlist-Adresse
	Register 13	Savearea-Adresse

Die Register 2 - 12 müssen sichergestellt werden, wenn sie in der Routine benötigt werden (SAVE/RETURN).

COBOL oder PL/I Routinen werden von QPAC intern dem Language Environment (LE) übergeben und von diesem aufgerufen. Dabei wird die LE-Funktion PIPI verwendet. PIPI kennt zwei Möglichkeiten, wie die Routinen zu behandeln sind. Diese werden durch die Art der Initialisierung bestimmt: Der SUBroutine Modus oder der MAINprogramm Modus.

Die Unterschiede dieser zwei Modi sind:

1. MAINprogramm
   Die WORKING STORAGE SECTION bei COBOL Programmen wird bei jedem Aufruf neu initialisiert.
   Das COBOL Programm muss mit der Option RENT kompiliert sein.
   Das PL/I Programm ist mit der Procedure Option MAIN kompiliert.

2. SUBroutine
   Die WORKING STORAGE SECTION bei COBOL Programmen behält den Inhalt eines vorausgehenden Aufrufes bei einem erneuten Aufruf.
   Das PL/I Programm muss zwingend mit der Procedure Option FETCHABLE kompiliert sein.

In einem QPAC Programm können diese zwei Modi nicht gleichzeitig vorkommen. Deshalb wird über die `PARM` Angabe der zu verwendende Modus bestimmt: `PARM=CALL=SUB` oder `PARM=CALL=MAIN`. Fehlt eine explizite `PARM` Angabe, wird `CALL=SUB` angenommen.

Assemblerformat der Linkage Konvention zur Exitroutine:

```
CALL-'Routine',wadr
```

Abb. 163: Format der CALL Instruktion für Assembler

wadr adressiert den QPAC internen Workbereich.

```
               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)               Laden QPAC int.
                                      Storage Adresse


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

Abb. 164: Assemblerbeispiel einer Exitroutine

Unabhängige Exitroutinen können zusätzlich mit Parametern versehen werden.

```
CALL-'Loadmodul',wadr [,wadr,... ,PCBn,PCB-Name ...] [,RC]
```

Es gelten die offiziellen Linkage Konventionen, wie sie vorhergehend beschrieben sind.
Werden Workbereich Adressen definiert, gilt der Defaultwert, wie vorausgehend beschrieben, nicht mehr, d.h. es werden der Unterroutine nur die definierten Adressen übergeben.

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

Abb. 165: Übergabe von Workbereichen an externe Routinen

Zusätzlich sei darauf hingewiesen, dass das Highorder Bit im Highorder Byte der letzten Adresse in der Parameter-Adressliste auf ON gesetzt ist.

Es können auch DL/I PCB Adressen übergeben werden:

`PCBn`	=	der *n*-te PCB im PSB
`PCB-Name`	=	PCB mit dem DB-Namen (`DBN=`)

Definierte Workbereichadressen können **nicht indiziert** werden.

```
CALL-'SUBROUT',WPOS5000+X1
```

Abb. 166: Indizierte Workbereichadressen sind NICHT ERLAUBT

RC kann als letzter Parameter definiert werden. Damit wird bestimmt, dass ein von der externen Routine übergebener Returncode durch QPAC übernommen werden soll.

Falls die Operanden der CALL Definition nicht vollständig in einem Statement angegeben werden können, besteht die Möglichkeit der Definitionsfortsetzung im Folgestatement, wenn mit Komma und Blank unterbrochen wird. Die erste Non-Blank Position im Folgestatement gilt danach als Fortsetzung.

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

Abb. 167: Definitionsfortsetzung bei CALL Instruktionen

Der Returncode, der durch eine externe Routine übergeben werden soll (oder im QPAC Coding selbst gesetzt wird (RC=)), kann abgefragt werden:

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

Abb. 168: Abfrage des Returncodes RC

RC ist ein Spezialregister Name und als solcher intern vorgegeben.

External Subroutines (LINK Exit Routines)

```
LINK-'Loadmodul'   [ ,Parameter,Parameter ]
```

Der LINK Befehl entspricht in seiner Funktion dem CALL Befehl mit dem Unterschied, dass die Routine jedes Mal neu geladen wird und nach dem Aufruf wieder freigegeben wird.

Die Definitionen der Operanden sind unter der Beschreibung des CALL Befehls nachzusehen.

```
LOAD-'Loadmodul',Ptrfeld[,Xn]

LOAD-Feldname,Ptrfeld[,Xn]
```

Mit diesem Ladebefehl können externe Tabellen (Loadmodule bei z/OS) geladen werden, die anschliessend im QPAC-Programm verarbeitet werden können. Als Operanden müssen ein Pointer-Feld und ein Indexregister definiert werden. Im Pointer-Feld steht anschliessend die Adresse der Speicherstelle, in der die Tabelle beginnt, und im Indexregister steht die Länge der Tabelle (Modul Länge).
Tabellen, die auf diese Weise geladen werden, können anschliessend mit einer Based Struktur angesprochen werden. Dabei muss die Struktur dem Aufbau der Tabellenelemente entsprechen.

```
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
```

Abb. 169: Beispiel Load Table Based Structure

Das Ende der Tabelle kann entweder durch ein Ende-Element erkannt werden oder indem die Element-Längen kumuliert werden und mit der Maximallänge im Indexregister verglichen wird.

Anstelle des direkt definierten Modulnamens kann auch der Name eines Feldes angegeben werden, in welches zur Ausführungszeit der Modulname gespeichert werden kann, bevor der `LOAD` Befehl durchgeführt wird.

Handelt es sich beim geladenen Modul um ein Assembler Programm, kann dieses anschliessend mit via `CALL-Ptrfeld,...` aufgerufen werden.

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

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

Abb. 170: Beispiel dynamisches Laden eines Modules

Löschen geladener Tabellen oder Subroutinen

```
DELMOD-'Loadmodul'
DELMOD-Symbolname
```

Mit diesem Befehl können Module, die vorausgehend mit LOAD geladen worden sind, wieder aus dem Speicher gelöscht werden.

QPAC als Subroutine (Aufruf aus User-Hauptprogramm)

Aus einem Assembler- oder Cobolprogramm kann ein QPAC-Batch Programm wie folgt als Subroutine aufgerufen werden:

```
LOAD   ..QPAC..                  QPAC-Nucleus laden und die
ST     R1,QPACENTR               ENTRY Adresse sicherstellen
```

Erster Aufruf

Der erste Aufruf des geladenen QPAC Nucleus dient dazu, das QPAC-Batch Programm zu assemblieren. Dabei wird ein Parameterbereich übergeben, in dem QPAC PARM Optionen definiert werden können. Als erster Operand muss im PARM Bereich die Option **SUBINIT** stehen. Diese legt fest, dass QPAC als Subprogramm aus einem übergeordneten Programm gestartet werden soll. Anschliessend können weitere Optionen angegeben werden, die QPAC selbst betreffen, z.B. NOLIST,NOLOGTIT,WORK=*nnnnn*

Der Source Code des QPAC Programmes wird unter z/OS standardmässig über //QPACIN DD * eingelesen. Falls dies nicht möglich ist, kann unter z/OS mit der PARM Definition QPGM=*Progname* das Einlesen über den PDS Dataset //QPACPGM gesteuert werden. Dabei ist *Progname* der Membername, unter dem der QPAC Source Code abgelegt ist.

Eine weitere Möglichkeit bildet die assemblierte Form als Loadmodul, das durch die PARM Definition QMOD=*Progname* gesteuert wird. Die obigen Möglichkeiten sind im Kapitel 1: Einführungsteil unter PARM Option beschrieben.

Eine dritte Möglichkeit ist die Übergabe des QPAC-Programmes via internen Workbereich. Wenn die Länge in den ersten zwei Bytes des PARMs grösser als 100 ist, wird automatisch angenommen, dass das QPAC-Programm in 80 Bytes Elementen dem SUBINIT Statement folgt. Die Länge enthält dann die Grösse des Programms, das mit einem END Statement abgeschlossen wird.

QPAC kann sich so auch selbst laden und aufrufen (LOAD-'QPAC'... ...CALL-*Pointer*,...).

Wenn QPAC als Subprogramm von einem QPAC Hauptprogramm aufgerufen wird, kann das Listing des Subprogrammes über das DD Statement //QPACSUBL DD SYSOUT=* vom Hauptprogramm separiert werden.

Als weiteres Parameter-Adresspaar kann nach Bedarf die Adresse eines Bereiches, des Externen Bereiches, und dessen Länge übergeben werden, der im QPAC-Programm via Feldsymbole angesprochen wird, vergleichbar mit einer DSECT in Assembler, bzw. einer LINKAGE SECTION in COBOL.

Ob ein solcher Bereich vorgesehen ist wird von QPAC aufgrund des gesetzten High-Order Bits erkannt, oder durch eine DUMMY Definition (Nullwert).
Ein vorhandener Bereich wird nicht ins QPAC-Programm übertragen. Die Länge kann maximal 3MB gross sein.

Im Subroutine-QPAC kann dieser Bereich durch implizite Symbole `XPOS`*`nnn`* oder Zuordnung von eigenen Symbolnamen nach der Regel *`nn`*`X=SYMBOL` angesprochen werden.

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

Abb. 171: Adressierung des Externen Bereiches

```
         L      R15,QPACENTR                              1)
         CALL   (15),(PARMS),VL                           2)
oder     CALL   (15),(PARMS,DUMMY)                        3)
oder     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)

oder

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) Initialisierungsaufruf ohne Bereich
3) ohne VL, ohne Bereich
4) mit Bereich
5) RC=0?
6) Fehler gefunden?
7) Längenwert ist 0

Abb. 172: QPAC als Subroutine: Erster Aufruf

Folgeaufrufe

Folgeaufrufe bedeuten jedes Mal eine Durchführung des QPAC-Batch Programmes. Dieses muss sinnvollerweise wie ein Unterprogramm konzipiert sein. Dem QPAC-Programm können nach Bedarf zusätzliche Workbereiche übergeben werden. Diese werden beim Aufruf in den QPAC internen Workbereich übertragen und stehen dort dem QPAC-Programm zur Verfügung.

Beim CALL Aufruf werden dazu 3 Angaben geliefert:
1. Position des internen Workbereiches, auf die der Bereich übertragen wird
2. Länge des Bereiches
3. Adresse des Bereiches

Vor der Rückkehr werden solche Bereiche wieder zurück übertragen und stehen darauf dem aufrufenden Programm wieder zur Verfügung.

Beim CALL Statement wird ein PARM Feld definiert, in dem das Schlüsselwort **SUBEXEC** steht. Dies signalisiert dem QPAC-Programm, dass es sich um einen Aufruf zur Programm-Durchführung handelt.
Ob zusätzliche Datenbereiche übergeben werden sollen, wird wiederum mit dem Highorder Bit oder einem DUMMY Feld (Nullwert) angegeben. Dasselbe gilt auch für die Anzahl definierter Bereiche, bzw. das Ende der Operanden.

```
         L     R15,QPACENTR
         CALL  (15),(PARMS),VL                          1)
oder     CALL  (15),(PARMS,DUMMY)                       2)
oder     CALL  (15),(PARMS,WPOS,WLEN,WAREA,...),VL
oder     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) Durchführen Programm ohne Bereich
2) ohne VL, ohne Bereiche
3) ohne VL, mit Bereichen

Abb. 173: QPAC als Subroutine: Folgeaufrufe

Letzter Aufruf

Ein letzter Aufruf muss erfolgen, um alle von QPAC-Batch dynamisch angelegten Bereiche und das assemblierte Programm wieder aus dem Speicher zu löschen.

Beim CALL Statement wird ein PARM Feld definiert, in dem das Schlüsselwort **SUBTERM** steht. Dies signalisiert QPAC, dass es sich um den Abbau aller dynamisch angelegten Bereiche und geladener Loadmodule handelt.
Danach ist das QPAC-Programm nicht mehr vorhanden.

```
         L     R15,QPACENTR
         CALL  (15),(PARMEND),VL                          1)
oder     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)
```

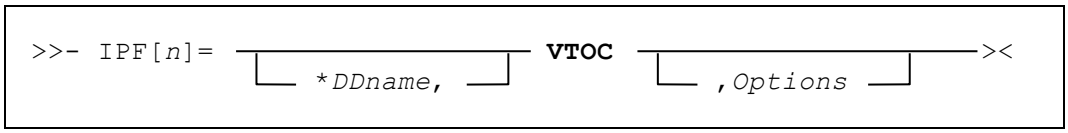
1) mit VL
2) ohne VL
3) Entry address

Abb. 174: QPAC als Subroutine: Letzter Aufruf

Kapitel 10. Systemlibraries und Systemkomponenten

VTOC

Grundformat der VTOC Filedefinition



DDname	Explizit definierter DD Name. Fehlt diese Angabe, wird IPF als DDname genommen
Options	Zusätzliche Options, wie im Kapitel 2: Input/Output Definitionen beschrieben
F4	Wird als Option bei der Filedefinition F4 angegeben, ist der erste VTOC Record der DSCB F4 (Format 4) Record, der ausgegeben wird.

Für das Lesen der VTOCs wird ein DD Statement benötigt, welches auf den Disk zeigen muss, von dem die VTOC Records gelesen werden sollen. Ein Filename (DSN=) ist nicht notwendig.

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

Abb. 175: DD Statement für z/OS VTOC-Records

Allgemeine Informationen zur VTOC Anwendung

Die VTOC des adressierten Disks wird logisch gelesen. Dabei werden die Format 1 Records und die zugehörigen Extent Angaben sowie die Format 8, Format 9 und Format 3 Records zur Verfügung gestellt.

Nachdem QPAC den VTOC File geöffnet hat, steht die Volume Serial Number in der FCA ab Displacement 2 zur Verfügung.

Die Positionen 1 bis 135 des Format 1 Records bleiben unverändert. Ab Position 136 und folgende werden durch QPAC alle Extent-Informationen eines zum Format 1 Record gehörenden Format 3 Records angehängt, so dass mit einem Lesebefehl alle diesbezüglichen Angaben verfügbar sind.

Pro Extent-Angabe werden 10 Bytes belegt. Das Ende dieser Extent Angaben wird durch das Vorhandensein von X'0000' erkannt.

Bei **EAV Volumes** werden auch Format 8 und Format 9 Records plus eventuell dazu gehörende Format 3 Records zurückgegeben. Es werden die kompletten Records (140 Byte) zurückgegeben. Das Record Layout ist im IBM Handbuch „DFSMSdfp Advanced Services“ beschrieben. Auf Position 45 steht das Format des Records.

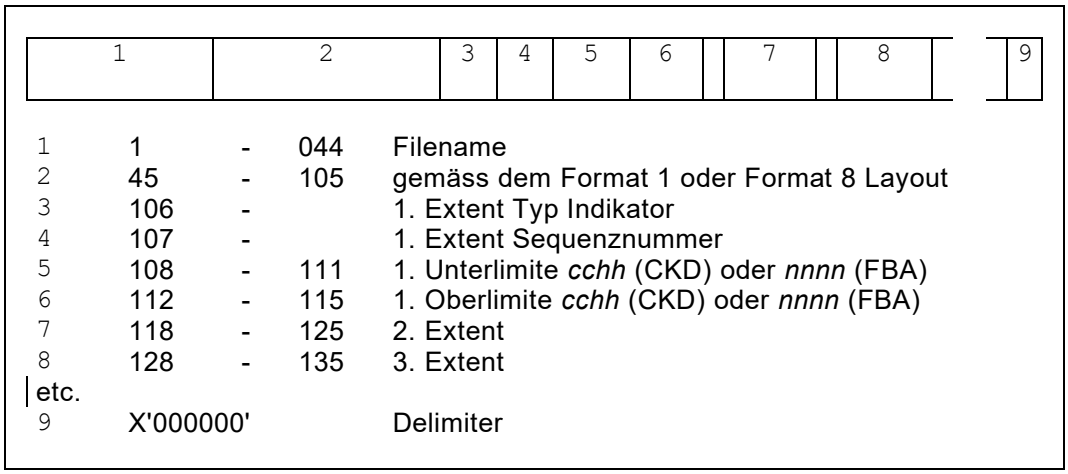


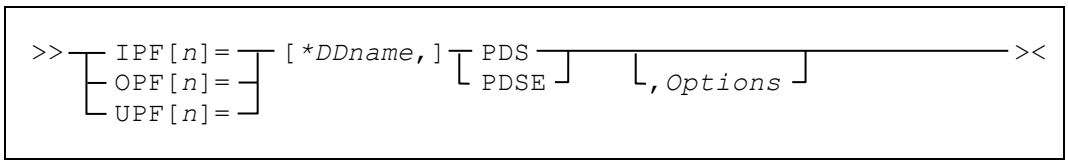
Abb. 176: Das QPAC VTOC Layout

```
//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
```

Abb. 177: Beispiel Lesen von VTOCs

z/OS-Libraries (Partitioned Data Sets)

Grundformat der z/OS-Library Filedefinition



DDname — Explizit definierter DD Name.
Fehlt diese Angabe, wird IPF als DDname genommen.

Options — Zusätzliche Options,
wie im Kapitel 2: Input/Output Definitionen beschrieben

DIR — Directory only
Anstelle von Datenrecords werden nur Directoryinformationen geliefert. Die Directory-Definition kann auch dynamisch über das reservierte Feldsymbol `..DIR` definiert werden. Dazu muss vor dem OPEN das Feld `..DIR` mit einem "D" abgefüllt werden.
z.B. `SET I1DIR = 'D'`

FCA=.... — die FCA Option ist wichtig. Die FCA wird dynamisch zugeordnet, sofern sie nicht explizit definiert worden ist

MN=.... — Membername oder Librarynummer
Einzelne Members oder Gruppen von Members können selektiv gelesen werden.
Die Definition erfolgt nach dem Prinzip des 'pattern matching', wobei die folgenden zwei Wildcards verwendet werden können:

- `*` repräsentiert **ein oder mehrere Zeichen**.
  Es ist nur ein Sternzeichen pro Definition erlaubt und dieses muss entweder am Anfang oder am Schluss stehen.
- `+` **Markiert eine Position** und repräsentiert **ein gültiges Zeichen**. Dieser Platzhalter kann irgendwo innerhalb der Definition stehen.

Diese Memberselektion kann auch dynamisch über das reservierte Feldsymbol `..MN` definiert werden. Dazu muss vor dem OPEN das Feld `..MN` mit dem Membernamen oder einem generischen Anteil abgefüllt werden.

NOE — Nach `SETGK/SETEK` soll kein E-Statuscode in der FCA zurückgegeben werden.

EOM — End of Member
Das Ende jedes Members wird angezeigt. Im `..RC2` Feld wird nach dem letzten Member Record ein "E" zurückgegeben, ohne Record in der Record Area.

MN=*name*	individueller Membername
MN=++++T	alle 5 Bytes langen Membernamen, die mit T enden
MN=*T	alle Membernamen, die mit T enden
MN=T*	alle Membernamen, die mit T beginnen
MN=*Z+	alle Membernamen mit Z in der zweitletzten Position

Abb. 178: Memberselektion für z/OS PDS Libraryfile

OPF[*n*]=PDS erlaubt, in einen bestehenden PDS neue Members einzufügen. Mit der PUT Instruktion werden die Memberrecords geschrieben. Zum Schluss muss der Membername in das FCA-Feld ..MEMNM gestellt werden; mit der Instruktion STOW-*id* wird das Member in die PDS-Directory eingetragen.
Im FCA-Feld ..STOWID wird signalisiert, ob ein bestehendes Member ersetzt ('R'), oder nur ein Neueintrag durchgeführt werden darf ('A').

UPF[*n*]=PDS erlaubt, bestehende Members inhaltlich zu verändern und neue Members einzufügen.
Mit der PUTA Instruktion werden neu zu erstellende Members geschrieben. Solche Members müssen mit der STOW Instruktion (wie vorausgehend unter OPF[*n*]=PDS beschrieben) abgeschlossen werden.
Mit der Instruktionsfolge GET - PUT können bestehende Memberrecords modifiziert, nicht aber Records hinzugefügt oder gelöscht werden.
Eine Veränderung der Recordanzahl kann durch „Umkopieren“ mit der Instruktionsfolge GET - PUTA erreicht werden.

Ebenfalls können mit der PUTD Instruktion Members gelöscht werden. Vor der Ausführung des Befehls PUTD muss der Membername ins FCA-Feld ..MEMNM gestellt werden. Nach der Ausführung steht im FCA-Feld ..RC2 der Returncode 'G', falls das Member nicht vorhanden war.

Allgemeine Hinweise zur z/OS-Library Anwendung

Die PDS Libraries können fixe Recordlänge besitzen oder als 'undefined' definiert sein. In jedem Falle wird die aktuelle Recordlänge in der FCA ab Displacement 12 (4 Bytes binär) übergeben.
Bei 'undefined' PDS ist zu beachten, dass die defaultmässige QPAC interne Recordarealänge 32760 Bytes gross ist. Dies kann zu einer WRONG-LENGTH Situation führen. Mit einer expliziten RL= Definition kann abgeholfen werden:

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

Abb. 179: Recordlängen Definition für PDS

Ein Partitioned Dataset kann als ganzer Libraryfile oder memberweise gelesen werden. In jedem Falle wird eine FCA zur gegenseitigen Kommunikation benötigt. Im Feld ..LENG steht die aktuelle Anzahl der gültigen Datenbytes in binärer Form des gelesenen PDS Records. Falls der PDS 'undefined' Format besitzt und die möglichen Recordlänge grösser als 32760 Bytes sein kann, muss in der Filedefinition der Operand RL=*nnnnn* definiert werden. Dieser Operand überschreibt die Defaultgrösse, welche für die interne Workbereichzuordnung benötigt wird.

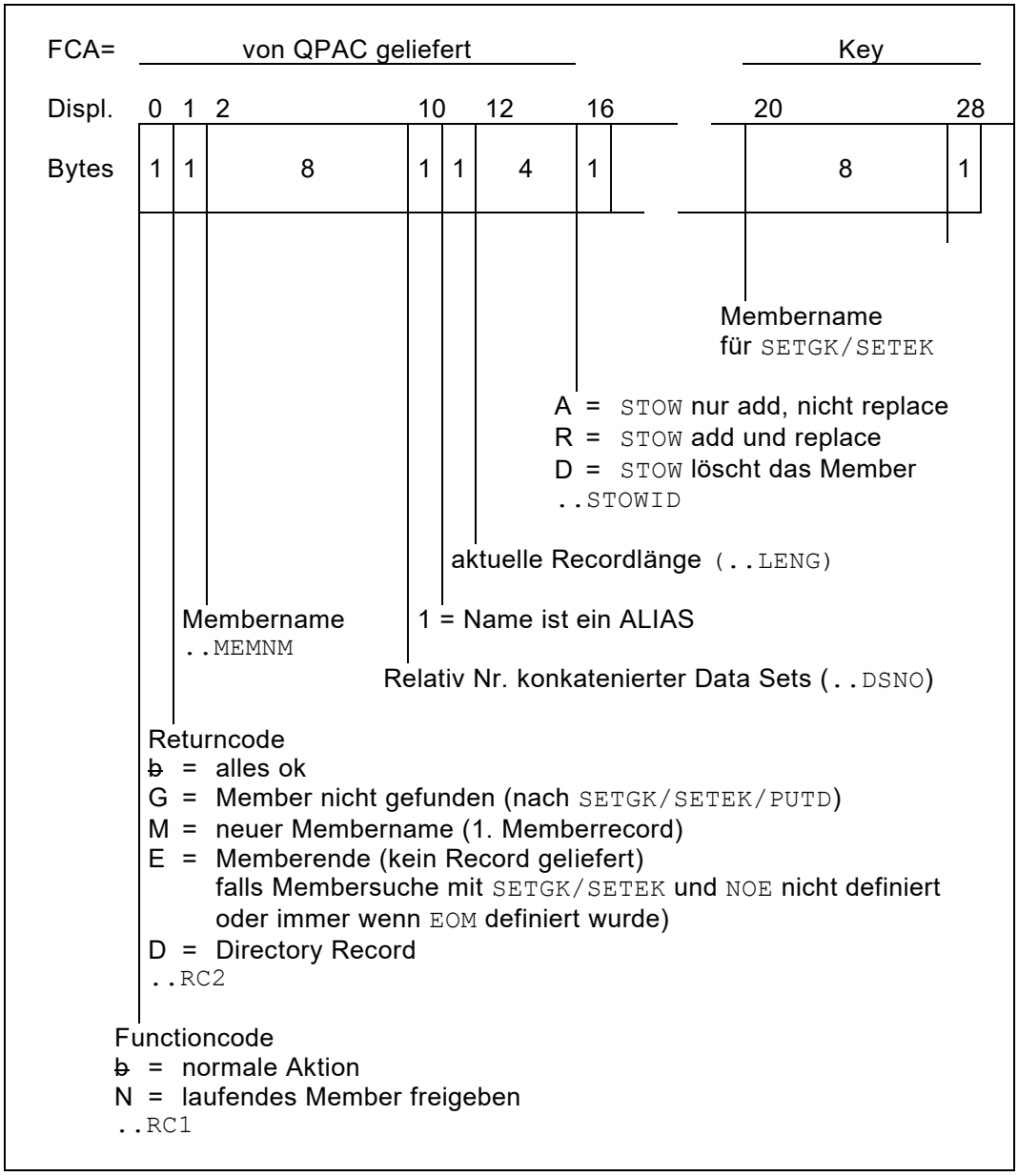


Abb. 180: FCA für PDS Libraryfile Zugriff

Bei PUT für Outputfiles bzw. PUTA bei Updatefiles werden neu die Statistik Informationen im Directory Record nachgeführt. Dabei können vor dem STOW Command zusätzlich zum STOWID Feld nach Bedarf auch die neuen Felder STOWVV, STOWMM oder STOWUSER abgefüllt werden.

..STOWVV,BL1	Version für die PDS Directory Statistik
..STOWMM,BL1	Modification Level für die PDS Directory Statistik
..STOWUSER,CL7	UserId für die PDS Directory Statistik

Nach dem ersten GET Befehl eines Members (M im ..RC2 Feld) stehen in der FCA zusätzliche Informationen aus dem Directory Statistik Record, in aufbereiteter Form. Diese können mit Hilfe der folgenden reservierten Felder abgefragt werden:

..MEMDIRVV,BL1	Version im Member Directory Statistik Record
..MEMDIRMM,BL1	Modification im Member Directory Statistik Record
..MEMDIRCRDT,PL5	Creation Date im Member Directory Statistik Record
..MEMDIRCHDT,PL5	Changed Date im Member Directory Statistik Record
..MEMDIRTIME,PL4	Time hhmmss im Member Directory Statistik Record
..MEMDIRSIZE,BL2	Anzahl Records im Member Directory Statistik Record
..MEMDIRINIT,BL2	Initial size im Member Directory Statistik Record
..MEMDIRUSER,CL7	UserId im Member Directory Statistik Record

Der ..RC2 Code "G" nach einem SETGK Befehl signalisiert, dass der Membername im vorgegebenen Keyfeld nicht vorhanden ist. Beim nachfolgenden GET Befehl wird das nächsthöhere Member angezogen oder EOF eingeleitet, wenn nicht zusätzlich EOM definiert ist.

Wird beim Code "G" nach einem SETEK Befehl mit GET weitergelesen, wird das 1. Member des PDS angezogen.

Der ..RC2 Code "D" nach einem GET Befehl signalisiert, dass ein Directory Record gelesen wurde.

SCAT (z/OS System Catalog)

Grundformat der z/OS System Catalog Filedefinition

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

SCATNM=	Catalog Name Fehlt dieser Operand, wird der ganze Katalog mitsamt den Alias-Katalogen gelesen. Dies kann sehr lange dauern! Der jeweils aktuelle Katalogname steht in der FCA.
SDSNM=	Selected Data Set Name Mit diesem Operanden können nach den Regeln des "pattern matching" Data Sets generisch selektiert werden:

z.B.

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

Options	Zusätzliche Options (die Sinn machen)
FCA=....	Die FCA kann in den internen Workbereich gelegt werden. Fehlt diese Option, wird die FCA dynamisch angelegt und kann nur via die vorgegebenen Symbolnamen angesprochen werden.
VOLID	Es werden nur DSN, Typ und VOLID aus dem Katalog zurückgegeben. Dies benötigt deutlich weniger Zeit und Systemressourcen.
DIR	Es werden die Informationen, die im Catalog vorhanden sind, zurückgegeben.
FULL	Es werden die Informationen aus dem Catalog, ergänzt mit VTOC und VSAM, zurückgegeben.
--	Fehlen VOLID, DIR oder FULL, werden die Informationen für VSAM nicht zurückgegeben, nur aus dem Catalog und VTOC.

Die FCA für den z/OS System Catalog besitzt den folgenden Aufbau:

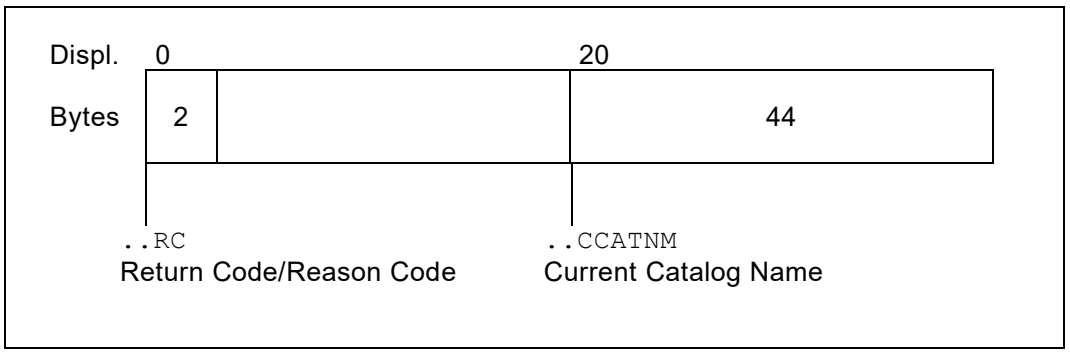


Abb. 181: FCA für z/OS System Catalog

Return Code und Reason Code sind im Messages Manual Volume 3 unter der Message IDC3009I zu finden.

Im Feld ..CCATNM Current Catalog Name steht während der Verarbeitung der Catalog Name, der gerade gelesen wird.
Die Data Set Informationen werden gemäss der nachfolgenden Record-Struktur zur Verfügung gestellt. Pro GET Instruktion wird jeweils ein Record übergeben.

01-44	DSN Data Set Name	
45-	Data Set Typ:	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 ?????? (wenn nicht vorhanden)	
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	

Abb. 182: SCAT Recordstruktur

Alle Felder sind im Character Format. Ob einzelne Feldinhalte vorhanden sind, hängt vom jeweiligen File Typ ab.

Der angesprochene Catalog Name und/oder die zu selektierenden Data Set Namen können auch vor dem OPEN dynamisch in die reservierten Feldnamen ..SCATNM bzw. ..SDSNM gestellt werden. Dies bedingt aber eine explizite Filedefinition.

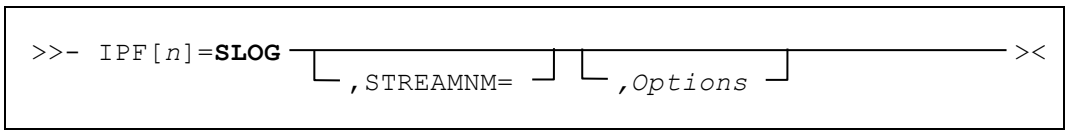
z.B.

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

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

Abb. 183: Beispiel SCAT

SLOG (z/OS System Logger)

Grundformat der z/OS System Logger Filedefinition



STREAMNM=	Log Stream Name Der Stream Name ist notwendig. Er kann auch dynamisch vor dem OPEN ins reservierte Feld I*n*STREAMNM gespeichert werden.

z.B.

