COMPLETE PROGRAM GENERATOR
User Manual

ND-60.219.1 EN

COMPLETE PROGRAM GENERATOR
User Manual

ND-60.219.1 EN

The information in this manual is subject to change without notice. Norsk Data A.S assumes no responsibility for any errors that may appear in this manual, or for the use or reliability of its software on equipment that is not furnished or supported by Norsk Data A.S.

Copyright © 1987 by Norsk Data A.S

Version 1 April 1987

Send all documentation requests to:

Norsk Data A.S
Graphics Center
P.O.Box 25 Bogerud
N-0621 Oslo 6
NORWAY

TABLE OF CONTENTS

Section | Page

Section Page

Section Page

APPENDIX

PREFACE

The product

This manual describes the Complete Program Generator, which is a 4th generation development tool.

Complete Program Generator, from now on called Complete-PG, is a part of ABM (Application Building and Maintenance). Complete-PG is an effective tool for program development and maintenance of computer systems.

Complete-PG generates error free and efficient FORTRAN and/or COBOL programs. The programs control an interactive dialogue between the user and the screen, and run as ordinary background programs.

Complete-PG is well suited for large data volumes and many concurrent users.

Complete-PG is registered with product numbers:
ND-211108 for ND-100
ND-211109 for ND-500

The reader

The manual is written mainly for programmers.

The manual

The manual gives an introduction to using the program generator Complete-PG, and explains requirements for the database structure and screen picture design.

The manual contains information on where to put additional code in the program generated code.

Various routines and program variables that can be used in addition to the automatically generated routines are also described.

Prerequisite knowledge

To use Complete-PG, it is necessary to have some knowledge about basic software on ND computers. It is assumed that the reader knows the SINTRAN operating system and the SIBAS database system, in addition to the programming languages FORTRAN or COBOL.

CHAPTER 1

DESCRIPTION OF COMPLETE-PG

1. DESCRIPTION OF COMPLETE-PG

Generation of programs

The program generator, which is a part of ABM, is an efficient tool during program development. It produces executable FORTRAN or COBOL code for screen functions.

> NOTE:
> Programs generated by Complete-PG are used to manage an interactive dialogue between the user and the screen picture.

Start

Start Complete-PG by giving the command 'Complete-PG' from the command line in ABM:

ABM command: COMPLETE-PG↵

Simpler maintenance

In addition to making the programming easier, Complete-PG also makes the maintenance work significantly easier.

When you want to modify a program, you first make the necessary changes using ABM. Then simply change the screen picture and generate the program again.

You use only two screen pictures to give commands and parameter values to Complete-PG.

When you have finished giving your input, the generation of the program can start. Depending on the input, one or more of the following files are generated:

- the source version of the program
- the binary code of the program
- program file

You then start the program by typing the name of the program as a SINTRAN command.

Complete-PG's starting point is a defined 'subschema' and 'subfunction', and requires that the screen picture be defined in ABM.

In addition to the rules that apply for definition of screen pictures in ABM, Complete-PG also places requirements on the design of the screen picture. This is described in chapter 3.

Programs generated by Complete-PG are short and compact. Most of the program code is built into standardized subroutines. These are described in chapter 7.

For complex functions it may be necessary to put in code manually, ie. code in addition to the code generated by Complete-PG. How to do this is described in chapter 5.

CHAPTER 2

DATABASE STRUCTURE REQUIREMENTS

2. DATABASE STRUCTURE REQUIREMENTS

When you want to use Complete-PG for program development, you must consider the following points when structuring the database:

- 'Sets' will not be treated automatically. Some additional programming is necessary.

- All realms that are to be maintained automatically by applications generated by Complete-PG must have at least one unique key.

  All realms that have records against which you want to use free text, must contain one free text item. (See page 20.)

  This item must be called:

  XXTNR in FORTRAN where XX = realm prefix.
  TNR in COBOL.

  The item must be defined as INTEGER*4, and be 9 character positions long.

  In addition, the realm D3TEXT must be defined in the database. This is the realm where the free text records are to be stored.

  A redefinition file for installing the realm D3TEXT in the user base is included when ABM with Complete-PG is delivered.

CHAPTER 3

SCREEN PICTURE LAYOUT REQUIREMENTS

3. SCREEN PICTURE LAYOUT REQUIREMENTS

3.1. THE SCREEN PICTURE

All communication between the user and the program is via screen pictures.

The screen picture of programs to be generated by Complete-PG must have a standard layout.

A screen picture consists of the following parts:

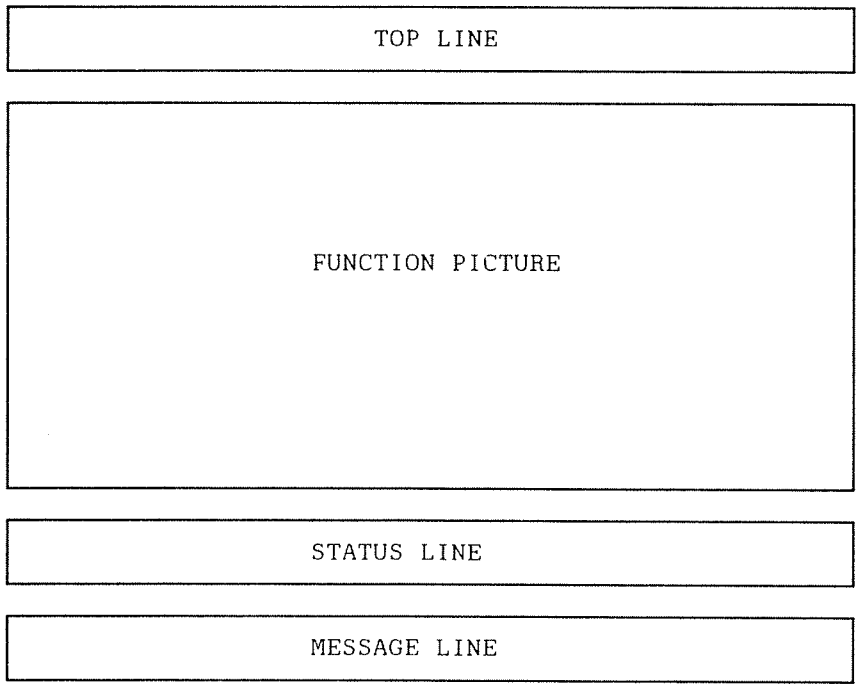


Fig. 3.1 - The layout of a screen picture.

The top line, the status line and the message line are generated by Complete-PG. The function picture, however, is defined by the user in the screen picture part of ABM.

On the next pages you will find a description of the various parts of the screen picture.

3.1.1. THE TOP LINE

The top line is the first line in all screen pictures, and it is generated automatically. It is not possible to edit this line when you define the function picture.

The top line consists of the following fields:

command field	system name	function name	date and time

Fig. 3.2 - Top line.

Command field

The first field in the top line is a command field that can be used in parallel with function keys to give commands. It is four characters long.

System name

System name is connected to your particular system, and must be specified in the file CP-SPEC:SYMB. This is described in chapter 9.

The system name is displayed automatically from position 6 in the top line. It can be up to 20 characters long.

Date and time

This field is used to display the date and time when the picture is written on the screen.

The generated program will automatically retrieve and display date and time from position 65 in the top line.

Function name

What this field is to contain, is optional.

The field is unique for each picture, and may for example contain the function name the user has defined in ABM. (In the FOCUS library, there is a subroutine which prints a text in a specified position in the picture. Use this subroutine if you want to display extra information in the picture.)

3.1.2. THE FUNCTION PICTURE

The function picture is the part of the screen picture the user defines/draws using the screen definition part of ABM.

Note that line two is the first line you can use to make your own screen picture. (The first line is reserved for the top line.)

When the program is generated, you use the function picture to enter, retrieve and modify data.

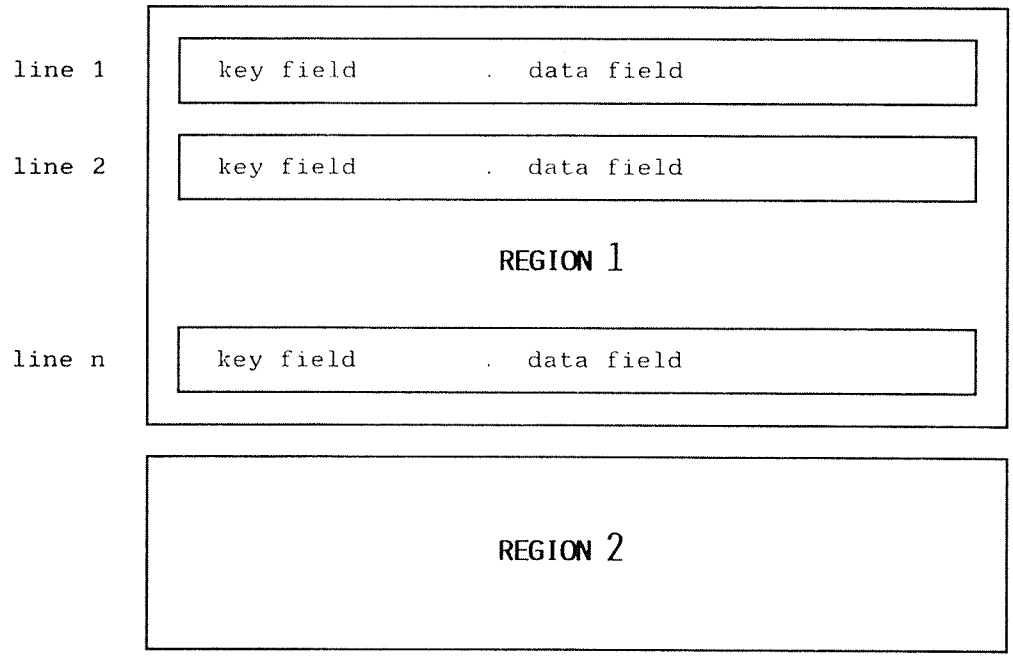


Fig. 3.3 - The function picture.

Region — A function picture consists of one or two regions.

Logical line — Each region may consist of one or more logical lines.

Physical line — A logical line may consist of one or more physical lines on the screen.

> NOTE:
> From now on, we will us 'line' to refer to 'logical line'.

Record

What we here call a line, is called a 'record' in the ABM manual, under the description of screen pictures.

Key field
Data field
Search key

A line consists of a key field and a data field.
A key field is a field which corresponds to search keys in the database. For each search key there may be one or more key fields. The values you enter here are used to search for data in the database. The data that is found, is then printed in the data fields on the screen.

Unique main key

One or more search keys may be connected to a line. One key must be defined as main key, and this must be unique in the database.

Data set

A data set consists of all data that can be displayed on a line, both in key fields and data fields.

Page

A page is defined as all data that can be displayed in a region, i.e. all data sets in a region.

Owner region
Member region

Data sets in two regions may 'belong together'. If two regions 'belong together', one of them is the owner region and the other a member region. An owner region can only contain one line.

Several lines

It is simple to define many lines in a region, using the screen picture part of ABM. It is only necessary to define the first line in the region completely. Then copy as many lines as you want using the COPY key. A new field name is generated automatically for the first field in each line that is copied. The field name will be different for each line.

When defining a field in the screen picture, you refer directly to the corresponding item in the database. If a database-item is not used, you must refer to a data type (defined in the data-description menu in ABM).

Key field requirements

For a field to be understood as a key field, the following requirements must be met:

- the element or group element the field refers to, must be specified as search key ('K') when creating a subschema in ABM.
- when generating a program in Complete-PG, you must specify that a field is to be a key field.

It is possible to define key fields for different search keys on the same line.

If you are to retrieve or check data against other registers, the search key for this data must be specified in subschema and Complete-PG.

3.1.3. STATUS LINE AND MESSAGE LINE

At the bottom of the screen picture are the status line and the message line. Like the top line, these are also written out automatically by Complete-PG. No editing is possible.

Status line

The status line comes right under the function picture, and shows the name of the current region (the region you work in at the moment). The status line also shows what you are doing with the data (what command you have given), and various other information.

Message line

The message line is the bottom line in the picture. This is where messages from the system/function are displayed.

There are two types of messages:

- Error messages
- Informative messages

Queue system for messages

If a message is longer than a line, it is put into a queue system. The same happens if there are several messages to be displayed in a series. By pressing any key, the next message will be displayed on the screen. In this way you are able to read all messages before the next one is printed.

3.1.4. FIELD TERMINATION

When a character is typed in the last position of a field in the screen picture, you may decide whether you want the cursor to remain in the field until CR is pressed, or you want it to move to the next field. The latter is done in the following way:

On the work-user area there is a file with various parameters for your system. The name of this file is CP-SPEC:SYMB. Read this file into an editor. You will see that one of the defined parameters is called NEXTFI. You may set NEXTFI to Y or N.

^DEF,NEXTFI,<Y>; causes the cursor to move to the next field when a character is typed in the last position of the current field.

^DEF,NEXTFI,<N>; causes the cursor to remain in the last position of the field until CR is pressed.

3.2. BK FIELD AND OK FIELD

Optional

You yourself decide whether you want a treatment code field (BK field) and/or OK field in the regions in a screen picture.

If you define BK fields and OK field in a picture, you must make sure they refer to data types (defined in data description in ABM) with fixed names:

Define with fixed names

OKCOD or OKCD1 for OK fields (COBOL/FORTRAN)
BKODE or BKOD1 for BK fields (COBOL/FORTRAN)

Both fields are of type x(1), i.e. one alphanumeric character.

For BK codes, it is optional what letters or numbers are presented to the end user. Decide this when installing Complete-PG. (See chapter 9.)

In this manual the letters Q, S, M and D are used as treatment codes for querying, storing, modifying and deleting.

When generating a program, you decide what BK codes will be allowed. All combinations of Q, S, M and D are possible. This is described in chapter 4.

3.3. TEXT FIELD

The free text function makes it possible to connect an unlimited amount of free text to a record.

The free text function requires that you have defined a field of type TTYPE on the line in the screen picture from where you call the free text function.

TTYPE is a data type with the format PIC X(1). This field is used only for output.

When a line has text connected to it, a 'T' is displayed in the TTYPE field on the screen. The field is empty if the line has no text connected to it.

If you want to read more about the free text function, see page 177.

CHAPTER 4

GENERATING PROGRAMS WITH COMPLETE-PG

4. GENERATING PROGRAMS WITH COMPLETE-PG

4.1. STARTING COMPLETE-PG

In order to start Complete-PG, the ABM database must be in 'running' state.

Start Complete-PG by giving the command 'Complete-PG' from the command line in ABM:

ABM command: COMPLETE-PG↵

You will then be shown the first screen picture in Complete-PG.

4.2. THE DIFFERENT SCREEN PICTURES IN COMPLETE-PG

Complete-PG consists of the two screen pictures PROGRAM DESCRIPTION and USE OF PROGRAM KEYS. By entering data into these pictures you decide how the program will work after being generated. It is therefore very important that these pictures are filled in correctly.

You have to fill in both pictures before Complete-PG can generate the program you want.

The two screen pictures are on two different 'levels'. See figure 4.1 on the next page. There may be several USE OF PROGRAM KEYS pictures in connection with one PROGRAM DESCRIPTION picture.

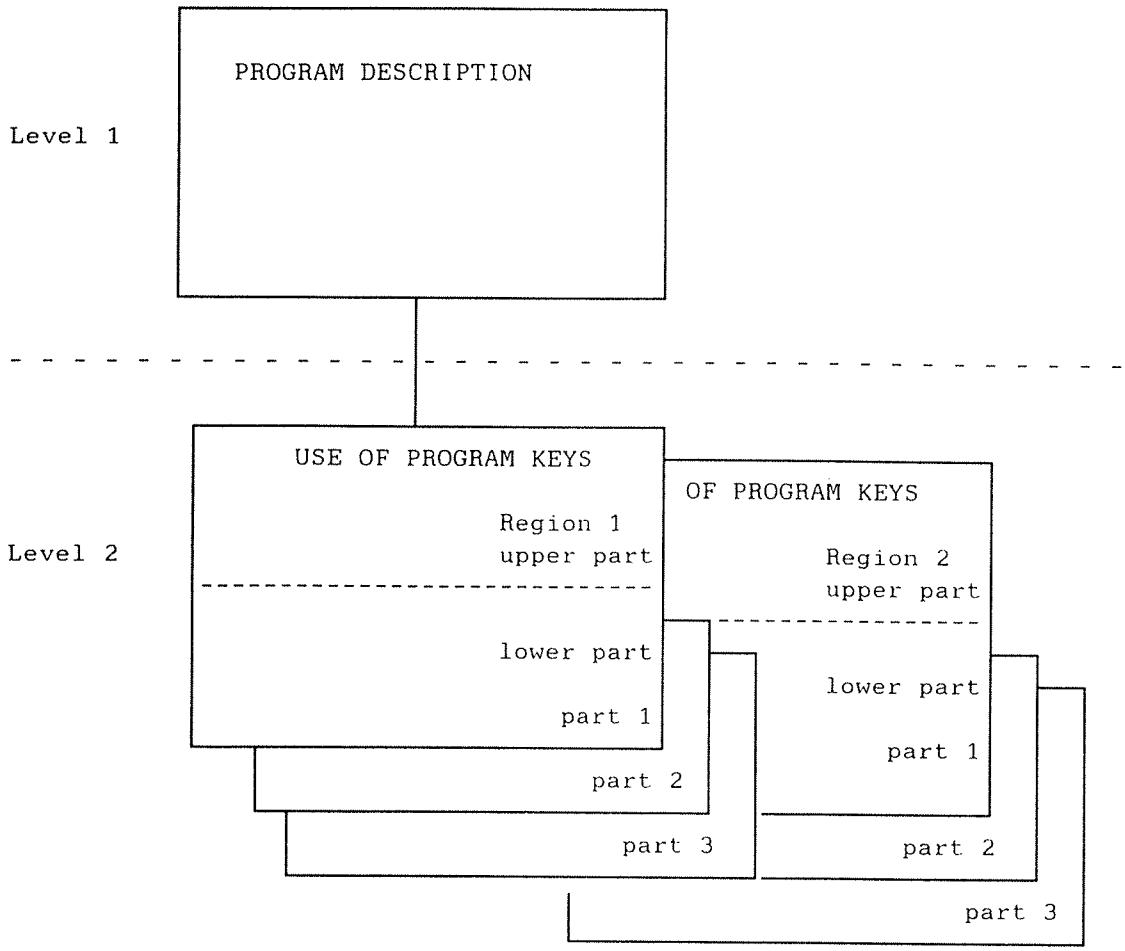


Fig. 4.1 - The screen pictures in Complete-PG are on two levels.

Navigating

You move from the first picture, PROGRAM DESCRIPTION, to the next one, USE OF PROGRAM KEYS, by

- hitting the '<>'- key in the command field.

You return to the picture PROGRAM DESCRIPTION by

- giving the command 'E' in the command field in the picture USE OF PROGRAM KEYS.

HELP

You can always ask for HELP, no matter where you are in the screen picture. You will then get a help picture with information concerning what you are doing.

4.3. THE SCREEN PICTURE PROGRAM DESCRIPTION

After you have given the command Complete-PG on the command line, you will get the first screen picture to be filled in:

```
PG > .          P R O G R A M   D E S C R I P T I O N

Program identification.
 subfunction  : ........  subschema: ........  form: ........
 author       : ............................................................
 program id   : ..........
 explanation  : ............................................................
              : ............................................................
              : ............................................................


Parameters for generate.
 object language : ....
 object filename : ...................................................
 load procedure  : ...................................................


Date of
   creation      : ........          last modification : ........
                                     last generated   : ........
```

The picture contains the fields SUBFUNCTION, SUBSCHEMA and FORM, which refer to the SUBFUNCTION, SUBSCHEMA and FORM in ABM.

In this picture you state the programming language you want the program to be generated in, the name of the file where the program is to be stored, as well as the generating procedure you want to use.

On the following pages you will find a more detailed description of the fields in this picture.

4.3.1. DESCRIPTION OF THE FIELDS IN THE SCREEN PICTURE PROGRAM DESCRIPTION

PG > .

This is the command field in the picture.