```
SET I1STREAMNM = 'SYSPLEX.OPERLOG'
```

Options	Zusätzliche Options
GMT=NO|YES	Bestimmt über Timestamp local (NO) oder Greenwich mean time (YES). Default ist NO.
ACTIVE|ALL	ACTIVE bestimmt, dass nur aktive Daten vom Logstream zurückgegeben werden. ACTIVE ist default. ALL bestimmt, dass aktive und inaktive Daten zurückgegeben werden.
Start Position:	
TIMESTAMP=	yyyymmddhhmmsstt Mit der TIMESTAMP Definition kann die Startposition festgelegt werden, ab der gelesen werden soll. Die Leserichtung ist abhängig von zusätzlichen Operanden. Dieser Parameter sollte durch TIMESTAMPFROM= ersetzt werden. Wenn TIMESTAMP= verwendet wird, wird die Warning QPAC182W ausgegeben.
TIMESTAMPFROM=	yyyymmddhhmmsstt Mit der TIMESTAMPFROM Definition kann die Startposition festgelegt werden, ab der gelesen werden soll. Die Leserichtung ist abhängig von zusätzlichen Operanden.
TIMESTAMPTO=	yyyymmddhhmmsstt Mit der TIMESTAMPTO Definition kann die Endposition festgelegt werden, bis zu der gelesen werden soll.
YOUNGEST OLDEST	Der jüngste Datenblock wird gelesen. Der älteste Datenblock wird gelesen. Default ist OLDEST.
Direction:	
OLDTOYOUNG YOUNGTOOLD	Es wird vom ältesten zum jüngsten Datenblock gelesen. Es wird vom jüngsten zum ältesten Datenblock gelesen. Default ist OLDTOYOUNG.

`FCA=....` Die FCA kann in den internen Workbereich gelegt werden. Fehlt diese Option, wird die FCA dynamisch angelegt und kann nur via die vorgegebenen Symbolnamen angesprochen werden.

`RC=YES` Return Code / Reason Code
Wird diese Definition angegeben, wird beim Auftreten eines Return/Reason Codes nicht abgebrochen, sondern dieselben in den FCA Feldern zurückgegeben.

Zu beachten: In diesem Falle müssen auch X'0846' und X'0848' explizit abgefragt werden.

Die FCA für den z/OS System Logger besitzt den folgenden Aufbau:

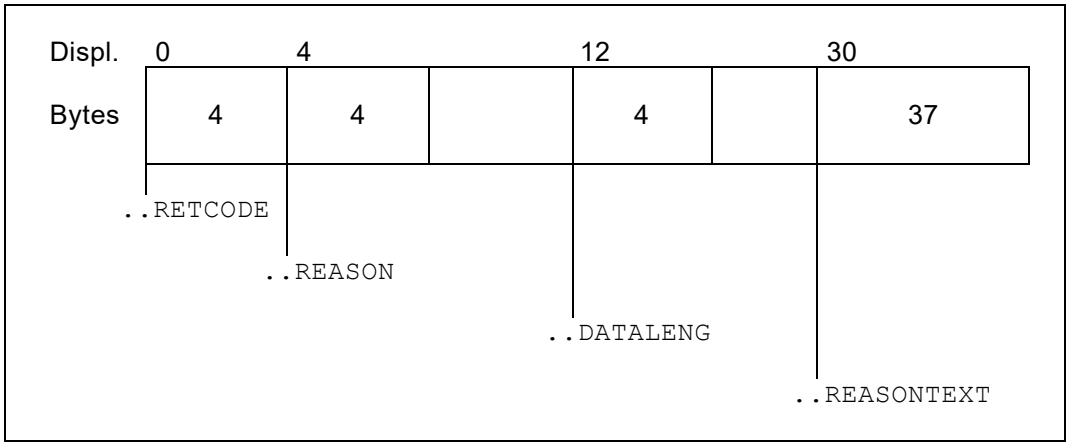


Abb. 184: FCA für z/OS System Logger

Return Code und Reason Code sind im Manual Authorized Assembler Services Reference Volume 2 unter IXGBRWSE Macro zu finden.

Die Struktur der übergebenen Stream Records ist abhängig vom jeweils definierten Stream Namen.

Zwei Reason Codes haben aus Sicht von QPAC keine direkte Bedeutung als Error.
Der Reason Code X'0846' sagt aus: Der Log Stream ist empty und wird als EOF interpretiert, wenn `RC=YES` nicht definiert ist.
Der Reason Code X'0848' sagt aus: No more data exists in the Log Stream und wird als EOF interpretiert, wenn `RC=YES` nicht definiert ist.

```
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
```

Abb. 185: Beispiel 1 SLOG

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

Abb. 186: Beispiel 2 SLOG

Kapitel 11. Integrierte Funktionen (Function Box)

Funktionenübersicht

BINTABS( )	Binary Table Search.
CALENDAR( )	Universelle Datum-Umrechnungsroutine für die einzelnen Datum-Komponenten: bürgerliches Format, julianisches Datum, Woche, Wochentag
CHANGEF( )	Ersetzen von Character Strings in Files
CHANGER( )	Ersetzen von Character Strings in Records.
CHANGEW( )	Ersetzen von Character Strings im int. Workbereich
COMPAREF( )	Vergleicht zwei Files miteinander
COMPARER( )	Vergleicht zwei Recordareas miteinander
IDCAMS( )	VSAM Catalog Functions.
IEBCOPY( )	z/OS Utility Functions.
PRINTF( )	Druckt einen File in Character/Hex-Format aus
PRINTR( )	Druckt den in der Recordarea befindlichen Record in Character/Hex-Format aus
PRINTW( )	Druckt den angegebenen Workbereich, Hiperspace oder Externen Bereich in Character/Hex-Format aus
SCANF( )	Durchsucht einen File nach einem Zeichenstring
SCANR( )	Durchsucht einen Record nach einem Zeichenstring
SCANW( )	Durchsucht einen Workbereich nach einem Zeichenstring
SEQCHK( )	Prüft einen File auf richtige Sortiersequenz
SETIME( )	Ermöglicht das Setzen von Zeitintervallen
SNAP( )	Erstellt einen „Schnappschuss“ vom Inhalt von Feldern wie Symbolen, Index Registern, ACCUs.
SORTF( )	Sortiert Inputfiles auf einen Outputfile
SORTR( )	Sortiert Records
SORTW( )	Sortiert einen internen Workbereich (Tabelle)

Abb. 187: Übersicht integrierte Funktionen

Anwendungshinweise

Die folgenden Spezifikationen betreffen alle Funktionen:

a) Eine Funktionsroutine ist dadurch definiert, dass dem Schlüsselwort als Funktionsname eine linke Klammer angeschlossen folgt.

```
COMPAREF(Parameter)
```

Abb. 188: Grundformat der Funktionsroutinen

b) Die Parameter-Angaben, die in Klammern stehen, sind ohne Blank-Stellen zu definieren.

c) Eine Funktionsdefinition mit den zugehörigen Parametern muss vollständig in einem Statement stehen.
Fortsetzungen über mehr als eine Zeile sind nicht unterstützt.

d) Die Parameterangaben können auf zwei Arten definiert bzw. an die Funktionsroutine übermittelt werden:

- man kann die Parameter direkt zwischen die Klammern platzieren

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

Abb. 189: Parameterangaben für Funktionsroutinen

- man kann die Parameter in den internen Workbereich von QPAC platzieren und sagt bei der Funktionsdefinition zwischen den Klammern nur, wo die Angaben stehen. Dazu ist das Schlüsselwort `PARM=` zu verwenden:

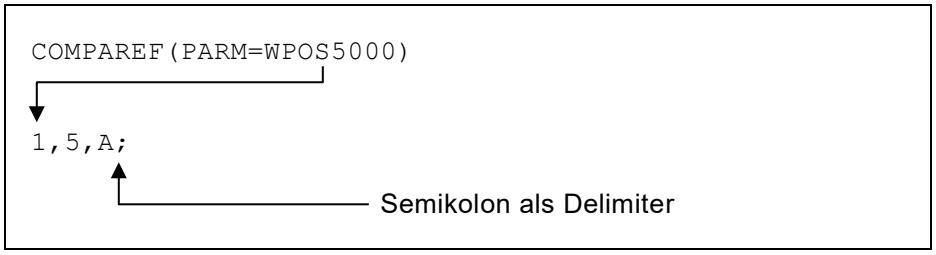


Abb. 190: Parameterangaben im internen Workbereich

Letztere Möglichkeit erlaubt, die Parameter während der Ausführung von QPAC dynamisch zu ermitteln und zu übergeben.

e) Die Parameter werden erst bei Aufruf der Funktionsroutine durch die Routine selbst auf ihre Richtigkeit geprüft und nicht schon durch QPAC zur Übersetzungszeit. Das Ende der Parameterfolge wird durch ein Semikolon (;) gekennzeichnet.

f) Im Allgemeinen ist die Reihenfolge der Parameterangaben wahlweise. Dabei gilt jedoch die Regel, dass von sich abhängige Parameterangaben in der Reihenfolge der Abhängigkeit definiert werden müssen, da deren Interpretation von links nach rechts erfolgt. Zum Beispiel müssen eventuelle File-ID Angaben vor deren Recordpositionsangaben stehen.

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

Abb. 191: Reihenfolge der Parameter

Diesbezügliche Vorgaben sind in den Detailbeschreibungen der einzelnen Funktionen zu beachten.

g) Einzelne Funktionen behandeln ganze Filebestände, z.B. `COMPAREF()` (Compare File). Dabei ist folgendes zu beachten:

- ist der Inputfile vor Aufruf der Funktion schon eröffnet (open), beginnt die Verarbeitung mit dem zuletzt gelesenen Record.
- ist der Inputfile noch nicht eröffnet, wird er durch die Funktion selbst eröffnet.
- ein Outputfile (Printer) wird durch die Funktion eröffnet, sofern er es noch nicht ist. Nach Funktionsende bleibt der Outputfile eröffnet, es sei denn, dass bei der Detailbeschreibung einer einzelnen Funktion explizit ein anderer Hinweis steht.
- Inputfiles werden nach Funktionsende geschlossen.

h) Durch die einzelnen Funktionen notwendige Printausgaben erfolgen normalerweise über den internen Systemprinter mit der Operation `PUTLST`. Wenn jedoch als Parametereintrag ein Outputfile definiert wird, kann der Output jederzeit auf einen definierten Printerfile umgeleitet werden.

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

Abb. 192: Umleitung von Printausgaben

i) Wenn die einzelne Funktionsroutine bei der Interpretation der Parameter Fehler entdeckt, wird eine Message ausgegeben und die Durchführung abnormal beendet.
Wird hingegen kein Abbruch gewünscht, kann dies mit dem Schlüsselwort `NOABEND` als Parameterdefinition angegeben werden. In diesem Falle wird durch die Funktionsroutine der Return Code auf 12 gesetzt, der anschliessend abgefragt werden kann.

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

Abb. 193: Abfrage des Funktions Return Codes

j) Wenn Funktionsroutinen mit Inline-Subroutinen kooperieren, kann es notwendig sein, dass von und zu der Funktionsroutine Steuerinformationen ausgetauscht werden müssen. Zu diesem Zwecke gibt es ein Spezial-Register `FC` (Functioncode).
`FC` ist ein 4 Byte binäres Register und kann als solches abgefragt oder mit einem Wert gefüllt werden.

k) Wenn als Parameter Fileidentifikationen (z.B. `IPF1` oder `ODB2`) angegeben werden können, gilt grundsätzlich die Möglichkeit, dass auch die Kurzschreibweise angewendet werden kann (z.B. `I1` od. `O2`).

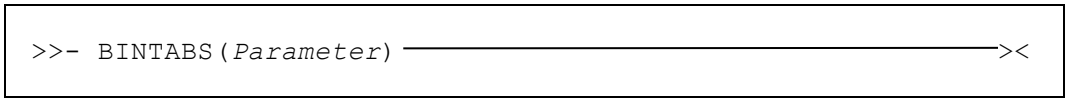
l) Es gibt Funktionsroutinen, bei denen die interne Arbeitsweise und das Ergebnis unmittelbar eine Folge von Vergleichszuständen ist.
Als Beispiel gelten die Compare-Funktionen, bei denen der Ungleich-Zustand gedruckt wird.
Bei vergleichsabhängigen Funktionen dieser Art besteht jeweils die Möglichkeit, anstelle der Printausgabe die Kontrolle einer Inlinesubroutine zu übergeben, in der nach Bedarf die zur Verfügung gestellten Records oder Bereiche verarbeitet werden können.

Diese Subroutine-Steuerung wird definiert durch Schlüsselwort-Parameter `EQ=`, `NE=` usw., denen der Subroutine-Name beigefügt wird. Die einzelnen Schlüsselwortparameters sind je nach Funktion unterschiedlich.

Funktionsroutinen sind **nicht rekursiv** aufrufbar, d.h. aus einer angesprungenen Subroutine darf die gleiche Funktionsroutine nicht mehr verwendet werden.

BINTABS( ): Binary Table Search

Mit dieser Funktion können grosse Tabellen, die sich im internen Workbereich oder dem Hiperspacebereich befinden, abgesucht werden. Die Funktion setzt voraus, dass die Tabellen aufsteigend mit fixen Elementlängen sortiert sind.
Die Funktion arbeitet nach dem Prinzip der Einmittung.



Über Parameter werden die notwendigen Informationen an die Funktionsroutine übergeben.

Parameter:

[WK=]WPOS*nnnn*:X*n* HPOS*nnnn*:X*n* *Symbol*:X*n*	Workarea *von-bis* Definition als dynamische Dimensionsangabe. Der Inhalt des Indexregisters (*bis* Angabe) versteht sich als Displacement und wird von der Funktion zum *von*-Wert addiert, um das Tabellen-Ende festzustellen.
RL=*nnn* X*n*	Länge eines Elementes in der Tabelle, als Direktwert oder als Indexregister.
KP=*nn-mm* *nn,l*	Key Position innerhalb des Tabellenelementes. Diese Sortierposition kann als *von-bis* oder als *von,länge* definiert werden. Es wird angenommen, dass die Tabelle nach diesen Positionen aufsteigend sortiert ist.
ARG=WPOS*nnnn*:X*m* *Symbol*:X*m*	Argument, nach dem in der Tabelle gesucht wird. Die Länge des Argumentfeldes muss der Länge des Keyfeldes (KP=) entsprechen. Dies wird angenommen. In das Indexregister wird der Offset des gefundenen Elementes zurückgegeben.
EQ=Subroutine	Subroutine, die angesprungen wird, wenn das Element in der Tabelle gefunden wird. Im Indexregister X*n* steht in diesem Falle der Offset des gefundenen Elementes zum Tabellenanfang.
NE=Subroutine	Subroutine, die angesprungen wird, wenn das Element in der Tabelle nicht gefunden wird.

Die Definition von Subroutinen ist optional. In jedem Falle steht nach der Rückkehr aus der Funktion im Function-Register FC 0 (null), wenn das Element gefunden wurde oder 1, wenn es nicht gefunden wurde.

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

Die Tabelle im internen Workbereich ab Position 10001 wird nach dem Wert im Argument-Feld `FIELD` abgesucht. Wenn das Element gefunden wird, steht der Offset zum Tabellenanfang im Indexregister `X2` und die Subroutine `SUBEQ` wird angesprungen. Nach der Rückkehr aus der Subroutine geht die Verarbeitung hinter der Funktion `BINTABS` weiter.

```
BINTABS(TABSTART:X4,KP=5,9,ARG=ARGFELD:X9,EQ=SUBEQ,RL=100)
```

Die Tabelle beginnt beim Feld `TABSTART` und befindet sich im Hiperspace. Sie ist nach den Positionen 5-13 sortiert.

```
BINTABS(HPOS1:X5,KP=1-9,RL=50,ARG=ARGUMENT:X6)
IF FC = 0  THEN found IFEND
```

Dieses Beispiel zeigt eine Abfrage, ob das Element in der Tabelle vorhanden ist oder nicht, ohne dass eine Subroutine definiert wurde.

Abb. 194: BINTABS() Beispiele

CALENDAR( ): Datum Umrechnungsroutine

Mit dieser Funktion kann auf einfache Art und Weise von einer Datumsform auf eine andere umgerechnet werden.

Unterstützt sind sechs Hauptformate:

- bürgerliches Datum *DDMMYY*
- bürgerliches Datum mit Jahrhundert (century) *DDMM**CC**YY*

- julianisches Datum *YYDDD*
- julianisches Datum mit Jahrhundert (century) ***CC**YYDDD*

- Wochenformat Datum *YYWWD*
- Wochenformat Datum mit Jahrhundert (century) ***CC**YYWWD*

Das **bürgerliche Datum** ist das gebräuchlichste Format in sechs möglichen Reihenfolgen der Glieder Jahrhundert, Jahr, Monat, Tag:

- `YMD` bedeutet Year Month Day
- `CYMD` bedeutet Century Year Month Day
- `MDY` bedeutet Month Day Year
- `MDCY` bedeutet Month Day Century Year
- `DMY` bedeutet Day Month Year
- `DMCY` bedeutet Day Month Century Year

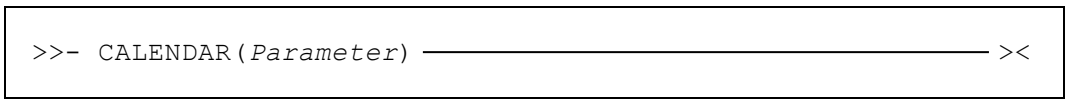
Das **julianische Datumformat** bezeichnet den fortlaufend gerechneten Tag innerhalb des Jahres: ([*CC*]*YYDDD* = Day within year).

Das **Wochenformat** bezeichnet die Darstellung in Wochen und Wochentag innerhalb des Jahres ([*CC*]*YYWWD* = Week and day of week within year).

Zusätzlich können zu einem Datum auch eine Anzahl Tage, Monate oder Jahre addiert bzw. subtrahiert werden.

Auch das Ermitteln der Differenz zwischen zwei Kalenderdaten wird unterstützt. Das Ergebnis wird in diesem Fall in Anzahl Tagen oder als "Date Duration yyyymmdd" in die Output-Workposition (OWP) geschrieben.

Alle Werte der verschiedenen Input, Output und Arithmetik-Formate beziehen sich immer auf ein gepacktes 8-Byte grosses Feld im Workbereich.



Über Parameterdefinitionen müssen die detaillierten Informationen übergeben werden.

Parameter:

`IWP=datef=wpos` Input-Workposition
definiert das Ausgangsformat des Datums und die Position innerhalb des Workbereiches, wo sich in einem **8-Byte gepackten Feld** das Datum befindet.

Beim Indikatorformat (`IYMD=` und `IJUL=`) wird der Indikator 0 als 1900 und der Indikator 1 als 2000 interpretiert.

datef = Datumsformat (date format), kann sein:

YMD	= YYMMDD
MDY	= MMDDYY
DMY	= DDMMYY
JUL	= YYDDD
YWK	= YYWWD
CYMD	= CCYYMMDD
IYMD	= 0IYYMMDD
DMCY	= DDMMCCYY
CJUL	= CCYYDDD
CYWK	= CCYYWWD
IJUL	= 0IYYDDD

wpos = Workbereichposition

`+WP=datef=wpos`
`-WP=datef=wpos`
Datums-Arithmetik Workposition (optional)
der im 8-Byte gepackten Feld (wpos) stehende Wert wird zu `IWP` addiert (`+WP=`) oder subtrahiert (`-WP=`). Das Ergebnis steht nach der Funktion in `OWP`. Die Werte in `wpos` können auch negativ sein. Pro `CALENDAR()` Anweisung ist dieser Parameter nur einmal zulässig.

Bei der Differenz Berechnung zwischen dem IWP-Datum und dem -WP-Datum wird das tiefere vom höheren Datum subtrahiert. Ist das IWP-Datum tiefer, wird das Ergebnis bei `OWP=DAY` oder `OWP=DUR` negativ.

datef = Datumsformat (date format), kann sein:

DAY	= Anzahl Tage
MON	= Anzahl Monate
YEAR	= Anzahl Jahre
CYMD	= `CCYYMMDD` (nur bei "`-WP=`" gültig)
DMCY	= `DDMMCCYY` (nur bei "`-WP=`" gültig)

wpos = Workbereichposition

`OWP=`*`datef`*`=`*`wpos`*

Output-Workposition
definiert das gewünschte Ergebnisformat des Datums und die Workbereichposition, wo sich in einem **8-Byte gepackten Feld** dasselbe nach Rückkehr befindet.

Zusätzlich werden alle reservierten `CALDR…` Felder mit dem entsprechenden Inhalt abgefüllt. Ausser wenn bei *`datef`* DUR bzw DAY angegeben wurde.

`datef` = gewünschtes Datumsformat, gleiche wie bei `IWP` plus zusätzlich:

`DAY` = Differenz zwischen IWP-Datum und –WP-Datum in Tagen

`DUR` = Differenz zwischen IWP-Datum und –WP-Datum in der Form yyyymmdd

`wpos` = Workbereichposition

`WP=`*`datef`*`=`*`wpos`*

Mit dieser Definitionsform wird die Datumsumformung im gleichen Feld zurückgegeben. Ebenfalls wird das Ergebnisformat vom Sendeformat, wie nachfolgend dargestellt, abgeleitet:

`YMD` ergibt `JUL`
`MYD` ergibt `JUL`
`DMY` ergibt `JUL`
`JUL` ergibt `YMD`

Es handelt sich um einen einfachen Wechsel zwischen bürgerlichem und julianischem Datumsformat.

`NOABEND`

Kein Abbruch bei fehlerhaftem Datum.
Ist das Eingabedatum ungültig, wird im Outputfeld 0 zurückgegeben, wenn `NOABEND` definiert ist.
Zusätzlich wird der Function Code (`FC=`) auf 8 gesetzt.

`WINDOW=nn|50`
`WDOW=`

Ist das Jahrhundert gleich 00 oder nicht angegeben, wird standardmässig ein Window mit der Jahresgrenze 50 angenommen.
Das heisst, dass die Jahre vor 50 mit Jahrhundert 20 gerechnet werden, die Jahre ab und inklusive 50 mit dem Jahrhundert 19.
Mit diesem Parameter kann der Standardwert von 50 überschrieben werden.

```
CALENDAR(IWP=YMD=WPOS6000,OWP=JUL=WPOS6010)
```

Das Datum soll vom Format YYMMDD ins julianische Format umgerechnet werden (YYDDD). Das Ergebnis steht in einem 8-Byte Feld auf Position 6010 in gepackter Form.

```
CALENDAR(IWP=JUL=WPOS6010,OWP=YWK=WPOS6900)
```

Das Datum in julianischem Format soll umgerechnet werden ins Wochenformat (YYWWD Jahr, Woche, Wochentag). Das Ergebnis steht im 8-Byte Feld auf Workarea Position 6900.

```
CALENDAR(IWP=YMD=WPOS6000,OWP=DMY=WPOS6000)
```

Das Datum im Format YYMMDD soll ins Format DDMMYY umgestellt werden. Das Ergebnis steht im gleichen Feld wie das Ausgangsdatum.

```
CALENDAR(IWP=CYMD=WPOS6000,+WP=DAY=WPOS6010,OWP=DMCY=WPOS6020)
```

Zum Datum in der Position 6000 soll die Anzahl Tage aus der Position 6010 addiert werden. Das Ergebnis wird in der Form DDMMCCYY in die Position 6020 geschrieben.

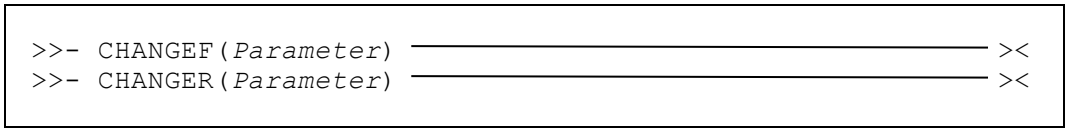
```
CALENDAR(IWP=DMCY=WPOS6040,-WP=CYMD=WPOS6050,OWP=DAY=WPOS6060)
```

Es soll die Differenz zwischen dem IWP-Datum und dem -WP-Datum berechnet werden. Die Anzahl Tage stehen als 8-Byte grosses Feld auf der Postition 6060. Mit „OWP=DUR=WPOS6060“ würde das Resultat als Date-Duration (yyyymmdd) gespeichert.

Abb. 195: CALENDAR() Beispiele

CHANGEF( ) / CHANGER( ): Ersetzen von Character Strings

Mit dieser Funktion können in Datenfiles Character Strings ersetzt werden. Die Organisation des Datenbestandes muss ein Updating zulassen, d.h. er muss sich auf Disk befinden und ein SAM oder VSAM Data Set sein.
`CHANGEF()` liest den ganzen Datenbestand bis EOF durch und schreibt die veränderten Records jeweils auf den Datenbestand zurück.
`CHANGER()` behandelt nur den Record, der sich im Moment in der I/O Area befindet und schreibt ihn nicht zurück.



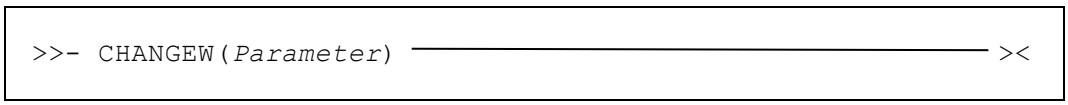
Über Parameter werden die notwendigen Informationen an die Funktionsroutine übergeben.

Parameter:

`UPF`*n*	File Ident, der bearbeitet werden soll. Er muss als Update-File definiert sein. Zu Testzwecken kann der File auch als `IPF`*n* angegeben werden. Damit wird die Veränderung nur simuliert und nicht auf den Bestand zurückgeschrieben.
`DLM='`	Delimiter für die Character-String Definitionen. Per Default wird ein Apostroph angenommen. Es kann hier ein anderer Character definiert werden.
'oldvalue'newvalue'	Alter String, der gesucht wird und neuer String, der den alten String ersetzt. Ist der alte String kürzer als der neue, wird der nachfolgende Teil des Records nach rechts verschoben. Ist der alte String länger als der neue, wird der nachfolgende Teil des Records nach links verschoben und an den neuen String angehängt.
`EQ=`*Subroutine*	Eine Subroutine kann definiert werden, die angesprungen wird, wenn der alte String im Record gefunden wird.
`NE=`*Subroutine*	Ein Subroutine kann definiert werden, in die verzweigt wird, wenn im Reocrd der alte String nicht gefunden wird.
`OPF`	Ein Printerfile Ident kann angegeben werden. In diesem Falle werden die Veränderungen auf diesen Printer gedruckt, andernfalls erfolgt der Print Output über QPACLIST (SYSLST).

CHANGEW( ): Ersetzen von Character Strings in Workarea-Tabellen

Mit dieser Funktion können in Workarea-Tabellen Characterstrings ersetzt werden. Die Tabellen müssen aus fixen Elementlängen bestehen. CHANGEW() beginnt am Anfang der Tabelle und durchsucht jedes Element nach dem definierten Characterstring und ersetzt ihn wenn vorhanden.



Über Parameter werden die notwendigen Informationen an die Funktionsroutine übergeben.

Parameter:

[WK=]WPOS*nnnn*:X*n* HPOS*nnnn* WPOS*nnnn*-WPOS*nnnn* WK=*Symbol*:X*n*	Workarea von-bis Definition als dynamische Dimensionsangabe. Der Inhalt des Indexregisters (bis Angabe) versteht sich als Displacement und wird von der Funktion zum von-Wert addiert, um das Tabellen-Ende festzustellen.
RL=*nnn* X*n*	Länge eines Elementes in der Tabelle, als Direktwert oder als Indexregister.
DLM='	Delimiter für die Character-String Definitionen. Defaultmässig wird ein Apostroph angenommen. Es kann hier ein anderer Character definiert werden.
'*oldvalue*'*newvalue*'	Alter String, der gesucht wird und neuer String, der den alten String ersetzt. Ist der alte String kürzer als der neue, wird der nachfolgende Teil des Records nach rechts verschoben. Ist der alte String länger als der neue, wird der nachfolgende Teil des Records nach links verschoben und an den neuen String angehängt.
X*n*	X*n* als erstes alleinstehendes Indexregster beinhaltet bei einem Eintritt in eine Subroutine den Offset des Elementes relativ zum Tabellenanfang. Dieser Parameter ist optional.
X*m*	X*m* als zweites alleinstehedes Indexregister beinhaltet bei einem Eintritt in eine Subroutine im EQ (Equal) Falle den Offset des gefundenen Wertes relativ zum Anfang des Elementes. Dieser Parameter ist optional.
X*c*	X*c* als drittes alleinstehendes Indexregister beinhaltet bei einem Eintritt in eine Subroutine die Elementnummer. Dieser Parameter ist optional.
EQ=*Subroutine*	Eine Subroutine kann definiert werden, die angesprungen wird, wenn der alte String im Element gefunden wird.

NE=*Subroutine*	Eine Subroutine kann definiert werden, in die verzweigt wird, wenn im Element der alte String nicht gefunden wird.
OPF[*n*]	Ein Printerfile kann angegeben werden. In diesem Falle werden die Veränderungen auf diesen Printer gedruckt, andernfalls erfolgt der Print Output über QPACLIST (SYSLST), sofern keine Subroutine definiert wurde.