Choose between these commands:

↵: stores a new record.
F : finds the first record.
L : finds the last record.
N : finds the next record.
P : finds the previous record.
S : sets/deletes the search region.
When 'set' search region is used, the cursor will be placed in the SUBFUNCTION field. Type in the lower limit. Afterwards you will be asked for the upper limit. These limits will be deleted the next time 'S' is given in the command field.
Q : clears the screen picture.
C : copies the screen picture to a file. The system will ask for the name of the file (output file name:):
M : modifies the current record.
D : deletes the current record.
<>: moves to the next screen picture (USE OF PROGRAM KEYS).
X : executes the command(s) in the field load procedure.
E : exits, returns to the ABM command line.

The commands F, L, N, P and S search for program descriptions that already have been generated by Complete-PG.

Subfunction

The name of the subfunction defined in ABM which corresponds to the function to be generated.

Subschema

The name of the subschema defined in ABM which is connected to the subfunction above. The name is fetched automatically after the subfunction has been retrieved, and cannot be changed.

Form

The name of the form belonging to the subfunction. This is fetched from the form file (specified in the file CP-SPEC:SYMB) and is displayed in this field. The name of the form cannot be changed here.

Author	The name of the person to be known as the originator of the program.
Program id	The name of the generated program. If you want to call the generated program as a subroutine from another program, you have to use this program id as the subroutine name.
Explanation	Space reserved for a short description of the program to be generated.
Object language	The programming language that the generated code will be written in. Choose between 'FORT' (FORTRAN) and 'COBL' (COBOL).
Object filename	The name of the file where the generated program will be stored. This will also be the name of the BRF/NRF file and any PROG files or domains.
Load procedure	These are the three alternatives: GENERATE : Generates program code for the function. COMPILE : Compiles the generated program. LOAD : Loads the necessary files. These possibilities may be combined, for example GENERATE/COMPILE. GENERATE/COMPILE/LOAD is the default value. The contents of this field determine what is to be executed when you give the command 'X' in the command field.
Date of creation	The time of the first generation of the program. This date is displayed automatically in the field by Complete-PG.
Last modification	The time when the last change was made to the function description. Generated by Complete-PG.
Last generated	The time of the last generation of program code. Generated automatically by Complete-PG.

4.4. THE SCREEN PICTURE USE OF PROGRAM KEYS

In this screen picture you establish the search keys for the program which is to be generated.

>< <>

The picture below will be displayed on the screen when you hit the '<>' key in the command field.

```
PG >.        U S E   O F   P R O G R A M   K E Y S

Subfunction: ........        fieldrecord: ...
      name: ............... owner: ........
      okcode: .              textfunction: .     action codes: ....

Realm    Key      Use D Ex  Realm    Key      Use D Ex  Realm    Key      Use D Ex
........ ........ .. . ..  ........ ........ .. . ..  ........ ........ .. . ..
........ ........ .. . ..  ........ ........ .. . ..  ........ ........ .. . ..
........ ........ .. . ..  ........ ........ .. . ..  ........ ........ .. . ..
........ ........ .. . ..  ........ ........ .. . ..  ........ ........ .. . ..

Initial values for realm: ........  key: ........

 Item     Lowlimit     Highlimit      Item     Lowlimit     Highlimit
 ........ ............ ............   ........ ............ ............
 ........ ............ ............   ........ ............ ............
 ........ ............ ............   ........ ............ ............
 ........ ............ ............   ........ ............ ............
```

4.4.1. DESCRIPTION OF THE FIELDS IN THE SCREEN PICTURE USE OF PROGRAM KEYS

PG > . This is the command field in the picture.

Choose between these commands:

F : finds the first record.
L : finds the last record.
N : finds the next record.
P : finds the previous record.
S : shifts between upper and lower parts of picture.
C : copies the screen picture to a file. The system will ask for the name of the file (output file name:).
M : modifies the current record.
E : returns to the previous picture (PROGRAM DESCRIPTION).

These commands are also valid for the other menus in ABM, apart from the command move (shift) between the upper and lower parts of a picture.

> NOTE:
> The screen picture USE OF PROGRAM KEYS consists of an upper and lower part. All commands affect the part of the picture where you are at the moment.

Use the command S to move the cursor between the upper and lower part of the picture.

The commands F, L, N and P have to do with the fact that several regions in a screen picture are connected to the program to be generated.

As shown in figure 4.2, several levels of the lower part of the screen picture may be connected to the same region.

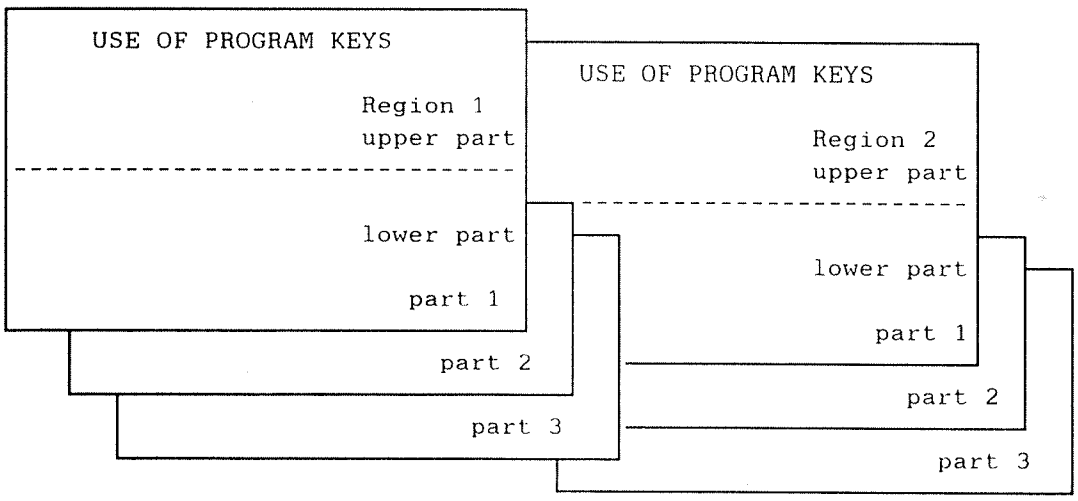


Fig. 4.2 - An example of the composition of regions.

An example of how to use the command N:

Let us look at an example of how you can move both horizontally and vertically in figure 4.2.

If you want to move horizontally from Region 1 to Region 2, you do the following:

- Make sure that the upper part of the screen picture in Region 1 is the current work area (by hitting S if necessary).
- Give the command N in the command field.

On the screen you will now see the screen picture for Region 2.

On the other hand, if you want to move vertically in the figure, from part 1 to 2 within the same region, you do the following:

- Make sure that the lower part (part 1) of the picture is the current work area, by hitting S.
- Give the command N.

The upper part of the picture remains unchanged, but you get a new picture (called part 2 in the figure) on the lower part of the screen.

- Give the command M to move the cursor to the lower part of the screen (the current work area).

Subfunction

Refers to a generated subfunction in ABM. This will be displayed automatically on the screen, and cannot be changed.

Fieldrecord

Record name that is generated by the screen picture part of ABM, and which defines the region uniquely. (An example of a record name is 'R1A'.)

Name

The name of the region where you are working at the moment. The name is shown on the status line in the function picture. If you move between the regions, the name on the status line changes accordingly.

Owner

The owner of the region to be defined.

So far there is fixed dependency between the first and next region. This field has therefore no importance for the time being.

OKCODE

If there is an OK field in the screen picture generated by ABM, a 'Y' will be displayed in this field. You will not be able to change this without removing/inserting the OK field in your screen picture.

Textfunction

This field gets the value 'T' if free text is used in the function; that is, if the data type TTYPE is used in the picture. (See page 20.) You will not be able to change this without removing/inserting the text field in your screen picture.

Action codes

A combination of legal access codes.

All combinations are possible.

Legal values are:

1 : query
2 : registration
3 : modification
4 : deletion

These may be different in the different regions.

Realm

The name of the register that the search key (shown in the key field) belongs to.

Key

This is where all the keys (indexes) marked with a 'K' in a subschema are listed.

These search keys may be used in different ways in the program:

- There must be only one search key that is a main key in the main register in the region.
- Some keys can be alternative search keys in the main register.
- Some search keys can be used to fetch data from registers other than the main register.
- Some search keys can be used for existence control towards other registers (i.e. to look up in other registers and see whether certain data is to be found there).

You decide how to use a search key by filling in the fields 'Use', 'D' and 'Ex'.

For each of the search keys you want to use in the region, you also have to fill in the lower part of the screen picture. Here you specify which limits (low limit and high limit) are to be used for looking up and searching with the current key in the register.

Use

This field can have three different values: MK, AK and K.

MK : main key.

The main key belonging to the main register in the region is marked MK.

> NOTE:
> Specify only one MK per region. If you want to update the main register, the main key must be unique within the entire database.

When new data is being registered, the MK is checked to find out whether the record already exists.

When modification or deletion is taking place, the record is retrieved by means of the MK, and the data belonging to the record is displayed in the screen picture.

When a query is being performed, the record in question is retrieved and displayed on the screen. If the record does not exist, and the high limit (see page 33) is set to space, the next record in the search region will be fetched and displayed. If low and high limit are equal, and the record in question is not found, a message will be given.

In order to search in the main register with certain key values, the lower part of the screen picture for this key has to be filled in correctly. (Low limit and high limit.)

AK or **A1, A2,.....A7**: alternative key.

If you want to use alternative search keys to search through the main register, these should be marked AK.

If you have many keys of this kind in the program, and would like to number them in order that they be input in a certain sequence, you mark them with an 'A' and a successive numbering from 1 to 7. A1 will be read from the screen before A2, etc.

> NOTE:
> Alternative keys are only valid for querying.

Because alternative keys are only valid for querying, these keys may be non-unique.

K : search key towards a register other than the main register where data is fetched from and/or checked.

The fields D and Ex are used to show what is to be performed on the registers.

D(isplay)

This field may have one of two values: blank or D.

BLANK : data will not be fetched from another register.

D : data will be fetched from another realm by means of this search key.

Retrieved data will be transferred to a screen buffer and then displayed. If no data is found, and the action code is modification or registration, an error message will be displayed. The values have to be input from the screen.

> NOTE:
> If data is to be displayed on the screen, the name of the item in the database must be the same as the name of the field in the screen picture (See the ABM manuel, DDTRNSF/DDTRNSC).
> Items which are to be retrieved must be marked in the subschema.

Ex(istence control)

This field may have one of three values: blank, M or E.

BLANK: No control towards another register.

M : Existence check against another register when storing new data or when modifying data.

E : Leads to existence check against another register when deleting records from the main register.

Initial values for realm

The register where the search key is to be found. Fetched automatically from the ABM database.

Key

The search key which is to be given a start value and/or a stop value. This value is read into the field automatically.

Item

Search keys specified under KEY are listed here. If the search key consists of a group key, all the elements in the group will be listed. A new record is generated for each search key, which makes it possible to navigate between them (F, L, N, P).

Lowlimit/ highlimit

These fields are used to set up search regions and retrieve requested data from the database registers.

In the further explanation of the fields in this screen picture, we give a few examples of pictures that are already filled in. The examples are fetched from chapter 10.

EXAMPLE 1

We want to make a program to maintain a register of employees. The register is called EMPLOYEE, and contains among other things:

EMPNO - employee's number
EMPNAME - employee's name
EMPDEPT - department
EMPPOS - position
EMPADDR - address
EMPPOST - postal code
EMPPHON - telephone number
EMPADM - place of administration

Employee's number is a unique key in the employee register.

The program should find the data for one particular employee when you type in the employee's number in the picture. It should also be possible to navigate forwards and backwards in the register (see 4.7).
Employee's number is a field in the screen picture. The name of the generated screen record in the screen picture is R1.

Filling in the screen picture:

```
  -
Realm    Key     Use   D   Ex     - - - - - - -
EMPLOYEE EMPNO   MK
  -
  -
-------------------------------------------------
Initial values for realm:        key: EMPNO
Item     Lowlimit   Highlimit     - - - - - - -
EMPNO    R1PNO                    FORTRAN example
--------------------------------
EMPNO    R1-EMPNO                 COBOL   example
  -
```

refers to the field in the screen picture from where the key value is to be fetched when searching.

NOTE:
For COBOL, specify which screen record the key item belongs to, in addition to the name of the key item itself.

EXAMPLE 2

We use the same register as in example number 1. The program still maintains information about employees, but now each department maintains information about its own employees. Users will only be allowed to retrieve information about employees within their own department.

The group key EMPK1 = EMPADM + EMPNO is used as the main key.

When users log on to the system, they are asked for the department they work in. The program fetches this information from the logging-on system, and moves this information to the program variable INADM.

Filling out the screen picture:

```
  -
Realm    Key     Use   D   Ex     -  -  -  -  -  -  -
EMPLOYEE EMPK1    MK
  -
  -
  -
-------------------------------------------------------
Initial values for realm:      key: EMPK1
Item     Lowlimit   Highlimit    -  -  -  -  -  -  -
EMPADM   INADM      INADM
EMPNO    R1PNO               FORTRAN example
---------------------------
EMPADM   INADM      INADM
EMPNO    R1-EMPNO            COBOL   example
  -
```

EXAMPLE 3

The program now maintains information both about the departments and about the employees within each department.

The screen picture consists of two regions. Region 1 contains the department number and various other information fetched from the department register.

Region 2 in the picture consists of a list of lines containing employee numbers and information about the employees.

The program will manage to navigate through the department register, and for each department all the employees will be listed in region 2.

The employee register has the group key:
EMPK2 = EMPDEPT + EMPNO

Filling in region 2 in the screen picture:

```
  -
Realm    Key     Use   D   Ex     - - - - - - - -
EMPLOYEE EMPK2   MK
  -
  -
  -
-------------------------------------------------
Initial values for realm:      key: EMPK2
Item     Lowlimit   Highlimit    - - - - - - - -
EMPDEPT  R1PDEPT    R1PDEPT
EMPNO    R2PNO                   FORTRAN example
-----------------------------
EMPDEPT  R1-EMPDEPT R1-EMPDEPT   COBOL   example
EMPNO    R2-EMPNO
  -
```

If you do not want to navigate in the main register by means of a main key, then low limit and high limit have to be filled in with the same variable/value for all the items in the main key.

Note that the first part of the group key comes from region 1, whereas the second part is fetched from region 2.

When the high limit for EMPNO is blank, all of the employees belonging to a certain department are listed.

EXAMPLE 4

We now want to introduce employee's name, EMPNAME, and position, EMPPOS, as keys in the employee realm. Instead of typing in the employee's number you will now be able to type in the employee's name and get a list of information on the employee. We also wish to navigate alphabetically between names in the employee realm.

We also want to have the possibility of finding an employee by typing in the position, that way using the position title to navigate through the realm.

Filling in the screen picture:

```
  -
  -
Realm    Key       Use   D   Ex   -  -  -  -  -  -  -
EMPLOYEE EMPNO     MK
EMPLOYEE EMPPOS    A2
EMPLOYEE EMPNAME   A1
  -
-------------------------------------------------------
Initial values for realm:        key: EMPNAME
Item     Lowlimit   Highlimit     -  -  -  -  -  -  -
EMPNAME  R1PNAME                 FORTRAN example
-------------------------------------------------------
EMPNAME  R1-EMPNAME              COBOL   example
.                                                      .
.                                                      .
.                                                      .
Initial values for realm:        key: EMPPOS
Item     Lowlimit   Highlimit     -  -  -  -  -  -  -
EMPPOS   R1PPOS                  FORTRAN example
-------------------------------------------------------
EMPPOS   R1-EMPPOS               COBOL   example
  -
```

EXAMPLE 5

In the employee realm, the postal code is stored, but not the postal address. The postal address is stored on a separate realm, the postal address realm POADR. The key to this realm is the postal code POSTCODE.

When searching in the employee realm, you want to be able to retrieve the postal address corresponding to the current postal code, and display it on the screen.

When modifying and storing employees, you want to be able to retrieve the postal address corresponding to the postal code you have typed in. If this postal code does not exist in the postal address realm, an error message will be displayed, and the postal code has to be retyped.

Filling in the screen picture:

```
  -
  -
Realm    Key       Use   D   Ex   -  -  -  -  -  -  -
EMPLOYEE EMPNO     MK
POADR    POSTCODE  K     D
  -
------------------------------------------------------
Initial values for realm:        key: POSTCODE
Item     Lowlimit   Highlimit     -  -  -  -  -  -  -
POSTCODE R1PPOST    R1PPOST      FORTRAN example
------------------------------------------------------
POSTCODE R1-EMPPOST R1-EMPPOST   COBOL   example
  -
```

the name of the field in the screen picture where you type in the postal code.

EXAMPLE 6

When maintaining the employee realm, you also state the position. All approved positions are stored on a separate realm called POSITION.

When storing and modifying information about employees, the position realm should be checked to see whether the position you have specified is registered there. If the position does not exist, an error message will be displayed, and you have to retype the position.

Filling in the screen picture:

```
  -
  -
Realm      Key       Use   D   Ex  -  -  -  -  -  -  -
EMPLOYEE   EMPNO     MK
POSITION   POS       K     M
  -
-------------------------------------------------------
Initial values for realm:        key: POS
Item     Lowlimit    Highlimit    -  -  -  -  -  -  -
POS      R1PPOS      R1PPOS        FORTRAN example
-------------------------------------------------------
POS      R1-EMPPOS   R1-EMPPOS     COBOL   example
  -
```

EXAMPLE 7

When maintaining the employee realm, it should not be permitted to delete an employee who is participating in a project. We have a separate project realm which contains the numbers of the employees. One of the keys in the realm is

PROK1 = PROEMP (employee's No.) + PRONO (project No.)

If the employee you are trying to delete exists on the project realm, an error message will be displayed, and the employee will not be deleted from the employee realm.

Filling in the screen picture:

```
  -
  -
Realm      Key       Use   D   Ex  -  -  -  -  -  -  -
EMPLOYEE   EMPNO     MK
PROJECT    PROK1     K         E
  -
-------------------------------------------------------
Initial values for realm:        key: PROK1
Item     Lowlimit    Highlimit    -  -  -  -  -  -  -
PROEMP   R1PNO       R1PNO         FORTRAN example
-------------------------------------------------------
PROEMP   R1-EMPNO    R1-EMPNO      COBOL   example
  -
PRONO
```

4.5. WHAT HAPPENS DURING THE GENERATING OF A PROGRAM?

Before you can instruct Complete-PG to generate a program, you have to do the following:

- Make the necessary additions/changes to the files:

  CP-SPEC:SYMB
  CP-PROGEN:MCRO

- Give Complete-PG the necessary information by filling in the pictures PROGRAM DESCRIPTION and USE OF PROGRAM KEYS.

- Start the generating by giving the command 'X' in the picture PROGRAM DESCRIPTION.

The result of the generating is a program that is ready to be executed.

On the following pages, we shall look at what really happens, but a user does not 'see', when a program is generated.