```
CHANGEW(WK=WPOS10001:X1,RL=020,DLM=/,/Value1/Value2/,
                                          X2,X3,X4,EQ=SUBEQ)
```

Die Tabelle im internen Workbereich ab Position 10001 wird nach dem Wert1 abgesucht. Wenn der Value1 im Element gefunden wird, erfolgt eine Aenderung auf Value2 und der Aufruf der Subroutine `SUBEQ`. Dann steht der Offset des Elementes zum Tabellenanfang im Indexregister `X2`, der Offset zum korrigierten Value2 innerhalb des Elementes im `X3` und die Elementnummer im `X4`.

```
CHANGEW(TABSTART:X4,RL=100,'Value1'Value2',NE=SUBNE,X8)
```

Die Tabelle beginnt beim Feld `TABSTART` und befindet sich im Hiperspace. Die Elementlänge ist 100. Bei Elementen ohne vorhandenem Value1 wird in die Suroutine `SUBNE` gesprungen.

```
CHANGEW(HPOS1-HPOS5001,RL=X5,'Oldvalue'Newvalue',OPF)
```

Dieses Beispiel zeigt eine Tabelle im Hiperspace. Gefundene Elemente werden über den Printfile `OPF` ausgelistet.

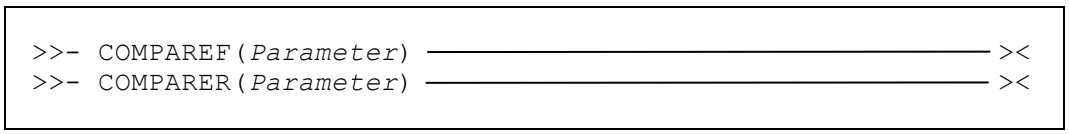
Abb. 196: CHANGEW() Beispiele

COMPAREF( ) / COMPARER( ): Compare Files / Recordareas

Diese Funktion vergleicht die Inhalte der Datenrecords zweier definierter Files miteinander und protokolliert Ungleichheiten über einen Printoutput oder übergibt die Kontrolle einer Inline-Subroutine.

Die Funktion `COMPAREF()` verarbeitet die zwei Files gesamthaft durch bis EOF, d.h. öffnet und schliesst sie selbständig. Nach Austritt aus der Funktionsroutine sind die zwei Files in geschlossenem Zustand.

Die Funktion `COMPARER()` liest selbst keine Records, sondern vergleicht nur den Inhalt der Record Areas. Diese Funktion entspricht in der Parameteranwendung der Funktion `COMPAREF()`.



Über Parameter können verschiedene Anweisungen an die Funktionsroutine übergeben werden.

Parameter:

`(*)`	keine näher spezifizierten Parameter vorhanden. Der Vergleich erfolgt eins zu eins über die ganze Recordlänge beider Files.
`IPF`*n*`,IPF`*m*	Damit wird explizit bestimmt, welche Inputfiles verglichen werden sollen. Fehlen Fileidentifikationsdefinitionen, sucht sich die Funktion selber die ersten zwei Inputfiles und vergleicht dieselben. Am Ende ist ersichtlich, welche Files verglichen wurden.
`OPF`*n*	Wenn der Printoutput auf einen bestimmten Printfile (`OPF=PR`) ausgegeben werden soll, kann die Fileidentifikation definiert werden. Ist sowohl `OPF`*n* wie auch Subroutinen (`EQ=`, `NE=` etc.) definiert, werden sowohl die Subroutinen angesprungen als auch der Print Output ausgegeben. Soll innerhalb einer der Subroutinen `NE`, `R1` oder `R2` kein Print Output ausgegeben werden, kann dies mit dem Function Code 4 (`FC=4`) mitgeteilt werden.
`SRT=`*s*`,`*l*`,A [,...]` `  `*s*`,`*l*`,D`	Damit wird festgelegt, dass die zwei Files in dieser vorgegebenen Weise sortiert sind. Der Funktion ist es dadurch auch möglich, festzustellen, ob Records fehlen. Bis zu 20 Sortierdefinitionen sind möglich. Fehlt eine `SRT=` Sequenzangabe, werden die zwei Files eins zu eins nachgelesen.
`NODUPREC`	Gilt nur für `COMPAREF()`: Records mit gleichem Sortierschlüssel werden unterdrückt. Nur der erste Record wird übernommen.

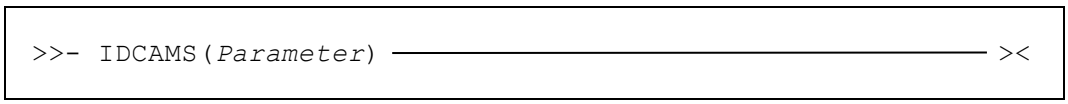
von-bis [,...]	Damit wird festgelegt, welche Positionszonen verglichen werden sollen; wenn beispielsweise die zwei Files unterschiedliche Recordlängen aufweisen, jedoch der Anfangsbereich gleich strukturiert ist. Bis zu 20 Bereichsdefinitionen sind möglich.
CHR	Der Printoutput soll nur in Characterformat erfolgen.
EQ=*Subname*	Wenn zwei gleiche Records verglichen werden, soll die Kontrolle dieser Subroutine übergeben werden.
NE=*Subname*	Wenn zwei zu vergleichende Records ungleich sind, soll die Kontrolle dieser Subroutine übergeben werden.
R1=*Subname*	Wenn bei der Sequenz-Kontrolle nur der Record von File 1 vorhanden ist, soll in diese Subroutine verzweigt werden. Fehlt diese Definition, geht die Kontrolle zu NE.
R2=*Subname*	Wenn bei der Sequenz-Kontrolle nur der Record von File 2 vorhanden ist, soll in diese Subroutine verzweigt werden. Fehlt diese Definition, geht die Kontrolle zu NE, sofern definiert.

COMPAREF(*)	Die ersten zwei Inputfiles werden verglichen
COMPAREF(IPF5,IPF9)	Die Inputfiles I5 und I9 werden verglichen
COMPAREF(SRT=1,4,A)	Die ersten zwei Inputfiles werden verglichen. Die Files sind nach den Kolonnen 1 - 4 aufsteigend sortiert.
COMPAREF(SRT=1,4,A,10,2,D,1-80)	Die ersten zwei Inputfiles werden miteinander verglichen. Die Files sind sortiert in Kolonnen 1-4 aufsteigend und 10-11 absteigend. Es sollen nur die Kolonnen 1-80 verglichen werden.
COMPAREF(IPF1,IPF3,OPF,CHR)	Die zwei Files I1 und I3 werden verglichen. Der Printoutput soll auf den Printerfile OPF ausgegeben werden und nur in Characterformat.
COMPARER(IPF1,IPF3,OPF,CHR)	Die *Recordareas* der zwei Files I1 und I3 werden verglichen. Der Printoutput soll auf den Printerfile OPF ausgegeben werden und nur in Characterformat.
COMPAREF(IPF1,IPF3,NE=SUBNE)	Die zwei Files I1 und I3 werden verglichen. Bei ungleichen Recordpaaren wird die Routine SUBNE angesprungen.
COMPARER(IPF1,IPF3,NE=SUBNE)	Die *Recordareas* der zwei Files I1 und I3 werden verglichen. Bei ungleichen Recordpaaren wird die Routine SUBNE angesprungen.

Abb. 197: COMPAREF() / COMPARER() Beispiele

IDCAMS( ): VSAM Catalog Functions

Mit dieser Funktion können die IDCAMS Utility Funktionen direkt im QPAC Programm integriert ausgeführt werden. Die zugehörigen Definitions-Statements werden der Funktion im internen Workbereich übergeben.
Diese Funktion steht nur in z/OS zur Verfügung.



Über Parameter werden die notwendigen Informationen an die Funktionsroutine übergeben. Dabei gibt es für die Übergabe der Statements an IDCAMS zwei Möglichkeiten:

1. Die zu übergebenen Statements werden in Form einer Tabelle im internen Workbreich übergeben.
2. Die Statements werden in einer Input-Subroutine einzeln übergeben. Die Routine wird sooft aufgerufen bis der Function Code 8 übergeben wird (`FC=8`).

Das Ergebnis aus IDCAMS kann ebenfalls auf zwei Arten empfangen werden:

1. Der Output wird über IDCAMLST ausgegeben, wenn keine Output Subroutine definiert wird.
   Fehlt das IDCAMLST DD Statement, wird es dynamisch angelegt, mit Output Class A. Eine andere Output Class kann mit dem Parameter `SYSOUT=x` zugeteilt werden.
2. Der Output kann über eine Output-Subroutine in Empfang genommen werden. Pro Zeile wird die Subroutine aufgerufen bis das Ende erreicht ist.

Nach der Rückkehr aus der Funktion wird ein eventueller interner MAXCC Return Code aus dem IDCAMS Utility im Function Return Code `FC` zurückgegeben.

Parameter:

`STMT=WPOS`*nnnn* `HPOS`*nnnn*	Workarea-Position ab welcher die Statements für die IDCAMS Funktion gespeichert sind. Die Statements entsprechen genau der Syntax, die das offizielle IDCAMS Utility voraussetzt. Die einzelnen Statements sind in der Länge von 80 Byte Elementen abzuspeichern. Das Ende ist durch ein /* oder Blank-Element gekennzeichnet.
`SYSOUT=`*x*	Fehlt eine Output Subroutine Definition, kann dem dynamisch allozierten IDCAMLST DD Statement eine Output Class zugeteilt werden.
`ISU=InSubroutine`	Name der Input Subroutine. Die IDCAMS Function ruft diese Subroutine statementweise auf. Das zu übergebende Statement steht in der `WPOS`*nnnn*, die durch den Parameter `IWP=WPOS`*nnnn* definiert ist. Die Statements entsprechen genau der Syntax, die das offizielle IDCAMS Utility voraussetzt, 80 Bytes lang. Das Ende wird durch die Übergabe des Function Codes 8 (`FC=8`) signalisiert.

`IWP=WPOS`*nnnn* Workarea Position, in die das durch die Input-Subroutine zu übergebende Statement gestellt werden muss, 80 Bytes lang.

`OSU=OutSubroutine` Output Subroutine Name. Diese Routine wird pro Ergebniszeile aufgerufen. Die Ergebniszeile steht an der `WPOS`*nnnn*, die durch den Parameter `OWP=WPOS`*nnnn* definiert werden muss.
Die einzelne Zeile ist 121 Bytes lang, mit führendem ASA Controlcharacter.

`OWP=WPOS`*nnnn* Workarea Position, in der die von der Output-Subroutine erhaltene Ergebniszeile steht, in der Länge von 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
```

Abb. 198: Beispiel 1 IDCAMS()

```
SET WPOS7001,CL80 = ' LISTCAT ALL '
SET WPOS7081,CL80 = '/*'

IDCAMS(STMT=WPOS7001,OWP=WPOS5200,OSU=OUTSUB)

  SUB-OUTSUB
     PUTLST
  SUBEND
```

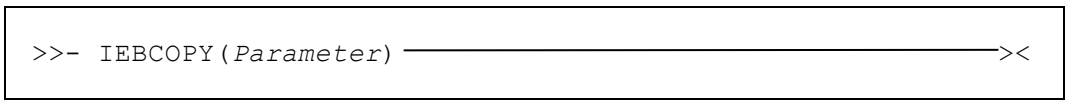
Abb. 199: Beispiel 2 IDCAMS()

```
SET WPOS7001,CL80 = ' LISTCAT ALL '
SET WPOS7081,CL80 = '/*'

IDCAMS(STMT=WPOS7001,SYSOUT=T)
IF FC NOT = 0 THEN ...internal MAXCC code ... IFEND
```

Abb. 200: Beispiel 3 IDCAMS()

IEBCOPY( ): z/OS Utility Functions

Mit dieser Funktion können die IEBCOPY Utility Funktionen direkt im QPAC Programm integriert ausgeführt werden. Die zugehörigen Definitions-Statements werden der Funktion im internen Workbereich übergeben.
Diese Funktion steht nur unter z/OS zur Verfügung.



Über Parameter werden die notwendigen Informationen an die Funktionsroutine übergeben. Dabei gibt es für die Übergabe der Statements an IEBCOPY zwei Möglichkeiten:

1. Die zu übergebenden Statements werden in Form einer Tabelle im internen Workbreich übergeben (`STMT=...`).

2. Die Statements werden in einer Input Subroutine einzeln übergeben. Die Routine wird so oft aufgerufen, bis der Function Code 8 übergeben wird (`FC=8`).

Ein //SYSIN DD .. Statement darf bei Verwendung dieser Funktion nicht vorhanden sein. Die QPAC Programm-Definitionen müssen in diesem Falle über das //QPACIN DD ... Statement eingelesen werden.

Das Ergebnis aus IEBCOPY kann ebenfalls auf zwei Arten empfangen werden:

1. Der Output wird über SYSPRINT ausgegeben, wenn nichts Spezielles definiert wird und ein //SYSPRINT DD .. vorhanden ist.

2. Der Output kann über eine Output Subroutine in Empfang genommen werden. Pro Zeile wird die Subroutine aufgerufen bis das Ende erreicht ist. In diesem Falle darf kein //SYSPRINT DD.. Statement vorhanden sein.

Nach der Rückkehr aus der Funktion wird ein eventueller interner Return Code aus dem IEBCOPY Utility im Function Return Code `FC` zurückgegeben.

Parameter:

`STMT=WPOS`*nnnn*
`HPOS`*nnnn*
: Workarea-Position ab welcher die Statements für die IEBCOPY Funktion gespeichert sind. Die Statements entsprechen genau der Syntax, die das offizielle IEBCOPY Utility voraussetzt.
Die einzelnen Statements sind in der Länge von 80 Byte Elementen abzuspeichern. Das Ende ist durch ein /* oder Blank-Element gekennzeichnet.

`ISU=`*InRoutine*
: Name der Input Subroutine. Die IEBCOPY Funktion ruft diese Subroutine pro Statement auf. Das zu übergebende Statement steht in der `WPOS`*nnnn*, die durch den Parameter `IWP=WPOS`*nnnn* definiert ist. Die Statements entsprechen genau der Syntax, die das offizielle IEBCOPY Utility voraussetzt, 80 Bytes lang. Das Ende wird durch die Übergabe des Function Codes 8 (`FC=8`) signalisiert.

`IWP=WPOS`*nnnn*
: Workarea Position. Hierhin wird das durch die Input Subroutine zu übergebende Statement gestellt. Länge 80 Bytes.

`OSU=`*`OutRoutine`* Name der Output Subroutine. Diese Routine wird pro Ergebniszeile aufgerufen. Die Ergebniszeile steht in der `WPOS`*`nnnn`*, die durch den Parameter `OWP=WPOS`*`nnnn`* definiert werden muss. Die einzelne Zeile ist 121 Bytes lang, mit führendem ASA Control Character.

`OWP=WPOS`*`nnnn`* Workarea Position, in der die von der Output Subroutine erhaltene Ergebniszeile steht, in der Länge von 121 Bytes.

`SVC2..` Wenn das Programm IEBCOPY APF protected ist, kann mit diesem Parameter der SVC einer Typ 3 SVC-Routine angegeben werden, durch die temporär auf APF umgestellt wird. Siehe mitgeliefertes Beispiel einer Routine.

```
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
```

Abb. 201: IEBCOPY() Beispiele

PRINTF( ) / PRINTR( ): Print File / Recordareas

Die Funktion `PRINTF()` druckt den ganzen Inputfile, Updatefile oder MQS*n* in Hexadezimal- oder Character-Form aus und berücksichtigt vorgegebene Filedefinitionen fix, variabel oder undefined. Bei MQS*n* wird intern die GEToptions Browse und Truncation gesetzt.

Die Funktion `PRINTR()` druckt den momentanen Inhalt der Recordarea des Inputfiles, das entweder vorbestimmt oder von der Funktion selbst gefunden wurde. Am Inhalt des Bereiches wird nichts verändert.

Die Funktion `PRINTR()` selbst liest keine Records, sondern setzt voraus, dass vor deren Aufruf ein Lesebefehl erfolgte. Im Übrigen entspricht diese Funktion in der Anwendung der Parameter der Funktion `PRINTF()`.

```
>>- PRINTF(Parameter)──────────────────────────────────────── ><
>>- PRINTR(Parameter)──────────────────────────────────────── ><
```

Über Parameterdefinitionen können verschiedene variierende Anweisungen an die Funktionsroutine übergeben werden.

Parameter:

`(*)`	keine näher spezifizierten Parameter vorhanden. Der Printoutput erfolgt nach den Defaultwerten, d.h. die Funktion nimmt den ersten Inputfile, Updatefile oder MQS*n*, den sie als Definition findet und druckt ihn in Character-/Hex-Form aus.
`IPF`*n* `|` `UPF`*n* `|` `MQS`*n*	Der explizit bestimmte File soll ausgedruckt werden.
`OPF`*n*	Der Printoutput soll auf diesen vorbestimmten Printfile und nicht über den internen Systemprinter (QPACLIST, SYSLST) ausgegeben werden.
`CHR`	Der Printoutput soll nur in Characterformat erfolgen.

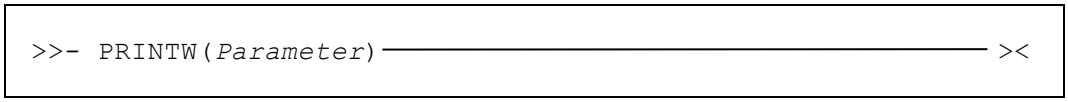
`PRINTF(*,OPF)`	Der erste Inputfile, der als Definition gefunden wird, wird über `OPF` ausgedruckt
`PRINTF(IPF5)`	Inputfile `IPF5` soll ausgedruckt werden
`PRINTF(OPF3)`	Der erste Inputfile, der als Definition gefunden wird, wird über den Printfile `OPF3=PR` ausgedruckt
`PRINTF(I1,CHR)`	Der Inputfile `IPF1` wird im Characterformat ausgedruckt
`PRINTR(I1,CHR)`	Die *Recordarea* des Inputfiles `IPF1` wird im Characterformat ausgedruckt

Abb. 202: PRINTF() / PRINTR() Beispiele

PRINTW( ): Print Workbereich , Hiperspace oder Externer Bereich

Diese Funktion druckt den internen Workbereich, den Hiperspace oder den Externen Bereich in Hexadezimal- und Character-Form aus. Es ist möglich, die ganze Area oder nur Teilbereiche auszudrucken.

Die Funktion verändert den Workbereich nicht.



Über Parameterdefinitionen können Teilbereiche angegeben, wie auch weitere Anweisungen an die Funktionsroutine übergeben werden.

Parameter:

(*)	keine näher spezifizierten Parameter. Der ganze Workbereich wird über den Systemprinter (QPACLIST, SYSLST) aufbereitet ausgegeben.
OPF*n*	Der Printoutput erfolgt über den Printerfile OPF*n* und nicht über QPACLIST oder SYSLST.
[WK=]WPOS*nnnn*-WPOS*nnnn* HPOS*nnnn*-HPOS*nnnn* XPOS*nnnn*-XPOS*nnnn*	Es soll der Bereich von Startposition bis Endposition ausgedruckt werden.
[WK=]WPOS*nnnn*:X*n* HPOS*nnnn*:X*n* XPOS*nnnn*:X*n*	Von Startposition bis zur durch das Indexregister X*n* definierten Endposition soll ausgedruckt werden. Der Inhalt von X*n* ist als Offset zu verstehen und wird von der Funktionsroutine, zur Bestimmung der *Bis-Limite*, zur *Von-Position* addiert.

PRINTW(*)	Der ganze Workbereich wird ausgedruckt
PRINTW(OPF,WPOS5000-WPOS9000)	Die ersten 4000 Bytes des Workbereiches werden über OPF ausgedruckt
PRINTW(HPOS10000:X1)	Im Indexregister X1 steht ein Offset, welches zu 10000 dazu addiert wird und die *Bis-Limite* ergibt, bis zu der ausgedruckt wird.

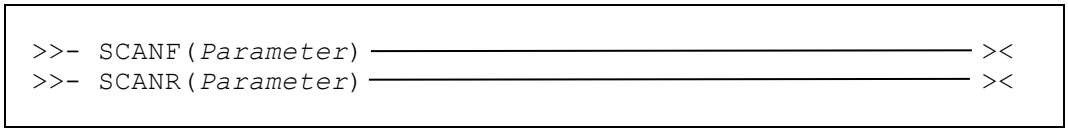
Abb. 203: PRINTW() Beispiele

SCANF( ) / SCANR( ): Scan File / Recordarea

Die Funktion SCANF() durchsucht einen Inputfile nach einem vorgegebenen Zeichen-String und schreibt diese Records heraus, in denen die Information gefunden wurde.

Die Funktion SCANR() durchsucht den momentanen Inhalt der Recordarea des Inputfiles, der entweder vorbestimmt oder von der Funktion selbst gefunden wurde. Am Inhalt des Bereiches wird nichts verändert.

Die Funktion SCANR() selbst liest keine Records, sondern setzt voraus, dass vorgängig des Aufrufes ein Lesebefehl erfolgte. Im übrigen entspricht diese Funktion in der Anwendung der Parameter der Funktion SCANF().



Über Parameterdefinitionen müssen die detaillierten Informationen übergeben werden. Diese sind teils notwendig und teils wahlweise.

Parameter:

DLM='	Delimitercharacter Dieser Parameter ist wahlweise und bestimmt den Character, der als String-Delimiter gelten soll. Defaultwert ist das '-Zeichen (Apostroph).
IPF*n*	Damit wird explizit definiert, welcher Inputfile durchsucht werden soll. Fehlt diese Definition, sucht sich die Funktion selbst den Inputfile und nimmt den ersten, den sie als Filedefinition findet.
'...' X'...'	Character- oder Hexstring zwischen den Delimiter-Zeichen, nach dem der Inputfile durchsucht werden soll. Der String kann maximal 100 Zeichen gross sein. Wird ein Hexstring angegeben, ist der Parameter DLM= ungültig.
von-bis [,...]	Damit wird festgelegt, welche Positionen nur abgesucht werden sollen, wenn beispielsweise ausserhalb dieser Scan-Zonen der Begriff vorkommen kann, aber diese Fälle nicht von Interesse sind.
CAPS=OFF|ON	Wird CAPS=OFF definiert, wird der Text auf Klein- und Grossbuchstaben durchsucht. Ist der Scan String als Hexstring definiert, wird automatisch auf CAPS=ON geschaltet. Andernfalls wird der Text wie definiert abgesucht. CAPS=OFF ist Default.
CHR	Gefundene Records sollen nur in Characterdarstellung ausgedruckt werden.
NOPRINT	Es sollen keine Records detailliert ausgedruckt werden. In jedem Falle wird am Schluss die Anzahl Records, die die Information enthalten, ausgegeben.
EQ=*Subname*	Wird ein gesuchter Begriff gefunden, wird die Kontrolle dieser definierten Subroutine übergeben.

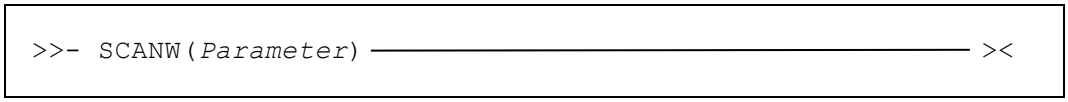
X*nn*	Im ersten Indexregister, sofern angegeben, steht der Offset des Begriffs, nach dem gesucht wurde.
X*nn*	Wurde der Begriff mehr als einmal gefunden, so enthält ein zweites Indexregister (Counter) die Anzahl Treffer.
NE=*Subname*	Ist ein Record vorhanden, der den gesuchten Begriff *nicht* enthält, wird die Kontrolle dieser Subroutine übergeben.

`SCANF('CHARACTERS')`	Durchsucht den Inputfile, der als erster gefunden wird, auf den Begriff CHARACTERS.
`SCANF(DLM=*,*CHARACTERS*)`	Das Delimiter-Zeichen soll ein Stern sein. Der Rest ist wie in Beispiel 1.
`SCANF(IPF5,'CHARACTERS',CHR)`	Der Inputfile `IPF5` soll durchsucht werden. Gefundene Records werden im Characterformat ausgedruckt.
`SCANF(IPF5,'CHARACTERS',NOPRINT)`	Gefundene Records werden nicht gedruckt. Es interessiert nur deren Anzahl, die am Ende ausgegeben wird.
`SCANF(DLM=:,:CHARACTERS:,20-80)`	Nur die Positionen 20-80 des Records werden durchsucht.
`SCANR('CHARACTERS',EQ=SUBR,X9,X10)`	Bei jedem Record, der den Begriff CHARACTERS enthält, wird in die Routine SUBR verzweigt. Im Indexregister `X9` steht der Offset, im Indexregister `X10` die Anzahl der Treffer.

Abb. 204: SCANF() / SCANR() Beispiele

SCANW( ): Scan Workbereichtabelle

Diese Funktion durchsucht die Elemente einer Tabelle, die sich im Workbereich befindet. Am Inhalt des Bereiches wird nichts verändert.



Parameter:

DLM='	Delimitercharacter Dieser Parameter ist wahlweise und bestimmt den Character, der als String-Delimiter gelten soll. Defaultwert ist das '-Zeichen (Apostroph).
[WK=]WPOS*nnnn*-WPOS*nnnn* WPOS*nnnn*:X*n* *field1*-*field2* *field1*:X*n*	Tabellenbereich im Working Storage, Hiper Space oder Single Field. Als Tabellenende kann ein Indexregister definiert werden, in welchem der Offset steht.
'...' X'...'	Character- oder Hexstring zwischen den Delimiter-Zeichen, nach dem der Record durchsucht werden soll. Der String kann maximal 100 Zeichen lang sein. Wird ein Hexstring angegeben, ist der Parameter DLM= ungültig.
RL=*nnnn*	Elementlänge in der Tabelle
CAPS=OFF|ON	Wird CAPS=OFF definiert, wird der Text auf Klein- und Grossbuchstaben durchsucht. Ist der Scan String als Hexstring definiert, wird automatisch auf CAPS=ON geschaltet. Andernfalls wird der Text wie definiert abgesucht. CAPS=OFF ist Default.
CHR	Printoutput im Characterformat
EQ=*Subname*,Xn_1,Xn_2	Wird ein gesuchter Begriff gefunden, wird die Kontrolle dieser definierten Subroutine übergeben. Im ersten Indexregister steht die Elementnummer und im zweiten Indexregister der Offset innerhalb des Elementes des gesuchten Begriffs.
NE=*Subname*,Xn_1	Wird ein gesuchter Begriff nicht gefunden, so wird die Kontrolle dieser Subroutine übergeben.
OPF*n*	Printer-Output

`SCANW('CHAR',WK=WPOS1-WPOS1000,RL=100)`	Durchsucht die Tabellenelemente mit Länge 100 auf den Begriff CHAR.
`SCANW(DLM=*,*CHARACTERS* ...)`	Delimiter-Zeichen soll ein Stern sein.
`SCANW('CHAR',EQ=SUBR,X1,X2 ...)`	Bei jedem Element, das den Begriff CHAR enthält, wird in die Routine SUBR verzweigt.

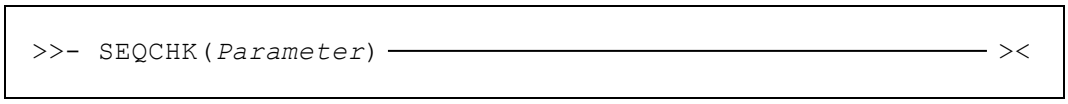
Abb. 205: SCANW() Beispiele

SEQCHK( ): Sequence Check

Diese Funktion liest den ganzen Inputfile durch und prüft ihn auf richtige Sequenz. Die für die Folgekontrolle notwendigen Informationen werden über Parameterangaben übergeben.

Die Funktion verarbeitet den ganzen File bis End of File, nachdem er durch die Funktion (sofern noch nicht) eröffnet wurde.

Fehler in der Sort-Sequenz werden angezeigt, indem die zwei Records mit falscher Sequenz ausgedruckt werden. Dadurch gilt der zuletzt gelesene Record als Basis für die Fortsetzung. Records mit **gleichen** Sortierwerten gelten als richtig.



Über Parameterdefinitionen müssen verschiedene variierende Anweisungen an die Funktionsroutine übergeben werden.

Parameter:

`IPF`*n*
: Damit wird explizit bestimmt welcher Inputfile geprüft werden soll. Diese Definition muss, wenn vorhanden, vor den Sortierangaben stehen.

`SRT=`*s*`,`*l*`,A [,....]`
`    `*s*`,`*l*`,D`
: Diese Parameterdefinition ist immer notwendig. Damit wird die Sortierung bestimmt, deren Richtigkeit geprüft werden soll.

  `s` = Sortierfeld Position im Record
  `l` = Länge des Feldes in Bytes
  `A` = Aufsteigend (ascending)
  `D` = Absteigend (descending)

  Es können bis zu 5 Sortierfelder definiert werden.

`OPF`*n*
: Damit wird bestimmt, dass die Ausgabe von Fehlerrecords auf diesen Printfile erfolgen soll.

`NOPRINT`
: Damit wird die Printausgabe der einzelnen Records, die in falscher Reihenfolge entdeckt werden, unterdrückt. In jedem Falle erfolgt am Schluss die Ausgabe der Anzahl der gefundenen Sortierfehler.

`EQ=`*Subname*
: Wenn zwei Records mit gleichen Sortkriterien gefunden werden, wird diese Routine angesprungen.

`LT=`*Subname*
: Wenn der erste Record kleiner dem zweiten Record ist, wird in diese Routine verzweigt.

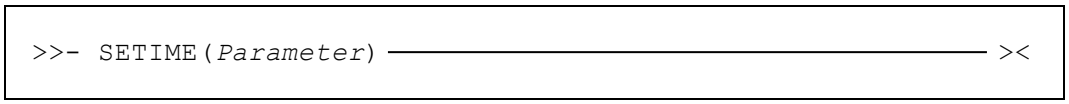
`GT=`*Subname*
: Wenn der erste Record grösser als der zweite Record ist, wird in diese Routine verzweigt.

`SEQCHK(SRT=1,4,A)`	Der Inputfile, der als erste Filedefinition gefunden wird, wird nach der Sortierung 1,4,A geprüft (Position 1, Länge 4 Bytes, aufsteigend).
`SEQCHK(IPF5,SRT=10,3,A,2,5,D)`	Der Inputfile `IPF5` wird geprüft, ob er sortiert ist nach - Position 10, Länge 3, aufsteigend - Position 2, Länge 5, absteigend.
`SEQCHK(SRT=15,9,D,NOPRINT)`	Der Inputfile, der als erste Filedefinition gefunden wird, wird nach der Sortierung Position 15, Länge 9, absteigend geprüft. Sequenz-Fehler sollen nicht detailliert gedruckt werden.
`SEQCHK(SRT=1,4,A,EQ=ROUTINE)`	Es wird geprüft, ob Records mit gleichen Sortierbegriffen vorkommen.

Abb. 206: SEQCHK() Beispiele

SETIME( ): Setzen von Zeitintervallen

Mit dieser Funktion kann ein Zeitintervall gesetzt werden, um nach Ablauf desselben in eine vordefinierte Routine zu springen, oder um solange in einen Wartestatus zu gehen.



Über Parameter werden die notwendigen Informationen an die Funktionsroutine übergeben.

Parameter:

INTERVAL=*nn* IV=*nn*	Mit dieser Definition wird das Zeitintervall definiert. Die Verarbeitung geht weiter. Nach Ablauf des Intervalls wird die momentane Verarbeitung unterbrochen und die vordefinierte Subroutine angesprungen. Nach Verlassen der Subroutine geht die Verarbeitung an der unterbrochenen Stelle weiter.
WAIT=*nn*	Mit dieser Definition wird das Zeitintervall definiert. Die Verarbeitung bleibt im Wartezustand, bis das Intervall abgelaufen ist. Danach wird, sofern definiert, die angegebene Subroutine angesprungen. Nach dem Verlassen der Subroutine geht die Verarbeitung mit den Instruktionen, die der Funktionsroutine folgen, weiter. Für z/OS bedeutet das definierte Zeitintervall **Hundertstels-Sekunden**.
SU=*Subname*	Damit wird eine Inline-Subroutine definiert, die beim Erreichen des gesetzten Zeitintervalls angesprungen wird. Die Definition der Subroutine ist optional. IV= ohne Subroutine ist jedoch sinnlos.

SNAP( ): Snapshot von QPAC Feldern und Registern

Mit dieser Funktion kann zu jedem Zeitpunkt während der Programmausführung ein Schnappschuss (Snapshot) von Workfeldern gemacht werden. Diese Informationen sind sehr nützlich für Traces und Problemanalysen.
Im Falle eines Program Checks wird diese Funktion von QPAC automatisch durchgeführt.

```
>>- SNAP(Parameter) ──────────────────────────────────────── ><
```

Über Parameter werden die notwendigen Informationen an die Funktionsroutine übergeben.

Parameter:

(*)	Keine näher spezifizierten Parameter vorhanden. Alle Feldtypen, die verwendet werden und deren Wert ungleich Null ist, werden auf dem Systemdrucker ausgedruckt.
OPF*n*	Der Printoutput soll auf diesen vorbestimmten Printfile und nicht über den internen Systemprinter (QPACLIST, SYSLST) ausgegeben werden.
XR	Es wird nur der Inhalt aller verwendeten Index Register gedruckt, deren Inhalt ungleich Null ist.
ACCU	Es wird nur der Inhalt aller verwendeten ACCUs gedruckt, deren Inhalt ungleich Null ist.
IO	Es werden nur die I/O-Areas aller verwendeten File Definitionen gedruckt, deren Inhalt ungleich Null ist.
SYM	Es wird nur der Inhalt aller verwendeten Symbole gedruckt, deren Inhalt ungleich Null ist.
SYM=*Feld1/Feld2* =*Feld** =*F+ld1*	Es wird nur der Inhalt der angegebenen Feldsymbole gedruckt. * repräsentiert eine oder mehrere Positionen, welche nicht geprüft werden + repräsentiert eine Positionen, welche nicht geprüft wird

```
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
```

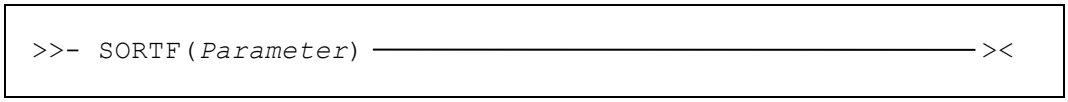
Abb. 207: Beispiel SNAP() Output

SORTF( ): Sort File

Mit dieser Funktion können ein oder mehrere Inputfiles während der Ausführung eines QPAC-Programmes sortiert und in einen Outputfile ausgegeben werden.

Die Funktion benutzt intern den IBM-Sort (bzw. kompatibler Sort eines anderen Herstellers).

Die Input- und Outputfiles werden durch die Funktion eröffnet und geschlossen. Nach der Ausführung der Funktion sind sie in geschlossenem Zustand.



Über Parameterdefinitionen werden die minimal notwendigen Informationen an die Funktionsroutine übergeben.

Parameter:

`IPF[n], UPF[n]` Damit wird explizit bestimmt, welche Inputfiles sortiert werden sollen. Fehlen die Definitionen über den zu sortierenden Input, sucht sich die Funktion alle Inputfiles selbständig zusammen.

`OPF[n]` Damit wird explizit bestimmt, welcher Outputfile den sortierten Bestand aufnehmen soll. Fehlt diese Angabe, sucht sich die Funktion den ersten definierten.

`SRT=s,l,A [,...]`

Die Sortfields-Definitionen sind im Minimum zu definieren. Wenn nur diese Angaben als Parameter übergeben werden, sucht sich die Funktion die restlichen Informationen selbständig zusammen:

- alle vorhandenen Inputfiles werden als zu sortierender Input genommen
- die erste vorhandene Outputfiledefinition gilt als sortierter Outputfile
- Format der Sortfields ist `BI` (binär)
- Anzahl Workfiles ist 1

```
SRT=s,l,f,A [,...]
      s,l,f,D
```

Die Sortfield-Definitionen können auch mit dem Feldformat-Attribut *f* versehen werden, wenn nicht alle Felder unter dem allgemeingültigen Format `BI` behandelt werden können.
Zulässige Format-Attribute sind:
`CH,ZD,PD,FI,BI,FL,Y2B,Y2C,Y2D,Y2P,Y2S,Y2Z.`

`NODUPREC` Records mit gleichem Sortierschlüssel werden unterdrückt. Nur der erste Record wird übernommen (SUM FIELDS=NONE).

`MSG` Mit dieser Definition wird gewünscht, dass die üblichen Sort-Messages über SYSOUT ausgegeben werden sollen. Default ist nur Ausgabe von Fehler-Messages.

NOABEND

Normalerweise bricht die Funktion die Verarbeitung ab, nachdem sie eine Fehlersituation festgestellt hat. Um einen vorzeitigen Abbruch zu vermeiden, kann dieser Parameter angegeben werden. Dadurch kehrt die Funktion nach einer festgestellten Fehlersituation sofort zum Aufrufpunkt zurück und übergibt einen Return Code ungleich 0. Der Return Code kann in diesem Falle abgefragt werden,

```
IF RC NOT = 0 THEN
```

und es ist möglich, die Funktion mit veränderten Parametern erneut aufzurufen.

WORK=*n*

Es wird angenommen, dass die zu sortierenden Files normalerweise im Mainstorage sortiert werden können, so dass kein externer Workfile benötigt wird. Falls der Input gross ist, kann mit diesem Parameter die Anzahl SORTWK*n*-Files angegeben werden. Je nach Anzahl (1-9) müssen die entsprechenden DD Statements vorhanden sein (SORTWK1-9).

WORKNM=*xxxx*

Falls mit Workfiles gearbeitet wird, können mit diesem Parameter die ersten 4 Stellen des Filenames der Workfiles (Default SORTWK*n*) durch einen eigenen Namen abgeändert werden:

```
WORKNM=QPAC
```

beispielsweise ergibt QPACWK1,QPACWK2 etc.

Y2PAST=*yyyy*

Damit kann das aktuelle Jahrhundert-Window definiert werden. Siehe entsprechende SORT-Literatur.

`SORTF(SRT=1,3,A)`	Alle, durch die Funktion gefundenen Inputfiles werden sortiert und auf den ersten, durch die Funktion gefundenen Outputfile, ausgegeben.
`SORTF(SRT=1,3,PD,A)`	Das Sortierfeld benützt das Format 'packed decimal' und wird dementsprechend im arithmetischen Sinne angewendet.
`SORTF(OPF5,SRT=10,4,D,1,5,A)`	Der sortierte Bestand soll auf den Outputfile 5 (`OPF5`) ausgegeben werden.
`SORTF(I1,I5,O5,SRT=7,9,CH,A)`	Die zwei Inputfiles `I1` und `I5` sollen zusammensortiert und auf `OPF5` ausgegeben werden. Am Schluss sind `I1`,`I5` und `OPF5` in geschlossenem Zustand.

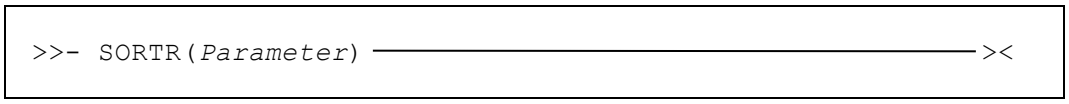
Abb. 208: SORTF() Beispiele

SORTR( ): Sortieren Records

Mit dieser Funktion können beliebige Records sortiert werden. Die Records werden der Funktion über eine Inline-Subroutine, die von der Funktionsroutine angesprungen wird, übermittelt. Nach dem Sortiervorgang werden die sortierten Records wiederum von der Funktionsroutine über eine angesprungene Inline-Subroutine zurückgegeben.

Die Funktion benutzt intern den IBM-Sort (bzw. kompatibler Sort eines anderen Herstellers).

Die Funktion selbst liest keine Records sondern setzt voraus, dass sie übermittelt werden. Im Übrigen gelten die gleichen Sortparameter-Regeln, wie sie bei der Funktion `SORTF()` beschrieben wurden.



Über Parameter werden die notwendigen Informationen an die Funktionsroutine übergeben.

Parameter:

`ISU=`*name*	Name der Input-Subroutine, die vor dem Sortvorgang so lange angesprungen wird um Records zu holen, bis über den `FC` (Functioncode) der Wert 8 übermittelt wird, der aussagt, dass keine weiteren Records folgen. Daraufhin wird sortiert. In der Input-Subroutine (Sort Exit) sind keine `GO....` Sprungbefehle erlaubt. Die Rückkehr ins Sort-Programm via `SUBEND` ist zwingend. Um einen Sort-Prozess abzubrechen, kann in der Input-Subroutine der Function Code auf 16 gesetzt werden (`FC=16` anstelle `FC=8`). Daraufhin wird keine Sortierung vorgenommen und gleich aus der `SORTR` Funktion zurückgekehrt. `FC=16` bleibt dabei bestehen und kann danach abgefragt werden.
`OSU=`*name*	Name der Output-Subroutine, die nach dem Sortiervorgang die Records in Empfang nimmt. Die Routine wird so lange angesprungen, bis alle Records übermittelt wurden.
`SU=`*name*	Anstelle von `ISU=` und `OSU=` kann eine gemeinsame Subroutine definiert werden. Der Anwender muss dabei in der Subroutine selbst kontrollieren, ob er sich im Input- oder Output-Zyklus befindet.
`IWP=`*nnnn*	Input-Workarea-Position des Bereiches, in dem die Funktion nach Rückkehr von der Input-Subroutine den zu sortierenden Record erwartet.
`OWP=`*nnnn*	Output-Workarea-Position des Bereiches, in dem die Funktionen die sortierten Records einzeln über die Output-Subroutine zurückgibt.
`WP=`*nnnn*	Anstelle von `IWP=` und `OWP=` kann ein gemeinsamer Bereich definiert werden.
`RL=`*nnn*	Länge der zu sortierenden Records (fixe Recordlänge).

`RL=V`*`nnn`*	Maximale Länge bei variablen Recordlängen.
`NODUPREC`	siehe unter `SORTF()`.
`SRT=`*`s`*`,`*`l`*`,A,...` *`s`*`,`*`l`*`,D` *`s`*`,`*`l`*`,`*`f`*`,A` *`s`*`,`*`l`*`,`*`f`*`,D`	Sortfields-Definitionen. Das Format dieser Definitionen entspricht dem des SORTs, mit oder ohne Feldformat-Attribut *`f`*. Werden keine Feldformat-Attribute angegeben, wird intern das Format `BI` angenommen. Zulässige Format-Attribute sind: `CH,ZD,PD,FI,BI,FL,Y2B,Y2C,Y2D,Y2P,Y2S,Y2Z`.
`MSG`	Mit dieser Definition wird veranlasst, dass die üblichen Sort-Messages über SYSOUT ausgegeben werden.
`WORK=`*`n`*	Anzahl SORTWK-Files (siehe auch `SORTF()`).
`WORKNM=`*`xxxx`*	Aendern der ersten 4 Stellen des SORTWK Filenamen (siehe auch `SORTF()`).
`Y2PAST=`*`yyyy`*	Damit kann das aktuelle Jahrhundert-Window definiert werden. Siehe entsprechende SORT-Literatur.

```
SORTR(ISU=INPUT,OSU=OUTPUT,WP=WPOS7001,RL=800,SRT=1,3,A)
...
   SUB-INPUT
      ...
       übertragen Record nach Position 7001
       oder setzen  FC=8  (keine weiteren Records folgen)
       ....
   SUBEND

   SUB-OUTPUT
      ...
      lesen Record von Position 7001 und verarbeiten
      ...
   SUBEND
```

Abb. 209: SORTR() Beispiel mit zwei definierten Subroutinen

```
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         (Ende des Inputs)
   IFEND
SUBEND
```

Abb. 210: SORTR() Beispiel mit einer definierten Subroutine

SORTW( ): Sortieren Workbereich

Mit dieser Funktion können z.B. Tabellen im internen Workbereich oder Hiperspace sortiert werden.

```
>>- SORTW(Parameter) ──────────────────────────────────────────── ><
```

Über Parameter werden die notwendigen Informationen an die Funktionsroutine übergeben.

Parameter:

[WK=]WPOS*nnnn*-WPOS*nnnn* HPOS*nnnn*-HPOS*nnnn*	Bereich *von-bis* Definition als feste Dimensionsangaben. Das durch die *bis*-Adresse adressierte Element wird nicht mehr mitsortiert.
[WK=]WPOS*nnnn*:X*n* HPOS*nnnn*:X*n*	Bereich *von-bis* Definition als dynamische Dimensionsangaben. Dieser definierte Bereich wird sortiert. Wird ein Indexregister angegeben, ist dessen Inhalt als Displacement zu verstehen und wird von der Funktionsroutine zum *von*-Wert dazu addiert. Das dadurch adressierte Element wird nicht mehr mitsortiert. Das Schlüsselwort `WK=` ist optional.
RL=*nnn* X*n*	Länge eines Elementes im Bereich, als Direktwert oder in einem Indexregister
SRT=*s*,*l*,A *s*,*l*,D	Sortfield-Definitionen, bis 5 Felder können angegeben werden. Feldattribute sind nicht zulässig.
NODUPREC	Duplikate werden mit Hex'FF' überschrieben und ans Ende des Workbereichs sortiert. Ein allfälliges Indexregister wird entsprechend verringert.

```
SORTW(WPOS6000-WPOS7000,RL=10,SRT=1,8,A)
```

Die Felder im 6000er Bereich werden aufsteigend sortiert.

```
SORTW(WPOS5000:X1,RL=80,SRT=5,10,D)
```

Die Records ab Adresse `WPOS5000` in der Länge, die durch `X1` gegeben wird, werden absteigend nach den Positionen 5 bis 14 sortiert.

```
SORTW(HPOS1:X1,RL=80,SRT=5,10,A,1,4,D,50,1,A)
```

Die Records ab Adresse `HPOS1` im Hiperspace in der Länge, die durch `X1` gegeben wird, werden sortiert. Positionen 5-14 aufsteigend, darin Positionen 1-4 absteigend, darin Position 50 aufsteigend.

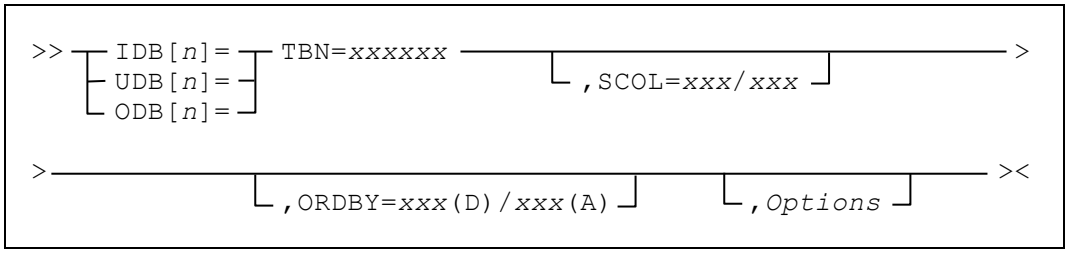
Abb. 211: SORTW() Beispiele

Kapitel 12. DB2 Support Feature

DB2 Datenbank Definition

IBM DB2 wird durch dieses Feature voll unterstützt. Es basiert auf der Voraussetzung, dass DB2 installiert ist und dass der Zugriff möglich ist.

Grundformat der DB2 DB Definition



TBN= Ist die Tabelle bzw. View, die verarbeitet werden soll.
Es können die folgenden Items spezifiziert werden:

- Lokation (für verteilte Datenbanken)
- Ersteller
- Tabellenname

```
IDBn=TBN=LOCATION.CREATOR.TABLENAME
```

Abb. 212: Eindeutige Spezifikation eines Tabellenobjektes

Der Tabellenname kann auch ein Synonym sein.
Diese Definition muss zwingend als erster Operand stehen.
An ihr erkennt QPAC, dass es sich bei dieser Datenbank um DB2 handelt.

SCOL= Selected Column Names.
Damit können aus der Tabellen-Row ausgewählte Columns selektiert verarbeitet werden. Dies wirkt sich so aus, als wären nur diese Columns vorhanden. Mehrere Column Namen werden durch einen Schrägstrich voneinander getrennt definiert.

```
SCOL=COLUMN1/COLUMN10
```

Abb. 213: Selektion von mehreren Columns

Wenn nicht alle Namen auf demselben Statement Platz finden, ist es möglich, die Definition mittels Schrägstrich-Blank (in der laufenden Zeile) auf dem Folgestatement fortzusetzen.
Der nächste Column Name wird danach auf dem Folgestatement definiert. Beliebige führende Blankstellen sind möglich.

```
SCOL=COLUMN1/
     COLUMN10
```

Abb. 214: Selektion von mehreren Columns (Forts.)

SCOL= kann nur bei IDB oder UDB definiert werden.

Bei ODB muss immer eine gesamte Row verarbeitet werden.

ORDBY= ORDER BY Klausel.
Damit kann die Sortierung der Row bzw. der ausgewählten Columns festgelegt werden. In dieser Sortierfolge werden die Rows zur Verfügung gestellt. Mehrere Column Namen werden durch einen Schrägstrich voneinander getrennt definiert.

```
ORDBY=COLUMN1/COLUMN10
```

Abb. 215: Sortieren von Rows und selektierten Columns

Wenn nicht alle Namen auf demselben Statement Platz finden, ist es möglich, die Definition mittels Schrägstrich-Blank (in der laufenden Zeile) auf dem Folgestatement fortzusetzen.
Der nächste Column Name wird danach auf dem Folgestatement definiert. Beliebige führende Blankstellen sind möglich.

```
ORDBY=COLUMN1/
      COLUMN10
```

Abb. 216: Sortieren von Rows und selektierten Columns (Forts.)

Die Sortierung einer Column kann auch absteigend sein. Anschliessend an den Column Namen kann in Klammer explizit die Sortierart angegeben werden:

COLUMN1(**D**)/ bedeutet absteigend (descending),
COLUMN2(**A**)/ bedeutet aufsteigend (ascending).
Wird nichts definert, wird (A) angenommen.

```
ORDBY=COLUMN1/COLUMN10(D)/COLUMN20
```

Abb. 217: Sortieren von Columns aufwärts oder abwärts

ORDBY= kann nur bei IDB definiert werden.

Options Die nachfolgend aufgeführten Optionen sind möglich:

WP=*nnnnn* Workbereich-Position
Mit dieser Definition wird bestimmt, dass die Row in den allgemeinen Workbereich geschrieben bzw. aus demselben gelesen wird. Die Definition bezieht sich auf den Platz innerhalb des Workbereiches, der für den zu bearbeitenden Row Bereich belegt wird. Wird WP= definiert, existiert für die entsprechende Filedefinition kein dynamischer Recordbereich.

FCA= File Communication Area
Mit dieser Definition wird bestimmt, an welcher Stelle innerhalb des allgemeinen QPAC-Workbereiches sich die Informations-Austauscharea für die zugehörige Filedefinition befinden soll. Ist FCA= nicht definiert, wird sie dynamisch angelegt.
Mit der FCA sind ausserdem folgende implizite Symbole vordefiniert, die nach Bedarf verwendet werden können:

```
..SQLCODE,BL4      = Returncode
..ROWLEN,BL4       = Länge der gelesenen Row
..TBN,CL18         = Tabellenname
```

`RC=YES` Returncode
Wird diese Definition angegeben, erfolgt bei einem eventuellen Fehler seitens DB2 durch QPAC kein Abbruch, sondern der Original SQLCODE wird im FCA Feld `..SQLCODE` an das Programm übergeben.

```
-927
```

Abb. 218: SQL Returncode im FCA-Feld ..SQLCODE

Der Returncode im FCA Feld `..SQLCODE` ist 0, wenn keine Fehler aufgetreten sind. Es ist wichtig, dass applikatorisch der Returncode in der FCA abgefragt wird, wenn dieser Operand definiert wurde.

```
IF I1SQLCODE = -927 THEN
IF I2SQLCODE NOT = 0 THEN
```

Abb. 219: Abfrage des SQL Returncodes

`PRFX=YES` Symbol Prefix vorsehen.
Bei unterschiedlichen Tabellen mit gleichen Column Namen kann bei einer Filedefinition diese Option definiert werden. Dies veranlasst, dass allen Column Namen zur Feldsymbolbildung die Fileidentifikation (Kurzform) vorangestellt wird.
Dadurch wird eine 'Duplicate Symbol' Fehlersituation verhindert.

CAF Support an Stelle des TSO Batchprogrammes IKJEFT01

Der PLAN Name sowie die DB2 System Id können via `PARM=` Definition angegeben werden. Damit wird intern der **CAF Support** initialisiert an Stelle der Verwendung des TSO Batch Programmes IKJEFT01.

`PLAN=`*`planname`*
Planname, wenn QPAC nicht unter dem TSO Batchprogramm IKJEFT01 aufgerufen wird.

`DB2ID=`*`sysid`* DB2 System Id, wenn QPAC nicht unter dem TSO Batchprogramm IKJEFT01 aufgerufen wird.
Siehe auch unter DB2 Job Control Beispiel.

CAF Return und Reason Codes

CAF Return und Reason Codes sind im IBM Manual *Application Programming and SQL Guide* unter "CAF Return Codes and Reason Codes" zu finden.

Verarbeitung von DB2 Datenbanken

Die Verarbeitung durch QPAC erfolgt in **z/OS** auf der Basis des DBRM Modules QPACBDB2 und eines DB2 Applikations-Planes. Darin muss das QPAC Interface QPACBDB2 als Programm definiert werden.
Danach können die Tabellen verarbeitet werden.

QPAC arbeitet auf der Basis von **Dynamic SQL**. Es orientiert sich aber weitestgehend selbst.

Es können in z/OS bis zu 39 DB2 Filedefinitionen angegeben werden. Sie müssen jedoch die Fileidentifikationen 1-39 oder das implizite Format verwenden.
Bitte beachten: Die Identifikationsnummern 1-9 sind **ohne "WITH HOLD"** declared und die implizite Form sowie die Identifikationsnummern 10-39 sind **mit "WITH HOLD"** declared.

```
IDB=     ODB=     UDB=
IDB1=    ODB8=    UDB3=
```

Abb. 220: DB2 Filedefinitionen

QPAC definiert und ordnet intern automatisch die Original Column Namen als QPAC-Feldnamen zu, mit den zugehörigen Längen und Format Attributen. Diese Feldnamen werden im I/O Bereich in der Reihenfolge zugeordnet, in der die einzelnen Columns in der Row definiert wurden.
Variable Felder (VARCHAR, LONG VARCHAR) werden im I/O Bereich mit einem **führenden 2 Byte Binärfeld** angelegt. Der Column Name zeigt jedoch auf den Anfang des Feldes.
Auf das vorangestellte 2 Byte Längenfeld wird ein eigener Name zugeordnet, bestehend aus dem Column Namen, dem ein **Nummernzeichen (#)** angehängt wurde.

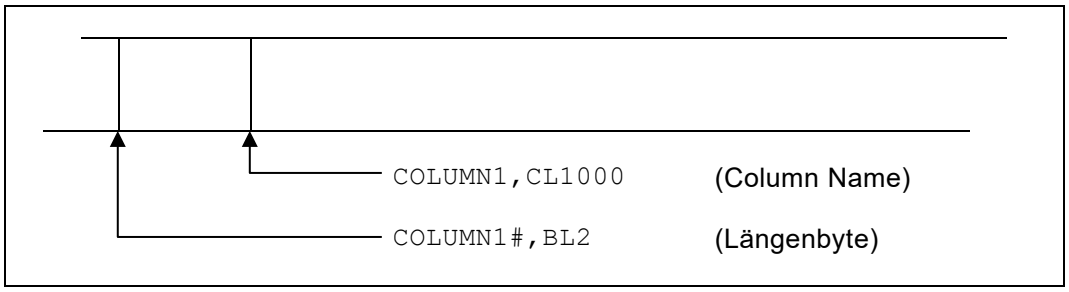


Abb. 221: Variable Felder mit 2 Byte Längenfeld

Diese automatisch zugeordneten Feldnamen können für die Verarbeitung verwendet werden.
Die Zuordnungen sind in der XREF Crossreference Liste ersichtlich.

Columns, die das Attribute NULL YES besitzen, d.h. die NULL sein können, bekommen noch eine zusätzliche Feldzuordnung:
Ein dynamisch zugeordnetes 2 Byte Binärfeld, dem der Column Name mit angehängtem **Alpha Zeichen (@)** als Feldname zugeordnet wird, kann zur Steuerung des NULL Zustandes verwendet werden. Ist nach einer Leseoperation dieses Indikatorfeld auf -1 gesetzt, gilt der Column Inhalt als NULL. Wird vor einer Ausgabeoperation dieses indikatorfeld auf -1 gesetzt, wird der Columninhalt als NULL ausgegeben.

```
SET COLUMN10@ = -1
IF COLUMN10@ = -1
   THEN                              (der Column-Inhalt ist NULL)
```

Abb. 222: Steuerung des NULL Zustandes

Ist ein Indikatorfeld (@) einer Column nach dem Lesen auf -1 gesetzt (NULL) und wird das Column Feld mit einem Wert gefüllt, muss das Indikatorfeld auf 0 gesetzt werden um den NOT NULL Status zu signalisieren, andernfalls würde der NULL Zustand beim `REWRITE` erhalten bleiben.

```
SET COLUMN10@ = 0
```

Abb. 223: Aufheben des NULL Zustandes

Gleitkomma Zahlen (Floating Point)

Columns mit dem Attribut FLOAT (floating point, short oder long precision) können von QPAC verarbeitet werden.

a) Ein Feld mit Attribut FLOAT wird mit einem `SET` Befehl automatisch ins Displayformat konvertiert und in ein Characterfeld gestellt.

```
SET WPOS7000,CL20 = COLUMN_FLOAT
```

Abb. 224: Verarbeitung von SQL Gleitkomma Feldern

Anschliessend steht linksbündig die Gleitkommazahl.

```
+0.5925E+02
```

Abb. 225: Darstellungsformat einer Gleitkomma Zahl

b) Soll ein Feld mit Attribut FLOAT mit dem `SET` Befehl gefüllt werden, ist der Sendeteil ein Characterfeld mit einer Gleitkommazahl als Inhalt, oder ein Characterliteral im Format einer Gleitkommazahl. Die Gleitkommazahl im Characterformat wird automatisch konvertiert und in das entsprechende Column-Feld gespeichert.

```
SET COLUMN_FLOAT = WPOS7000,CL20
SET COLUMN_FLOAT = +0.5925E+02
SET COLUMN_FLOAT = '+0.5925E+02'
```

Abb. 226: Darstellung von Gleitkomma Zahlen

Weitergehende Gleitkomma-Arithmetik ist in QPAC nicht unterstützt.