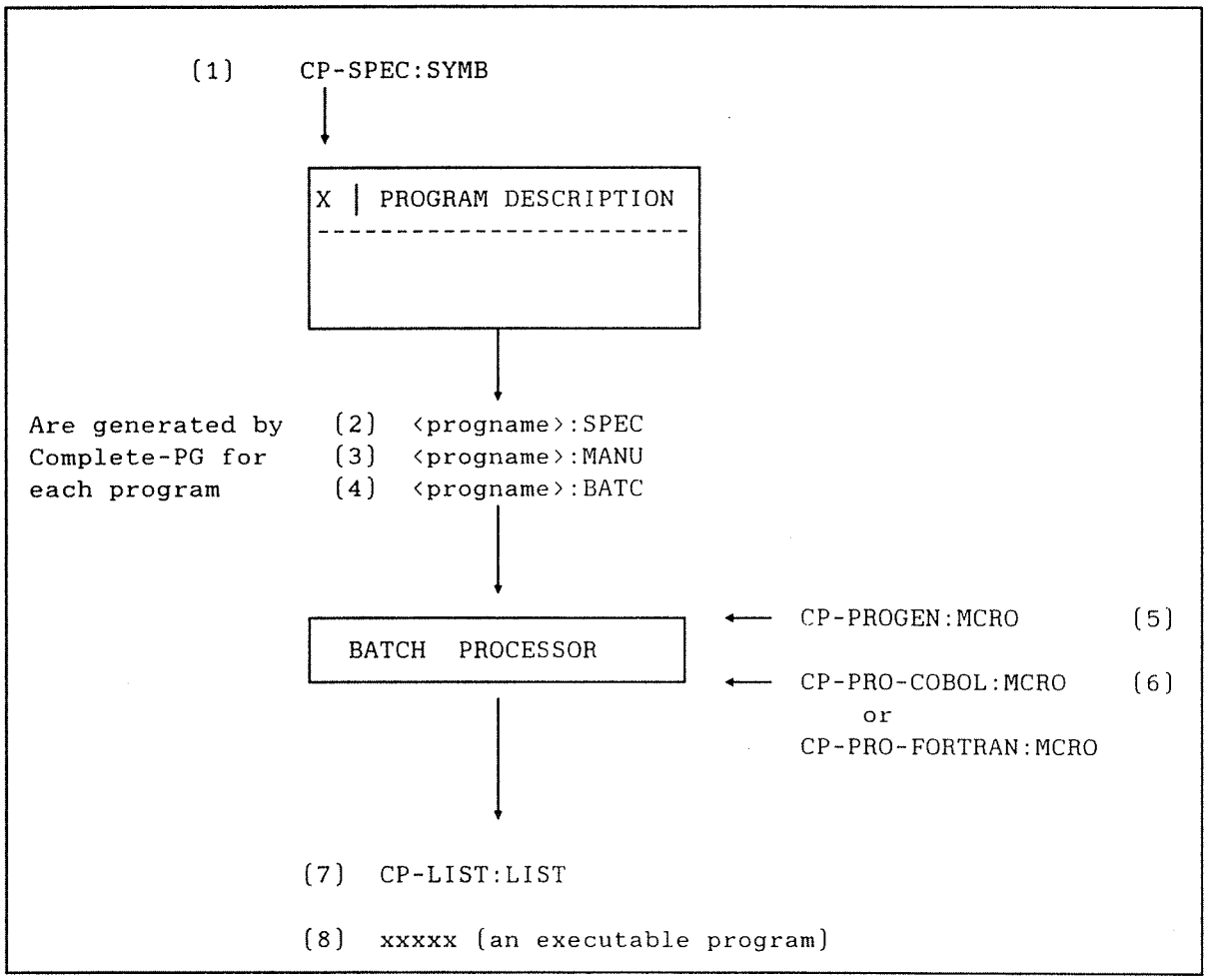


Fig. 4.3 - What happens when a program is generated.

(1) See chapter 9 about adapting CP-SPEC.

(2) This is a preliminary COBOL/FORTRAN program which contains information specific to the program in question.

(3) This file is generated the very first time a program is generated. Later on, when the program is generated again, it may contain additional manual code, i.e. code that the user has programmed herself/himself. This additional code is taken care of and will always be inserted in the correct place in the program, whether the information in the picture USE OF PROGRAM KEYS has been altered or not.

(4) Input file for the batch processor. Contains the name of the macro to be run. This is a job that generates, compiles or loads (or a combination of these options).

(5) Macro to generate, compile or load a generated program. The macro has to be adapted to your own installation. Type in the necessary names of the users and libraries.

(6) General COBOL and FORTRAN routines (macros) which are inserted in the generated program <progname>:SPEC.

(7) This is a list from the job which has been run in the batch processor.

(8) The resulting executable program (if you have specified 'generate/compile/load').

Progname is the name you have typed in in the field 'prog id' in the picture PROGRAM DESCRIPTION.

During the generating of a program, these files are established:

<progname>:BATC
<progname>:SYMB
<progname>:MANU
<progname>:BRF/NRF
<progname>:PROG/DOMAIN

When you start the generating by giving the command 'X', you will be asked whether you want to generate a main program. If you reply 'Y', the following files will be generated, too:

<m-function-name>:SYMB
<m-function-name>:BRF/NRF

Note that Complete-PG only generates subroutines. If you do not have a menu control system, you have to reply 'Y(es)' when you are asked if you want a main program to be generated.

4.6. THE GENERATED PROGRAM - DIALOGUE BETWEEN USER AND SCREEN PICTURE

- Start the generated program by specifying the program name as if it were a SINTRAN command.

The picture will then be displayed, and the cursor will be placed within the first key field in the first region.

There are two different ways of giving commands to the generated program:

- Give commands in the generated command field in the upper left-hand corner of the picture.
- Use the function keys.

Key Command Comment	F1 DELE Delete record	F2 STOR Store a new record	F3	F4 CREG Clear a region
Key Command Comment	F5 CLIN Clear a line	F6	F7 QUER Find data	F8 MODI Modify data

NOTE:
Function keys used as action code can only be used in the command field and in key fields. They have no effect in the other fields.

See description of action codes on page 49.

Querying

Command : **QUER**

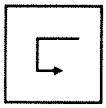
Function key : F7

Description : Find data

1. Hit the function key F7 or the action code you have defined for the command QUER.

   The cursor will be placed in the first field belonging to the main key. This field will be shown in inverse video on the screen.

2. Type in the data you want to search for and press CR, or hit the EXECUTE key.



The value you type in in the key field determines what data will be listed. On the first line, data will be listed for the search key that has the same value as the one typed in, or for the search key that has the closest higher value. On the next lines, the data will be listed and sorted on ascending value of the search key until the page is full (i.e. until data sets are displayed on all the lines in the region) or until there are no more data sets in the search region in the database.

If there is more than one search key for the line(s) in the region, you may move the cursor to the search key you wish to use, and specify the start value for that one instead. The data sets will then be listed on ascending value of the specified search key.

If you ask for data in an owner region, the data set in the owner region will be presented first. Afterwards, the data sets in the member region will be sorted on ascending value of the main key.

Registration

Command : **STOR**

Function key : F2

Description : Store data

1. Hit the function key F2 or type in the action code you have defined for storing of data.

   Fields which are part of the main key are shown in inverse video.

   The cursor will be placed in the first field that is part of the main key.

2. Type in the value of the main key.

   If the value of the main key does not exist in the database already, you can proceed to store data in the other fields.

3. Hit the EXECUTE key, and the data will be stored.

   After the data has been stored, the cursor will be placed in the first field of the next line if there are more lines in the picture. But if this is the last line in the picture, the screen will be cleared, and the last line will be redisplayed on the first line of the picture. The cursor will be placed on line number two.

   You may edit data that is filled in in advance, by first specifying that you want to make a query. The data which is then displayed will remain on the screen until it is modified by overwriting.

   In order to clear the contents of a region while you are storing data, hit the F4 key. If you only want to clear a line, you may use the F5 key.

Modification

Command : **MODI**

Function key : F8

Description : Modify data

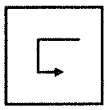
When data is to be modified, the system will search for the record in question. If the record does not exist, you get a message on the screen telling you so.

1. Hit the function key F8 or type in the action code you have defined for modification of data.

   Fields which are part of the main key are shown in inverse video on the screen. The cursor will be placed in the first key field.

2. Move the cursor by means of the arrow keys to the line and the data field where you wish to alter the data.

3. Type on top of the data that is already shown in the field.



4. Hit the EXECUTE key, and the modification will be stored.

   You may also modify data directly, without going via querying. This is done by specifying a legal value for the main key before altering the data field.

Deletion

Command : **DELE**

Function key : F1

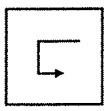
Description : Delete data

When data is to be deleted, the system will search for the record in question. If the record does not exist, you will get a message on the screen.

1. Hit the function key F1 or the corresponding action code.

   Fields that make up the main key, will be shown in inverse video on the screen. The cursor will be placed in the first key field.

2. Move the cursor to the line you want to delete.



3. Hit the EXECUTE key.

   Before the data is deleted, a control determines whether deletion is permitted. (For instance, you cannot delete an 'owner' that has 'members' connected to it.)

   You may also delete data directly without going via querying. This is done by specifying a legal value for the main key before deleting the data.

> NOTE:
> When a line is deleted from the database, it will be shown in low intensity on the screen. This is done to show that the line is deleted, but that you still have the opportunity of cancelling the command (i.e. you may store the line again).

Action codes

Instead of specifying what kind of action is wanted by means of function keys, you have the option of using action codes. In that case, you have to make room for an action code field in the picture when you define it.

In the action code field, you specify what kind of action is to be performed. You may choose among: querying, modification, storing, and deletion.

A letter or a number represents each of the action codes. The letters/numbers are optional, and may vary from one system to another. The action codes are fetched fromn the files:

CP-SMESS-NO-BOO:SYMB for the Norwegian version
CP-SMESS-EN-BOO:SYMB for the English version

A screen picture with action code fields may look like this:

```
Bk  Key     Description
.   .....   ....................................
.   .....   ....................................
.   .....   ....................................
```

In each region, one, several or all action codes may be permitted: query, modify, store and delete. Various combinations may be permitted for the different regions.

4.7. COMMANDS AND FUNCTION KEYS

On the following pages, you will find an overview of the various function keys together with their meaning, as well as suggested command words for the functions that can be chosen in the command field. The command words can be chosen freely, and are fetched from the files:

CP-SMESS-NO-BOO:SYMB for the Norwegian version
CP-SMESS-EN-BOO:SYMB for the English version

We distinguish between commands/keys that are valid for separate regions, and those which are valid for the entire screen picture.

COMMANDS AND FUNCTION KEYS FOR SEPARATE REGIONS:

COMMAND	FUNCTION KEY	DESCRIPTION
SN	→|	Choose next search key
SP	|←	Choose previous search key
	==>	Move to next region
	<==	Move to previous region
NL	↓	Move to next line (if last line - move to next region)
PL	↑	Move to previous line (if first line - move to previous region)
	COPY	Copy field from previous line
FIRS	SHIFT + ←↑	Show first page (scroll)
-	←↑	Show previous page (scroll)
LAST	SHIFT + →↓	Show last page (scroll)
+	→↓	Show next page (scroll)
CLIN	F5	Clear line (and member lines) The mode for clearing a line when storing data lasts until you give the command to switch off this mode. The status is displayed on the status line.
OFFCL	SHIFT + F5	Switches off the clearing of a line during storing of data.

COMMANDS AND FUNCTION KEYS FOR THE ENTIRE SCREEN PICTURE:

COMMAND	FUNCTION KEY	DESCRIPTION
CREG	F4	Clear region (and member regions)
NR	> >	Choose next region
PR	< <	Choose previous region
PRIN		Copy picture to a printer/file
	HELP	Show help information
	SHIFT + HELP	Activate advanced help function (see chapter 13).
NAPL		Jump to next predefined function and transfer main key
PAPL		Jump to previous predefined function and transfer main key
X	EXIT	Exit from this program
	↖	Jump to/from command field
TEXT	⇶	Call of text function (see chapter 12).
	↳	Execute chosen function (modify, store or delete)

If an OK field is defined, this may be used instead of the EXECUTE key. The OK field is a field consisting of one character, where you may type in 'Y' if updating is to be performed, or 'N' if you do not want to update.

An example of a screen picture with an OK field:

```
Key      Description                          OK
.....    ..................................    .
.....    ..................................    .
.....    ..................................    .
```

COMMANDS / DESCRIPTION

Scroll

Commands: **FIRS, +, LAST, -**

FIRS, +, LAST and - (and corresponding function keys) are commands which concern the current region. They may only be used if the page in the region already is filled in. (See page 15.)

One page often contains only a small part of all the data to be found in the database for the current search key. The purpose of these commands is, in a simple way, to present the selection of data that you are interested in. The data that is presented is always sorted on ascending value of the current search key.

By typing in FIRS, you will be shown a page where the first data set is connected to the search key in the database with the lowest value.

'+' will display the following page, i.e. the data sets which follow the ones that are currently shown on the screen.

Similarly, '-' will display the previous page, and LAST the last page.

If you scroll in an owner region, the connected data in the member region will also be displayed. This is done in the same way as described under Querying.

Move to new region

Commands: **NR, PR**

When the program is started, the name of the first region in the picture is shown on the status line. This means that the first region is the current one, and that this is where you are working just now. (See explanation of the region notion on page 15.)

If you want the next region to become the current one, you type in the command NR. The command PR causes the previous region to become the current one.

Print screen picture

Command: **PRIN**

If you give the PRIN commant when the cursor is in the command field, you will get a print-out of the picture on your screen.

Jump to next application

Command: **NAPL**

Direct transfer from one program to a new, following program (defined in the menu system), with transfer of the main key. Search using the main key in the called program.

Jump back to previous application

Command: **PAPL**

Like NAPL, but with a jump back to a program defined as the previous one in the menu system.

Exit from function

Command: **X**

X (exit) causes the function you are at to be stopped.

Go to text function

Command: **TEXT**

The command TEXT only works if free text is defined for the program. You may give the command when querying or modifying. When querying in the calling program, you may only query in the free text function.

When modifying in the calling program, you may query, store, modify, and delete in the free text function.

If free text is registered in a record, a 'T' is displayed in the field for free text; otherwise this field is blank. When returning from the free text function, you are brought back to the place in the picture where the free text function was called.

CHAPTER 5

ADDITIONAL PROGRAMMING IN FORTRAN AND COBOL

5. ADDITIONAL PROGRAMMING IN FORTRAN AND COBOL

It may be necessary to touch up programs that are generated automatically by the program generator. In such additional programming there would naturally be calls to ABM's SIBAS and FOCUS overhead. Additional programming may be necessary in one or more regions. Below, we have described the various forms of additional code that may be useful, and where to insert this code in the generated code. The rules are the same for each region.

Several read calls

Complete-PG is based on the fact that all data fields (fields apart from main key, command word, BK field, and OK field) are read in one read call. However, if you want an instant input control on fields, the reading must be split into several read calls.

An example of this is shown in chapter 10.

Inserting additional code: Before and after CPREAD.

Control/calculation before updating

When all data is read or presented, control and/or calculation of fields may be required before the database is updated.

An example of this is shown in chapter 10.

Inserting additional code: After CPOKCOD, or after the last CPREAD call if CPOKCOD is not used.

Updating several realms

An updating sequence is surrounded by CPBTRANS and CPETRANS, which execute SUBEG og SUEND respectively. If several realms are to be updated, the additional code must be inserted before CPETRANS.

An example of this is shown in chapter 10.

Inserting additional code: After CPUPDATE.

5.1. INSERTING ADDITIONAL CODE

Belonging subroutine call

In the following description, we use the term 'belonging subroutine call'. A belonging subroutine is one of the subroutines that appear in the automatically generated program, and the manual code always has to be inserted immediately before or after this subroutine. For the type of additional code described above, the belonging subroutine is the one specified after 'Inserting additional code:'.

Below, we have described how the additional code is inserted in the generated code.

5.1.1. BEFORE BELONGING SUBROUTINE CALL

If the additional code is to be inserted before a belonging subroutine call, you have to test whether this subroutine is to be executed. Whether a subroutine is to be executed or not depends on the navigating in the picture.

Necessary syntax for additional code before a call to a subroutine is shown below.

COBOL syntax will be:

```
CALL 'CPIENABLE' USING TRIGGER-<flag> RESULT
IF RESULT = 1 THEN
  additional code
  -
```

FORTRAN syntax will be:

```
IF (CPABLED(FL<flagg>,1)) THEN
   additional code
   -
ENDIF
```

5.1.2. AFTER BELONGING SUBROUTINE CALL

When a subroutine is executed, an 'execute' flag will be set. If it is set, it will be reset automatically during the execution of the next subroutine.

This flag may be used in the following additional code to test whether it should be executed. In this way, you prevent the execution of the additional code every time the DO loop is performed.

The necessary syntax for additional code after a call to a subroutine is shown below.

COBOL syntax will be:

```
IF EXECUTE = 1 THEN
  additional code
  -
END-IF
```

FORTRAN syntax will be:

```
IF (EXECUTE) THEN
   additional code
   -
ENDIF
```

5.2. HOW TO DISCERN BETWEEN ADDITIONAL CODE AND GENERATED CODE

Storing the additional code

Certain functions demand manual programming in addition to the generated code. The first time an application is generated, a file named <program-name>:MANU is created. This file contains a series of empty macros. Additional code must be inserted in these macros if it is not to disappear after regenerating the program.

```
NOTE:
All additional code should be inserted in the macros
on the file <program-name>:MANU.
```

Name standard

The name of each macro consists of letters and numerals, for example: ecprea1.

The first letter is'f' or 'e', which specifies either before or after the belonging subroutine.

The next letters are the same as the first letters in the belonging subroutine.

The numeral shows which region the manual code belongs to.

An example of a macro as it is to be found on the file (this is the macro following CPREAD in region 1):

```
%,%% Manual code inserted after CPREAD 1;
DEF,ecprea1,^CRMOD;<>^ICRMOD;;
```

An example of a macro containing additional code:

```
%,%% Manuel code inserted after CPREAD 1;
DEF,ecprea1,^CRMOD;<
*        This is where the additional code starts.
         IF (EXECUTE) THEN
            -
            -
         ENDIF
*        End of additional code.
>^ICRMOD;;
```

5.3. MESSAGES

All messages used in a system should be stored on a file, with one file for each language you wish to run the system in.

Message file

The message file should be as follows:

% Sentences beginning with % are comments.
^ Sentences beginning with ^ specify the message number.

The messages should be written as described in the ABM manual.

Let us have a look at an example of a message file:

```
% Messages to my system
^sysid = 0
^SSI = 0
^0    MY SYSTEM
^1
0:-> First message'
^2
0:-> Second message'
^3
+:-> Error message'
```

Message database

This message file is 'compiled' by means of UEER-CONVERT into a message database (see next page). The name of the message database must be:

UE-UMESS-XX-BZZ

where ZZ is the revision number (00 - 99) and XX the language code:

NO - Norwegian
EN - English
TY - German
FR - French
SV - Swedish
DA - Danish
FI - Finnish
IS - Icelandic
IT - Italian
HO - Dutch
PO - Portuguese
SP - Spanish

The message database should belong to the user area where you run the system.

```
NOTE:
Before compiling the message file,
take a copy of the existing database, so
that you can start again if anything goes wrong.
```

Compiling

Compiling the message database:

```
@(ABM-SYS)UEER-CONVERT-B
<Message database>
<message file 1>
<message file 2>
 :
<message file n>
```

The file (ABM-SYS)UEER-CONVERT must be the B version or a later version.

You can also get a list of all the messages in a message database by giving the command:

```
@(ABM-SYS)UEER-LIST-B
<Message database>
<List file>
<from error no.> default lowest number
<to error no.> default highest number
```

The file (ABM-SYS)UEER-LIST must be the B version or a later version.

If you run the system under USER-ENVIRONMENT, the language code for the current user will be fetched, and the message database for that language will be opened. Without USER-ENVIRONMENT, the English message database will be opened.

Fetch and display a message

When messages are to be fetched and displayed, you use the following subroutine:

CPGETMSG(MSGNO)

Parameter list:

INTEGER MSGNO : the number of the message to be fetched

This subroutine concerns both FORTRAN and COBOL.

The message is placed in CTEXT (FORTRAN) / TEXT (COBOL). If you want to, you can edit the message before it is displayed, by means of CPMESS (or DDWMSGE(ITEXT,MSTA)).

5.4. FORTRAN AND COBOL EXAMPLES

Here we shall show a few examples of how to apply useful variables and routines. These are viewed in connection with problems often come across.

EXAMPLE 1

Problem: Several CPREAD calls are required. This is important if you want immediate INPUT control.