Wird die gleiche Tabelle in mehr als einer Filedefinition angegeben, entsteht durch die automatische Feldsymbolzuordung der Column Namen eine 'Duplicate Symbol' Situation. Diese entsteht auch, wenn in unterschiedlichen Tabellen gleiche Column Namen vorhanden sind.
Diese Situation kann durch den Operanden `PRFX=YES` in der Filedefinition vermieden werden.
Durch diese Definition wird jedem Column Namen, der als Feldsymbol zugeordnet wird, die Fileidentifikation (Kurzform) vorangestellt. Dadurch unterscheiden sich die Symbolnamen eindeutig.

```
IDB1=TBN=TABLE1,PRFX=YES
IDB2=TBN=TABLE1,PRFX=YES

IF I1COLUMN1 = ...
IF I2COLUMN1 = ...

  oder

IDB1=TBN=TABLEX
IDB2=TBN=TABLEX,PRFX=YES

IF COLUMNX = ...
IF I2COLUMNX = ...
```

Abb. 227: PRFX=YES verhindert "Duplicate Symbol" Situationen

☞ Diese Prefixzuordnung gilt nur für die Feldsymbole, nicht aber für anderweitige Column Definitionen, wie beispielsweise: `SCOL=`, `ORDBY=`, bzw. `WHERE-`*`id`* Conditions etc.

Die FCA dient dazu, zwischen dem Verarbeitungsprogramm und DB2 eventuell notwendige Informationen auszutauschen.
Es handelt sich dabei im Wesentlichen um Return Codes.

Die FCA für DB2 besitzt das folgende Format:

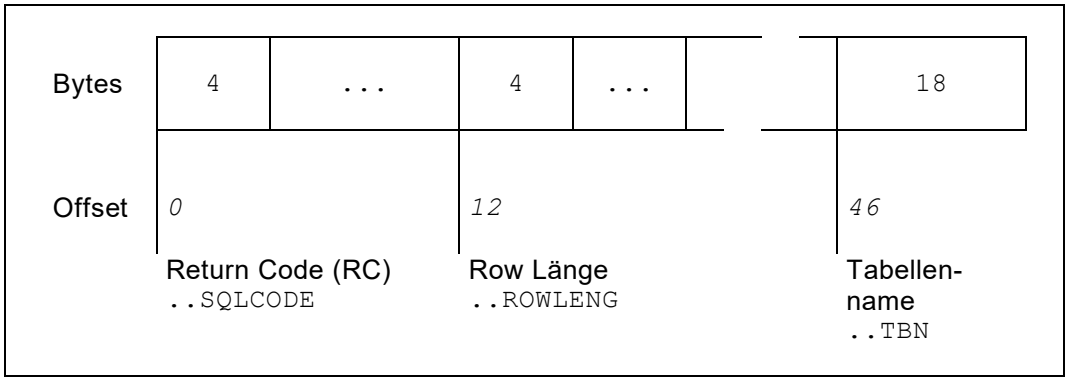


Abb. 228: FCA für DB2

RC	DB2 Return Code in binärem Format als negativer Wert. z.B. `-927` Dieser Return Code wird nur in die FCA gestellt, wenn bei der Filedefinition der Parameter `RC=YES` definiert wurde. Andernfalls erfolgt bei einem SQL Fehler automatisch ein Verarbeitungsabbruch. Dies dürfte der Normalfall sein. Diesem Feld ist das Symbol `..SQLCODE,BL4` zugeordnet.
Row Länge	Ist die Länge der ganzen Row (alle Columns aneinander in der I/O Area), inkl. der eventuellen Längenbytes bei VARCHAR Columns. Diesem Feld ist das Symbol `..ROWLENG,BL4` zugeordnet.
Tabellenname	Ist der Tabellenname der zu bearbeitenden Tabelle (siehe auch `TBN=`). Diesem Feld ist das Symbol `..TBN,CL18` zugeordnet.

Hinweise zur Verarbeitungslogik

Grundsätzlich kann eine Tabelle (bzw. einzelne Rows daraus) gelesen (`IDB` mit dem Befehl `GET`), modifiziert (`UDB` mit der Befehlsfolge `GET - PUT`) bzw. erstellt werden (`ODB` mit dem Befehl `PUT`).

Hinzufügen (Additions) von Rows zu einer bestehenden Tabelle bzw. **Löschen** (Deletions) von Rows aus einer Tabelle sind im Updatemodus (`UDB`) unterstützt.

Die Verarbeitung erfolgt sequenziell row-weise und zwar in der Reihenfolge, wie sie DB2 liefert, sofern die ORDER BY Klausel (`ORDBY=`) nicht definiert wurde.

Die einzelnen Columns einer Row werden aneinandergereiht in der I/O Area zur Verfügung gestellt. Die Felder können via die Column-Namen angesprochen werden. Es ist auch möglich (aber nicht unbedingt zu empfehlen) die einzelnen Felder direkt zu adressieren:

```
I1POS1,CL80
```

Abb. 229: Nicht zu empfehlen: Direktadressierung von Columns

Werden nur `GET` verwendet (`IDB` als Definition), erfolgt die Verarbeitung von Anfang bis Ende der Tabelle. Intern entspricht dies der folgenden Funktionsfolge:

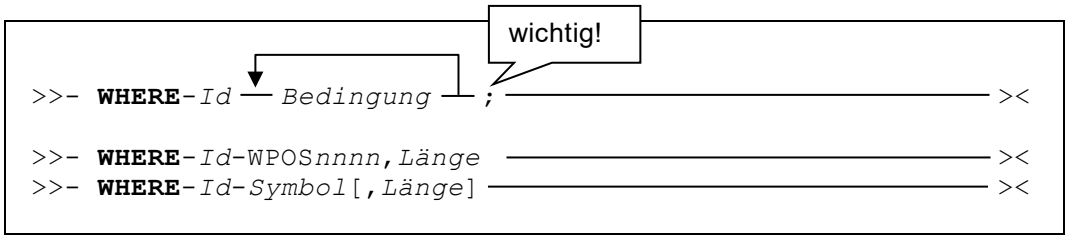
DECLARE .....,
OPEN Tabelle ..,
FETCH ...,
CLOSE Tabelle.

Wird anstelle von `IDB` (Input DB2 Data Base Table) `UDB` definiert (Update Data Base Table), entspricht die Befehlsfolge `GET PUT` der folgenden Funktionsfolge:

DECLARE ... FOR UPDATE OF ...,
OPEN ....,
FETCH ....,
UPDATE .... WHERE CURRENT OF .....,
CLOSE ...

Die WHERE Instruktion

Eine Tabelle kann mit dem Befehl `WHERE` für eine **selektive Verarbeitung** vorbereitet werden. Er entspricht der Bedeutung der WHERE Klausel in einem SQL SELECT Statement.
Die `WHERE` Anweisung besitzt folgende zwei Formate:



Die Bedingungen müssen genau das Format der WHERE Klausel im SQL SELECT Satz besitzen. Das **Ende aller Bedingungsdefinitionen** wird mit einem **Semikolon (;)** angegeben. Die Bedingungsdefinitionen werden intern in der definierten Form übernommen und **erst durch DB2 syntaktisch analysiert**. Eine Ausnahme bilden mögliche Host-Variablen, die in einer Bedingung vorhanden sein können.
Eine **Host-Variable** wird gemäss SQL Konvention mit einem vorangestellten **Doppelpunkt (:)** gekennzeichnet. Vorhandene Host-Variablen werden zum Zeitpunkt der Befehlsausführung inhaltlich aufgelöst. Die Host-Variable selbst muss zum Zeitpunkt der Programm-Umsetzung (Syntaxanalyse) ein definiertes Feld sein, andernfalls erfolgt eine Fehlerausgabe 108 (Undefined Symbol).

Beim zweiten Format werden die Bedingungen dynamisch zur Ausführungszeit ins definierte Feld resp. in den definierten Bereich gestellt. Diese Bedingungen dürfen **keine Hostvariablen** mehr enthalten. Sie können zu diesem Zeitpunkt keine Syntaxanalyse mehr durchlaufen. Sie werden unverändert dem DB2 übergeben. Auch das abschliessende Semikolon (;) muss fehlen.

Der `WHERE` Befehl ist ein vorbereitender Befehl. Sind mehrere Rows vorhanden, die die Selektionskriterien erfüllen, werden sie mit den nachfolgenden `GET` einzeln zur Verfügung gestellt.

```
WHERE-I1 COLUMN10 = 'name' AND
         COLUMN5  > :X28 ;

NORMAL
GET-I1   (nur Rows, deren Columns die obigen
         Bedingungen erfüllen, erscheinen hier)
...
```

Abb. 230: Anwendung der WHERE Anweisung (Beispiel 1)

```
WHERE-I1 COLUMN10 = 'name' AND
         COLUMN5  > :X28 ;

DO-FOREVER
   GET-I1   AT-EOF DOQUIT ATEND
            (nur Rows, deren Columns die obigen
            Bedingungen erfüllen, erscheinen hier)
   ...
DOEND
```

Abb. 231: Anwendung der WHERE Anweisung (Beispiel 2)

```
0S=WHEREAREA,CL100
SET WHEREAREA = 'COLUMN10 = "name" AND '/
                'COLUMN5 > 116         '
...
WHERE-I1-WHEREAREA
```

Abb. 232: Anwendung der WHERE Anweisung (Beispiel 3)

Der `WHERE` Befehl beinhaltet einen `OPEN`, sofern der Filezustand "closed" ist.

Die FETCH Instruktion

An Stelle der `GET` Instruktion kann auch die `FETCH` Instruktion verwendet werden. Der Vorteil liegt darin, dass bei einer EOF-Situation kein automatischer Close erfolgt und kein GET-Block übersprungen wird.

```
>>- FETCH-Id ──────────────────────────────────────────────────────────><
```

Die PUTA Instruktion

Eine **komplette Row** kann mit `PUTA` (Put Addition) in eine DB2 Tabelle **eingefügt** werden. Die einzelnen Columns müssen vorher in der I/O Area mit Daten gefüllt werden, gemäss DB2 Konvention.
NULL Werte müssen mit dem Column-Namen@ und -1 als Inhalt bekannt gegeben werden.

Bei VARCHAR Feldern muss ausserdem die aktuelle Stringlänge im Feld Column-Name# (2 Byte binär) bekannt gegeben werden. In der I/O Area ist jedoch stets die maximale Feldgrösse zugeordnet. Die einem VARCHAR Feld folgende Column ist folglich immer anschliessend der maximal möglichen Länge plaziert.

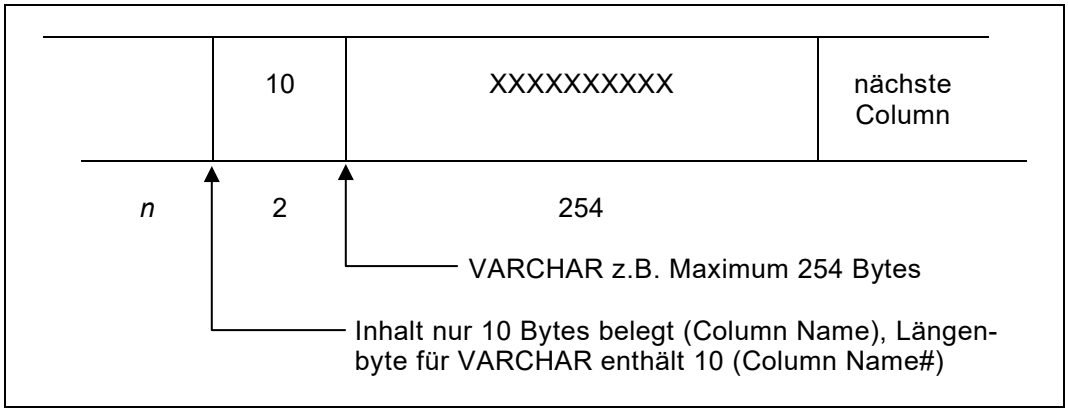


Abb. 233: Angabe der Stringlänge bei VARCHAR Feldern

`PUTA` ist nur im Updatemodus (`UDB=`) möglich.

Die PUTD Instruktion

Einzelne Rows können aus einer Tabelle **gelöscht** werden. Die Row, die gelöscht werden soll, muss vorausgehend mit `GET` gelesen worden sein, d.h. es wird durch den Befehl `PUTD` die jeweils zuletzt gelesene Row gelöscht.

`PUTD` ist nur im Updatemodus (`UDB=`) möglich.

Die ODB= Definition (Initial Load)

Bestehende Rows in einer Tabelle werden zur Openzeit gelöscht.
Der Befehl `PUT` fügt in diesem Falle neue Rows in die Tabelle ein, ohne dass die alten Rows erhalten bleiben. Die einzelnen Columns in der Row müssen, wie unter `PUTA` beschrieben, vorausgehend mit Daten gefüllt werden.
Zu beachten sind dabei die Längenbytes bei VARCHAR Columns bzw. die Indikatorfelder (@), wenn NULL Werte gesetzt werden sollten.

Hinweise zu Job Control und Durchführung

Unter **z/OS** wird QPAC, welches DB2 Tabellen verarbeitet, mit dem **TSO Batch Programm IKJEFT01** aufgerufen und durchgeführt. Die Anwendung entspricht der offiziellen Form für Batch Programme.

Nachfolgend ist ein Job Control Beispiel aufgeführt:

```
//         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
```

Abb. 234: DB2 Job Control (z/OS)

Es kann auch auf das TSO Batch Programm IKJEFT01 verzichtet werden.
In diesem Falle muss der Plan und die System Id über `PARM` definiert werden.

```
//         EXEC PGM=QPAC
//QPACLIST DD SYSOUT=*
//QPACIN   DD *
PARM=PLAN=qpacplan,DB2ID=db2id
IDB=TBN=...
```

Abb. 235: DB2 Aufruf ohne IKJEFT01

Erweiterte SQL Command Functions

Nachfolgende Befehle beziehen sich auf erweiterte Möglichkeiten im Zusammenhang mit der Bearbeitung von DB2 Tabellen. Der hohe Grad an Flexibilität dieser Command-Gruppe entspricht fast dem Umfang der offiziellen IBM SPUFI Funktionen.
Jeder Command dieser erweiterten Gruppe wird durch das Schlüsselwort **EXECSQL** eingeleitet und durch ein **Semikolon (;)** abgeschlossen. Die dazwischenliegenden Definitionen werden grösstenteils unverändert dem DB2 übergeben. Einzig die Beziehung zu den DB-Tabellen Definitionen und eventuelle Hostvariablen, wo letztere erlaubt sind, werden vorher aufgelöst.

QPAC intern basieren alle diese EXECSQL Commands auf **Dynamic SQL**. Dynamic SQL auferlegt jedoch gewisse Einschränkungen, die von QPAC nicht umgangen werden können. Diese Einschränkungen sind aber minimal und normalerweise von Seite der Anwenderbedürfnisse nicht ermangelt.

Die Kommunikation zwischen QPAC und DB2 im Zusammenhang mit diesen EXECSQL Commands erfolgt über neue reservierte Feldnamen. Dabei sind unter anderem offizielle Bezeichnungen der **SQLCA Struktur** implementiert worden, wie z.B. SQLCODE, SQLSTATE etc.
Die Inhalte dieser SQLCA Felder werden nach jedem EXECSQL Command verfügbar gemacht und von einem nachfolgenden SQL Command jeweils wieder überschrieben.
Die zusätzlichen reservierten Feldnamen sind weiter unten aufgeführt.

Ergebnisse bzw.Column-Inhalte stehen nach einem EXECSQL Command in den jeweiligen Column Feldern der zugehörigen DB-Tabellen Definition. Handelt es sich jedoch um ein sogenanntes "Expression" Ergebnis, steht dieses Ergebnis in einem reservierten Feldnamen, der sich nach der Art, d.h. dem Format des Ergebnisses bildet. Ein solches Feld beginnt immer mit dem Stamm **RESULTxn**, gefolgt vom Format (**B**=Binary, **C**=Character, **V**=VarCharacter, **P**=Packed) und der Position als Nummer, wo die korrespondierende **Column-Position** innerhalb des Command Strings steht.

z.B.
```
EXECSQL SELECT COUNT(*), MAX (SALARY), MIN(SALARY)
  FROM IDB2 ;
```

ergibt RESULTB1, da COUNT im SELECT Statement an Position 1 steht und das Ergebnis ein Binärwert ist, RESULTP2, da MAX (SALARY) an Position 2 steht und das Ergebnis eine gepackte Zahl ist und RESULTP3, da MIN(SALARY) an Position 3 steht und ebenfalls eine gepackte Zahl ist.

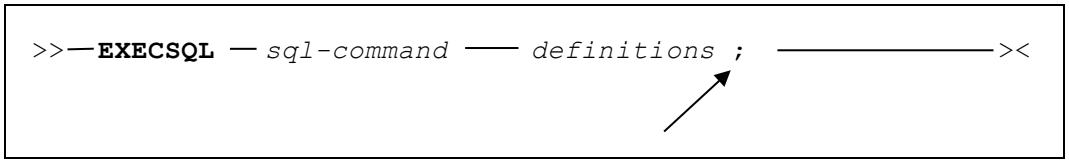
Reservierte Feldnamen (nur bei EXECSQL gültig)

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

Die EXECSQL Single Instruktion

Dem Schlüsselwort `EXECSQL` folgt der eigentliche SQL Command, der seinerseits mit einem **Semikolon (;)** abgeschlossen wird.
Innerhalb des SQL Commands werden eventuelle Tabellendefinitionen nicht mit dem originalen Tabellennamen angegeben, sondern mit der Kurz-Id einer vorausgehenden QPAC DB-Tabellendefinition.
Jede Tabelle, mit Ausnahme von Tabellen in SubSelects, die bearbeitet wird, muss als QPAC DB-Tabellendefinition vorausgehend definiert sein. Durch diese Definition werden die Column-Namen als Felder angelegt, die dann auch nach Bedarf abgefüllt werden. Dabei werden eventuelle Präfixe berücksichtigt.

Column-Namen innerhalb der Command-Definition dürfen dagegen keine Präfixe aufweisen, da sie direkt dem DB2 übergeben werden. Es gelten diesbezüglich die gleichen Regeln wie sie schon bei der `WHERE-Id` Instruktion beschrieben worden sind.

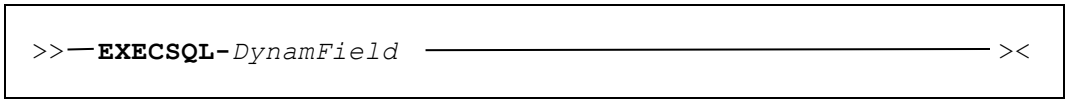


Es werden sämtliche SQL Commands unterstützt, welche dynamisch aufbereitet (prepared) werden können. Eine vollständige Liste aller unterstützten SQL Commands befindet sich im Appendix C des SQL Reference Manuals.

Die *definitions* müssen genau dem Format des offiziellen SQL Commands INSERT, UPDATE, DELETE, SELECT etc. entsprechen, mit Ausnahme der angesprochenen Tabellen Definitionen.
Die Tabellen Definitionen werden nur mit der QPAC DBId (z.B. `IDB2` oder `I2`) angegeben, die dann zugleich die Verbindung zur vorausgehenden DB-Tabellendefinition herstellt und von QPAC zur Execution-Time durch den originalen Tabellen-Namen ersetzt wird.
Hostvariablen werden nach SQL-Konvention mit einem vorangestellten **Doppelpunkt (:)** gekennzeichnet. Die Hostvariable selbst muss zum Zeitpunkt der Programm-Umsetzung ein definiertes Feld sein, andernfalls erfolgt eine Fehlermeldung `108` (undefined symbol). Details hierzu sind in den nachfolgenden Beispielen aufgeführt.

Der `EXECSQL SELECT` Command im obigen Format entspricht einem sogenannten **Single-SELECT**, d.h. es wird nur eine Row zur Verfügung gestellt. Wenn mehrere Rows gleicher Conditions vorhanden wären wird neben der ersten Row zugleich der SQLCODE –811 zurückgegeben.

Die `EXECSQL` Instruktion kann auch im dynamischen Format angewendet werden. In diesem Falle wird der eigentliche SQL Command SELECT, INSERT, DELETE bzw. UPDATE in einem vordefinierten Bereich zur Verfügung gestellt. Der Feldname dieses Bereiches wird dann an den `EXECSQL` Command mit Bindestrich angehängt.



Beim dynamischen Format dürfen keine Hostvariablen mit Doppelpunkt **(:)** vorhanden sein, da diese zur Ausführungszeit nicht mehr aufgelöst werden können. Ebenfalls ist das Semicolon (;) nicht zu definieren. Dieses ist Bestandteil des im dynamischen Feld vorhandenen Commands.

Beispiele Single Instruktion

```
* 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
```

Abb. 236: EXECSQL - Beispiele DB Tabellen Definitionen

```
* 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
```

Abb. 237: EXECSQL - Beispiele DB Tabellen Definitionen (Forts.)

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
```

Abb. 238: EXECSQL - Beispiel 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
```

Abb. 239: EXECSQL - Beispiel 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
```

Abb. 240: EXECSQL - Beispiel 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
```

Abb. 241: EXECSQL - Beispiel 1 Single SELECT static format

Die erste Row, die der WHERE Klausel entspricht wird in die Column Felder der Tabelle `IDB1` abgefüllt.
Der **Returncode –811** signalisiert, dass weitere Rows die Condition erfüllen würden.

```
* 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
```

Abb. 242: EXECSQL - Beispiel 2 Single SELECT static format

Die Anzahl der Rows in der Tabelle `IDB2` wird in das reservierte Feld **`RESULTB1`** abgefüllt, das das Format 4 Bytes **Binary** und die **Positionsnummer 1** besitzt, da nach dem SELECT Command die Funktion COUNT an Position 1 steht.

```
* 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
```

Abb. 243: EXECSQL - Beispiel 3 Single SELECT static format

Die Ergebnisse dieser Funktionen sind 16 Bytes packed. SUM steht an Position 1, MAX an Position 2 und MIN an Position 3. Das Feld mit dem angehängten Alphazeichen (@) signalisiert ein eventueller NULL Wert.

```
* 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
```

Abb. 244: EXECSQL - Beispiel 4 Single SELECT static format

Das Feld `I2LASTNAME` der Tabellendefinition `IDB2` wird abgefüllt.

```
* 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
```

Abb. 245: EXECSQL - Beispiel 5 Single SELECT static format

In obigem Beispiel werden Correlation Namen verwendet. Solche Namen dürfen keine DB-Ids sein (z.B. `IDB1`) !
Correlation Namen müssen bei allen Column Namen definiert werden, wenn die zugehörige DB-Id Definition einen Correlation Namen besitzt.
Die Column Namen in der `SELECT` Definition dürfen die QPAC Präfixe nicht enthalten. Im `SELECT` werden nur eventuelle DB-Ids und Hostvariablen mit Doppelpunkt (:) aufgelöst bevor der String sonst unverändert dem DB2-SQL übergeben wird.

```
* 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
```

Abb. 246: EXECSQL - Beispiel 6 Single SELECT static format

Ergebnisse sind in den reservierten Feldern mit dem Format C (CHARACTER) und den Positionsnummern 1-3. Diese Character Felder sind 256 Bytes gross. Siehe diesbezüglich bei den reservierten Feldern.

Dynamic Format:

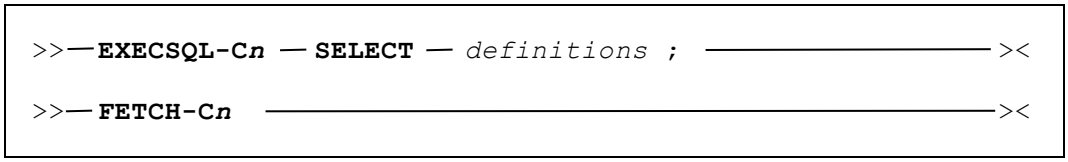
```
* Example DELETE from DB2 table IDB1
* ----------------------------------

1W=DYNAMAREA,CL800

SET DYNAMAREA = 'DELETE FROM I1     '/
'WHERE COLUMN_CHA = '/
''CHARACT400'' ; '

EXECSQL-DYNAMAREA

IF SQLCODE < 0 THEN .. .. .. IFEND
```

Abb. 247: EXECSQL - Beispiel Dynamic Format

Die EXECSQL Cursor Instruktion für SELECT Commands

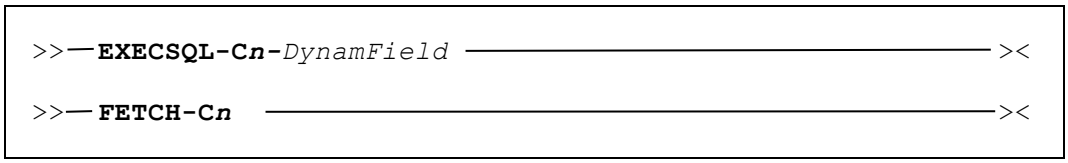
Neben der EXECSQL Instruktion für einzelne SQL Commands gibt es noch die Variante, bei der der EXECSQL Command mit einem Cursor verknüpft wird und dadurch die Möglichkeit bietet, mehrere Rows sequenziell durchzuarbeiten.

Hierfür gibt es den zusätzlichen Command FETCH, der eine Row nach der anderen in die Column Felder abfüllt. Bezüglich der SELECT Definitionen gelten die gleichen Regeln, wie sie vorausgehend bei der EXECSQL Single Instruktion beschrieben worden sind.



Dem EXECSQL Command wird mit Bindestrich eine Identifikation (C*n*) angehängt. Diese kann C1 bis C19 sein. Es handelt sich dabei, vergleichbar mit den File-Identifikationen, um eine Referenz-Id, die einerseits den Hinweis gibt, dass es sich nicht um einen Single SELECT handelt und andererseits auch bei einem nachfolgenden korrespondierenden FETCH-C*n* Command angegeben werden muss. Es handelt sich zugleich um den Cursornamen, der intern zugeordnet wird.
Bitte beachten: Die Cursor C1-C9 sind intern **ohne "WITH HOLD"** declared und die Cursor C10-C19 sind intern **mit "WITH HOLD"** declared.

Die EXECSQL Instruktion mit Cursor kann ebenfalls im dynamischen Format angewendet werden. In diesem Falle wird der Feldname des Bereiches wiederum mit Bindestrich an die Id angehängt. Der vollständige SELECT Command wird dann vor Aufruf des EXECSQL in diesen Bereich gestellt, mit der bereits vorausgehend erwähnten Einschränkung der Hostvariablen mit Doppelpunkt (:).



Beispiele SELECT Cursor Instruktion

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
```

Abb. 248: EXECSQL - Beispiel 1 Static Format mit 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
```

Abb. 249: EXECSQL - Beispiel 2 Static Format mit Cursor

In diesem Beispiel ist ein Subselect aufgeführt. An dritter Position ist eine Textkonstante definiert. Dieser Text bzw. `BB.PROJNO` steht nach Ausführung im Ergebnisfeld `RESULTV3`. Dieses Feld besitzt das Attribute `V` (VARCHAR) und ist 254 Bytes gross. Die effektive Textlänge steht im zugehörigen Feld `RESULTV3#`.

Dynamic Format mit 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
```

Abb. 250: EXECSQL - Beispiel Dynamic Format mit Cursor

Die Definition in der `DYNAMAREA` darf keine Hostvariablen mit Doppelpunkt (:) enthalten. Jedoch können in der `SET` Anweisung Characterfelder konkateniert werden, deren Inhalt dann zur Ausführungszeit übernommen wird.
Der Inhalt der `DYNAMAREA` wird, nachdem die DB-Id `IDB1` aufgelöst worden ist, unverändert dem DB2-SQL übergeben.

Beispiele weiterer 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;
```

Abb. 251: EXECSQL – Beispiele weiterer SQL Commands

Die gleichen Definitionen sind auch im Dynamic Format möglich:

```
0S=DYNAMICAREA,CL80

SET DYNAMICAREA = 'GRANT BIND ON PLAN TESTQPAC TO PUBLIC'
EXECSQL-DYNAMICAREA
```

Abb. 252: EXECSQL – Beispiele SQL Commands im Dynamic Format

Auto Commit

Das reservierte Feld `DB2COMMIT` ist ein 8 Bytes grosses Counter Feld. Es kann dazu dienen, nach einer gewünschten Anzahl von INSERTs und UPDATEs durch `EXECSQL` und `PUTA-U`*`n`* bzw. `PUTD-U`*`n`*, `PUT-O`*`n`* bei den Basis Commands automatisch einen COMMIT auszulösen.
Diese gewünschte Anzahl kann im Feld `DB2COMMIT` festgelegt werden. Nach Erreichen dieses Wertes und somit nach einem COMMIT wird erneut mit dem Zählen begonnen.

```
SET DB2COMMIT = 10

SET DB2COMMIT = 0
```

Abb. 253: DB2 Auto Commit

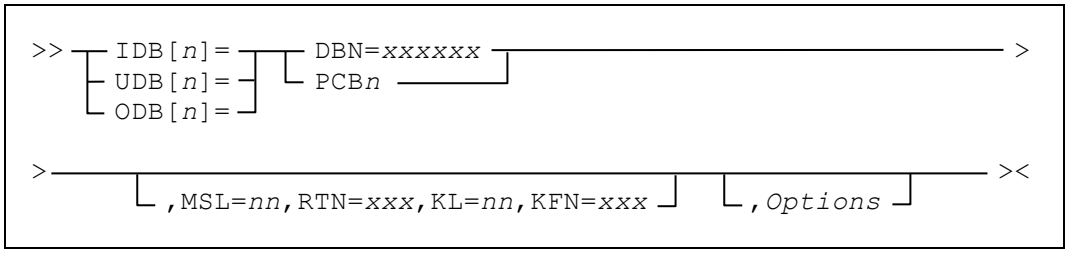
Wird der Inhalt von `DB2COMMIT` auf 0 (Null) gesetzt, wird die Auto Commit Funktion ausgeschaltet. Default ist 0.

Kapitel 13. DL/I Support Feature

DL/I Datenbank Definition

IBM DL/I ist durch dieses Feature voll unterstützt. Es basiert auf der Voraussetzung, dass DL/I installiert ist und dass bestimmte DL/I Basisroutinen im System vorhanden sind.

Grundformat der DL/I DB Definition



DBN= PCB*n*	Ist der Datenbankname. Es wird vorausgesetzt, dass der zugehörige PSB eine DB-Definition unter diesem Namen enthält. Anstelle des Datenbanknamen kann aber auch die PCB-Nummer innerhalb des PSBs definiert werden.
MSL=	Ist die maximale Segmentlänge. Fehlt diese Angabe, wird die Segmentlänge für den I/O-Buffer aus den internen DL/I Blöcken geholt, **sofern möglich**. Diese Definition ist notwendig, wenn die internen DL/I Blöcke nicht verfügbar sind. Das ist der Fall, wenn sich DL/I in einem anderen Adressraum befindet (QPACBMP, DBCTL unter z/OS).
RTN=	Rootsegment Name
KFN=	Keyfeldname des Rootsegmentes
KL=	Keylänge des Rootsegmentes Diese Angaben sind immer dann notwendig, wenn sich das DL/I Database Management in einem anderen Adressraum befindet, als das QPAC Programm selbst, und wenn gleichzeitig die Befehle SETGK undSETEK benützt werden. QPACBMP bzw. DBCTL verlangen diese Definition immer. Ansonsten werden bei QPACDLI diese Informationen automatisch aus den internen Blöcken geholt.
Options	Die nachfolgend aufgeführten Optionen stehen zur Verfügung:
CLR= CLE=	Diese Options entsprechen den Angaben, die im Kapitel 2: Input/Output Definitionen beschrieben sind. CLR= bezieht sich auf die Outputarea, CLE= bezieht sich auf die Inputarea. Es gelten bei DL/I bezüglich der Basislöschungen der I/O-Areas die gleichen Regeln wie bei den übrigen Fileorganisationen: die Outputarea wird auf X'00', die Inputarea auf X'FF' gesetzt, wenn keine Options aufgeführt werden.

FCA=

File Communication Area
Mit dieser Definition wird bestimmt, an welcher Stelle innerhalb des allgemeinen QPAC-Workbereiches sich die Informations-Austauscharea für die zugehörige Filedefinition befinden soll. Die FCA wird standardmässig dynamisch angelegt.

SSEG=

Selected Segments
Wenn nicht alle in der DBD vorhandenen Segmente verfügbar sein sollen, können mit dieser Definition die gewünschten Segmentnamen angegeben werden. Mehrere Segmentnamen werden durch / voneinander getrennt.

```
SSEG=NAMEA/NAMEB/NAMEX
```

Abb. 254: Angabe von mehreren Segmentnamen

Eine Fortsetzung auf dem Folgestatement, wenn mehrere Segmentnamen definiert werden müssen, wird erreicht, wenn mit dem / auf dem ersten Statement aufgehört wird und der nächste Name auf dem folgenden Statement steht.

```
SSEG=NAMEA/NAMEB/
  NAMEX
```

Abb. 255: Angabe von Segmentnamen auf dem Folgestatement

SSEG= ist nur bei Input bzw. Update zulässig (IDB=, UDB=).

WP=*nnnnn*

Workbereich Position
Mit dieser Definition wird bestimmt, dass das Segment in den allgemeinen Workbereich geschrieben bzw. aus demselben gelesen werden soll. Die Definition bezieht sich auf den Platz innerhalb des Workbereiches, der für das zu bearbeitende Segment belegt wird. Wird WP= definiert, existiert für die entsprechende Filedefinition kein dynamischer Recordbereich.

NOGE

Diese Definition unterdrückt den Return Code GE, der in Zusammenhang mit SETGK oder SETEK auftreten kann. Siehe diesbezüglich auch unter 'Hinweise zur Verarbeitungslogik', Punkt 4 und 5. Es ist dabei zu beachten, dass in diesem Falle anstelle des Return Codes das nächstliegende Segment (Rootsegment) übergeben wird, gleichzeitig mit der internen Umschaltung von 'Record orientiert' auf 'DB orientiert'.

NOII

Diese Definition unterdrückt den Return Code II, der bei PUTA auftreten kann, wenn das Segment mit gleichem Key schon vorhanden ist. Wird NOII definiert, erfolgt Abbruch, wenn das Segment nicht eingefügt werden kann. Wird NOII nicht definiert, steht nach PUTA in der FCA der Return Code II, falls das Segment schon vorhanden ist.

RC=YES

Return Code.
Der DL/I Return Code wird in das FCA-Feld ..RC zurückgegeben, ohne dass im Fehlerfalle die QPAC-Verarbeitung abbricht.
Dieser Return Code ist blank, falls kein DL/I Return Code aufgetreten ist.

STAT	Data Base Statistik Beim Verarbeitungsende (close time) wird eine Data Base Statistik ausgegeben.

Die Verarbeitung durch QPAC erfolgt mit dem GET-Befehl sequenziell, auf der Basis der logischen DBs, die durch PSB-Definitionen beschrieben sind. Dabei wird Segment um Segment verfügbar gemacht, in Abhängigkeit etwelcher Sensitivität.

Es sind alle üblichen DB-Organisationen unterstützt. In QPAC sind keine Komponenten enthalten, die den offiziellen DB- und PSB-Definitionen widersprechen würden. QPAC orientiert sich ausschliesslich am definierten PSB.

Die FCA dient dazu, zwischen der Verarbeitung und DL/I eventuell notwendige Informationen auszutauschen. Es handelt sich dabei im Wesentlichen um Return Codes, die eine sachlogische Bedeutung besitzen und denen notwendigerweise keine Fehlersituation zugrunde liegt. Es kann sich aber auch um Key-Werte handeln, die für das Aufsetzen mit der SETGK-Funktion bzw. SETEK an DL/I übergeben werden.

Die FCA für DL/I besitzt den folgenden Aufbau:

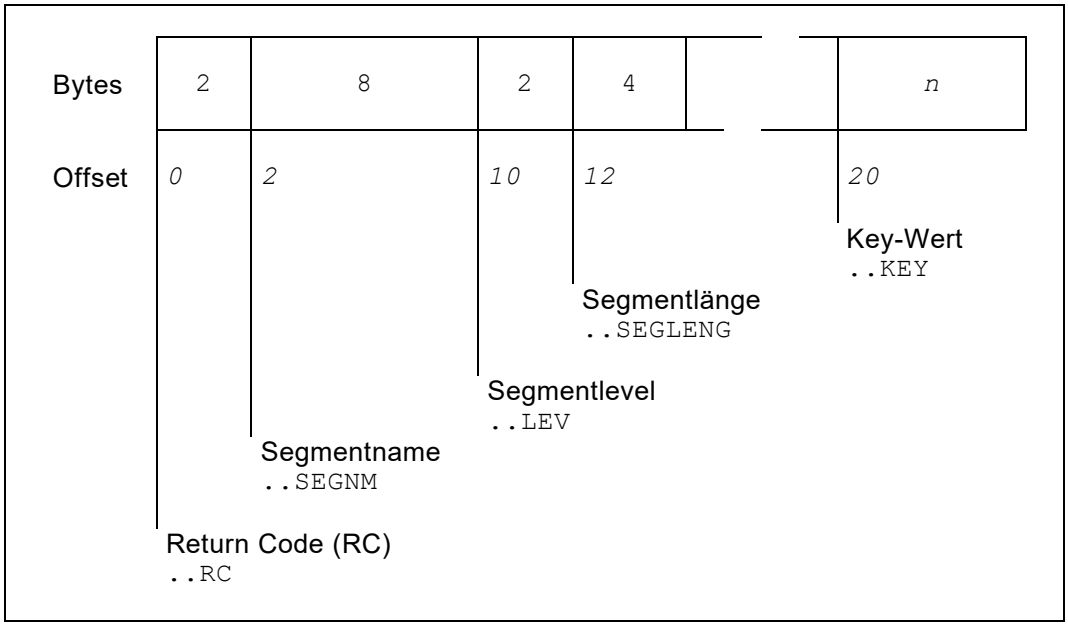


Abb. 256: FCA für DL/I

Return Code (RC)

Folgende Return Codes werden durch QPAC an die Verarbeitung übergeben, sofern RC=YES **nicht** definiert wurde:

bei Input/Update:

~~bb~~ = Normalsituation ohne Bedingungen
GA = Crossed Hierarchical Boundary
GE = Segment nicht gefunden
GK = Verschiedene Segmenttypen auf gleichem Level
II = Segment kann mit PUTA nicht eingefügt werden

bei Output:

~~bb~~ = Normalsituation ohne Bedingungen

	Return Codes, die nicht mit G oder Blank beginnen, führen immer zu Abbruch; bedingte Ausnahme ist II. Der Return Code GB (End of DB) führt zu EOF-Situation und wird nur übergeben, wenn `RC=YES` definiert worden ist.
Segmentname	Der Segmentname wird nach jeder Leseoperation (`GET`) zur Verfügung gestellt. Bei Output muss vor dem `PUT`-Befehl der Segmentname des zu schreibenden Segmentes in die FCA gestellt werden.
Segmentlevel	2 Bytes numerisch EBCDIC
Segmentlänge	in binärer Form, 1 Fullword, jeweils nach einem `GET`-Befehl. Sie wird nur zur Verfügung gestellt, wenn die internen DL/I Blöcke zugänglich sind, d.h. wenn das DL/I Management sich im gleichen Adressraum befindet.
Key-Wert	Schlüsselwert auf den mit Set Generic Key `(SETGK)` oder Set Equal Key `(SETEK)` aufgesetzt werden kann. Siehe Beschreibung unter `SETGK` bzw. `SETEK`.
Füllwert (`IDB` & `UDB`)	Bei `IDB` und `UDB`-Definition ist nach EOF der in maximaler Länge von 4096 Bytes ausgelegte Segmentbereich (Recordarea) jeweils mit X'FF' gefüllt. Wenn ein anderes Füllzeichen gewünscht wird, kann dies mit der Option `CLE=` in der Filedefinition explizit definiert werden.

```
IDB=DBN=EDB888,CLE=X'40'
```

Abb. 257: Option für spezielles Füllzeichen

Das DL/I Support Feature bedient sich der offiziellen Verarbeitungsroutinen des DL/I Systems. Sie müssen zu diesem Zwecke unter den offiziellen Namen in den DL/I- bzw. System-Libraries enthalten sein.

Bei DL/I-z/OS können keine für QPAC bestimmten EXEC-PARM Angaben definiert werden. Anstelle dessen muss das QPAC `PARM`-Steuerstatement verwendet werden. Dieses hat ein fixes Format: d.h. ab **Kolonne 1-5** muss `PARM=` stehen, und es muss **als erstes Statement** eingelesen werden:

```
//        EXEC    PGM=    (QPACDLI      or
//DD1     DD               QPACBMP)
//DD2     DD
...
//QPACIN DD *
PARM=LIST,NOLOG,LINECNT=66
IDB=DBN=DBTEST,MSL=150
OPF=PR
...
...
END
```

Abb. 258: DL/I Beispiel für z/OS

Die PARM Options entsprechen genau denen, die im Kapitel 1: Einführungsteil unter PARM Option beschrieben sind.

Der Job Control-mässige Aufruf von QPAC erfolgt standardmässig nach DL/I-Konvention. Dabei ist nicht zu unterscheiden, ob der PSB für PL/I oder COBOL/ASSEMBLER generiert wurde. In jedem Falle muss bei z/OS-Systemen als Hauptprogramm eines der folgenden Interfaces aufgerufen werden:

QPACDLI	für reine DL/I Batchverarbeitungen
QPACBMP	für DL/I Databases in separatem Adressraum (z.B. DBCTL)

(PSBTEST ist generiert für COBOL Programme)

```
//EXEC PGM=DFSRRC00,PARM='DLI,QPACDLI,PSBTEST,15'
//EXEC PGM=DFSRRC00,PARM='DLI,QPACBMP,PSBTEST,..'
```

Abb. 259: DL/I Aufruf

Ein definierter PSB kann mehrere PCBs enthalten. Aufgrund der DBN= Angabe in der DB-Filedefinition wird die gewünschte DB ausgewählt und nur diese gelangt in den Zugriff von QPAC.

Anstelle der DBN (Database Name) Definition, die aufgrund des Namens den PCB im PSB sucht, kann der PCB auch durch seine Platz-Nummer definiert werden. Dabei bedeutet PCB1 = 1. PCB im PSB, PCB4 = vierter PCB.

Einfache HISAM DBs können mit SETGK nur verarbeitet werden, wenn gleichzeitig die Option NOGE definiert wird. Diese Definition schliesst die Voraussetzung aus, dass eine Parentage gesetzt ist.

Die Segmentlänge kann nur bei der Verwendung von QPACDLI (d.h. bei reiner Batchverarbeitung) in der FCA zurückgegeben werden. Bei den Zugriffen auf Online-Datenbanken stehen diese Informationen nicht zur Verfügung.

Hinweise zur Verarbeitungslogik

Grundsätzlich kann eine DB gelesen (IDB), modifiziert (UDB) bzw. erstellt (ODB) werden.

Einfügungen (Additions) zu einer bestehenden DB bzw. **Löschungen** (Deletions) aus einer DB sind im Updatemode (UDB) unterstützt.

Die Verarbeitung erfolgt segmentweise und zwar in der Reihenfolge der hierarchischen Ordnung (top to bottom, left to right). Es werden nur sensitive Segmente zur Verfügung gestellt.
Es ist die Aufgabe des Anwenders, das jeweils zur Verfügung gestellte Segment zu identifizieren. Der Segmenttyp-Wechsel gleicher Stufe bzw. die hierarchische Rückstufung wird jeweils mit dem zugehörigen Return Code angezeigt. Ebenfalls wird in der FCA der jeweils zugehörige Segmentname übergeben.

Werden nur GETs verwendet, erfolgt die Verarbeitung von Anfang bis Ende der DB. Dies entspricht der Funktion GN (Go next). Ist anstelle IDB (Input DL/I Database) UDB definiert (Update DL/I Database), entspricht der QPAC GET der Funktion GHN und ein eventuell nachfolgender PUT der Funktion REPL. Ein verwendeter Befehl PUTA entspricht ISRT und PUTD der Funktion DLET.

Die SETGK Instruktion

Eine erweiterte Anwendungsmöglichkeit bietet die SETGK Operation. **Set Generic Key** ermöglicht das Aufsetzen auf einem bestimmten **Rootsegment**.

SETGK selbst stellt das Rootsegment nicht zur Verfügung, erst ein nachfolgender GET Befehl.

Bevor ein SETGK Befehl durchgeführt wird, muss in die FCA der Key-Wert gespeichert werden. Die Länge des Keys erkennt QPAC aus der DBD. Der nachfolgende GET basiert auf der Funktion GHU und stellt das Rootsegment mit gleichem oder nächst höherem Key zur Verfügung. Wird dagegen anstelle eines Segmentes das Ende der DB festgestellt, erfolgt der Return Code GE (No Segment found).

Dem ersten GET folgende GETs entsprechen darauf der Funktion GNP (GHNP) und zwar so lange bis das Ende des DB-Records erreicht ist. Darauf wird Return Code GE (No Segment found) übergeben.

```
SET WPOS9020,CL8 = 'KEY'
SETGK-I1  GET-I1
DO-UNTIL I1RC = 'GE'
...
...
GET-I1
```

Abb. 260: Beispiel SETGK für DL/I Zugriff

Es ist zu beachten, dass bei Verwendung der SETGK Instruktion die Verarbeitungssteuerung **DB recordorientiert** erfolgt, d.h. pro erreichtem DB Recordende wird der Return Code GE gesetzt und kein Segment übergeben. Bei Update würde in diesem Falle ein nachfolgender PUT (REPL) als fehlerhaft erscheinen (Return Code DJ). Wird keine SETGK Operation verwendet, erfolgt die Verarbeitung **DB orientiert**.
DB recordorientiert heisst: Mit SETGK wird auf ein Rootsegment aufgesetzt. Die nachfolgenden GET-Operationen basieren auf der DL/I Funktion GNP (Get next

within parent). Das Ende des DB-Records (vor nächstfolgendem Rootsegment) wird mit Return Code GE signalisiert. Eine eventuell folgende `GET` Operation macht das nächstfolgende Rootsegment verfügbar und schaltet automatisch um auf DB orientierte Verarbeitung, oder ergibt möglicherweise EOF-Status (GB wenn `RC=YES` definiert ist), wenn es sich vorausgehend um den letzten DB-Record gehandelt hat.

DB-orientiert heisst: Mit jeder `GET` Operation wird das nächstfolgende sensitive Segment verfügbar gemacht, bis das DB Ende erreicht wird. Die `GET` Operation basiert auf der DL/I Funktion GN (Get next). Der Wechsel vom Ende eines DB-Records zum nächstfolgenden Rootsegment wird mit dem Return Code GA signalisiert, wie jede hierarchische Rückstufung innerhalb von Dependent Segmenten.

z.B.	QPAC Operation	DL/I Funktion	FCA Return Code	Verfügbarkeit
a)	`SETGK`			(kein Segment)
	`GET`	GU		Rootsegment
	`GET`	GNP		Dependent Segment
	`GET`	GNP		Dependent Segment
	`GET`	GNP	GE	(kein Segment)
	`SETGK`			(kein Segment)
	`GET`	GU		Rootsegment
	.			
	.			
b)	`SETGK`			(kein Segment)
	`GET`	GU		Rootsegment
	`GET`	GNP		Dependent Segment
	`GET`	GNP		Dependent Segment
	`GET`	GNP	GE	(kein Segment)
	`GET`	GN		Rootsegment
	`GET`	GN		Dependent Segment
	`GET`	GN		Dependent Segment
	`GN`	GN	GA	Rootsegment
	`GET`	GN		Dependent Segment
c)	`SETGK`			(kein Segment)
	`GET`	GHU		Rootsegment
	`PUT`	REPL		
	`GET`	GHNP		Dependent Segment
	`PUT`	REPL		
	`GET`	GHNP		Dependent Segment
	`GET`	GHNP	GE	(kein Segment)
	`PUT`	REPL	DJ Fehler	
d)	.			
	.			
	`GET`	GN		Segment
	`GET`	GN		Segment
	.			
	.			
	`GET`	GN	GB	End of data base sofern `RC=YES` definiert wurde

Die SETEK Instruktion

Eine weitere Anwendungsmöglichkeit bietet die **SETEK** Operation.

Set Equal Key ermöglicht das Aufsetzen auf einem bestimmten Rootsegment mit gleichem Key.

SETEK selbst stellt das Rootsegment noch nicht zur Verfügung, erst ein nachfolgender GET Befehl.

Bevor ein SETEK Befehl durchgeführt wird, muss in die FCA der Key-Wert gespeichert werden. Die Länge des Keys erkennt QPAC aus der DBD. Der nachfolgende GET basiert auf der Funktion GU (GHU) und stellt das Rootsegment mit gleichem Key zur Verfügung. Wird kein Segment mit gleichem Key gefunden, erfolgt der Return Code GE (No Segment found).

Dem ersten GET folgende GETs entsprechen darauf der Funktion GNP (GHNP) und zwar solange, bis das Ende des DB-Records erreicht ist. Darauf wird Return Code GE (No Segment found) übergeben.

```
SET WPOS9020,CL8 = 'KEY'
SETEK-I1  GET-I1
DO-UNTIL WPOS9000,CL2 = 'GE'
...
...
GET-I1
```

Abb. 261: Beispiel SETEK für DL/I Zugriff

Es ist zu beachten, dass bei Verwendung der SETEK Operation die Verarbeitungssteuerung **DB recordorientiert** erfolgt, d.h. pro erreichtem DB-Recordende wird der Return Code GE gesetzt und kein Segment übergeben. Bei Update würde in diesem Falle ein nachfolgender PUT (REPL) als fehlerhaft erscheinen (Return Code DJ).

SETEK unterscheidet sich im Übrigen nicht von der Operation SETGK. Es ist deshalb zu beachten, was über DB-orientierte Verarbeitung unter SETGK geschrieben steht.

Die PUTA Instruktion

Segmente können mit dem Befehl **PUTA** (Put Addition) in eine DB **eingefügt** werden. In die FCA muss der Segmentname platziert werden und in der Recordarea muss das Segment stehen, das eingefügt werden soll.

Als Return Code kann in der FCA ~~bb~~ oder II zurückgegeben werden: letzterer nur wenn nicht NOII als Option definiert wurde.

PUTA ist nur im Updatemodus (UDB) möglich und nur, wenn der PSB 'Inserts' zulässt.

Die PUTD Instruktion

Segmente können mit dem Befehl **PUTD** (Put Delete) aus der DB **gelöscht** werden. Das Segment, das gelöscht werden soll, muss vorausgehend mit GET gelesen worden sein, d.h. es wird durch den Befehl PUTD das jeweils zuletzt gelesene Segment gelöscht.

PUTD ist nur im Updatemodus (UDB=) möglich und nur, wenn der PSB 'Deletes' zulässt.

Die ODB= Definition (Initial Load)

Eine Datenbank kann auch neu erstellt werden. Dazu muss ein PSB mit PROCOPT LS bzw. L verwendet werden. In das FCA-Feld ..SEGNM (Segmentname) muss vor der Ausführung des PUT-Befehls der Segmentname gestellt werden. In der I/O Area steht der Segment-Record.

DL/I Datenbank bezogene Befehle mit SSAs

Eine separate Kategorie von DL/I Befehlen steht zur Verfügung. Diese Befehle verwenden jedoch SSA-Felder, deren Inhalt zusätzlich zu definieren ist.
Es stehen bis zu 8 SSA-Felder zu je maximal 256 Bytes zur Verfügung. Die Namen sind `..SSA1` bis `..SSA8`. Die Verwendung muss in der Reihenfolge der Nummern erfolgen. Die Anzahl der benutzten SSA-Felder wird im Feld `..SSAN` (SSA numbers) bekannt gegeben.

Der Inhalt der SSA-Felder muss dem offiziellen DL/I-Format entsprechen. QPAC setzt die Richtigkeit des Formates voraus.

Diese Befehlskategorie gilt für `IDB` und `UDB`, **nicht** aber für `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

Abb. 262: DL/I-Befehle mit 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
```

Abb. 263: Beispiel der Anwendung

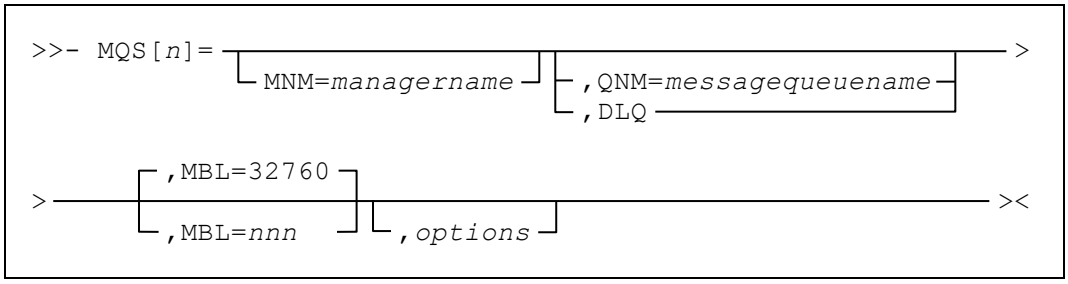
Im FCA-Feld `..RC` (Return Code) steht nach Ausführung der offizielle DL/I Return Code zur Verfügung.
Bei dieser Befehlskategorie ist `RC=YES` standardmässig gesetzt. Es erfolgt daher im Fehlerfalle kein Abbruch.

Kapitel 14. MQSeries Support Feature

MQSeries Single Message Queue Definition

IBM MQSeries für z/OS ist durch dieses Feature voll unterstützt.
Es basiert auf der Voraussetzung, dass MQSeries installiert und die entsprechende Autorisierung gegeben ist.

Grundformat der MQSeries Message Queue Definition



`MNM=` Ist der aktuelle Queue Manager Name des MQSeries Systems. Der Manager Name kann auch, bevor der `CONNECT` Befehl durchgeführt wird, dynamisch ins reservierte Feld **`Qn`MGRNAME** gestellt werden.

`QNM=` Ist der Message Queue Name der Queue, die hier gesendet oder gelesen werden soll. Dieser Queue Name kann auch, bevor der `OPEN` Befehl durchgeführt wird, dynamisch ins reservierte Feld **`Qn`QNAME** gestellt werden.

`DLQ` Dead Letter Queue
Durch diese Definition, anstelle einer Queue-Name Angabe (`QNM=`) wird bestimmt, dass diese Filedeklaration zum Lesen der Dead-Letter Queue vorzusehen ist.
Dadurch wird durch QPAC nach dem Connect-Vorgang der Name der Dead-Letter Queue automatisch eruiert und dynamisch zugeordnet.
Bei den `GET` Commands wird der Dead-Letter-Header (DLH) durch QPAC in die separate Dead-Letter-Area gestellt (`QnDLH_AREA`) und steht dort dem Anwender zur Verfügung. Die eigentliche Message steht in der Bufferarea.

`MBL=` Mit diesem Schlüsselwort kann die maximale Buffer Länge, die für die Message Queue vorgesehen werden soll, definiert werden. Die Angabe kann in **Bytes oder in Megabytes** definiert werden.
Megabytes werden in den Grössen **1M 2M 3M 4M** angegeben. 100M ist der grösste Wert, der in z/OS zulässig ist.
Fehlt diese Definition, wird per default eine Buffergrösse von 32760 Bytes angenommen. Dieser Wert kann nicht dynamisch zugeordnet bzw. vergrössert, jedoch nach Bedarf über das Schlüsselwort **`Qn`BUFFLENG** verkleinert werden.

`Options` Die nachfolgend aufgeführten Optionen können zusätzlich verwendet werden:

`RC=YES`

Der CompletionCode und der ReasonCode werden in der FCA zurückgegeben und es erfolgt bei einem Fehler kein Abbruch. Die zugeordneten reservierten FCA Felder lauten **`Qn`COMPCODE** und **`Qn`REASON**.
Bei einem Fehler steht ausserdem im FCA Feld `Qn`CMDTEXT der aktuelle Command und im Feld `Qn`REASONTEXT der Reason Code eigene Klartext.

`FCA=`

File Communication Area
Mit dieser Definition wird bestimmt, an welcher Stelle innerhalb des allgemeinen QPAC-Workbereiches sich die Informations-Austauscharea für die zugehörige MQSeries Filedefinition befinden soll. Die FCA wird normalerweise nicht explizit in den Workbereich gelegt. Sie wird, wenn nicht explizit angegeben, dynamisch angelegt und über die zugeteilten reservierten Feldsymbole angesprochen.
Der Aufbau der FCA ist weiter unten grafisch dargestellt.

`CLR=X'00'`
`CLR=NO`

Mit dieser Option kann der Löschwert angegeben werden, mit dem die Message Queue Area nach einem `PUT` Command gelöscht werden soll. Fehlt diese Definition, wird `X'00'` angenommen, d.h. die Queue Area wird jeweils auf LowValue gelöscht. Mit der Definition `CLR=NO` kann das Löschen verhindert werden.

`WP=`*`nnnnn`*

Workbereich Position
Mit dieser Definition wird bestimmt, dass die MQSeries Message Queue Area in den allgemeinen Workbereich gelegt wird. Entsprechend gross muss dabei der Allgemeine Workbereich mit dem `PARM` Parameter **`PARM=WORK=`*`nnnnn`** definiert sein. Wird `WP=` definiert, existiert kein zusätzlicher dynamisch angelegter Queue Area Bereich.

Verarbeitung von MQSeries Message Queues

Die Verarbeitung von Message Queues durch QPAC erfolgt mit den Commands **CONNECT**, **OPEN**, **PUT** bzw. **PUT1**, **GET**, **COMMIT**, **CLOSE**, **DISCONNECT** und in speziellen Fällen **INQUIRE** und **SET**.

In einem QPAC Programm können gleichzeitig mehrere Message Queues definiert werden.
Die Länge aller maximal adressierbaren Buffergrössen darf jedoch 16 MB nicht überschreiten, d.h. es könnten beispielsweise 4 Message Queues mit je maximal 4 MB (MBL=4M) definiert werden.

Neben den Message Queue Definitionen können alle übrigen File- bzw. Datenbank-Organisationen, wie VSAM, SAM, DB2 etc. in beliebiger Kombination definiert werden. Die Deklaration erfolgt immer nach dem gleichen Prinzip, indem dem Fileident-Schlüsselwort eine Zahl von 1-99 beigefügt wird.

```
IPF1=...   IDB2=...   MQS3=...   OPF4=...
```

Abb. 264: Explizite Filedefinitionen

Die Zahl ist bei den Input/Output Befehlen bzw. bei den impliziten Positionssymbolen Bestandteil der Referenz.
Beispielsweise bezieht sich das Symbol mit der Positionsangabe Q3POS1 auf die erste Position der Message Queue Area der File Deklaration MQS3=. Dasselbe gilt für den Command CONNECT-Q3.

Bevor eine Message bearbeitet werden kann, gelesen oder gesendet,.muss mit dem **CONNECT** Command die Verbindung zum Message Queue Manager hergestellt werden. Der CONNECT Befehl muss für jede einzelne MQS*n* Definition durchgeführt werden. QPAC prüft intern, ob schon eine vorausgehende Connection zum gleichen Queue Manager existiert und stellt intern die Beziehung her. Dasselbe gilt für den **DISCONNECT** Befehl.

Der Zugang zu einer Message muss ausserdem durch einen OPEN Befehl eröffnet werden. Entsprechend der nachfolgend gewünschten Verarbeitung muss die Open Option gesetzt werden. Die Open Option wird ins vordefinierte Feld **Q*n*OPENOPT** gespeichert. Dazu stellt QPAC die offiziellen symbolischen Bezeichnungen (wie im CMQA Macro/Book ersichtlich) zur Verfügung, die an Stelle der direkten Zahlen als Sendefelder verwendet werden können.
Beispielsweise kann für zu sendende Messages die Option OUTPUT wie folgt gesetzt werden:

```
SET Q3OPENOPT = MQOO_OUTPUT
```

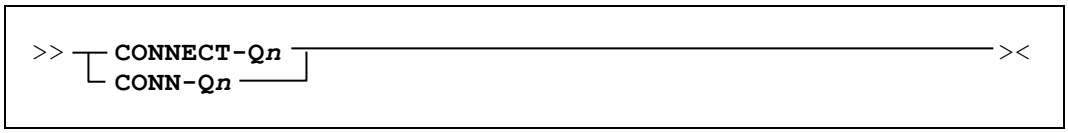
Abb. 265: Output Option

Ist bei der Filedefinition **RC=YES** definiert worden, muss der Completion Code abgefragt werden. Auch hier können die offiziellen symbolischen Bezeichnungen gemäss CMQA Macro verwendet werden.