Suggested solution in **FORTRAN:**

```
*     Start of additional code
        FLNEXT = FLREAD
*     End of additional code
        CALL CPREAD (1, --- ,FLKEY,FLNEXT)
*     Start of additional code
        poss. test of field read in CPREAD(1,--)
        CALL CPREAD (2, --- ,FLREAD,FLREAD)
        poss. test of field read in CPREAD(2,--)
        CALL CPREAD (3, --- ,FLREAD,FLOKCOD)
        poss. test of field read in CPREAD(3,--)
*     End of additional code
```

Suggested solution in **COBOL:**

```
*     Manual code inserted before CPREAD (1...
*     Start of additional code
         MOVE TRIGGER-READ TO TRIGGER-NEXT
*     End of additional code
        CALL CPREAD (1, --- ,TRIGGER-KEY,TRIGGER-NEXT)
*     Manual code inserted after CPREAD (1...
*     Start of additional code
        poss. test of field read in CPREAD(1,--)
        CALL CPREAD (2, --- ,TRIGGER-READ,TRIGGER-READ)
        poss. test of field read in CPREAD(2,--)
        CALL CPREAD (3, --- ,TRIGGER-READ,TRIGGER-OK)
        poss. test of field read in CPREAD(3,--)
*     End of additional code
```

Comment: Here we have used three CPREAD calls. In the parameter lists, somewhat different parameters appear. The first parameter is the counter, which is increased from 1 to the total number of CPREAD calls. The parameters that control the field termination will also vary.

In the first CPREAD call, the backward arrow will cause a jump to the key (FLKEY or TRIGGER-KEY), the next to last parameter in the call. The ENTER key or forward arrow will cause a jump to the next CPREAD call (FLREAD, TRIGGER-READ).

For CPREAD call number 2, hitting the same keys will bring the previous CPREAD call or the next CPREAD call. For the last CPREAD call, the same field termination will bring the previous CPREAD call or CPOKCOD. The OK code will be read, or the EXECUTE key will be prompted for.

Instead of several CPREAD calls, you can use CPINVER. All fields may then be read by means of one CPREAD call, and a control will be performed for all of the fields. The fields that are not OK, and therefore have to be read once more, are shown in inverse video using CPINVER, and CPREAD is called again. You thereby get to know which fields have to be corrected. The combination of several CPREAD calls and use of CPINVER may also be applied. See example in chapter 10.

EXAMPLE 2

Problem: We want INPUT control of certain fields, possibly together with an error message, as well as activation of the same CPREAD call (i.e. to read the same field again).

Suggested solution in **FORTRAN:**

```
CALL CPREAD (n, ---)

IF (EXECUTE) THEN
    IF (IACTCOD.EQ.2.OR.IACTCOD.EQ.3) THEN
       <test field values if any>
       IF <error> THEN
           CALL CPGETMSG(9)
           CALL CPMESS
       ENDIF
    ENDIF
ENDIF

CALL CPREAD (n+1, ---)
```

Comment: CPREAD must have been activated (EXECUTE = TRUE). If the action code is R(egistration) or M(odification) (i.e. IACTCOD = 2 or 3), a control of the screen value will be performed in this additional programming.

If the INPUT value is not correct, the routine CPMESS will be called. An error message will be displayed, and the same CPREAD call will be re-activated.

Suggested solution in **COBOL**:

```
CALL 'CPREAD' USING n, ---

IF EXECUTE = 1
    AND (MAINTAB(5)=2 OR MAINTAB(5)=3)
    <test field values if any>
    IF <feil>
        CALL 'CPGETMSG' USING 9
        CALL 'CPMESS'.

CALL 'CPREAD' USING n+1, --- .
```

Comment: CPREAD must have been activated (EXECUTE = 1). If the action code is R(egistration) or M(odification) (i.e. MAINTAB(5) = 2 or 3), a control of the screen value will be performed in this additional programming.

If the INPUT value is not correct, the routine CPMESS will be called. An error message is displayed, and the same CPREAD call is re-activated.

5.5. ERROR HANDLING

If an error should occur, the internal error handling in Complete-PG will list all the involved routines in a hierarchical sequence, i.e. the routine on the highest level is printed first, then the routine on the next level, etc. Type of error, error status and other information will also be printed.

All of these error messages will be written on the file CP-ERROR:LOGG. On the screen, you will get information about which error has occurred. The program will be stopped.

5.5.1. ERROR HANDLING IN MANUALLY DEFINED SUBROUTINES

You may also use Complete-PG's error handling in your own subroutines. The standard layout for this is shown below.

COBOL syntax will be:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. name.
DATA DIVISION.
WORKING-STORAGE SECTION.
     COPY (ABM-SYS)CP-PROBOT-COM:COPY
LINKAGE SECTION.
     -
     -
PRODECURE DIVISION USING....
     MOVE 'name    ' TO CSUB
     CALL 'CPIN' USING CSUB.
     CALL 'RUTINE1' USING...
     IF error
        MOVE 'RUTINE1' TO CSUB
        CALL 'CPABORT'
        CALL 'CPOUT' USING CSUB
     END-IF
     EXIT-PROGRAM
```

FORTRAN syntax will be:

```
SUBROUTINE name( -- )
    $INCLUDE (ABM-SYS)CP-PROBOT-COM:INCL

    csub = 'name'
    CALL CPIN(isub)
    csub = 'routine1'
    CALL routine1
    IF (error) GOTO 9900
        -
        -
        -
9900 IF (error) CALL CPABORT
        CALL CPOUT(isub)
        RETURN
    END
```

In chapter 6 you will find a description of the routines CPIN, CPABORT and CPOUT.

5.6. SEVERAL CPREAD CALLS

By introducing a counter in the parameter list for the subroutine CPREAD, it is possible to split READ calls. For each new CPREAD call that you want, you have to increase the first parameter by 1. The CPREAD calls will thus be 'numbered' from 1 to n, where n is the total number of CPREAD calls. The system will arrange a sequential execution of all the CPREAD calls.

5.7. READCO

READCO is an INTEGER variable that corresponds to the counter in CPREAD. In each CPREAD call this variable will be increased or decreased by 1 depending on the termination of the field. This is done by the system. CPREAD calls will only be executed if READP = READCO. Through additional programming, you may for instance use READCO to skip the reading of certain fields in certain situations.

5.8. SELECTING RECORDS

The CPGET routine uses GETN calls towards SIBAS, and retrieves as many records from the database as there are lines in the screen picture. All of these records are then displayed in the CPDISP routine.

If you need to fetch one record at a time (for example to display only those records that fulfill certain criteria), the parameter EVERYLIN may be set to 1 (it is otherwise equal to 0). Then CPGET will fetch one record, which is placed in the screen buffer and the database buffer. This may now be tested. If it is accepted, i.e. it is to be displayed, you have to set the flag FLOK (FORTRAN) or TRIGGER-OK (COBOL): It is done like this:

FORTRAN:

```
CALL  CPENABLE(FLOK)
```

COBOL:

```
CALL 'CPIENABLE' USING TRIGGER-OK
```

If it is not accepted, you have to reset the OK flag (this is not done automatically), and the record will not be displayed. CPGET will then be repeated, until the screen picture is full or there are no more records left in the database.

5.9. FLOK

FORTRAN:
The FLOK flag is used to indicate which records are to be accepted. This flag is set if the record is accepted, and reset if the record is rejected.

5.10. TRIGGER-OK

COBOL:
The TRIGGER-OK flag is used to indicate which records are to be accepted. This flag is set if the record is accepted, and reset if the record is rejected.

CHAPTER 6

PROGRAM VARIABLES AND ROUTINES AVAILABLE TO A PROGRAMMER

6. PROGRAM VARIABLES AND ROUTINES AVAILABLE TO A PROGRAMMER

The other program variables and routines that you may use for additional programming, are:

IACTCOD, MAINTAB(5), OWNMESS, NOERR, FLNEXT, TRIGGER-NEXT, CTEXT, TEXT, CRSPNS, TERMCOD, CPABLED, CPIENABL, CPIN(), CPOUT() and CPABORT.

We shall now describe each one:

6.1. IACTCOD

FORTRAN:
An INTEGER variable that always keeps track of the action code last used, and thereby may be used to test the current action code.

The action codes are:

```
IACTCOD  = 1 : querying
         = 2 : storing
         = 3 : modifying
         = 4 : deleting
```

6.2. MAINTAB(5)

COBOL:
An INTEGER variable that always keeps track of the last used action code, and thereby may be used to test the current action code.

The action codes are:

```
MAINTAB(5) = 1 : querying
           = 2 : storing
           = 3 : modifying
           = 4 : deleting
```

6.3. OWNMESS

FORTRAN:
A flag (logical variable) used to overrule messages in the Complete-PG routines.

By setting OWNMESS to .TRUE. before a PG routine, and moving the preferred message to CTEXT, your own message will be displayed instead of the standard message from the PG routine.

OWNMESS must be reset to .FALSE. immediately after the return from the PG routine.

COBOL:
A variable used to overrule messages in the Complete-PG routines.

By setting OWNMESS to 1 before a PG routine, and moving the preferred message to TEXT, your own message will be displayed instead of the standard message from the PG routine.

OWNMESS must be reset to 0 immediately after the return from the PG routine.

6.4. NOERR

FORTRAN:
A flag (logical variable) that is .TRUE. as long as no error occurs.

COBOL:
A variable that has the value of 1 as long as no error occurs.

6.5. FLNEXT

FORTRAN:
A flag used in connection with several CPREAD calls. Complete-PG sets FLNEXT = FLOKCOD. If several CPREAD calls are to be used in the additional programming, you must set FLNEXT = FLREAD after CPGET.

6.6. TRIGGER-NEXT

COBOL:
A variable used in connection with several CPREAD calls. Complete-PG sets TRIGGER-NEXT=TRIGGER-OK. If several CPREAD calls are to be used in the additional programming, you must set TRIGGER-NEXT=TRIGGER-READ after CPGET.

6.7. CTEXT

FORTRAN:
A character string containing the message which is to be displayed for the user. The message in CTEXT is displayed by means of the routine CPMESS. The routine CPGETMSG(I) moves message I, from the message file, into CTEXT. CTEXT may be edited, but the message in the variable must have ABM/FOCUS format when CPMESS is called.

6.8. TEXT

COBOL:
A character string containing the message which is to be displayed for the user. The message in TEXT is displayed by means of the routine CPMESS. The routine CPGETMSG(I) moves message I, from the message file, into TEXT. TEXT may be edited, but the message in the variable must have ABM/FOCUS format when CPMESS is called.

6.9. CRSPNS

FORTRAN AND COBOL:
A variable containing the last used command word.

(This one is the same for both FORTRAN and COBOL).

6.10. TERMCOD

TERMCOD (INTEGER variable) contains the last chosen termination code (termination from a field on the screen). The alternatives available are:

Alternatives	Meaning
TERMCOD = 0	--> , CR
TERMCOD = 1	↖
TERMCOD = 2	<--
TERMCOD = 3	==>
TERMCOD = 4	<==
TERMCOD = 5	-->!
TERMCOD = 6	!<--
TERMCOD = 7	↓
TERMCOD = 8	↑
TERMCOD = 9	EXIT
TERMCOD = 10	<== SHIFT ↑
TERMCOD = 11	<== ↑
TERMCOD = 12	==> SHIFT ↓
TERMCOD = 13	==> ↓
TERMCOD = 14	F5
TERMCOD = 15	F4
TERMCOD = 16	>>
TERMCOD = 17	<<
TERMCOD = 18	CNTR C
TERMCOD = 19	PRINT
TERMCOD = 20	↳
TERMCOD = 21	<>
TERMCOD = 22	><
TERMCOD = 23	⤓
TERMCOD = 24	F3
TERMCOD = 25	MARK
TERMCOD = 26	SHIFT F5
TERMCOD = 27	F7
TERMCOD = 28	F2
TERMCOD = 29	F8
TERMCOD = 30	F1

6.11. LOGICAL FUNCTION CPABLED(FLXXXX,1)

FORTRAN:

Parameter list:

INTEGER FLxxxx (Input): The parameter value of the routine to be tested.

INTEGER (Output)

Function description: The function tests whether the specified flag is set. TRUE is returned if this is the case.

6.12. SUBROUTINE CPIENABL(TRIGGER-XXXX,RESULT)

COBOL:

Parameter list:

INTEGER TRIGGER-xxxx (Input): The parameter value of the routine to be tested.

INTEGER RESULT (Output)

Routine description: The routine tests whether the specified flag is set.

RESULT=1 : is returned if the specified flag is set.
RESULT=0 : is returned if the specified flag is not set.

6.13. SUBROUTINE CPIN(ISUB)

FORTRAN:

Parameter list:

INTEGER*2 ISUB(4) (Input): The name of the current routine.

COBOL:

Parameter list:

PIC X(8) (Input): The name of the current routine.

Routine description: Names of all routines involved are put in a table. If an error occurs, you may use this table to retrieve the name of the routine. If this routine is used, it must be called at the beginning of a subroutine.

6.14. SUBROUTINE CPOUT(ISUB)

FORTRAN:

Parameter list:

INTEGER*2 ISUB(4) (Input): Name of the routine that was last executed.

COBOL:

Parameter list:

77 ISUB PIC X(8) (Input): Name of the routine that was last executed.

Routine description: The name of the last routine executed without error, is removed from the table.

6.15. SUBROUTINE CPABORT

FORTRAN AND COBOL:

Parameter list: None

Routine description: If an error occurs in a SIBAS call or a FOCUS call, CPABORT should be called. CPABORT resets all flags so that the DO loop is ended and CPEND is called. CPEND will, if MSTA≠0 or KSTAT≠0 and KSTAT≠1, write en error message on the error message file.

> NOTE:
> An error message is written by CPEND only if MSTA≠0, or KSTAT≠1 and KSTAT≠0.

CHAPTER 7

DOCUMENTATION OF ROUTINES IN THE GENERATED PROGRAM

7. DOCUMENTATION OF ROUTINES IN THE GENERATED PROGRAM

7.1. THE STRUCTURE OF THE GENERATED PROGRAM

Here we shall give an overview of the program logic used in the generated program.

Programs generated by Complete-PG are short. A FORTRAN program consists of 11 to 17 subroutines, while a COBOL program contains 13 to 19 subroutines.

Out of these subroutines, 11 (in FORTRAN) and 13 (in COBOL) will always be part of a program generated by Complete-PG, no matter what the function picture will look like (i.e. the contents of the regions).

7.2. AN EXAMPLE OF A GENERATED PROGRAM IN FORTRAN

The following is part of a generated FORTRAN program. Additional code is not included in the program. Neither OK code nor action code is contained in the program, and the program only handles one realm.

```
PROGRAM name
<heading>
<declarations>
<include files>
CALL CPHJON
CALL CPBEGIN( -- )                %fetches picture and opens database
DO WHILE (one region active)
 DO WHILE (this region active)
  CALL CPREGION( -- )

  citmsub(1) = key field
  CALL CPKEY( -- )                %reads value in key field

  CALL CPGET( -- )                %fetches records of interest

  CALL CPINRC( -- )               %fetches data from database buffer

  citmsub(1) = key + data field
  CALL CPDISP( -- )               %transfers data to screen

  citmsub(1) = data field
  CALL CPREAD( -- )               %reads data field

  CALL CPBTRANS( -- )             %starts transfer

  citmsub(1) = all fields in record
  CALL CPUPDATE( -- )             %updates database

  CALL CPETRANS                   %ends transfer

  CALL CPRSPNS( -- )              %reads response code

  CALL CPOVER( -- )

  CALL CPHELP

 ENDDO
ENDDO

CALL CPEND                        %terminates program
END
```

The DO loops are characteristic for the generated program. The first one encloses all the regions in the program. The other DO loop(s) enclose(s) one region each. In this case the other DO loop encloses 9 of the subroutines.

The length of the program increases for each region

In general a program consists of one DO loop for each region (maximum two regions). For each region included in the screen picture, the code will increase by n sets of subroutine calls (in addition to CPBEGIN and CPEND). Two regions will for instance generate at least 18 subroutine calls inside the DO loop.

The other 6 subroutines

In addition to the 11 regular subroutines shown on the previous page, you might get calls generated for the following 6 subroutines, depending on which information you have given to the generating program:

CPACTCOD(--) is generated if there is an action code field in the screen picture.

CPOKCOD(--) is generated if there is an OK field in the screen picture.

CPOTHER(--) is generated if data is to be fetched from other realms.

CPEXIST(--) is generated if there is an existence control towards other realms.

CPFRTXT(--) is generated if free text is used in the screen picture.

CPTDISP(--) is generated if free text is used - puts a 'T' in the field on the screen.

Jump out of a DO loop

A jump out of a DO loop only takes place if you press the EXIT key, or if an error occurs.

7.3. AN EXAMPLE OF A GENERATED PROGRAM IN COBOL

Below, you will find part of a generated COBOL program. Additional code is not included in this program.

```
IDENTIFICATION DIVISION.
<name>
<heading>
ENVIRONMENT DIVISION.
   CONFIGURATION DIVISION.
DATA DIVISION.
   WORKING-STORAGE SECTION.
   <declarations>
   LINKAGE SECTION.
   <declarations>
PROCEDURE DIVISION.
   MAIN SECTION.
      PERFORM STARTUP.
      PERFORM REGION UNTIL (no region active)
      PERFORM ROUNDUP.
   EXIT.

   STARTUP SECTION.
      CALL 'CPHJON'                    %sets help function available
      <assignment statements>
      CALL 'CPBEGIN' USING ---         %fetches picture and
                                        opens database
   EXIT.

   REGION SECTION.
      PERFORM REGION-1 UNTIL (region-1 not active)
   EXIT.


   REGION-1 SECTION.
      CALL 'CPREGION' USING ---

      CALL 'CPCURKC' USING ---

      ddc-select = key field
      CALL 'CPKEY' USING ---           %reads value in key field

      CALL 'CPKEYNC' USING ---

      CALL 'CPGET' USING ---           %fetches records of interest

      CALL 'CPINRC' USING ---          %fetches data in database buffer

      ddc-select = key + data field
      CALL 'CPDISP' USING ---          %transfers data to screen

      ddc-select = data field
      CALL 'CPREAD' ---                %reads data field
```

Continued on next page...

```
      CALL 'CPBTRANS' USING ---          %starts transfer

      ddc-select = all fields in
                   record
      CALL 'CPUPDATE' USING ---          %updates database

      CALL 'CPETRANS'                    %ends transfer

      CALL 'CPRSPNS'                     %reads response code

      CALL 'CPOVER' USING ---

      CALL 'CPHELP' USING ---

   EXIT.