```
IF Q3COMPCODE = MQCC_OK     THEN ...
IF Q3COMPCODE = MQCC_FAILED THEN ...
```

Abb. 266: Beispiele der Completion Code Abfrage

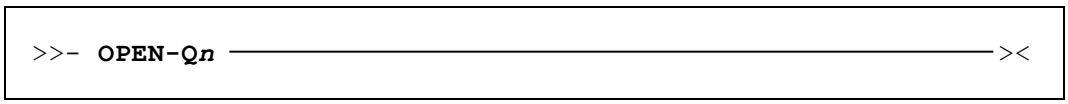
MQSeries Commands



Vor dem CONNECT Command kann ins Feld **Q*n*MGRNAME** ein Queue Manager Name eingegeben werden, sofern er nicht schon in der Filedefinition definiert wurde.

```
SET Q1MGRNAME = 'MANAGER'
CONNECT-Q1
IF Q1COMPCODE NOT = 0 THEN oops an error IFEND
```

Abb. 267: Beispiel MQSeries CONNECT Command



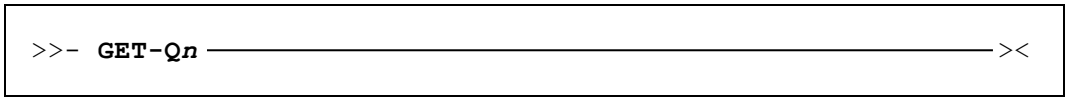
Vor dem OPEN Command muss ins Feld **Q*n*OPENOPT** die Open Option eingetragen werden. Siehe diesbezüglich die Open-Option-Liste unter **'Values Related to MQOPEN** Open Options'. Ebenfalls kann ins Feld **Q*n*QNAME** ein Message-Queue Name angegeben werden, sofern er nicht schon in der Filedefinition angegeben wurde.

```
SET Q1OPENOPT = MQOO_INPUT_SHARED
OPEN-Q1
IF Q1COMPCODE NOT = 0 THEN oops an error
```

oder

```
SET Q1QNAME   = 'MESSAGE_QUEUE_NAME'
SET Q1OPENOPT = MQOO_OUTPUT
OPEN-Q1
IF Q1COMPCODE NOT = 0 THEN oops an error IFEND
```

Abb. 268: Beispiele MQSeries OPEN Command



Vor dem GET Command sind nach Bedarf ins Feld QnGMO_OPTIONS die Options zu setzen, die in der Liste "Get Message Options" aufgeführt sind. Weiter Felder, die nach Bedarf zu beachten sind, können in der MQGET OPTIONS Area nachgesehen werden.
Siehe diesbezüglich die Equates unter **'Values Related to MQGMO'**.

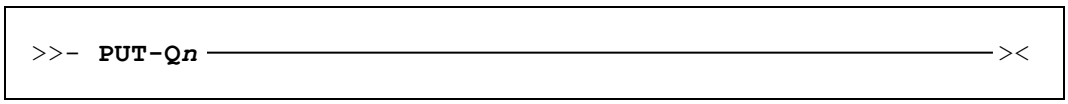
Nach einem GET Command steht in der Input-Output Area die Message, zu verstehen wie ein Record eines konventionellen Data Sets. Im FCA Feld QnDATALENG steht die aktuelle Länge der Message. Diese kann kleiner sein als die Bufferlänge, d.h. die maximale Input-Output Area. Wenn dagegen das FCA Feld Completion Code QnCOMPCODE nicht 0 enthält, ist ein Fehler aufgetreten, dessen genauerer Grund im FCA Feld Reason Code QnREASON steht. Zusätzlich steht im FCA Feld QnREASONTEXT ein kurzer Klartext zum Reason Code.

Ein GET Command ist nur erlaubt, wenn die Verarbeitung mit einer Input Open Option eröffnet wurde, beispielsweise MQOO_BROWSE oder MQOO_INPUT_AS_Q_DEF.

```
GET-Qn AT-EMPTY ... wenn Message Queue leer ist ... ATEND
```

Abb. 269: Beispiel MQSeries GET Command

AT-EMPTY ist gleichbedeutend mit AT-EOF nur mit dem Unterschied, dass kein automatischer CLOSE erfolgt. AT-EMPTY gibt es nur bei MQSn= Definitionen und kann angegeben werden, wenn RC=YES nicht definiert wurde.



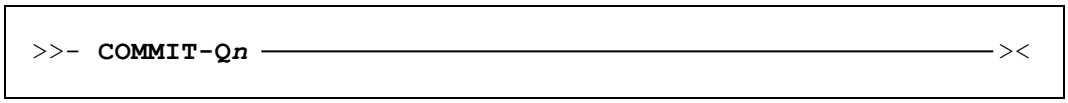
Vor dem PUT Command muss in die Output Area die Message gespeichert werden. Ins FCA Feld QnDATALENG sollte die aktuelle Länge gespeichert werden. Fehlt diese, wird die maximale Bufferlänge angenommen (MBL=).
Ebenfalls sind je nach Bedarf auch die PUT Options zu beachten. Siehe diesbezüglich die Equates unter **'Values Related to MQPMO'**.

Der PUT Command ist nur erlaubt, wenn die Verarbeitung mit der Open Option MQOO_OUTPUT eröffnet wurde.

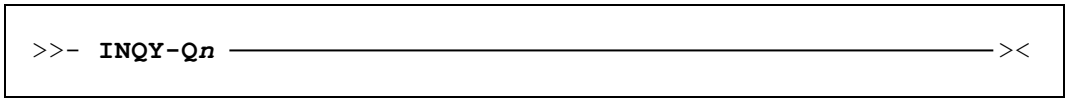
```
SET Q1POS1,CL100 = 'MESSAGE DATA'
SET Q1DATALENG   = 100
PUT-Q1
IF Q1COMPCODE NOT = 0 THEN oops an error IFEND
```

Abb. 270: Beispiel MQSeries PUT Command

Der Completion Code QnCOMPCODE muss abgefragt werden, wenn bei der Filedeklaration RC=YES angegeben wurde. Ist RC=YES nicht definiert, bricht QPAC bei einem Fehler automatisch ab.



Den COMMIT Command kann man nach Bedarf nach einem PUT Command angeben. Die Bedeutung ist selbsterklärend.



Mit dem INQUIRE Command können Informationen abgerufen werden.

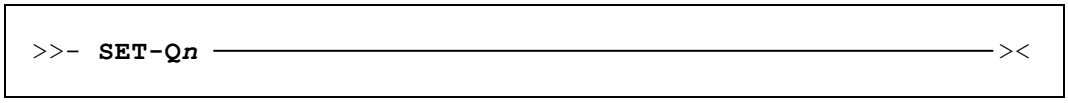
Vor dem Command müssen die SELECTOR Werte gesetzt werden. Dazu stehen maximal 16 Selector Felder zur Verfügung. Sie sind vordefiniert und heissen **QnSELECTOR1,BL4** bis **QnSLECTOR16,BL4**. Das Symbol **QnSLECTORS,CL64** spannt sich über alle Einzelfelder. Zusätzlich gibt es ein Feld **QnSELCNT,BL4**, in das jeweils die Anzahl benutzter Selector Felder gesetzt werden muss. Dabei gibt es zwei Gruppen, Character Attribute Selectors und Integer Attribute Selectors.

Das Ergebnis der Character Attribue Selectors steht im vordefinierten Feld **QnCHARATTAREA,CL256** und das Ergebnis der Integer Attribute Selectors im Feld **QnINTATTARRAY,CL64**. QnINTATTARRAY sind 16 aneinanderliegende Felder vom Format BL4. Diese 16 Integer Attribute Felder besitzen ebenfalls je einen eigenen Namen: **QnINTATT1** bis **QnINTATT16**.

Zu beiden Areas gibt es noch ein Längen- bzw. ein Counterfeld, die ebenfalls vordefiniert sind. Sie heissen **QnCHARATTLENG,BL4** und **QnINTATTCNT,BL4**. In diese Felder ist die Länge der benutzten Character Attribute Area bzw. die Anzahl der benutzten Integer Attribute Felder zu setzen.

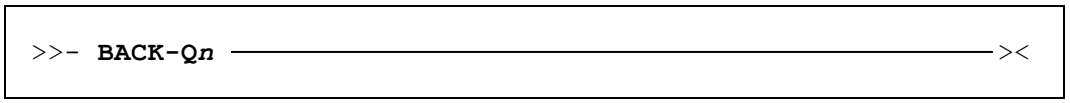
```
SET Q1SELCNT      = 1
SET Q1SELECTOR1   = MQCA_CREATION_DATE
SET Q1INTATTCNT   = 0
SET Q1CHARATTLENG = 12
INQY-Q1
IF Q1COMPCODE NOT = 0 THEN oops an error IFEND
```

Abb. 271: Beispiel MQSeries INQY Command

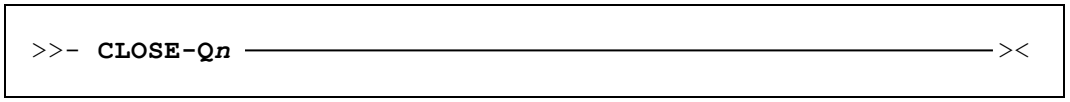


Mit dem SET Command können neue Attribute gesetzt werden.
Der SET Command ist vom Prinzip her das Gegenteil vom INQY (INQUIRE) Command.

Vor dem SET Command müssen im umgekehrten Sinne zum INQuire Command die Selector Felder, die zu speichernden Character Werte bzw. die Integer Attribute abgefüllt werden. Es gelten dabei die gleichen Felder wie sie unter dem INQY Command beschrieben sind.



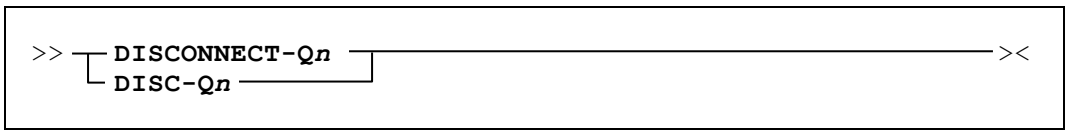
Mit diesem Command kann ein Rollback durchgeführt werden. Alle GET-Q*n* und PUT-Q*n* Operationen werden rückgängig gemacht. Dieser Command ist nur VOR einem CLOSE-Q*n* effektiv.



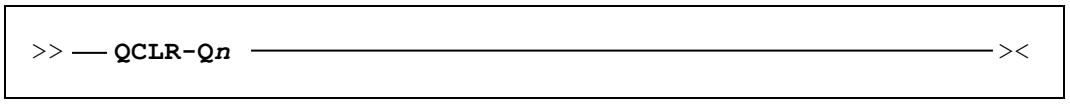
Vor dem CLOSE Command muss ins Feld Q*n*CLOSEOPT die Close-Option eingetragen werden.
Siehe diesbezüglich die Close-Option-Liste unter '**Values Related to MQCLOSE** Close Options'.

```
SET Q1CLOSEOPT = MQCO_NONE
CLOSE-Q1
```

Abb. 272: Beispiel MQSeries CLOSE Command



Vor dem DISCONNECT Command sind im Normalfalle keine zusätzlichen Options zu definieren. Mit dem DISCONNECT Command wird die Verbindung zum Queue Manager definitiv beendet.



Mit diesem Command kann der Inhalt der referenzierten Queue gelöscht werden. Der Command arbeitet intern mit dem MQSeries Utility CSQUTIL. Die JCL Statements //SYSIN DD .. und //SYSPRINT DD .. werden dabei intern alloziert und sollten daher im Job Step nicht definiert sein.

Im Falle eines Fehlers steht im FCA Feld **Q*n*REASON** der Reason Code und im FCA Feld **Q*n*CMDTEXT** der Command bzw. im FCA Feld **Q*n*REASONTEXT** der zugehörige Reason Code Klartext gemäss CMQA Macro, beispielsweise `MQRC_NOT_OPEN_FOR_OUTPUT`.

Die FCA für MQSeries besitzt folgenden Aufbau:

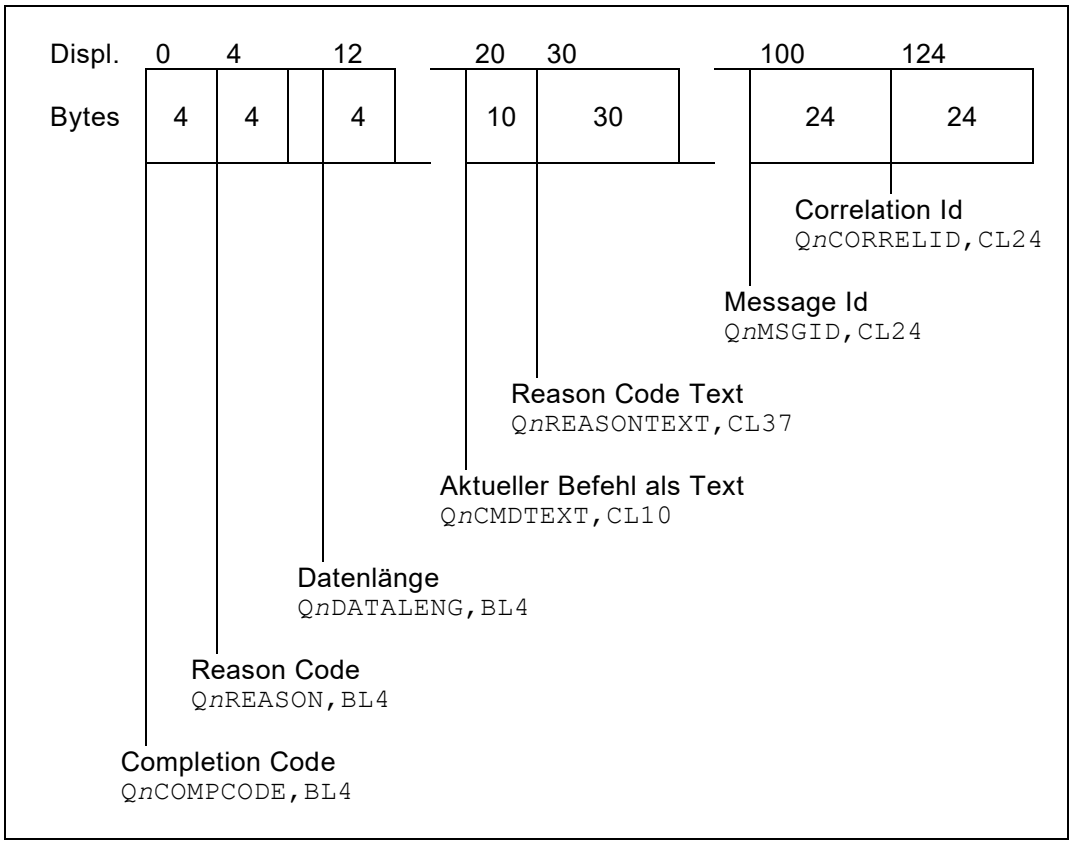


Abb. 273: FCA für MQSeries

Ins FCA Feld **Q*n*DATALENG** muss vor einem `PUT` Command die aktuelle Message Länge gespeichert werden. Fehlt diese Länge, bzw. ist der Inhalt binär 0, wird die Maximale Buffer Länge als Message Länge genommen (**MBL=***nnn*).

Nach einem `GET` Command steht in dem FCA Feld **QnDATALENG** die aktuelle Message Länge, die in der Queue Area empfangen wurde. Ist die definierte maximale Buffer Länge zu klein, wird ein Buffer-Längenfehler zurückgegeben.

Areas

Den internen Areas, die bei den einzelnen Commands als Parameter zugeordnet sind, sind nachfolgend aufgeführte reservierte Feldnamen fest zugeteilt.
Die Symbolnamen entsprechen den offiziellen Bezeichnungen, wie sie in den Macros/Books CMQODA, CMQMDA, CMQGMOA, CMQPMOA vorgegeben werden.

Object Descriptor Area

Symbol	Offset	Länge	Fmt	Bezeichnung
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	Länge	Fmt	Bezeichnung
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	Länge	Fmt	Bezeichnung
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	Länge	Fmt	Bezeichnung
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	Länge	Fmt	Bezeichnung
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	Länge	Fmt	Bezeichnung
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	Länge	Fmt	Bezeichnung
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 der verschiedenen Options- bzw. Fieldvalues

Den verschiedenen Werten, die je nach den notwendigen Anwendungsbedürfnissen in die Felder der Areas gesetzt werden müssen, sind offizielle Symbolnamen zugeteilt, die auch in QPAC verwendet werden können.
Die wichtigsten sind hier aufgeführt. Im Uebrigen entsprechen sie genau den Angaben, wie sie in den MQSeries Manuals der IBM bzw. in den Copy-Books der Programmiersprachen zu finden sind.
Die nähere Bedeutung ist in den MQSeries Manuals der IBM nachzulesen.

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
```

Kapitel 15. CICS External Interface Support Feature (EXCI)

EXCI External Batch to CICS Communication Definition

Die Kommunikation zwischen Batch und CICS ist durch den Sunset von QPAC for CICS (QPAC-Online) ab QPAC Release 9.10 nicht mehr verfügbar.

Kapitel 16. ISPF/PDF Support Feature

ISPF/TSO Command Definition

IBM ISPF unter TSO MVS ist durch dieses Feature voll unterstützt. Es ermöglicht die Programmierung ganzer ISPF-Applikationen an Stelle von beispielsweise CLIST. Durch dieses Feature ist es möglich, in einer ISPF Applikation gleichzeitig neben den unter CLIST unterstützten Fileorganisationen wie PDS und SAM auch VSAM, Datenbanken wie DB2 oder SYSOUT Daten, sogar MQSeries zu verarbeiten. Der ganze Sprachumfang von QPAC-Batch steht dazu zur Verfügung.

Grundformat der ISPF Command Definitionen

Alle offiziellen ISPF Commands, die unten aufgeführt sind, sind unterstützt. Dabei ist die Command-Konstante, wie sie unter CLIST normalerweise beim CALL ISPLINK-Befehl definiert wird, bei QPAC der Befehl selbst, gefolgt von einem Bindestrich.
Die Operanden entsprechen der normalen Syntax, wie sie im IBM Manual "**ISPF Services Guide"** oder "**ISPF Reference Summary"** beschrieben sind.
Nicht unterstützt sind die Operanden, die sich auf DBCS beziehen.

Nachfolgend sind die Commands aufgeführt. Die zugehörigen Beschreibungen der Operanden sind sinnvollerweise im IBM Manual zu konsultieren. Hier sind nur Hinweise gegeben bzw. ein Überblick.

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-

Syntax Beispiel: QPAC Programm Beispiel 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: Beispiel 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 Beispiel

```
 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
```

Anhang A. Basis Instruktionsformate (Zus.fassung)

Überblick

In diesem Anhang sind die wichtigsten Instruktionsformate des **ehemaligen QPAC-Batch Basisteils** zusammengefasst.
Diese Instruktionsformate können weder von der symbolischen Adressierung noch von der automatischen Datenkonversion profitieren und sollten daher, wenn immer möglich, durch das neue High-Level Format der QPAC `SET`-Instruktion ersetzt werden, oder aber mindestens durch die Impliziten Positionssymbole.

```
IPOS1,OPOS1,80
```

Imperative Instruktionen und Operationen

Grundformat

```
VON-ADRESSE,NACH-ADRESSE,OPERATION,LAENGEN

1. VON,NACH,OPERATION,VON-LAENGE,NACH-LAENGE

2. VON,NACH,OPERATION,VON-LAENGE

3. VON,NACH,VON-LAENGE

4. VON,NACH,EDITMASKE
```

VON kann sein:	*iadr*	= Inputarea Adresse
	wadr	= Workarea Adresse
	lit	= Literal / Konstante
NACH kann sein:	*oadr*	= Outputarea Adresse
	wadr	= Workarea Adresse

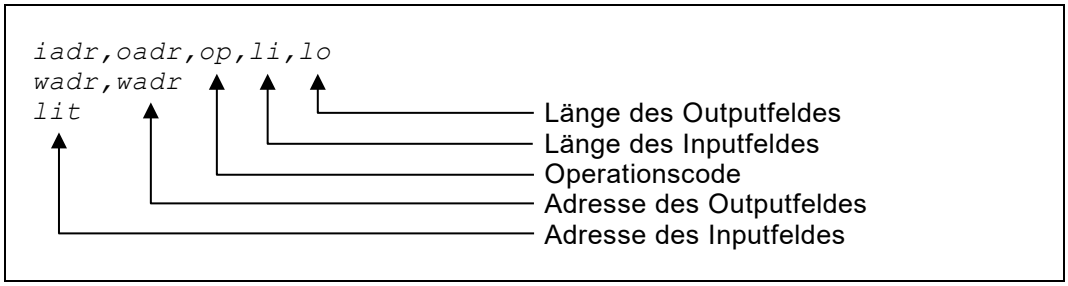
OPERATION ist ein Operationscode

LAENGE ist immer in Bytes zu verstehen

EDITMASKE ist ein Editierliteral

Für weitere, detailliertere Informationen konsultieren Sie bitte das Kapitel 7: Die High-Level-Format Instruktion SET.

Format 1



Dieses Format gilt bei allen Instruktionen, bei denen zwei Längen angegeben werden können, z.B. zur Verarbeitung von gepackten Feldern.

Format 2

```
iadr,oadr,op,l
wadr,wadr
lit
```

Dieses Format gilt bei allen Instruktionen, bei denen nur eine Länge angegeben werden muss, z.B. gewisse Übertragungsoperationen.

Format 3

```
iadr,oadr,l
wadr,wadr
lit
```

Spezialformat für die einfache Übertragung.
Es ist kein Operationscode notwendig. Die Übertragung findet von *iadr* nach *oadr* statt, in der angegebenen Länge, welche innerhalb der Recordlängen liegen muss.

Format 4

```
iadr,oadr,editmask
wadr,wadr
```

Spezialformat für Editier-Operationen.

Für weitere, detailliertere Informationen konsultieren Sie bitte das Kapitel 7: Die High-Level-Format Instruktion SET.

Literals / Konstanten

`C'....'` `'....'`	**Character Konstanten** Der Characterstring zwischen Apostrophs gilt als Wert. Das Attributzeichen C kann fehlen (Default). Ein Apostroph innerhalb der Konstanten kann definiert werden, indem man es verdoppelt. z.B. `C'JOHN''S'`
`X'....'`	**Hexadezimal Konstante** Die Hexadezimalziffern zwischen Apostrophs werden als Konstante verwendet. z.B. `X'4040F1'`
`P'....'`	**Gepackte Konstante** Die Dezimalziffern zwischen Apostrophs werden in gepackte Form umgesetzt und als arithmetischer Wert verwendet. Es kann ein Minuszeichen definiert werden. Fehlt ein Solches, gilt der Wert als positiv. z.B. `P'-10'` `P'10-'`
`F'....'`	**Fullword Konstante** Die Dezimalziffern zwischen Apostrophs werden in binäre Form umgesetzt und als arithmetischer Wert verwendet. Es kann ein Minuszeichen definiert werden. Fehlt ein Solches, gilt der Wert als positiv. Eine Fullword Konstante ergibt immer eine 4 Bytes lange Binärkonstante. z.B. `F'-100'` `F'100-`

Für weitere, detailliertere Informationen konsultieren Sie bitte das Kapitel 7: Die High-Level-Format Instruktion SET.

Einfache Übertragungsoperation

```
iadr,oadr,l
wadr,wadr
lit
```

- *iadr* wird logisch nach *oadr* übertragen.
  Die Länge bezieht sich auf Bytes.

```
1,1,80
180,25,30
```

- Anstelle von *iadr* und *oadr* kann *wadr* definiert werden, eine Working Storage Adresse von 5000 - 20999.

```
5000,7100,100
6230,6510,8
```

- Anstelle von *iadr* kann ein **beliebiger Konstantentyp** stehen.

- Ist die Übertragungslänge durch eine Konstante bestimmt, kann auf die Längendefinition verzichtet werden. Wird aber eine Länge angegeben, muss sie mit derjenigen des Literals übereinstimmen.

```
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
```

- Wird bei einfachen Übertragungsoperationen keine Länge angegeben, wird eine **Länge von 1** angenommen.

```
6010,6020 [,1]
```

Für weitere, detailliertere Informationen konsultieren Sie bitte das Kapitel 7: Die High-Level-Format Instruktion SET.

Boolsche Operationen

Boolsches AND

```
iadr,oadr,AND,l
wadr,wadr
C' '
X' '
```

- Verknüpfungsbefehl nach Boolscher UND Logik.
- *oadr* wird mit *iadr* UND verknüpft.
- Das Resultat steht in *oadr*.
- Die Länge bezieht sich auf Bytes.
- Wird keine Länge definiert, wird eine **Länge von 1** angenommen.

```
                          1 + 1 = 1
1,7000,AND,100            1 + 0 = 0      Beide 1 => 1
X'40',5000,AND            0 + 1 = 0      alle anderen Fälle => 0
                          0 + 0 = 0
```

Boolsches OR

```
iadr,oadr,OR,l
wadr,wadr
C' '
X' '
```

- Verknüpfungsbefehl nach Boolscher ODER Logik.
- *oadr* wird mit *iadr* ODER verknüpft.
- Das Resultat steht in *oadr*.
- Die Länge bezieht sich auf Bytes.
- Wird keine Länge definiert, wird eine **Länge von 1** angenommen.

```
                          1 + 1 = 1
1,7000,OR,100             1 + 0 = 1      Beide 0 => 0
X'F0',5700,OR             0 + 1 = 0      alle anderen Fälle => 1
                          0 + 0 = 0
```

Für weitere, detailliertere Informationen konsultieren Sie bitte das Kapitel 7: Die High-Level-Format Instruktion SET.

Boolsches XOR

```
iadr,oadr,XOR,l
wadr,wadr
C' '
X' '
```

- Verknüpfungsbefehl nach Boolscher EXKLUSIV ODER Logik.
- *oadr* wird mit *iadr* EXKLUSIV ODER verknüpft.
- Das Resultat steht in *oadr*.
- Die Länge bezieht sich auf Bytes.
- Wird keine Länge definiert, wird eine Länge von 1 angenommen.

```
                          1 + 1 = 0
1,7000,XOR,100            1 + 0 = 1     Beide gleich => 0
X'F0',5100,XOR            0 + 1 = 1     alle anderen Fälle => 0
                          0 + 0 = 0
```

Für weitere, detailliertere Informationen konsultieren Sie bitte das Kapitel 7: Die High-Level-Format Instruktion SET.

Algebraische Operationen

Addition

```
iadr,oadr,A,l,l
wadr,wadr
P' '
```

- *iadr* wird nach *oadr* **addiert**
- Beide Felder sind gepackte Datenfelder.
- Die Längen beziehen sich auf Bytes.

```
1,5,A,8,8
10,2,A,8,4
```

- Anstelle von *iadr* und *oadr* kann *wadr* definiert werden, eine Working Storage Adresse von 5000 - 20999.

```
6000,6050,A,8,8
100,6200,A,5,8
```

- Anstelle von *iadr* kann eine **gepackte Konstante** stehen.

- Ist die Übertragungslänge durch eine Konstante bestimmt, kann auf die Längendefinition verzichtet werden. Wird aber eine Länge angegeben, muss sie mit derjenigen des Literals übereinstimmen.

```
P'125',6020,A,,8
P'125',6020,A,2,8
P'125-',6030,A,,8
```

- Wird bei Operationen mit gepackten Feldern keine Länge angegeben, wird eine **Länge von 8** angenommen.

```
6010,6020,A    [,8,8]
100,6010,A,5   [,8]
P'125',6020,A  [,2,8]
```

- Diese Längenannahme gilt bei allen Operationen, die gepackte Felder bearbeiten: A, S, M, D, ZA, CB, CD, P, U.

- Ausnahme: Gepacktes Literal **ohne** Operationscode:

```
P'100',60
```

Für weitere, detailliertere Informationen konsultieren Sie bitte das Kapitel 7: Die High-Level-Format Instruktion SET.

Subtraktion

```
iadr,oadr,S,l,l
wadr,wadr
P' '
```

- *iadr* wird von *oadr* **subtrahiert**.
- Beide Felder sind gepackte Datenfelder.
- Die Längen beziehen sich auf Bytes.

```
1,5,S,8,8
10,2,S,8,4
```

- Anstelle von *iadr* und *oadr* kann *wadr* definiert werden, eine Working Storage Adresse von 5000 - 20999.

```
6000,6050,S,8,8
100,6200,S,5,8
```

- Anstelle von *iadr* kann eine **gepackte Konstante** stehen.

- Ist die Übertragungslänge durch eine Konstante bestimmt, kann auf die Längendefinition verzichtet werden. Wird aber eine Länge angegeben, muss sie mit derjenigen des Literals übereinstimmen.

```
P'125',6020,S,,8
P'125',6020,S,2,8
P'125-',6030,S,,8
```

- Wird bei Operationen mit gepackten Feldern keine Länge angegeben, wird eine **Länge von 8** angenommen.

```
6010,6020,S    [,8,8]
100,6010,S,5   [,8]
P'125',6020,S  [,2,8]
```

- Diese Längenannahme gilt bei allen Operationen, die gepackte Felder bearbeiten: `A, S, M, D, ZA, CB, CD, P, U.`

- Ausnahme: Gepacktes Literal **ohne** Operationscode.

```
P'100',60
```

Für weitere, detailliertere Informationen konsultieren Sie bitte das Kapitel 7: Die High-Level-Format Instruktion SET.

Multiplikation

```
iadr,oadr,M,l,l
wadr,wadr
P' '
```

- *oadr* wird mit *iadr* **multipliziert**.
- Das Resultat (Produkt) steht in *oadr*.
- Beide Felder sind gepackte Datenfelder.
- Die Längen beziehen sich auf Bytes.
- Überlaufdaten werden im Resultat **ohne Warnung** abgeschnitten.

```
5,10,M,8,8
15,22,M,4,8
```

Anstelle von *iadr* und *oadr* kann *wadr* definiert werden, eine Working Storage Adresse von 5000 - 20999.

```
6000,6050,M,8,8
100,6200,M,5,8
```

- Anstelle von *iadr* kann eine **gepackte Konstante** stehen.

- Ist die Übertragungslänge durch eine Konstante bestimmt, kann auf die Längendefinition verzichtet werden. Wird aber eine Länge angegeben, muss sie mit derjenigen des Literals übereinstimmen.

```
P'125',6020,M,,8
P'125',6020,M,2,8
P'125-',6030,M,,8
```

- Wird bei Operationen mit gepackten Feldern keine Länge angegeben, wird eine **Länge von 8** angenommen.

```
6010,6020,M    [,8,8]
100,6010,M,5   [,8]
P'125',6020,M [,2,8]
```

- Diese Längenannahme gilt bei allen Operationen, die gepackte Felder bearbeiten: `A, S, M, D, ZA, CB, CD, P, U.`

- Ausnahme: Gepacktes Literal **ohne** Operationscode.

```
P'100',60
```

Für weitere, detailliertere Informationen konsultieren Sie bitte das Kapitel 7: Die High-Level-Format Instruktion SET.

Division

```
iadr,oadr,D,l,l
wadr,wadr
P' '
```

- *oadr* wird durch *iadr* **dividiert**.
- Das Resultat (Quotient) steht in *oadr*, *iadr* ist der Divisor.
- Ein Rest steht nicht zur Verfügung.
- Beide Felder sind gepackte Datenfelder.
- Die Längen beziehen sich auf Bytes.
- Die Länge des Divisors sollte 8 Bytes nicht übersteigen, sonst werden führende Ziffern abgeschnitten.

```
5,10,D,8,8
15,22,D,4,8
```

- Anstelle von *iadr* und *oadr* kann *wadr* definiert werden, eine Working Storage Adresse von 5000 - 20999.

```
6000,6050,D,8,8
100,6200,D,5,8
```

- Anstelle von *iadr* kann eine **gepackte Konstante** stehen.

- Ist die Übertragungslänge durch eine Konstante bestimmt, kann auf die Längendefinition verzichtet werden. Wird aber eine Länge angegeben, muss sie mit derjenigen des Literals übereinstimmen.

```
P'125',6020,D,,8
P'125',6020,D,2,8
P'125-',6030,D,,8
```

- Wird bei Operationen mit gepackten Feldern keine Länge angegeben, wird eine **Länge von 8** angenommen.

```
6010,6020,D    [,8,8]
100,6010,D,5   [,8]
P'125',6020,D  [,2,8]
```

- Diese Längenannahme gilt bei allen Operationen, die gepackte Felder bearbeiten: `A, S, M, D, ZA, CB, CD, P, U.`

- Ausnahme: Gepacktes Literal **ohne** Operationscode.

```
P'100',60
```

Für weitere, detailliertere Informationen konsultieren Sie bitte das Kapitel 7: Die High-Level-Format Instruktion SET.

Konversions Operationen

Gepackt zu Binär Konversion

```
iadr,oadr,CB,l,l
wadr,wadr
P' '
```

- Konvertieren **gepackt zu binär**.
- Die *iadr* Feldlänge kann zwischen 1 und 16 Bytes umfassen.
- Die *oadr* Feldlänge hat eine Länge zwischen 1 und 4 Bytes und muss nicht auf Wortgrenze aliniert sein.

```
5,10,CB,8,4
10,22,CB,16,4
```

- Anstelle von *iadr* und *oadr* kann *wadr* definiert werden, eine Working Storage Adresse von 5000 - 20999.

```
6000,6050,CB,8,4
100,5200,CB,5,4
```

- Anstelle von *iadr* kann eine **gepackte Konstante** stehen.

- Ist die Übertragungslänge durch eine Konstante bestimmt, kann auf die Längendefinition verzichtet werden. Wird aber eine Länge angegeben, muss sie mit derjenigen des Literals übereinstimmen.

```
P'125',7250,CB,,4
P'125',7250,CB,2,4
```

- Wird keine Länge angegeben, wird für gepackte Felder eine **Länge von 8** und für binäre Felder eine **Länge von 4** angenommen.

```
6010,5010,CB    [,8,4]
100,5110,CB,5   [,4]
P'125',5020,CB  [,2,4]
```

Für weitere, detailliertere Informationen konsultieren Sie bitte das Kapitel 7: Die High-Level-Format Instruktion SET.

Binär zu Gepackt Konversion

```
iadr,oadr,CD,l,l
wadr,wadr
F' '
```

- Konvertieren **binär zu gepackt**.
- Die *iadr* Feldlänge kann zwischen 1 und 4 Bytes umfassen und muss nicht auf Wortgrenze aliniert sein.
- Die *oadr* Feldlänge hat eine Länge zwischen 1 und 16 Bytes.

```
5,10,CD,4,8
10,22,CD,4,5
```

Anstelle von *iadr* und *oadr* kann *wadr* definiert werden, eine Working Storage Adresse von 5000 - 20999.

```
5000,6000,CB,4,8
100,6200,CB,4,4
```

- Anstelle von *iadr* kann eine **Fullword Konstante** stehen.

- Ist die Übertragungslänge durch eine Konstante bestimmt, kann auf die Längendefinition verzichtet werden. Wird aber eine Länge angegeben, muss sie mit derjenigen des Literals übereinstimmen.

```
F'1',6000,CD,,8
F'1',6020,CD,4,8
```

- Wird keine Länge angegeben, wird für binäre Felder eine **Länge von 4** und für gepackte Felder eine **Länge von 8** angenommen.

```
5010,6010,CD    [,4,8]
100,6110,CD,4   [,8]
F'1',6020,CD    [,4,8]
```

Für weitere, detailliertere Informationen konsultieren Sie bitte das Kapitel 7: Die High-Level-Format Instruktion SET.

Pack Operation

```
iadr,oadr,P,l,l
wadr,wadr
```

- Packen zoned-decimal Feld *iadr* nach *oadr*.
- Das Vorzeichen in *oadr* wird auf F gesetzt, sofern der zoned-decimal Wert nicht negativ ist (Vorzeichen D).
- Demzufolge ist es möglich, Blank zu packen.

```
5,10,P,4,8
10,22,P,4,5
```

- Anstelle von *iadr* und *oadr* kann *wadr* definiert werden, eine Working Storage Adresse von 5000 - 20999.

```
5000,6000,P,16,8
100,6200,P,8,4
```

- Anstelle von *iadr* kann eine **zoned-decimal Konstante** stehen (Character Konstante mit numerischem Inhalt).

- Ist die Übertragungslänge durch eine Konstante bestimmt, kann auf die Längendefinition verzichtet werden. Wird aber eine Länge angegeben, muss sie mit derjenigen des Literals übereinstimmen.

```
'123',6000,P,,8
C'123',6000,P,,8
'123',6020,P,3,8
```

- Wird keine Länge angegeben, wird für zoned-decimal Felder eine **Länge von 16** und für gepackte Felder eine **Länge von 8** angenommen.

```
5010,6010,P    [,16,8]
100,6110,P,4   [,8]
'123',6020,P   [,3,8]
```

Für weitere, detailliertere Informationen konsultieren Sie bitte das Kapitel 7: Die High-Level-Format Instruktion SET.

Unpack Operation

```
iadr,oadr,U,l,l
wadr,wadr
```

- Entpacken (Unpack) gepacktes Feld *iadr* nach zoned-decimal Feld *oadr*.

```
5,10,U,8,16
10,22,U,2,3
```

- Anstelle von *iadr* und *oadr* kann *wadr* definiert werden, eine Working Storage Adresse von 5000 - 20999.

```
6000,5000,U,8,16
6215,100,U,4,8
```

- Wird keine Länge angegeben, wird für zoned-decimal Felder eine **Länge von 16** und für gepackte Felder eine **Länge von 8** angenommen.

```
6010,5010,U    [,8,16]
100,5110,U,4   [,16]
```

Zero Add Operation

```
iadr,oadr,ZA,l,l
wadr,wadr
P' '
```

- *iadr* ist ein gepacktes Feld und wird **arithmetisch** nach *oadr* übertragen.
- Der Inhalt von *oadr* vor der Operation geht verloren.
- Längen beziehen sich auf Bytes.
- Anstelle von *iadr* und *oadr* kann *wadr* definiert werden.
- Anstelle von *iadr* kann eine **gepackte Konstante** definiert werden.

```
1,5,ZA,8,8
6000,5000,ZA
P'0',5100,ZA,1,8
```

Für weitere, detailliertere Informationen konsultieren Sie bitte das Kapitel 7: Die High-Level-Format Instruktion SET.

Hexadezimal Konversion

```
iadr,oadr,CX,l
wadr,wadr
```

- *iadr* wird nach *oadr* in ein **hexadezimal printbares** Format aufbereitet, um es z.B. auszudrucken.
- Die Länge von *oadr* ist zweimal die Länge von *iadr*.
- Längen beziehen sich auf Bytes.

```
'QPAC',5000                    resultiert in:
TO-O1
5000,1,CX,4                    1...5....10...5....20..
PUT-O1                         D8D7C1C3
```

Für weitere, detailliertere Informationen konsultieren Sie bitte das Kapitel 7: Die High-Level-Format Instruktion SET.

Editier Operationen

Benutzerdefinierte Edit Masken

```
iadr,oadr,E'literal'
wadr,wadr
```

- Der Inhalt des gepackten Feldes *iadr* wird nach *oadr* **editiert**, unter Verwendung des Literals als Edit Maske.
- Die Edit Maske bestimmt die Länge des Outputs und muss deshalb die Länge des Inputfeldes *iadr* in Betracht ziehen.
- Jeweils das erste Zeichen in der Edit Maske ist der **Füllcharacter**.
- Ziffernstellen werden mit 9 definiert; Stellen, bei denen Nullen Unterdrückung gewünscht wird, werden mit Z definiert.
- Interpunktionen oder irgendwelche Zeichen können eingefügt werden, wie auch das Minuszeichen (-) am Schluss.

```
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'
```

Benutzerdefinierte Hexadezimal Edit Maske

```
iadr,oadr,EX'literal'
wadr,wadr
```

- Es gelten die gleichen Editierregeln wie unter E.
- Die Maske wird hexadezimal gemäss Assemblerkonventionen definiert.

```
15,10,EX'40F9F9F9F9F9F9'
6000,1,E' 999999'
```

Vordefinierte Edit Masken

```
iadr,oadr,EDA,l1,l2
          EDAZ
          EDAS
wadr,wadr
                    Maskentyp A = ▲▲▲▲▲▲–
```

- Das gepackte Feld *iadr* wird nach *oadr* gemäss Maskentyp A editiert.
- Diese erweiterte Editieroperation ermöglicht das Editieren ohne Interpunktionen.
- Ein negatives gepacktes Feld wird rechtsbündig durch ein Minuszeichen (-) gekennzeichnet.
- **Nullen Unterdrückung** ist möglich, wenn EDAZ oder EDAS definiert wird:
  EDAS ergibt Nullen Unterdrückung **exklusive** letzter Ziffernstelle.
  EDAZ ergibt Nullen Unterdrückung **inklusive** letzter Ziffernstelle.
- *l1* ist die Länge von *iadr* in Bytes, *l2* ist die Anzahl Bytes, die, von rechts betrachtet, ohne Berücksichtigung der Vorzeichenposition, vom aufbereiteten Wert effektiv benötigt wird.
- Wird für *l1* keine Länge angegeben, wird **Länge 8** angenommen.
- Wird *l2* nicht angegeben, wird die Länge gemäss *l1* berechnet.

```
10,65,EDAZ,5,4
```

```
Pos.10        =  X'019376219D'  l1=5
voll editiert =     19376219-
Pos.65        =           219-  l2=4
```

```
iadr,oadr,EDB,l1,l2
wadr,wadr
                    Maskentyp B = ▲▲.▲▲.▲▲.▲▲
```

- Das gepackte Feld *iadr* wird nach *oadr* gemäss Maskentyp B editiert.
- Diese erweiterte Editieroperation ermöglicht das Editieren in Gruppen von 2 dezimalen Stellen mit Punktetrennung.
- Ein negatives gepacktes Feld wird nicht als solches gekennzeichnet.
- Nullen Unterdrückung ist nicht möglich.
- *l1* ist die Länge von *iadr* in Bytes, *l2* ist die Anzahl Bytes, die, von rechts betrachtet, unter Berücksichtigung der Punktationen, vom aufbereiteten Wert effektiv benötigt wird.
- Wird für *l1* keine Länge angegeben, wird ***Länge 8*** angenommen.
- Wird *l2* nicht angegeben, wird die Länge gemäss *l1* berechnet.

```
10,65,EDB,5,8

                    Pos.10        =  X'019376219D' l1=5
                    voll editiert =   19.37.62.19
                    Pos.65        =      37.62.19  l2=8
```

Die Editierregeln für die folgende Edit Maske sind dieselben, wie sie für EDB **beschrieben sind:**

```
iadr,oadr,EDC,l1,l2
wadr,wadr
                    Maskentyp C = ▲▲:▲▲:▲▲:▲▲:▲▲
```

Die Editierregeln für die folgenden Edit Masken sind dieselben, wie sie für EDA **beschrieben sind:**

```
iadr,oadr,EDD,l1,l2
          EDDZ
          EDDS

                    Maskentyp D = ▲▲▲.▲▲▲.▲▲▲,▲▲▲
```

```
iadr,oadr,EDE,l1,l2
          EDEZ
          EDES

                    Maskentyp E = ▲▲▲,▲▲▲,▲▲▲.▲▲–
```

```
iadr,oadr,EDF,l1,l2
          EDFZ
          EDFS
```

Maskentyp F = ▲▲▲'▲▲▲'▲▲▲.▲▲–

```
iadr,oadr,EDG,l1,l2
          EDGZ
          EDGS
```

Maskentyp G = ▲▲▲ ▲▲▲ ▲▲▲–

```
iadr,oadr,EDH,l1,l2
          EDHZ
          EDHS
```

Maskentyp H = ▲▲▲,▲▲▲,▲▲▲–

```
iadr,oadr,EDI,l1,l2
          EDIZ
          EDIS
```

Maskentyp I = ▲▲▲.▲▲▲.▲▲▲–

```
iadr,oadr,EDK,l1,l2
          EDKZ
          EDKS
```

Maskentyp K = ▲▲▲▲▲▲▲▲▲▲,▲▲–

Spezialwert Instruktionen

Grundformat

Zur Initialisierungszeit generiert QPAC **Spezialregister**, deren Inhalt für den Benutzer von Interesse sein kann.

Diese Register können vom Benutzer nicht verändert, sondern nur abgefragt werden. Ihr Inhalt wird durch spezielle Instruktionen zur Verfügung gestellt, die syntaktisch einer einfachen Übertragung gleichkommen.

Anstelle von *iadr* wird der Name des Spezialregisters verwendet. Die Länge wird hierbei implizit durch den Namen des Registers definiert.

Systemdatum (System Date)

```
DATE,oadr
     wadr                          8 Byte Wert
```

Nach Durchführung dieser Operation enthält *oadr* bzw. *wadr* das aktuelle Systemdatum, zur QPAC Startzeit, im folgenden Format:

```
C'DD.MM.YY'
```

Systemzeit (System Time)

```
TIME,oadr
     wadr                          8 Byte Wert
```

Nach Durchführung dieser Operation enthält *oadr* bzw. *wadr* die aktuelle Systemzeit, zur QPAC Startzeit, im folgenden Format:

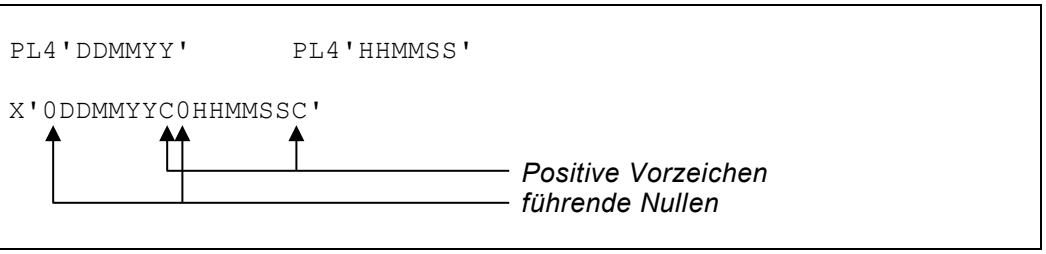
```
C'HH:MM:SS'
```

Für weitere, detailliertere Informationen konsultieren Sie bitte das Kapitel 7: Die High-Level-Format Instruktion SET.

Laufendes Datum/Zeit (Current Date/Time)

```
CDTIME,oadr
       wadr                          2 x 4 Bytes gepackt (8 Bytes)
```

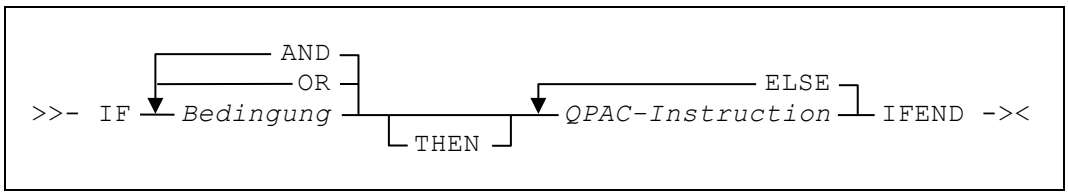
Nach Durchführung dieser Operation enthalten die ersten 4 Bytes das laufende Datum, die zweiten 4 Bytes die laufende Zeit, im folgenden gepackten Format:



Achtung: `CDTIME` ist nur im Basis Instruktionsformat gültig!

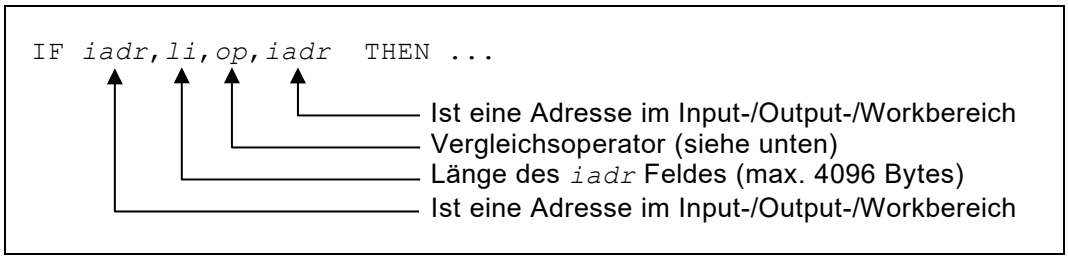
Die IF THEN ELSE Instruktion (Basis Format)

Grundformat der Bedingungsinstruktion



Für weitere, detailliertere Informationen konsultieren Sie bitte das Kapitel 8: Die Ablaufsteuerungs-Befehle**.**

Format 1 - Logischer Vergleich (CLC)

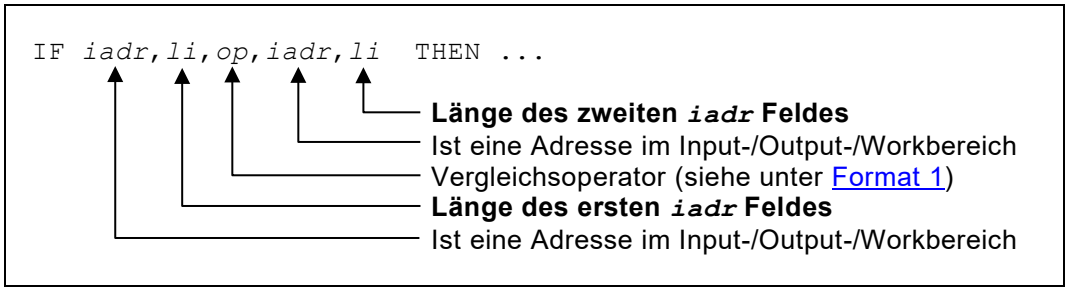


- Gültige Vergleichsoperatoren:

EQ	= gleich (equal)
NE	= ungleich (not equal)
GT	= grösser als (greater than)
LT	= kleiner als (less than)
GE	= grösser oder gleich (greater or equal)
LE	= kleiner oder gleich (less or equal)

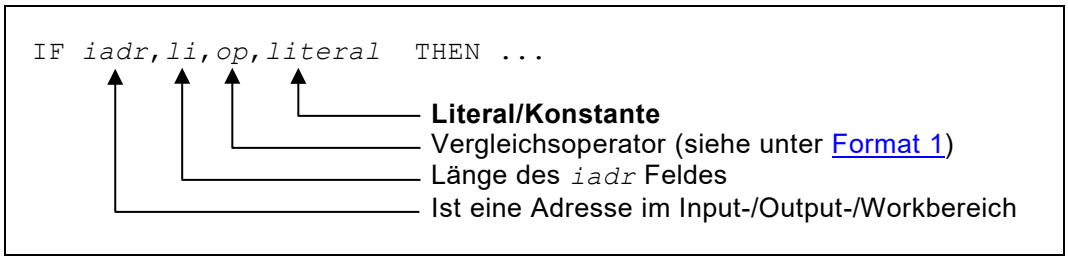
- Anstelle von *iadr* kann jederzeit auch eine *wadr* oder eine *oadr* angegeben werden.

Format 2 - Arithmetischer Vergleich von Gepackten Feldern (CP)



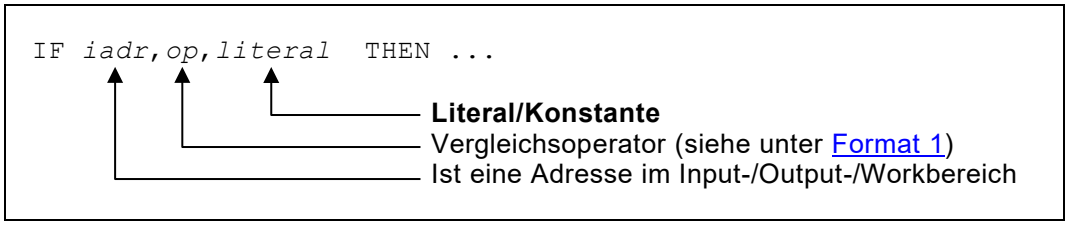
- Wenn eine **zweite Länge zusätzlich** zur zweiten *iadr* **definiert** wird, wird automatisch angenommen, es handle sich um einen **arithmetischen Vergleich von gepackten Feldern**.

Format 3 - Vergleich mit Konstanten (mit angegebener Länge)



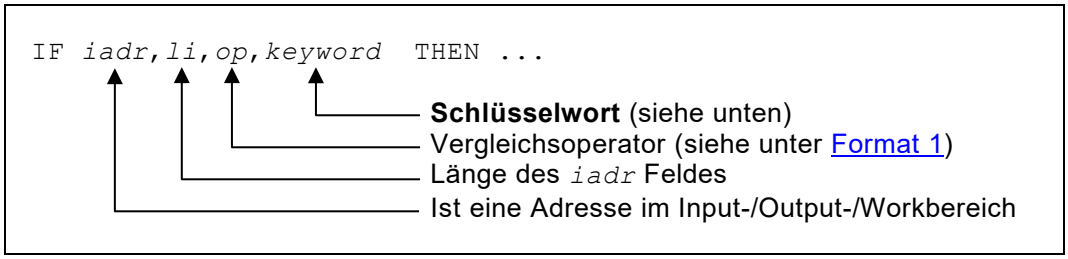
- Der zweite Vergleichsoperand kann als **Literal** definiert werden.
- Die angegebene Länge muss mit derjenigen des Literals übereinstimmen.
- Im Falle eines gepackten Literals wird die Länge wie gewünscht angegeben.

Format 4 - Vergleich mit Konstanten (ohne angegebene Länge)



- Handelt es sich beim Literal um eine Character- (C) oder Hexadezimalkonstante (X), kann die erste Längenangabe und deren Positionskomma weggelassen werden. Wird die Feldlänge angegeben, muss sie mit der resultierenden Bytelänge der Konstante übereinstimmen.
- Bei gepackten Literals muss die Feldlänge definiert sein, andernfalls wird für die Länge des *iadr* Feldes 8 angenommen.
  Bei Fullword Literals wird als Feldlänge 4 Bytes angenommen.
- Der Vergleich erfolgt logisch oder arithmetisch nach der Regel, wie unter Format 3 angegeben.

Format 5 - Logischer Vergleich mit Schlüsselwort



- Gültige Schlüsselwörter sind:

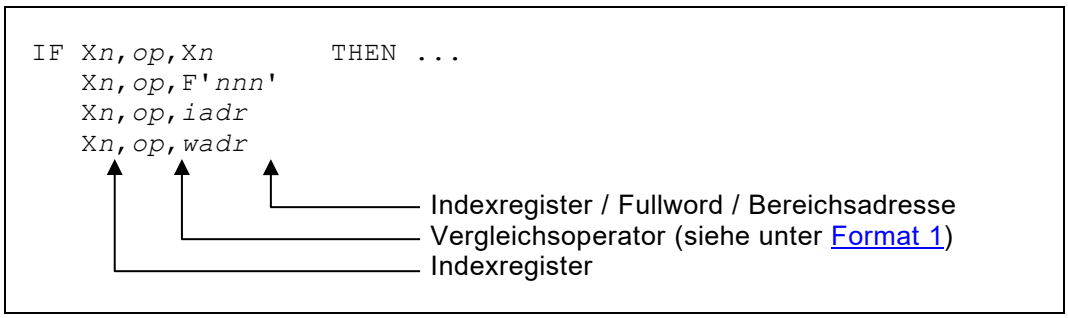
NUMERIC	= Zoned-Decimal Format
SPACE	= Blank
ZERO	= Zoned-Decimal Format

- In jedem Falle handelt es sich bei diesen Vergleichen um Zoned-Decimal- bzw. Characterfelder. Wird die Länge nicht angegeben, wird 1 Byte als Länge angenommen. Maximale Länge ist 4095 Bytes.

PACKED	= gepacktes Format

- Es wird auf korrekten gepackten Feldinhalt geprüft.
  Bei fehlender Längenangabe werden 8 Bytes angenommen.

Format 6 - Binärer Arithmetischer Vergleich

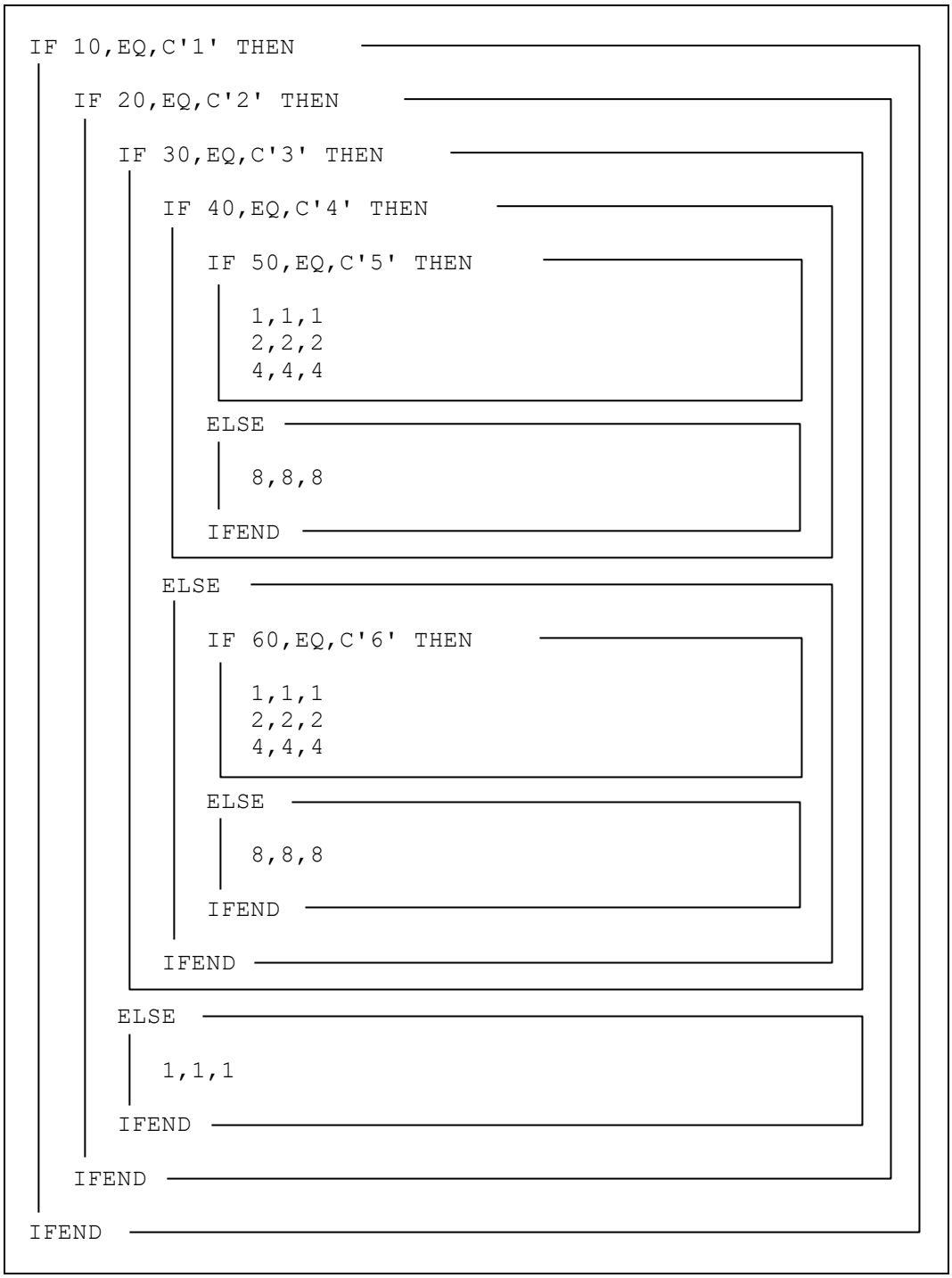


- **Indexregister** können auf ihren Inhalt abgefragt werden, indem mit einem anderen Indexregister, einem Fullword, einer Fullword Konstante, einer Workbereichadresse oder einem Input-/Outputfeld verglichen wird.

- Als Länge werden grundsätzlich **4 Bytes** angenommen, weshalb diese nicht explizit angegeben werden sollte.

Für weitere, detailliertere Informationen konsultieren Sie bitte das Kapitel 8: Die Ablaufsteuerungs-Befehle.

```
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
```



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