   ROUNDUP SECTION.
      CALL 'CPEND' USING ---             %terminates the program
   EXIT.
```

The program logic is described in more detail in chapter 8.

The length of the program increases with each region

In general, a program consists of one DO loop for each region. (We may have a maximum of two regions.) For each region included in the screen picture, the code will increase by n sets of subroutine calls (apart from CPBEGIN and CPEND).

The other 6 subroutines

In addition to the regular subroutines shown in this example, you may have calls generated for the following 6 subroutines, depending on which information you have given to the generating program:

CPACTCOD(--) is generated if there is an action code field in the screen picture.

CPOKCOD(--) is generated if there is an OK field in the screen picture.

CPOTHER(--) is generated if data is fetched from other realms.

CPEXIST(--) is generated if there is an existence control towards other realms.

CPFRTXT(--) is generated if free text is used in the screen picture.

CPTDISC(--) is generated if free text is used - puts a 'T' in the field in the screen picture.

Jump out of a DO loop	A jump out of a DO loop only takes place if you press the EXIT key, or if an error occurs.

7.4. ROUTINES IN THE GENERATED PROGRAM

These routines are always generated:

Routine	Type
CPBEGIN	Subroutine
CPREGION	Function
CPCURKC (COBOL)	Subroutine
CPCURKK (FORTRAN)	Function
CPKEY	Subroutine
CPKEYNC (COBOL)	Subroutine
CPKEYNK (FORTRAN)	Function
CPGET	Subroutine
CPINRC	Subroutine
CPDISP	Subroutine
CPREAD	Subroutine
CPBTRANS	Subroutine
CPUPDATE	Subroutine
CPETRANS	Subroutine
CPRSPNS	Subroutine
CPEND	Subroutine

These routines may be generated, depending on parameters:

Routine	Type
CPACTCOD	Subroutine
CPOKCOD	Subroutine
CPOTHER	Subroutine
CPEXIST	Subroutine
CPFRTXT	Subroutine
CPTDISP/CPTDISC	Subroutine

7.4.1. THE MOST USED PARAMETERS

The most often used parameters are described here:

COBOL	FORTRAN	Explanation
DDC-REF-TABLE	REFTAB	Reference table (described in ABM manual chap. 6/7).
DDS-xx-SUBSCHEMA	MITEM	Total element list information for the screen picture. (See ABM manual chap. 6.1).
SCV-xx	MRECxx	Screen value buffer for all the elements described in MITEM.
DDB-zz-SUBSCHEMA	KITEM	Total element list information for the database (described in the ABM manual chap. 4.1).
DBV-zz	KREC	Database value buffer for all the elements described in KITEM.

xx = Picture record name
zz = Realm name/Realm prefix

7.4.2. TABLES WITH VARIABLES IN FORTRAN AND COBOL

The following is an overview of all variable names used in this manual. In these tables you will see the corresponding variable names in FORTRAN and COBOL.

FORTRAN :	COBOL :
ANTFEIL	TELL-FEIL
BKODE	SCV-DUMMY-BKODE
CITMSUB	DDS-SELECT
CNAME	ITEM-NAME
COMTAB	COMTAB
CPABLED	CPIENABL
CPFRTXT	CPFRTXC
CPKEYNK	CPKEYNC
CPCURKK	CPCURKC
CRSPNS	CRSPNS
CSUB	CSUB
CTEXT	TEXT
CTYPE	SCREEN-VALUE
ENTEXT	ENTEXT
EVERYLIN	EVERYLIN
EXECUTE	EXECUTE
FLACTCOD	TRIGGER-ACTCODE
FLCOMMAN	TRIGGER-COMMAN
FLDISPLY	TRIGGER-DISPLAY
FLFRTXT	TRIGGER-FRTXT
FLGET	TRIGGER-GET
FLKEY	TRIGGER-KEY
FLNEXT	TRIGGER-NEXT
FLOK	TRIGGER-OK
FLOKCOD	TRIGGER-OKCODE
FLREAD	TRIGGER-READ
FLRESPON	TRIGGER-RESPONS
FLTRANS	TRIGGER-TRANS
FLUPDATE	TRIGGER-UPDATE
IACTCOD	MAINTAB(5)
ISUB	CSUB
ITEMSUB	DDS-SELECT
ITEXT	TEXT
KEYNR	CURRENT-KEY-NO
	KEY-NO
KIKEY	DBKI--
KITEM	DDB--SUBSCHEMA
KNREA	DBR-NO-OF-REALMS
KPMOD	DBR-REALM-PROTECT
KREALMS	DBR-REALM-NAMES

COBOL :	FORTRAN :
COMTAB	COMTAB
CPFRTXC	CPFRTXT
CPIENABL	CPABLED
CPKEYNC	CPKEYNK
CPCURKC	CPCURKK
CRSPNS	CRSPNS
CSUB	CSUB/ISUB
CURRENT-KEY-NO	KEYNR
DBKI--	KIKEY
DBKV--	KVKEY
DBR-NO-OF-REALMS	KNREA
DBR-REALM-NAMES	KREALMS
DBR-REALM-PROTECT	KPMOD
DBR-REALM-USAGE	KUMOD
DBV-	KREC
DBV--XXTNR	XXTNR
DDB--SUBSCHEMA	KITEM
DDC-REF-TABLE	REFTAB
DDS-SELECT	CITMSUB
DDS-SELECT	ITEMSUB
DDS-XX-SUBSCHEMA	MITEM
ENTEXT	ENTEXT
EVERYLIN	EVERYLIN
EXECUTE	EXECUTE
INDX	RCPOINT
ITEM-NAME	CNAME
ITEM-VALUE	TXTNR
KEY-NO	KEYNR
KSTAT	KSTAT
MAINTAB(5)	IACTCOD
MSTA	MSTA
NOERR	NOERR
OWNMESS	OWNMESS
READCO	READCO
SCREEN-NAME	CTYPNAM
SCREEN-VALUE	CTYPE
SCV-DUMMY-BKODE	BKODE
SCV-DUMMY-OKODE	OKODE
SCV-XX	MREC
TELL-FEIL	ANTFEIL

Continued on the next page...

FORTRAN :	COBOL :
KREC	DBV-
KSTAT	KSTAT
KUMOD	DBR-REALM-USAGE
KVKEY	DBKV--
MITEM	DDS-XX-SUBSCHEMA
MREC	SCV-XX
MSTA	MSTA
NOERR	NOERR
OKODE	SCV-DUMMY-OKODE
OWNMESS	OWNMESS
RCPOINT	INDX
READCO	READCO
REFTAB	DDC-REF-TABLE
TERMCOD	TERMCOD
TXTNR	ITEM-VALUE
CTYPNAM	SCREEN-NAME
XXTTNR	DBV--XXTTNR

COBOL :	FORTRAN :
TERMCOD	TERMCOD
TEXT	CTEXT
TEXT	ITEXT
TRIGGER-ACTCODE	FLACTCOD
TRIGGER-COMMAN	FLCOMMAN
TRIGGER-DISPLAY	FLDISPLY
TRIGGER-FRTXT	FLFRTXT
TRIGGER-GET	FLGET
TRIGGER-KEY	FLKEY
TRIGGER-NEXT	FLNEXT
TRIGGER-OK	FLOK
TRIGGER-OKCODE	FLOKCOD
TRIGGER-READ	FLREAD
TRIGGER-RESPONS	FLRESPON
TRIGGER-TRANS	FLTRANS
TRIGGER-UPDATE	FLUPDATE

READP : The number of the CPREAD call. The number is inserted directly as a parameter.

REG : The number of the current region is inserted directly.

TNR : Corresponds to the value of TXTNR (FORTRAN), ITEM-VALUE (COBOL).

7.5. DOCUMENTATION OF THE ROUTINES

In the following routine descriptions, the column marked 'I/O' means the following:

I : input parameter in the routine.
O : output parameter in the routine.

7.5.1. CPBEGIN

Routine name: **CPBEGIN**

Parameters :

COBOL	FORTRAN	I/O	Explanation
DDC-REF-TABLE,	REFTAB,	I/O	See ABM manual chapter 7/6.
DBR-NO-OF-REALMS	KNREA,	I	Number of realms to be readied.
DBR-REALM-NAMES	KREALMS,	I	Names of realms.
DBR-REALM-USAGE(1)	KUMOD,	I	Usage mode for the realms.
DBR-REALM-PROTECT(1)	KPMOD	I	Protection mode for the realms.

Routine description: CPBEGIN is an initiation routine that fetches the picture for the current function from a file, and transfers it to the screen. The routine also readies the realms to be used. It opens the correct error message file and initiates all the variables.

7.5.2. CPREGION

Routine name: **CPREGION**

Parameters :

COBOL	FORTRAN	I/O	Explanation
DDC-REF-TABLE	REFTAB,	I/O	See ABM manual chapter 7/6.
REG	REG,	I	Active region (is specified as a constant).
DDS-xx-SUBSCHEMA	MITEMxx,	I	See ABM manual chapter 7/6.
SCV-zz	MRECzz,	I	See ABM manual chapter 7/6.
TRIGGER-nnnn	FLnnnn	I	Next routine to be executed.

xx = Realm name
zz = Picture record name
nn = Routine name

Routine description:

Checks to see whether the current region is set active. If so, the parameter TRIGGER-nnnn or FLnnnn will specify the next routine to be called.

During registration, when all the lines in the picture are filled in, the routine will clear the region, display the last line on line number 1, and place the cursor on line number 2.

The routine moves the values for the current line into the screen buffer (DDGETRC).

7.5.3. CPCURKC

Routine name: **CPCURKC** (COBOL)
CPCURKK (FORTRAN)

Parameters :

COBOL	FORTRAN	I/O	Explanation
CURRENT-KEY-NO	KEYNO	O	Current key number.

Routine description:

Returns the number of the search key currently being used.

7.5.4. CPKEY

Routine name: **CPKEY**

Parameters :

COBOL	FORTRAN	I/O	Explanation
DDC-REF-TABLE	REFTAB,	I	See ABM manual chapter 7/6.
DDS-xx-SUBSCHEMA	MITEMxx,	I	See ABM manual chapter 7/6.
SCV-xx	MRECxx,	O	See ABM manual chapter 7/6.
TRIGGER-nn	FLnnnn,	I	Previous routine which may be activated.
TRIGGER-nn	FLnnnn	I	Next routine that will be activated.

xx = Picture record name
nn = Routine name

Routine description: When the program is started, and several action codes are permitted, the default action code ('query') is switched on. If only one action code is permitted, that one will be switched on.

If there is a key field in the picture, you may type in values for the key, and the action code may be set and changed. If there is no key field in the picture (only legal for querying and storing), you have to specify in the command field what kind of action you require.

CPKEY activates the routines CPGET, CPDISP and the routine that is specified as input parameter.

7.5.5. CPKEYNC

Routine name: **CPKEYNC** (COBOL)
CPKEYNK (FORTRAN)

Parameters :

COBOL	FORTRAN	I/O	Explanation
CURRENT-KEY-NO	KEYNO	O	Current key number.

Routine description: When querying, the current key number is returned. Otherwise it is set to 1 (i.e. main key).

7.5.6. CPGET

Routine name: **CPGET**

Parameters :

COBOL	FORTRAN	I/O	Explanation
EVERYLIN	EVERYLIN,	I	*
DDC-REF-TABLE	REFTAB,	I	See ABM manual chapter 7/6.
CURRENT-KEY-NO	KEYNR,	I	Number of current search key.
DBKI-xx-zz	KIzzzzz,	I	ABM manual chapter 5/4.
DBKV-xx-zz	KVzzzzz,	I	ABM manual chapter 5/4.
DDS-rr-SUBSCHEMA	MITEMrr,	I	ABM manual chapter 7/6.
SCV-rr	MRECrr,	O	ABM manual chapter 7/6.
DDB-xx-SUBSCHEMA	KITEM	I	ABM manual chapter 5/4.

xx = Realm name
yy = Item name
zz = Index name
rr = Picture record name

* EVERYLIN: The value 0 specifies that all records within the search region may be displayed without any control.

The value 1 specifies that the user wants each record to be tested before it is displayed. See example on page 162.

Routine description: Finds records in the database and transfers the values to a screen buffer.

When EVERYLIN=0 for querying, as many records are fetched from the database as there are lines in the region.

When EVERYLIN=1 for querying, one record is fetched from the database at a time. CPGET is executed repeatedly until all the lines in the picture are filled in, or until there are no more records left in the database.

When modifying and deleting, the record belonging to the key specified in CPKEY is fetched. If no record is found, you get an error message, and the CPKEY routine is re-activated.

When storing, there is a check to see if a record with the specified key value exists or not. If the record exists, a message is displayed, and the user has to give a new key value.

When querying by means of a key, the search region is established in accordance with the low/high limits specified in the screen picture 'Use of Program Keys' during the generating of the program. Records with a key value starting with the given key are fetched from the database.

7.5.7. CPINRC

Routine name: **CPINRC**

Parameters :

COBOL	FORTRAN	I/O	Explanation
INDX	RCPOINT,	I	The line the database record is to be fetched from.
DDB-xx-SUBSCHEMA	KITEMxx,	I	xx = Realm name/Realm prefix.
DBV-xx	KRECxx	O	xx = Realm name/Realm prefix.

Routine description: Fetches database values for the current line from the total database buffer, and places them in a local database buffer (i.e. the database buffer in the input parameter).

7.5.8. CPDISP

Routine name: **CPDISP**

Parameters :

COBOL	FORTRAN	I/O	Explanation
DDC-REF-TABLE	REFTAB,	I/O	ABM manual chapter 7/6.
DDS-xx-SUBSCHEMA	MITEMxx,	I	ABM manual chapter 7/6.
SCV-xx	MRECxx	O	ABM manual chapter 7/6.

xx = Picture record name

Routine description: When the action code is query, the desired page will be displayed on the screen. If you are modifying or deleting, the desired logical line will be displayed on the screen.

7.5.9. CPREAD

Routine name: **CPREAD**

Parameters :

COBOL	FORTRAN	I/O	Explanation
READP	READP,	I	Number of the current PREAD call.
DDC-REF-TABLE	REFTAB,	I/O	ABM manual chapter 7/6.
DDS-xx-SUBSCHEMA	MITEMxx,	I	ABM manual chapter 7/6.
SCV-xx	MRECxx,	O	ABM manual chapter 7/6.
TRIGGER-nn	FLnnnn,	I	Previous routine that can be activated.
TRIGGER-nn	FLnnnn	I	Next routine that will be activated.

Routine description:

The routine is only executed if a variable READCO=READP. READCO is set to 1 each time the subroutine CPKEY is executed. Otherwise, it is up to the user to control READCO so that the correct CPREAD call is executed.

CPREAD tests if the action code is storing or modifying. If the test is positive and the EXECUTE key is not pressed, then all the fields mentioned in the item list will be read. When you press the EXECUTE key, the reading will be terminated, and flag for updating will be set.

If the last field is read and no updating flag is set, you will be asked if you want to update. If you do want to update, press the EXECUTE key, and flags for updating will be set (i.e. CPBTRANS, CPETRANS and CPUPDATE).

If you want an automatic updating after the last CPREAD, i.e. the question about updating and pressing the EXECUTE key is not wanted, you have to call the subroutine CPUPMODE after CPREGION. See description of CPUPMODE in appendix A.

7.5.10. CPBTRANS

Routine name: **CPBTRANS**

Parameters :

COBOL	FORTRAN	I/O	Explanation
DDS-xx-SUBSCHEMA	MITEMxx,	I	ABM manual chapter 7/6.
SCV-xx	MRECxx,	I	ABM manual chapter 7/6.
DDB-zz-SUBSCHEMA	KITEMzz,	I	ABM manual chapter 5/4.
DBV-zz	KRECzz	O	ABM manual chapter 5/4.

xx = Picture record name
zz = Realm name/Realm prefix

Routine description: Starts the updating transaction by calling SUBEG. In addition, the screen buffer is transferred to the database buffer.

7.5.11. CPUPDATE

Routine name: **CPUPDATE**

Parameters :

COBOL	FORTRAN	I/O	Explanation
DDC-SELECT	ITEMSUB,	I	Subitem list. ABM manual chapter 7/6.
DDB-xx-SUBSCHEMA	KITEMxx,	I	ABM manual chapter 5/4.
DBV-xx	KRECxx	I	ABM manual chapter 5/4.

xx = Realm name/Prefix

Routine description: Depending on the action code this routine will execute one of the following functions on the main realm: store, modify, or delete.

7.5.12. CPETRANS

Routine name: **CPETRANS**

Parameters : None

Routine description: Ends a critical sequence (SUEND) and resets the updating flag. Displays a message saying which updating has taken place.

7.5.13. CPRSPNS

Routine name: **CPRSPNS**

Parameters :

COBOL	FORTRAN	I/O	Explanation
DDC-REF-TABLE	REFTAB,	I/O	See ABM manual chapter 7/6.
DDS-xx-SUBSCHEMA	MITEMxx,	I	See ABM manual chapter 7/6.
SCV-zz	MRECzz	I	See ABM manual chapter 7/6.

Routine description: Increased the line counter if there are several lines in the picture. Reads the command in the command field or reacts to predefined function keys. Sets flags for further execution, depending on the given command.

Only a legal command or a FOCUS error causes a return to the main program.

7.5.14. CPEND

Routine name: **CPEND**

Parameters : None

Routine description: The routine terminates the program, and checks if an error has occurred. If an error has occurred, i.e. KSTAT≠1, KSTAT≠0 or MSTA≠0, an error message will be written to the error message file. A message saying that an error has occurred will also be displayed on the screen.

7.5.15. CPACTCOD

Routine name: **CPACTCOD**

Parameters :

COBOL	FORTRAN	I/O	Explanation
DDC-REF-TABLE	REFTAB,	I/O	ABM manual chapter 7/6.
DDS-xx-SUBSCHEMA	MITEMxx,	I	ABM manual chapter 7/6.
SCV-xx	MRECxx,	O	ABM manual chapter 7/6.
SCV-DUMMY-BKODE	BKODE,	O	The chosen action code.
TRIGGER-nn	FLnnnn,	I	Previous routine that may be activated.
TRIGGER-nn	FLnnnn	I	Next routine to be activated.

Routine description:

Action code field is read.

If only one action code is legal, this will be standard. If no action code is legal, querying will become the standard code.

The current action code is written as default, and the cursor will be positioned in the next field. If you want to give or change the action code, you have to move the cursor back to this field by means of the left-arrow key.

Legal action codes:

1 : querying
2 : storing
3 : modifying
4 : deleting

7.5.16. CPOKCOD

Routine name: **CPOKCOD**

Parameters :

COBOL	FORTRAN	I/O	Explanation
DDC-REF-TABLE	REFTAB,	I/O	ABM manual chapter 7/6.
DDS-xx-SUBSCHEMA	MITEMxx,	I	ABM manual chapter 7/6.
SCV-xx	MRECxx,	O	ABM manual chapter 7/6.
SCV-DUMMY-OKODE	OKODE,	O	Specified OK code.
TRIGGER-nn	FLnnnn	I	Previous routine that may be activated.

Routine description:

Waits until the OK code is 'Y' or 'N', or until you press the HOME key (\). If OK = 'Y', the flag that signals updating will be made ready.

You can press the EXECUTE key instead of 'Y'.

7.5.17. CPOTHER

Routine name: **CPOTHER**

Parameters :

COBOL	FORTRAN	I/O	Explanation
DDC-REF-TABLE	REFTAB,	I	See ABM manual chapter 7/6.
DBKI-xx-zz	KIzzzzz,	I	See ABM manual chapter 5/4.
DBKV-xx-zz	KVzzzzz,	I	See ABM manual chapter 5/4.
DDB-xx-SUBSCHEMA	KITEM,	I	See ABM manual chapter 5/4.
DBV-xx	KRECxx,	I	See ABM manual chapter 5/4.
DDS-rr-SUBSCHEMA	MITEMrr,	I	See ABM manual chapter 7/6.
SCV-rr	MRECrr	O	See ABM manual chapter 7/6.

xx = Realm name
yy = Item name
zz = Index name
rr = Picture record name

Routine description:

Fetches data from the specified realm, and moves the data to a screen buffer.

When you modify/store, data is fetched and displayed on the screen. If data is not found, a message is displayed and the last executed CPREAD must be executed again.

7.5.18. CPEXIST

Routine name: **CPEXIST**

Parameters :

COBOL	FORTRAN	I/O	Explanation
DBKI-xx-yy	KIzzzzz,	I	ABM manual chapter 5/4.
DBKV-xx-yy	KVzzzzz	I	ABM manual chapter 5/4.

Routine description:

When you store or modify data, this routine checks if data is found in the specified realm. If not, a message is displayed and the last CPREAD call must be executed again.

When you delete data, the routine checks that there is no data in the specified member's realm. If there is any data, an error message is displayed and the cursor is positioned in the key field.

During the generating of the program, a search is performed of the realm using the key values stated in low/high limit in the screen picture 'Use of Program Keys'.

7.5.19. CPFRTXT

Routine name: **CPFRTXT** FORTRAN
CPFRTXC COBOL

Parameters :

COBOL	FORTRAN	I/O	Explanation
DDC-REF-TABLE	REFTAB,	I	See ABM manual chapter 7/6.
SCREEN-NAME	CTYPNAM,	I/O	Name of TT mark in picture.
ITEM-VALUE	TXTNR,	I/O	Text number.
SCREEN-VALUE	CTYPE,	I/O	Value of SCV-xx-TTYPE/ITTYPE.
DDS-xx-SUBSCHEMA	MITEMxx,	I	ABM manual chapter 6/7.
SCV-xx	MRECxx,	I	ABM manual chapter 6/7.
ITEM-NAME	INAME,	I	Name of DB item with value TXTNR.
DDS-zz-SUBSCHEMA	MITEMzz,	I	ABM manual chapter 6/7.
SCV-zz	MRECzz,	I	ABM manual chapter 6/7.
DDS-xx-SUBSCHEMA	MITEMxx,	I	ABM manual chapter 6/7.
SCV-xx	MRECxx,	I	ABM manual chapter 6/7.
MAINTAB(4)	MAINTAB(4)	I	Line number in the picture.
INFOTXT	INFOTXT		Information may be placed here.

Routine description:

Calls free text function. Checks if it is permitted to call the function. If it is not permitted, a message is displayed and you are returned to the calling function.

When querying in the calling function, you may only query in the free text function.

When modifying in the calling function, you may query, store, modify, and delete in the free text function.

When returned, you are returned to the place in the picture from where the function was called.

In the free text function, you may store N lines of text per free text.

7.5.20. CPTDISP/CPTDISC

Routine name: **CPTDISP** FORTRAN
CPTDISC COBOL

Parameters :

COBOL	FORTRAN	I/O	Explanation
DDC-REF-TABLE	REFTAB,	I	See ABM manual chapter 7/6.
DDS-xx-SUBSCHEMA	MITEMxx,	I	ABM manual chapter 6/7.
SCV-xx	MRECxx,	I	ABM manual chapter 6/7.
SVC-xx-TTYPE	XXTTYPE,	I	Screen picture field where 'T' is to be written.
'TTYPE '	"XXTTYPE",	I	Name of screen picture field.
DDB-zz-SUBSCHEMA	KITEMZZ,	I	ABM manual chapter 5/4.
DBV-ZZ	KRECZZ,	O	ABM manual chapter 5/4.
DBV-zz-TNR	ZZTTNR,		Database buffer values for the free text realm.
EVERYLIN	EVERYLIN	I	See page 94.

Routine description: Displays a 'T' in the field XXTTYPE when the line in the screen picture has additional text connected to it. If not, the field is blank.

CHAPTER 8

PROGRAM LOGIC IN THE GENERATED PROGRAM

8. PROGRAM LOGIC IN THE GENERATED PROGRAM

This chapter is written especially for FORTRAN, but the principles also are relevant to COBOL.

The following topics are important for understanding how the generated program works:

- COMTAB
- subroutine CPENABLE(flag)
- subroutine CPDISABL(flag)

8.1. COMTAB

COMTAB is a table defined in FORTRAN as INTEGER*4 COMTAB(8,2), where

'8' specifies the maximum number of regions (0-7) (at present, only 2 regions are available), and

'2' specifies that there are two words (of 32 bits each) containing information about each region in the picture.

COMTAB's function

The flags, which describe each region, are represented by one bit each in the COMTAB elements.

COMTAB(n,1)

COMTAB(n,1) is important for understanding the program logic. This part of the table is used by the program to describe which routines in the generated program are to be executed. Each routine call in the DO loop causes COMTAB(n,1) to be used, in order to test if the routine in question is to be executed.

After an error exit, or when X(exit) is pressed in the command field, COMTAB(n,1) will be reset to 0. This indicates that no routine is ready for execution, and the DO loop is therefore ended.

COMTAB(n,2)

COMTAB(n,2) contains a description of each region, based upon the layout of the screen picture and values specified in the 'Use of Program Keys' picture.

The routine names with unambiguous bits in COMTAB(n,1) are listed together with their bit numbers in this table:

COMTAB(n,1)	
Routine	Bit
CPREGION	31
CPACTCOD	30
CPRSPNS	29
CPKEY	28
CPREAD	27
CPOKCOD	26
CPUPDATE	25
CPGET	24
CPDISPL	23
CPBTRANS/	22
CPETRANS	22
OK-flag	21
CPFRTXT	20
Unused	19 - 0

(OK flag is used when one record is fetched at a time.)

Below, we describe the meaning of each of the bits in COMTAB(n,2):

COMTAB(n,2)		
Bit	Flag	Significance
31	FLREADBK	Specifies whether action code is to be read or not.
30	FLREADOK	Specifies whether OK code is to be read.
29	FLSUBLEV	Determines whether current region has members.
28-26	FLOWNER	Specifies which region, if any, is the owner of the current region.
25-23	FLMAXKEY	Maximum number of search keys in the region (max.7)
22-20	FLKEYNR	Number of current key field.
19-16	FLLEGACT	Specifies which action codes are permitted. 19 : querying 18 : storing 17 : modifying 16 : deleting
15	FLTXT	Specifies whether free text is in use or not.
14-0	unused	

8.2. CPENABLE(FLAG) AND CPDISABL(FLAG)

These routines call two FORTRAN library routines which set and reset flags respectively.

The purpose of the routines

The purpose of CPENABLE and CPDISABL is to activate the different routines.

When a flag is set, it means that the corresponding routine is executed when the program gets to this routine.

When a flag is reset, it means that the corresponding routine is to be skipped during further execution, until the routine's flag is set again.

8.3. SUBROUTINE CALLS

As mentioned before, the program will run in a loop until the user presses the EXIT key, or until an error occurs. Each run executes calls of each subroutine which is part of the DO loop.

Whether the subroutine in question is to be executed in full or not depends on the corresponding bit in COMTAB(n,1) being set. At the beginning of each subroutine, this is tested. If the correct bit is not set, the subroutine terminates and returns to the main program, and the next subroutine is called. If the correct bit is set, this bit will be reset immediately, and the entire subroutine can be executed.

8.4. HOW THE INDIVIDUAL BITS ARE SET

When the program is started, the flag for the routine CPREGION in the first region is set. Afterwards, the flags will be set and reset according to which command or navigation is given in the screen picture.

An example:

If you type in a command for the next region, a flag is set for CPREGION in the next region. If you type in a command for a new search key, a flag for CPKEY will be set (i.e. new read of key).

8.5. THE USE OF FLAGS IN THE COMPLETE-PG ROUTINES

Let us have a closer look at the use of flags in Complete-PG routines, and what criteria must be fulfilled before the routines are executed.

We shall also look at which flags the routine sets when it is executed, and which flags are set if it is not executed.

In the column 'Flag reset', it is specified whether or not the flag is turned off when the routine is executed.

ROUTINE	Criteria for execution	Flag reset
CPREGION	FLCOMMAN is set or FLCOMMAN set and MAINTAB(1)=MAINTAB(3).	YES
CPAKTCOD	FLAKTCOD is set.	YES
CPKEY	FLKEY is set.	YES
CPKEYNK	FLGET is set.	NO
CPGET	FLGET is set.	YES
CPDISP	FLDISPLY is set.	YES
CPREAD	FLREAD is set and READCO=READP.	YES
CPOKCOD	FLOKCOD is set.	YES
CPBTRANS	FLTRANS is set.	NO
CPETRANS	FLTRANS is set.	YES
CPUPDATE	FLUPDATE is set.	YES
CPRSPNS	FLRESPON is set.	YES
CPFRTXT	FLFRTXT is set.	YES
CPOTHER	EXECUTE is set to .TRUE./1 .	NO
CPEXIST	EXECUTE is set to .TRUE./1 .	NO

Setting of flags in the routines:

If the requirements for the routine to be executed are fulfilled, and the routine is executed without any error occurring, then EXECUTE is set to .TRUE./1 (both in FORTRAN and COBOL).

If the routine is not to be executed, then EXECUTE is set to .FALSE./0 (both in FORTRAN and COBOL).

If the routine is executed, the following flags are set in addition to EXECUTE:

ROUTINE	Normally executed	Display several regions	Flags that may be set in certain cases
CPREGION	If BK field in picture: FLAKTCOD, otherwise FLKEY.		
CPAKTCOD	PREV/FLCOMMAN or NEXT/FLKEY.		
CPKEY	PREV/FLAKTCOD or NEXT/CPREAD as well as FLGET and FLDISPLY.		When KEY is changed, FLKEY is set. FLRESPON. FLFRTXT.
CPKEYNK	No flags are set.		
CPGET			When EVERYLIN=1 FLGET is set until page is full. FLCOMMAN and FLRESPON are set when scrolling and no more records found. FLKEY if record does not exist.
CPDISP		When scrolling: sets FLGET, FLDISPLY in a subregion, if any.	FLRESPON.
CPREAD	PREV or NEXT. FLUPDATE and FLTRANS if EXECUTE key is hit		FLFRTXT. FLRESPON.
CPOKCOD	PREV can be set. FLUPDATE and FLTRANS if EXECUTE key is hit after 'Y'.		FLFRTXT.
CPBTRANS	No flags are set.		
CPETRANS	No flags are set.		
CPUPDATE	No flags are set.		
CPRSPNS	Can set most flags depending on navigation and position in picture.		FLKEY. FLFRTXT. FLGET. FLDISPLY. FLRESPON. FLCOMMAN.
CPFRTXT	Sets MAINTAB(6), last used routine.		
CPOTHER	Can set FLREAD.		
CPEXIST	Can set FLREAD.		FLKEY when deleting.

CHAPTER 9

INSTALLATION

9. INSTALLATION

This chapter gives an overview of the necessary preparations before using Complete-PG.

9.1. BASIC SOFTWARE REQUIREMENTS

In order to run Complete-PG, you must have the following or newer versions of ND software:

SYSTEM:	VERSION/RELEASE:
PED or WP	
ABM	C
SIBAS II	E
FOCUS	G
GPM	
JEC	B
USER-ENVIRONMENT	B

9.2. CP-SPEC AND CP-PROGEN

CP-SPEC:SYMB

This file contains installation parameters that must be initiated for each project:

progen : short name for the system, is shown in the heading of each generated program.

pgver : version name for the system, is shown in the heading of each generated program.

decuser : indicates where the generated DEC files from ABM are stored.

assuser : indicates where the generated ASS files from ABM are stored.

csysn : system name that is shown in the upper left-hand corner of the screen picture for each generated function.

formfil : name of the file where all the pictures are stored.

brfuser : where the BRF/NRF versions of the programs are to be stored.

symbuser : where the SYMB versions of the programs are to be stored.

workuser : the user area where one is working.

proguser : indicates where the PROG/DOMAIN versions of the programs are to be stored.

CP-PROGEN:MCRO

CP-PROGEN contains the procedure for generating, compiling and loading the program to be generated.

If you have symbolic versions, BRF/NRF versions or similar on various users, then you need to type in the user names for these files. In addition, you need to insert any personal subroutine libraries in the load procedure, as well as libraries such as COBOL, FORTRAN, SIBAS (together with the correct user names).

You must also create the files CP-LIST:LIST and CP-ERROR:LOGG on the workuser area:

CP-LIST:LIST

Generating, compiling and loading is run on BATCH-processor number 1. Output from this job is stored on CP-LIST:LIST.

CP-ERROR:LOGG

On this file, all the error messages from SIBAS, FOCUS and SINTRAN are gathered, if they occurred during the execution of the programs.

When an error occurs during the execution of a program, you will get a message about this at the bottom of your screen. Press any key to terminate the program.

A detailed error message is written to CP-ERROR:LOGG.

> NOTE:
> Make sure the files CP-LIST and CP-ERROR are cleared from time to time.

When using free text:

If the free text function is to be used in programs generated by Complete-PG, the realm D3TEXT has to be inserted in the database the programs are going to access. This can be done by running the redefinition file CP-REDEF-TEXT:SYMB.

Before running the file, type in the name of the database, OS file name and system realm:

- Type in the necessary information on the redefinition file CP-REDEF-TEXT:SYMB.
- Run CP-REDEF-TEXT:SYMB.

When using the advanced HELP function:

If you want to use the advanced HELP function (see chapter 13), you have to redefine the database by means of the file CP-REDEF-HELP:SYMB.

Before running the file, type the name of the database, OS file name, and system realm.

CHAPTER 10

A PROGRAMMING EXAMPLE

10. A PROGRAMMING EXAMPLE

We will demonstrate the use of Complete-PG by using a programming example. The example will show a function for maintenance of firms and their employees.

We have four realms:

- Employee realm
- Firm realm
- Postal code/area realm
- Position category realm

The database structure is as follows:

10.1. DATABASE DESCRIPTION

Database description Realm: **A1ANSAT - EMPLOYEE REALM**

- Contains all necessary information about each employee in each firm.

Itemname	Term	Type	Char	Explanation
A1ANSNR	EMPLOYEE NUMBER	N	4	Employee number
A1ENAVN	SNAME	AN	15	Surname
A1FNAVN	FNAME	AN	20	First name
A1STILL	POSITION	AN	10	Position title
A1BEDNR	FIRM NUMBER	N	4	Firm number
A1BENAV	FIRM NAME	AN	30	Name of firm
A1INTLF	EXTENSION	N	4	Extension
A1KODE	CODE	N	1	Code number
A1TTNR	TEXT P	N	9	Free text number for person
A1ATTNR	TEXT A	N	9	Free text number for seniority

Key: A1BEDAN AN
A1BEDNR
A1ANSNR

A1STILL AD

A1NAVNE AD
A1BEDNR
A1ENAVN

Database description Realm: **B1BEDRI - FIRM REALM**

- Alle firms of interest are stored here.

Itemname	Term	Type	Char	Explanation
B1BEDNR	FIRM NUMBER	N	4	Firm number
B1BENAV	FIRM NAME	AN	30	Name of firm
B1KONAV	SHORT NAME	AN	8	Short name of firm
B1ADRES	ADDRESS	AN	30	Address of firm
B1POSNR	POSTAL CODE	N	4	Postal code
B1TLFNR	TELEPHONE NUMBER	N	8	Telephone number of firm
B1KODE	CODE	N	1	Code number
B1SOPD	not screen field	N	8	Date when firm realm was last updated

Key:

B1BEDNR	AN
B1KONAV	AD
B1POSNR	AD

Database description Realm: **P1POST - POSTAL CODE REALM**

- All postal codes of interest are stored here.

Itemname	Term	Type	Char	Explanation
P1POSNR	POSTAL CODE	N	4	Postal code
P1PONAV	POSTAL AREA	AN	30	Name of postal area

Key: P1POSNR AN

Database description Realm: **S1STILL - POSITION CATEGORY REALM**

- Contains all position categories of interest.

Itemname	Term	Type	Char	Explanation
S1STILL	POSITION	AN	10	Position category

Key: S1STILL AN

10.2. MAINTAINING FIRMS AND THEIR EMPLOYEES

On the basis of this database structure we are going to make the following function:

Purpose:	The function is used to store and maintain firms and employees within each firm.
Type:	On-line, updating
Main keys:	Firm: Firm number Employee: Firm number + Employee number
Scrolling keys:	Firm: Short name Employee: Firm number + Surname
Automatic transfer to other functions:	None.
Automatic transfer from other functions:	None.
Use of function:	STORING : In accordance with the program generator. MODIFYING : In accordance with the program generator. DELETING : In accordance with the program generator. QUERYING : In accordance with the program generator.
Messages:	151 : "0:-> Surname must be specified.'" 152 : "0:-> Position category does not exist in the system.'" 153 : "0:-> Which text do you want (P=1,S=2) : '"
Use of database:	(C=Create, M=Modify, D=Delete, R=Retrieve) B1BEDRI - Firm realm C M D R A1ANSAT - Employee realm C M D R P1POST - Postal code/area realm R S1STILL - Position category realm R

10.3. THE SCREEN PICTURE:

This screen picture is designed in the screen picture part of ABM:

```
COURSE SYSTEMS            MAINTENANCE          yy.mm.dd   99:99
                             FIRM

---------------------------------------------------------------

Number of firm : ....   Firm    : ............................
Short name:........     Address : ............................
Postal code: ....       Area    : ............................
Tel.: ........          Code    : .

---------------------------------------------------------------
Employee :
                                            Code
No.        Surname          First name        v Position   Ext.
....       ................ ................... . ........... ....
....       ................ ................... . ........... ....
....       ................ ................... . ........... ....
....       ................ ................... . ........... ....
....       ................ ................... . ........... ....
....       ................ ................... . ........... ....
....       ................ ................... . ........... ....
....       ................ ................... . ........... ....
....       ................ ................... . ........... ....
```

10.4. DESCRIPTION OF FIELDS:

TERM	MEANING	INPUT CONTROL	REALM REFERENCE
NUMBER OF FIRM	Number belonging to each firm that is registered.		B1BEDRI
FIRM	Official name of the firm.		B1BEDRI
SHORT NAME	Short name of the firm.		B1BEDRI
ADDRESS	Address of the firm.		B1BEDRI
POSTAL CODE	Postal code of the firm.		B1BEDRI
AREA	Name of the postal area.		B1BEDRI
TEL	Telephone number of the firm. Also includes the area code.	Legal values: 01100000 - 09999999	B1BEDRI
CODE		1.....9	B1BEDRI
EMPLOYEE NO.	Internal number of each employee.		A1ANSAT
CODE		1.....9	A1ANSAT
SURNAME	The surname of each employee.		A1ANSAT
FIRST NAME	The first name of each employee.		A1ANSAT
POSITION	Name of position of each employee.		A1ANSAT
EXT.	Extension for each employee.	Legal values: 1000 - 9999	A1ANSAT

10.5. HELP PICTURES:

None.

10.6. FILLING IN THE SCREEN PICTURES

Before you fill in the two screen pictures in Complete-PG, you have to do the following:

- Define data types and database in ABM.
- Design the picture in the screen picture part of ABM.
- Define subschema and subfunction.

In this example, we have chosen to give the picture, subschema and subfunction the same name: **AJBEDR**.

- Call Complete-PG from the command line in ABM by typing:

ABM command: COMPLETE-PG↵

The first picture of Complete-PG will then appear on the screen.

- Fill in the Program Description picture.

Below, this picture is filled in for the program that is to be generated:

```
 PG  > .  |              P R O G R A M   D E S C R I P T I O N
-----------------------------------------------------------------
Program identification.
 subfunction   : AJBEDR   subschema: AJBEDR   form: AJBEDR
 author        : ERIC BROWN
 program id    : AJBEDR
 explanation   : MAINTENANCE OF FIRMS AND THEIR EMPLOYEES.
               :
               :

Parameters for generate.
 object language : FORT
 object filename : 101-AJBEDR
 load procedure  : GENERATE/COMPILE

Date of
   creation      : 86 05 22            last modification : 00 00 00
                                       last generated    : 00 00 00
```

All the fields that appear in the two screen pictures in Complete-PG, are described in chapter 4.

In the field 'load procedure' we have chosen GENERATE/COMPILE. We have not inclued LOAD, because this function is to be loaded together with a menu system, for example TRUE, which is not described here.

When this picture is registered, the picture Use of Program Keys will appear on the screen.

Here you see the picture already filled in for the two regions. Note that MK is a search key in both regions.

Region 1 :

```
PG  >.          U S E   O F   P R O G R A M   K E Y S

Subfunction: AJBEDR          fieldrecord: M1A
       name: BEDRIFT         owner:
       okcode: N             textfunction: N     action codes: 1234

Realm    Key      Use D Ex Realm    Key      Use D Ex  Realm  Key  Use D Ex
B1BEDRI  B1BEDNR  MK       A1ANSAT  A1STILL
B1BEDRI  B1KONAV  AK       A1ANSAT  A1NAVNE
B1BEDRI  B1POSNR
A1ANSAT  A1BEDAN

Initial values for realm: B1BEDRI   key: B1BEDNR

  Item     Lowlimit     Highlimit       Item     Lowlimit     Highlimit
  B1BEDNR  M1BEDNR
```

Region 2 :

```
PG  >.          U S E   O F   P R O G R A M   K E Y S

Subfunction: AJBEDR          fieldrecord: M2A
       name: EMPLOYEE        owner: BEDRIFT
       okcode: N             textfunction: N     action codes: 1234

Realm    Key      Use D Ex Realm    Key      Use D Ex  Realm  Key Use D Ex
B1BEDRI  B1BEDNR           A1ANSAT  A1STILL
B1BEDRI  B1KONAV           A1ANSAT  A1NAVNE  AK
B1BEDRI  B1POSNR
A1ANSAT  A1BEDAN  MK

 Initial values for realm: A1ANSAT   key: A1BEDAN

  Item     Lowlimit     Highlimit       Item     Lowlimit     Highlimit
  A1BEDNR  M1BEDNR      M1BEDNR
  A1ANSNR  M2ANSNR
```

When you have filled in both 'Use of Program Keys' pictures, do the following:

- Return to the first picture (Program Description) by pressing E in the command field.
- Start the program generating by pressing X in the command field in the Program Description picture.
- The generating of the program and the compiling will now be started on batch processor 1, with the (Work user)CP-LIST:LIST file as output file.

10.7. THE RESULTING GENERATED PROGRAM

The resulting program will look like this in FORTRAN (see page 134 for COBOL):

```
C=====================================================================
      SUBROUTINE AJBEDRIV
C=====================================================================
C
C     Programmer : ERIC BROWN

C
C
C=====================================================================

      CHARACTER CTYPE*2, CNAME*8, CTYPNAM*8, INFOTXT*50
      INTEGER*4 TXTNR
      INTEGER   CPCURKK, EVERYLIN, CPKEYNK
      INTEGER*2 IBKODE1,IBKODE2,IOKCOD1,IOKCOD2, INAME(4),TYPNAM(4)
      LOGICAL   ENTEXT
      EQUIVALENCE (CNAME,INAME),
     +            (CTYPNAM,TYPNAM)
      $INCLUDE (abm-user)CP-TRIGGER-TAB:INCL
      $INCLUDE (abm-user)CP-PROBOT-COM:INCL

C     %%% ABM interface

      $INCLUDE (k-1-abm)DECDDI-AJBEDRIV

C     %%% ABM interface

      $INCLUDE (k-1-abm)ASSDDI-AJBEDRIV


C     %%% Init picture name
      FORMFILE    =  '(k-1-abm)KURS:FABM'

C     %%% Init leaving field function
      NEXTFI      =  'Y'


      EVERYLIN = 0

C     %%%  Init function name, project name, no. of regions and language
      CMAIN        =  'AJBEDRIV'
      CSYSTEM      =  'K-1 kurs'
      MAINTAB(2)   =  2
      LANGUAGE     = 'FORT'
C     %%%  Make help function available
      CALL CPHJON


C     %%% Description and name of region 1
      COMTAB(1,2)   =    555679744
      NAMTAB(1)     =  'Firm
```

```
C     %%% Description and name of region 2
      COMTAB(2,2)    =     85950464
      NAMTAB(2)      =  'Employee

C     %%% Get picture

      CALL CPBEGIN(REFTAB,KNREA,KREALMS,KUMOD,KPMOD)

C     %%% Loop until an error occurs or "EXIT" key is pressed


      DO WHILE (MAINTAB(1).NE.0)

       DO WHILE (COMTAB(1,1).NE.0 .AND. MAINTAB(1).NE.0)

         EVERYLIN = 0
         CALL CPREGION(REFTAB,1,MITEMM1,MRECM1,FLACTCOD)

C     %%% Main key item list region 1


         OVITEM( 1) = 'B1BEDNR'
         OVANT =  1

C               %%% Read keys

         IF (CPCURKK().EQ. 1) THEN
            CITMSUB(1) = '+:M1BEDNR *
         ENDIF
         IF (CPCURKK().EQ. 2) THEN
            CITMSUB(1) = '+:M1KONAV *
         ENDIF
         CALL CPKEY(REFTAB,MITEMM1,MRECM1,FLACTCOD,FLREAD)

C               %%% Get record

         IF (CPKEYNK().EQ. 1) THEN
            LB1BED1 = M1BEDNR
         ENDIF
         IF (CPKEYNK().EQ. 2) THEN
            LB1KON1 = M1KONAV
         ENDIF

         CALL CPGET(EVERYLIN,REFTAB, 1,KIB1BED,KVB1BED,
     *              MITEMM1,MRECM1,KITEMB1)

         CALL CPGET(EVERYLIN,REFTAB, 2,KIB1KON,KVB1KON,
     *              MITEMM1,MRECM1,KITEMB1)


         IF (EXECUTE .AND. IACTCOD.NE.2 .AND. EVERYLIN.EQ.0) THEN
            I = REFTAB(7)
C           %%% Get data from database buffer
            CALL CPINRC(I,KITEMB1,KRECB1)
         ELSEIF (EXECUTE .AND. IACTCOD.NE.2 .AND. EVERYLIN.EQ.1) THEN
            I = 1
C           %%% Get data from database buffer
            CALL CPINRC(I,KITEMB1,KRECB1)
         ENDIF
```

```
C         %%% Display on screen

          CITMSUB(1) = '-:*'
          CALL CPDISP(REFTAB,MITEMM1,MRECM1)

          FLNEXT = FLOKCOD

C                %%% Read from screen

          CITMSUB(1) = '-:M1BEDNR *

          CALL CPREAD(1,REFTAB,MITEMM1,MRECM1,FLKEY,FLNEXT)


C         %%% Begin transaction

          CALL CPBTRANS(MITEMM1,MRECM1,KITEMB1,KRECB1)

C         %%% Update record
          CITMSUB(1) ='O:*'

          CALL CPUPDATE(ITEMSUB,KITEMB1,KRECB1)

C         %%% End transaction

          CALL CPETRANS

C         %%% Read response kode

          CALL CPRSPNS(REFTAB,MITEMM1,MRECM1)
          CALL CPOVER(MITEMM1,MRECM1,KITEMB1,KRECB1)

C         %%%  Swap to help application
          CALL CPHELP(REFTAB,MITEMM1,MRECM1,MITEMM2,MRECM2)


        ENDDO

        DO WHILE (COMTAB(2,1).NE.O .AND. MAINTAB(1).NE.O)

          EVERYLIN = O
          CALL CPREGION(REFTAB,2,MITEMM2,MRECM2,FLACTCOD)

C      %%% Main key item list region 2


          OVITEM( 1) = 'A1BEDNR'
          OVITEM( 2) = 'A1ANSNR'
          OVANT =  2

C                %%% Read keys

          IF (CPCURKK().EQ. 1) THEN
             CITMSUB(1) = '+:M2ANSNR *
          ENDIF
          IF (CPCURKK().EQ. 2) THEN
             CITMSUB(1) = '+:M2ENAVN *
          ENDIF
          CALL CPKEY(REFTAB,MITEMM2,MRECM2,FLACTCOD,FLREAD)
```

```
C               %%% Get record

        LA1BED1 = M1BEDNR
        LA1NAV1 = M1BEDNR
        HA1BED1 = M1BEDNR
        HA1NAV1 = M1BEDNR
        IF (CPKEYNK().EQ. 1) THEN
           LA1BED2 = M2ANSNR
        ENDIF
        IF (CPKEYNK().EQ. 2) THEN
           LA1NAV2 = M2ENAVN
        ENDIF

        CALL CPGET(EVERYLIN,REFTAB, 1,KIA1BED,KVA1BED,
     *             MITEMM2,MRECM2,KITEMA1)

        CALL CPGET(EVERYLIN,REFTAB, 2,KIA1NAV,KVA1NAV,
     *             MITEMM2,MRECM2,KITEMA1)


        IF (EXECUTE .AND. IACTCOD.NE.2 .AND. EVERYLIN.EQ.0) THEN
           I = REFTAB(7)
C          %%% Get data from database buffer
           CALL CPINRC(I,KITEMA1,KRECA1)
        ELSEIF (EXECUTE .AND. IACTCOD.NE.2 .AND. EVERYLIN.EQ.1) THEN
           I = 1
C          %%% Get data from database buffer
           CALL CPINRC(I,KITEMA1,KRECA1)
        ENDIF

C       %%% Display on screen

        CITMSUB(1) = '-:*'
        CALL CPDISP(REFTAB,MITEMM2,MRECM2)

        FLNEXT = FLOKCOD

C               %%% Read from screen

        CITMSUB(1) = '-:M2TTYPE M2ANSNR *

        CALL CPREAD(1,REFTAB,MITEMM2,MRECM2,FLKEY,FLNEXT)

        A1BEDNR  = M1BEDNR

C       %%% Begin transaction

        CALL CPBTRANS(MITEMM2,MRECM2,KITEMA1,KRECA1)

C       %%% Update record
        CITMSUB(1) ='O:*'

        CALL CPUPDATE(ITEMSUB,KITEMA1,KRECA1)

C       %%% End transaction

        CALL CPETRANS

C       %%% Read response kode
```

```
          CALL CPRSPNS(REFTAB,MITEMM2,MRECM2)
          CALL CPOVER(MITEMM2,MRECM2,KITEMA1,KRECA1)

C         %%%  Swap to help application
          CALL CPHELP(REFTAB,MITEMM2,MRECM2,MITEMM2,MRECM2)


        ENDDO

       ENDDO

C      %%% Exit from program

       CALL CPEND


       END
```

The program is now ready to be loaded and run.

We also include an example of the equivalent COBOL program:

```
*****************************************************************
*
 IDENTIFICATION DIVISION.
 PROGRAM-ID.            AJBEDRIV
 AUTHOR.                ERIC BROWN

*
*****************************************************************
*
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.       ND.
 OBJECT-COMPUTER.       ND.
*

 DATA DIVISION.
 WORKING-STORAGE SECTION.
     COPY (pg-2C)CP-TRIGGER-TAB:COPY
*
*         ABM interface
*
 01  DECDDC-INFO.
     COPY (k-1-abm)DECDDC-AJBEDRIV
*
*
*         Local declarations.
*
 77  CURRENT-KEY-NO                COMP.
 77  KEY-NO                        COMP.
 77  NULL                          COMP       VALUE 0.
 77  INDX                          COMP.
 77  STOPP                         COMP.
 77  ITEM-VALUE                    COMP.
 77  EVERYLIN                      COMP       VALUE 0.
 77  ENTEXT                        COMP.
 77  SCV-DUMMY-BKODE     PIC X(1).
 77  SCV-DUMMY-OKCOD     PIC X(1).
 77  ITEM-NAME           PIC X(8).
 77  SCREEN-NAME         PIC X(8).
 77  SCREEN-VALUE        PIC X(2).
 77  INFOTXT             PIC X(50).

*
*
*
 LINKAGE SECTION.
*
     COPY (pg-2c)CP-PROBOT-COM:COPY

 PROCEDURE DIVISION.
*
*
 MAIN SECTION.
 START-MAIN-10.
     PERFORM STARTUP.
     PERFORM REGION UNTIL NULL = MAINTAB (1).
     PERFORM ROUNDUP.
```

```
 EXIT-MAIN-99.
     EXIT PROGRAM.


 STARTUP SECTION.
 START-STARTUP-10.
*
*         ABM interface
*
     COPY (k-1-abm)ASSDDC-AJBEDRIV


*    Init form file name
*
     MOVE '(k-1-abm)KURS:FABM' TO FORMFILE.

*    Init leaving field function
     MOVE 'Y' TO NEXTFI.


*    Init function name, project name, no. of regions and language
*
     MOVE 'AJBEDRIV' TO CMAIN.
     MOVE 'K-1 kurs' TO CSYSTEM.
     MOVE  2  TO MAINTAB(2).
     MOVE 'COBL' TO LANGUAGE.


*    Description and name of region 1
*
     COMPUTE COMTAB(2, 1) =   555679744 .
     MOVE 'Firm            ' TO NAMTAB(1).


*    Description and name of region 2
*
     COMPUTE COMTAB(2, 2) =    85950464 .
     MOVE 'Employee        ' TO NAMTAB(2).

*
*         Set help function available.
*
     CALL 'CPHJON'
*
*         Get picture
*
     CALL 'CPBEGIN'  USING DDC-REF-TABLE, DBR-NO-OF-REALMS,
                           DBR-REALM-NAMES, DBR-REALM-USAGE(1),
                           DBR-REALM-PROTECT(1).

 EXIT-STARTUP-99.
     EXIT.


 REGION SECTION.
 START-REGION-10.
     IF NULL NOT EQUAL COMTAB(1, 1)
         PERFORM REGION-1.
     IF NULL NOT EQUAL COMTAB(1, 2)
         PERFORM REGION-2.
```

```
 EXIT-REGION-99.
     EXIT.


 REGION-1 SECTION.
 START-REGION1-10.
*
*         Define region
*
     MOVE 0 TO EVERYLIN.
     CALL 'CPREGION'  USING DDC-REF-TABLE, 1, DDS-M1-SUBSCHEMA,
                            SCV-M1, TRIGGER-ACTCODE.

*    Main key item list region 1


     MOVE 'B1BEDNR' TO OVITEM( 1).
     MOVE  1 TO OVANT.

*         %%% Read keys

     CALL 'CPCURKC' USING CURRENT-KEY-NO.
     IF CURRENT-KEY-NO =  1
         MOVE '+:B1BEDNR *'
              TO DDC-SELECT.
     IF CURRENT-KEY-NO =  2
         MOVE '+:B1KONAV *'
              TO DDC-SELECT.

*
     CALL 'CPKEY' USING DDC-REF-TABLE, DDS-M1-SUBSCHEMA,
                        SCV-M1,
                        TRIGGER-ACTCODE, TRIGGER-READ.

*         %%% Get record

     CALL 'CPKEYNC' USING KEY-NO.
     IF KEY-NO =  1
         MOVE SCV-M1-B1BEDNR    TO DBKV-B1BEDRI-B1BEDNR-LOW-1.
     CALL 'CPKEYNC' USING KEY-NO.
     IF KEY-NO =  2
         MOVE SCV-M1-B1KONAV    TO DBKV-B1BEDRI-B1KONAV-LOW-1.

*
     CALL 'CPGET' USING EVERYLIN, DDC-REF-TABLE, 1,
                        DBKI-B1BEDRI-B1BEDNR ,
                        DBKV-B1BEDRI-B1BEDNR , DDS-M1-SUBSCHEMA,
                        SCV-M1, DDB-B1BEDRI-SUBSCHEMA.

*
     CALL 'CPGET' USING EVERYLIN, DDC-REF-TABLE, 2,
                        DBKI-B1BEDRI-B1KONAV ,
                        DBKV-B1BEDRI-B1KONAV , DDS-M1-SUBSCHEMA,
                        SCV-M1, DDB-B1BEDRI-SUBSCHEMA.

     IF EXECUTE = 1
         AND EVERYLIN = 0
         AND MAINTAB(5) NOT = 2
         MOVE SCC-START-RW-LINE TO INDX
```

```
*        Get data from database buffer
         CALL 'CPINRC' USING INDX, DDB-B1BEDRI-SUBSCHEMA,
                             DBV-B1BEDRI
     ELSE
         IF EXECUTE = 1
            AND EVERYLIN = 1
            AND MAINTAB(5) NOT = 2
            MOVE 1 TO INDX
*           Get data from database buffer
            CALL 'CPINRC' USING INDX, DDB-B1BEDRI-SUBSCHEMA,
                                DBV-B1BEDRI.

*
*       Display on screen
*
     MOVE  '-:*'   TO DDC-SELECT.
     CALL 'CPDISP' USING DDC-REF-TABLE, DDS-M1-SUBSCHEMA,
                         SCV-M1.

     MOVE TRIGGER-OKCODE TO TRIGGER-NEXT.

*         %%% Read from screen

     MOVE '-:B1BEDNR *'
         TO DDC-SELECT.

     CALL 'CPREAD' USING 1, DDC-REF-TABLE,  DDS-M1-SUBSCHEMA,
                         SCV-M1, TRIGGER-KEY, TRIGGER-NEXT.


*
*        Begin transaction
*
     CALL 'CPBTRANS' USING DDS-M1-SUBSCHEMA, SCV-M1,
                           DDB-B1BEDRI-SUBSCHEMA, DBV-B1BEDRI.
*
*         Update record
*
     MOVE 'O:*'   TO DDC-SELECT.

     CALL 'CPUPDATE' USING DDC-SELECT,
                           DDB-B1BEDRI-SUBSCHEMA, DBV-B1BEDRI.

*
*        End transaction
*
     CALL 'CPETRANS'.

*
*        Read response kode
*
     CALL 'CPRSPNS' USING DDC-REF-TABLE,
                          DDS-M1-SUBSCHEMA, SCV-M1.
     CALL 'CPOVER' USING DDS-M1-SUBSCHEMA, SCV-M1,
                         DDB-B1BEDRI-SUBSCHEMA, DBV-B1BEDRI.

*   %%%  Swap to help application
     CALL 'CPHELP' USING  DDC-REF-TABLE,
                          DDS-M1-SUBSCHEMA, SCV-M1,
                          DDS-M2-SUBSCHEMA, SCV-M2.
```

```
 EXIT-REGION1-99.
     EXIT.

 REGION-2 SECTION.
 START-REGION2-10.
*
*         Define region
*
     MOVE 0 TO EVERYLIN.
     CALL 'CPREGION'  USING DDC-REF-TABLE, 2, DDS-M2-SUBSCHEMA,
                            SCV-M2, TRIGGER-ACTCODE.

*    Main key item list region 2


     MOVE 'A1BEDNR' TO OVITEM( 1).
     MOVE 'A1ANSNR' TO OVITEM( 2).
     MOVE  2 TO OVANT.

*         %%% Read keys

     CALL 'CPCURKC' USING CURRENT-KEY-NO.
     IF CURRENT-KEY-NO =  1
         MOVE '+:A1ANSNR *'
              TO DDC-SELECT.
     IF CURRENT-KEY-NO =  2
         MOVE '+:A1ENAVN *'
              TO DDC-SELECT.

*
     CALL 'CPKEY' USING DDC-REF-TABLE, DDS-M2-SUBSCHEMA,
                        SCV-M2,
                        TRIGGER-ACTCODE, TRIGGER-READ.

*         %%% Get record

     MOVE SCV-M1-B1BEDNR    TO DBKV-A1ANSAT-A1BEDAN-LOW-1.
     MOVE SCV-M1-B1BEDNR    TO DBKV-A1ANSAT-A1NAVNE-LOW-1.
     MOVE SCV-M1-B1BEDNR    TO DBKV-A1ANSAT-A1BEDAN-HIGH-1.
     MOVE SCV-M1-B1BEDNR    TO DBKV-A1ANSAT-A1NAVNE-HIGH-1.
     CALL 'CPKEYNC' USING KEY-NO.
     IF KEY-NO =  1
         MOVE SCV-M2-A1ANSNR    TO DBKV-A1ANSAT-A1BEDAN-LOW-2.
     CALL 'CPKEYNC' USING KEY-NO.
     IF KEY-NO =  2
         MOVE SCV-M2-A1ENAVN    TO DBKV-A1ANSAT-A1NAVNE-LOW-2.

*
     CALL 'CPGET' USING EVERYLIN, DDC-REF-TABLE, 1,
                        DBKI-A1ANSAT-A1BEDAN ,
                        DBKV-A1ANSAT-A1BEDAN , DDS-M2-SUBSCHEMA,
                        SCV-M2, DDB-A1ANSAT-SUBSCHEMA.

*
     CALL 'CPGET' USING EVERYLIN, DDC-REF-TABLE, 2,
                        DBKI-A1ANSAT-A1NAVNE ,
                        DBKV-A1ANSAT-A1NAVNE , DDS-M2-SUBSCHEMA,
                        SCV-M2, DDB-A1ANSAT-SUBSCHEMA.
```

```
          IF EXECUTE = 1
              AND EVERYLIN = 0
              AND MAINTAB(5) NOT = 2
              MOVE SCC-START-RW-LINE TO INDX
*             Get data from database buffer
              CALL 'CPINRC' USING INDX, DDB-A1ANSAT-SUBSCHEMA,
                                  DBV-A1ANSAT
          ELSE
              IF EXECUTE = 1
                 AND EVERYLIN = 1
                 AND MAINTAB(5) NOT = 2
                 MOVE 1 TO INDX
*                Get data from database buffer
                 CALL 'CPINRC' USING INDX, DDB-A1ANSAT-SUBSCHEMA,
                                     DBV-A1ANSAT.

*
*        Display on screen
*
          MOVE  '-:*'   TO DDC-SELECT.
          CALL 'CPDISP' USING DDC-REF-TABLE, DDS-M2-SUBSCHEMA,
                              SCV-M2.

          MOVE TRIGGER-OKCODE TO TRIGGER-NEXT.

*          %%% Read from screen

          MOVE '-:TTYPE   A1ANSNR *'
              TO DDC-SELECT.

          CALL 'CPREAD' USING 1, DDC-REF-TABLE,  DDS-M2-SUBSCHEMA,
                              SCV-M2, TRIGGER-KEY, TRIGGER-NEXT.

          MOVE SCV-M1-B1BEDNR    TO DBV-A1ANSAT-A1BEDNR .

*
*         Begin transaction
*
          CALL 'CPBTRANS' USING DDS-M2-SUBSCHEMA, SCV-M2,
                                DDB-A1ANSAT-SUBSCHEMA, DBV-A1ANSAT.
*
*          Update record
*
          MOVE 'O:*'   TO DDC-SELECT.

          CALL 'CPUPDATE' USING DDC-SELECT,
                                DDB-A1ANSAT-SUBSCHEMA, DBV-A1ANSAT.

*
*         End transaction
*
          CALL 'CPETRANS'.

*
*         Read response kode
*
          CALL 'CPRSPNS' USING DDC-REF-TABLE,
                               DDS-M2-SUBSCHEMA, SCV-M2.
          CALL 'CPOVER' USING DDS-M2-SUBSCHEMA, SCV-M2,
```

```
                        DDB-A1ANSAT-SUBSCHEMA, DBV-A1ANSAT.

*   %%%  Swap to help-application
     CALL 'CPHELP' USING  DDC-REF-TABLE,
                          DDS-M2-SUBSCHEMA, SCV-M2,
                          DDS-M2-SUBSCHEMA, SCV-M2.


 EXIT-REGION2-99.
     EXIT.

 ROUNDUP SECTION.
 START-ROUNDUP-01.
*        Exit from program
*
     CALL 'CPEND'.
 EXIT-ROUNDUP-99.
     EXIT.
```

10.8. EXTENDING THE EXAMPLE

In order to show more of the possibilities in Complete-PG, we will now introduce a few changes in the function. We shall look at the following:

- Existence control
- Display of data from another realm
- Manual code
- Several CPREAD calls
- CPREAD calls in manual code
- CPINVER instead of several CPREAD calls
- Overruling of messages in PG routines
- Calculating fields
- Updating other realms
- Free text in the application
- Several free texts on the same record in the application
- Selection of records

10.8.1. EXISTENCE CONTROL

We do not want it to be possible to delete a firm that has employees registered. Complete-PG can deal with this automatically. The changes that have to be done are shown in a section of picture number 2 (Use of Program Keys) in Complete-PG, region 1 :

Realm	Key	Use	D	Ex	Realm	Key	Use	D	Ex	Realm	Key	Use	D	Ex
B1BEDRI	B1BEDNR	MK			A1ANSAT	A1STILL								
B1BEDRI	B1KONAV	AK			A1ANSAT	A1ENAVN								
B1BEDRI	B1POSNR													
A1ANSAT	A1BEDAN	K		E										

Initial values for realm: A1ANSAT key: A1BEDAN

Item	Lowlimit	Highlimit	Item	Lowlimit	Highlimit
A1BEDNR	M1BEDNR	M1BEDNR			
A1ANSNR					

We have marked the key A1BEDAN in use (K), and specified that we shall use it for existence control when deleting (E). The values we use for search towards A1ANSAT are firm number + employee number. A1ANSNR has not been given any low/high limit, because we use minimum/maximum.

When the change has been mande, the function has to be generated again. The change of program code as Complete-PG generates it, is shown in the following part of the program:

```
      CALL CPREAD(1,REFTAB,MITEMM1,MRECM1,FLKEY,FLNEXT)

      IF (EXECUTE .AND. IACTCOD.EQ.4) THEN

C        %%% Existence control against other realms

         LA1BED1 = M1BEDNR
         HA1BED1 = M1BEDNR

         CALL CPEXIST(KIA1BED,KVA1BED)
      ENDIF


C     %%% Begin transaction

      CALL CPBTRANS(MITEMM1,MRECM1,KITEMB1,KRECB1)
```

And the same example in COBOL:

```
          CALL 'CPREAD' USING 1, DDC-REF-TABLE,  DDS-M1-SUBSCHEMA,
                              SCV-M1, TRIGGER-KEY, TRIGGER-NEXT.

          IF 1 = EXECUTE
              AND MAINTAB(5) = 4


              MOVE SCV-M1-B1BEDNR TO DBKV-A1ANSAT-A1BEDAN-LOW-1
              MOVE SCV-M1-B1BEDNR TO DBKV-A1ANSAT-A1BEDAN-HIGH-1

     *        Existing control against other realms
              CALL 'CPEXIST' USING DBKI-A1ANSAT-A1BEDAN,
                                   DBKV-A1ANSAT-A1BEDAN.



     *
     *        Begin transaction
     *
          CALL 'CPBTRANS' USING DDS-M1-SUBSCHEMA, SCV-M1,
                                DDB-B1BEDRI-SUBSCHEMA, DBV-B1BEDRI.
```

10.8.2. DISPLAY OF DATA FROM ANOTHER REALM

You may want to fetch data from another realm, using a key value either from a field in the screen picture or from an item in the main realm in the region, and display this information on the screen. In this example we want the name of a post-office area to be fetched automatically from P1POST when we type in the postal code.

Realm, item and key for P1POST must be inserted in the subschema AJBEDR in ABM. In the picture, the field 'Area' must be changed to refer to the P1POST realm.

The changes that must be made in Complete-PG, are shown in a section of picture number 2 (Use of Program Keys), region 1 :

Realm	Key	Use	D	Ex	Realm	Key	Use	D	Ex	Realm	Key	Use	D	Ex
B1BEDRI	B1BEDNR	MK			A1ANSAT	A1STILL								
B1BEDRI	B1KONAV	AK			A1ANSAT	A1ENAVN								
B1BEDRI	B1POSNR				P1POST	P1POSNR	K	D						
A1ANSAT	A1BEDAN	K		E										

Initial values for realm: P1POST key: P1POSNR

Item	Lowlimit	Highlimit	Item	Lowlimit	Highlimit
P1POSNR	M1POSNR	M1POSNR			

Display in Complete-PG also works as an existence control. If the postal code that was typed in does not exist in P1POST, CPOTHER will call the CPREAD call again, if CPREAD was the last call before CPOTHER.

When the change has been made, the function must be generated again. The change in the program code as Complete-PG generates it, is shown in this section of the program:

```
      CALL CPGET(EVERYLIN,REFTAB, 2,KIB1KON,KVB1KON,
     *           MITEMM1,MRECM1,KITEMB1)


      IF (EXECUTE .AND. IACTCOD.NE.2 .AND. EVERYLIN.EQ.0) THEN
         I = REFTAB(7)
C        %%% Get data from database buffer
         CALL CPINRC(I,KITEMB1,KRECB1)
      ELSEIF (EXECUTE .AND. IACTCOD.NE.2 .AND. EVERYLIN.EQ.1) THEN
         I = 1
C        %%% Get data from database buffer
         CALL CPINRC(I,KITEMB1,KRECB1)
      ENDIF

C     %%% Get data from other realms
      IF (EXECUTE) THEN

         IF (NOFOUND.GT.1) REFTAB(7) = 1

C        %%% Loop for every line in region

         DO FOR I = REFTAB(7), REFTAB(7)+NOFOUND-1

C           %%% Get data from database buffer
            IF (EVERYLIN.EQ.0 .AND. MAINTAB(5).EQ.1) THEN
               CALL CPINRC(I,KITEMB1,KRECB1)
            ENDIF

C           %%% Get data from screen buffer
            REFTAB(7) = I
            CALL DDGETRC(REFTAB,MITEMM1,MRECM1,MSTA)
            IF (MSTA.NE.0) CALL CPABORT
C


         LP1POS1 = M1POSNR
         HP1POS1 = M1POSNR

C          %%% Get and display data from other realm
            CALL CPOTHER(REFTAB,KIP1POS,KVP1POS,KITEMP1,KRECP1,
     *                   MITEMM1,MRECM1)

         ENDDO

         IF (NOFOUND.GT.1) REFTAB(7) = 1
      ENDIF

C     %%% Display on screen

      CITMSUB(1) = '-:*'
      CALL CPDISP(REFTAB,MITEMM1,MRECM1)
```

The same example in COBOL:

```
        CALL 'CPGET' USING EVERYLIN, DDC-REF-TABLE,  2,
                           DBKI-B1BEDRI-B1KONAV ,
                           DBKV-B1BEDRI-B1KONAV , DDS-M1-SUBSCHEMA,
                           SCV-M1, DDB-B1BEDRI-SUBSCHEMA.

        IF EXECUTE = 1
            AND EVERYLIN = 0
            AND MAINTAB(5) NOT = 2
            MOVE SCC-START-RW-LINE TO INDX
*           Get data from database buffer
            CALL 'CPINRC' USING INDX, DDB-B1BEDRI-SUBSCHEMA,
                                DBV-B1BEDRI
        ELSE
            IF EXECUTE = 1
               AND EVERYLIN = 1
               AND MAINTAB(5) NOT = 2
               MOVE 1 TO INDX
*              Get data from database buffer
               CALL 'CPINRC' USING INDX, DDB-B1BEDRI-SUBSCHEMA,
                                   DBV-B1BEDRI.

*       Get data from other realms
*
        IF 1 = EXECUTE
            AND NOFOUND GREATER THAN 1
            MOVE 1 TO SCC-START-RW-LINE.

        IF 1 = EXECUTE

            COMPUTE STOPP = SCC-START-RW-LINE + NOFOUND - 1

*           Loop for every line in region
*
            DO FOR INDX FROM SCC-START-RW-LINE BY 1 TO STOPP

*               Get data from database buffer
                IF EVERYLIN = 0
                   AND MAINTAB(5) = 1
                   CALL 'CPINRC' USING INDX, DDB-B1BEDRI-SUBSCHEMA,
                                       DBV-B1BEDRI
                END-IF

*               Get data from screen buffer
                MOVE INDX TO SCC-START-RW-LINE
                CALL 'DDGETRC' USING DDC-REF-TABLE, DDS-M1-SUBSCHEMA,
                                     SCV-M1, MSTA


            MOVE SCV-M1-B1POSNR TO DBKV-P1POST-P1POSNR-LOW-1
            MOVE SCV-M1-B1POSNR TO DBKV-P1POST-P1POSNR-HIGH-1

*               Get and display data from other realms
                CALL 'CPOTHER' USING DDC-REF-TABLE,
                                     DBKI-P1POST-P1POSNR,
                                     DBKV-P1POST-P1POSNR,
                                     DDB-P1POST-SUBSCHEMA,
                                     DBV-P1POST,
```

```
                                        DDS-M1-SUBSCHEMA, SCV-M1

          END-DO.
          IF NOFOUND GREATER THAN 1
              MOVE 1 TO SCC-START-RW-LINE.

*
*       Display on screen
*
     MOVE  '-:*'   TO DDC-SELECT.
     CALL 'CPDISP' USING DDC-REF-TABLE, DDS-M1-SUBSCHEMA,
                         SCV-M1.
```

10.8.3. MANUAL CODE

We shall now have a look at a few cases where manual programming is necessary. The manual code (additional code) must be inserted in the macros on AJBEDR:MANU. We will list parts of the program, show where the manual code should be inserted, and what it consists of.

10.8.4. SEVERAL CPREAD CALLS

The field 'Surname' in the screen picture is an alternative key. We therefore want to add a check to make sure that a value is specified in this field during registration and modification.

We also want to make sure that the value of the field 'Position' only is accepted as a legal input value during registration and modification if the value also exists on the S1STILL realm. This will then be a manually programmed existence control.

Realm and key for S1STILL must be inserted in the subschema AJBEDR in ABM. Another change that must be made is the addition of manual code. The problem of several CPREAD calls may be solved in two ways: by adding more CPREAD calls in manual code, or by using the generated CPREAD call along with additional manual code using CPINITEM and CPINVER. Both alternatives will be shown here.

10.8.5. CPREAD CALLS IN MANUAL CODE

Division into several CPREAD calls means that manual code must be inserted immediately before and immediately after the generated CPREAD call. The fields that are to be read by the generated CPREAD call in CITEMSUB, must be inserted immediately before the CPREAD call, and FLNEXT is set to FLREAD. The remaining CPREAD calls, with possible tests of input data, should be inserted immediately after the generated CPREAD call.

In our example we have chosen to use four CPREAD calls. This is in order to have a user interface that is as interactive as possible. We check the input value from the field 'Surname'. If value = blank, message number 151 is displayed, and CPMESS causes the CPREAD call for reading of the field to be executed again.

We check the input value from the field 'Position' by means of a CPEXIST call. If this value does not exist, the message 'does not exist' will be displayed automatically, and the CPREAD call will be repeated.

Manual code is inserted in the AJBEDR:MANU file, and the program is generated again. The changes to the code are shown here:

```
C                %%% Read from screen

         CITMSUB(1) = '-:M2ANSNR *'

C........%%% Manual code before CPREAD: read surname. Set FLNEXT.
         CITMSUB(1) = '+:M2ENAVN *'
C
         FLNEXT = FLREAD
C......... End of manual code before CPREAD.

         CALL CPREAD(1,REFTAB,MITEMM2,MRECM2,FLKEY,FLNEXT)

C........%%% Manual code after CPREAD: Check surname.
         IF (EXECUTE.AND.IACTCOD.NE.4.AND.M2ENAVN.EQ.' ') THEN
              CALL CPGETMSG(151)
              CALL CPMESS
         ENDIF

C        %%% Read first name and code.
         CITMSUB(1) = '+:M2FNAVN M2KODE  *'
         CALL CPREAD(2,REFTAB,MITEMM2,MRECM2,FLREAD,FLREAD)

C        %%% Read position and check if value is legal.
         CITMSUB(1) = '+:M2STILL *'
         CALL CPREAD(3,REFTAB,MITEMM2,MRECM2,FLREAD,FLREAD)
         IF (EXECUTE.AND.IACTCOD.NE.4) THEN

C           %%%  Existence control against S1STILL realm
            LS1STI1 = M2STILL
            HS1STI1 = M2STILL
            CALL CPEXIST(KIS1STI,KVS1STI)
         ENDIF

C        %%% Read extension.
         CITMSUB(1) = '+:M2INTLF *'
         CALL CPREAD(4,REFTAB,MITEMM2,MRECM2,FLREAD,FLOKCOD)
C........Manual code after CPREAD ends.

C        %%% Begin transaction
```

10.8.6. DUMMY CPREAD CALL

The question about updating and the reading of the EXECUTE key take place in the last CPREAD call. If there is manual code (value tests, search of other realms) after the last CPREAD call, that code will be executed after the EXECUTE key has been pressed. If you want the search and/or value tests to be performed before the EXECUTE key is pressed, you can insert a dummy CPREAD call after the manual code. CITMSUB/DDC-SELECT then has to be '+:*' for the CPREAD call. If you want a dummy CPREAD call in the example above, the code must look like this:

```
C       %%%Read extension.
        CITMSUB(1) = '+:M2INTLF*'
        CALL CPREAD(4,REFTAB,MITEMM2,MRECM2,FLREAD,FLREAD)

C.......Look-up and testing

C.......Dummy read call
        CITMSUB(1) = '+:*'
        CALL CPREAD(5,REFTAB,MITEMM2,MRECM2,FLREAD,FLOKCOD)

C.......End of manual code.
```

10.8.7. CPINVER INSTEAD OF SEVERAL CPREAD CALLS

In this case the manual code must be inserted immediately after the generated CPREAD call, possibly together with testing of the input data. If the input data is not OK, CPINITEM is called with the name of the field to be re-read as input value. When all manual tests have been performed, and before any CPEXIST calls, CPINVER is called. If CPINITEM has been called, CPINVER will display these fields in inverse video and CPREAD will be executed again. After CPINVER, you can use CPEXIST calls, overruling the message from the CPEXIST call.

Manual code is added to the AJBEDR:MANU file, and the program must be generated again. The changes to the code are shown below:

```
C                %%% Read from screen

         CITMSUB(1) = '-:M2ANSNR *'

         CALL CPREAD(1,REFTAB,MITEMM2,MRECM2,FLKEY,FLNEXT)

C........%%% Manual code after CPREAD: Check surname.
         IF (EXECUTE.AND.IACTCOD.NE.4) THEN
C            %%% Check surname.
             IF (M2ENAVN.EQ.' ')  CALL CPINITEM(REFTAB,"M2ENAVN ")
C            %%% If necessary, show fields in inverse video and read again.
             CALL CPINVER(REFTAB,MITEMM2,MRECM2)
C            %%%  Existence control against S1STILL  realm
             IF (EXECUTE) THEN
                LS1STI1 = M2STILL
                HS1STI1 = M2STILL
                OWNMESS = .TRUE.
                CALL CPGETMSG(152)
                CALL CPEXIST(KIS1STI,KVS1STI)
                OWNMESS = .FALSE.
           ENDIF
         ENDIF
C.........End of manual code.

C        %%% Begin transaction
```

10.8.8. OVERRULING OF MESSAGES IN PG ROUTINES

It is possible to overrule messages that are displayed in PG routines. In our example we have chosen to overrule the message in CPEXIST in the example, using several CPREAD calls. We want message number 152 to be displayed if the input position does not exist on the S1STILL realm. The modified manual code is added to the AJBEDR:MANU file, and the program is generated again.

Part of the manual code where the change has been made is shown here:

```
C         %%% Read position, and check if value is legal.
          CITMSUB(1) = '+:M2STILL *'
          CALL CPREAD(3,REFTAB,MITEMM2,MRECM2,FLREAD,FLREAD)
          IF (EXECUTE.AND.IACTCOD.NE.4) THEN
C            %%%  Existence control against S1STILL realm
             LS1STI1 = M2STILL
             HS1STI1 = M2STILL
C            %%%  Shows that own message is to be applied.
             OWNMESS = .TRUE.
C            %%%  Moves own message to text string.
             CALL CPGETMSG(152)
             CALL CPEXIST(KIS1STI,KVS1STI)
             OWNMESS = .FALSE.
          ENDIF
C         %%% Read extension.
```

10.8.9. CALCULATION OF FIELDS

The calculation of database items takes place on the basis of fields read during registration and modification. The calculation can be done immediately after the fields are read, or when all the fields are read and the user has ordered the updating of the record. We have chosen to do it in the latter way. In that case, the calculations are only done when necessary, and all the calculations are collected in one place in the program.

In our example, we want to give the item B1SOPD today's date if the record is updated. We have done it in the following way:

```
         CITMSUB(1) = '-:M1BEDNR *                                  '

         CALL CPREAD(1,REFTAB,MITEMM1,MRECM1,FLKEY,FLNEXT)
C........%%% Manual code : If updating.
         IF (CPABLED(FLUPDATE,1)) THEN
C           %%% Get today's date
            CALL CPDATUM(CDATUM,DATO)
            B1SOPD = DATO
C........%%% End of manual code.

C        %%% Begin transaction

         CALL CPBTRANS(MITEMM1,MRECM1,KITEMB1,KRECB1)


C        %%% Update record

         CITMSUB(1) ='O:*'
         CALL CPUPDATE(ITEMSUB,KITEMB1,KRECB1)

C        %%% End transaction

         CALL CPETRANS
```

10.8.10. UPDATING OF OTHER REALMS

Sometimes it is necessary to update several realms. Complete-PG only updates the main realm in each region. The updating of other realms has to be added in manual code. The manual code then has to be inserted after CPUPDATE. Make a test to see whether updating is to be performed by testing whether CPUPDATE has been performed, and testing whether EXECUTE is true.

A temporary database key must exist on the record that is to be modified or deleted on the other realm.

In our example we do not need to update other realms. If we had needed to do so, the code would have looked like this:

```
C         %%% Begin transaction

          CALL CPBTRANS(MITEMM1,MRECM1,KITEMB1,KRECB1)

C         %%% Update record

          CITMSUB(1) ='O:*'
          CALL CPUPDATE(ITEMSUB,KITEMB1,KRECB1)

C........%%% Manual code after CPUPDATE: If updating.
          IF (EXECUTE) THEN
C            %%% If deleting
             IF (IACTCOD.EQ.4) THEN
                CALL SRASE(KTDBKXX,O,KSTAT)
                IF (KSTAT.NE.1) CALL CPABORT
                KTDBKXX = O
C            %%% If modifying
             ELSEIF (IACTCOD.EQ.3) THEN
                CITMSUB(1)='+: -----  *'
                CALL DDMDFY(KTDBKXX,ITEMSUB,KITEMXX,KRECXX,KSTAT)
                IF (KSTAT.NE.1) CALL CPABORT
C            %%% If registering
             ELSEIF (IACTCOD.EQ.2) THEN
                CITMSUB(1)='+: -----  *'
                CALL DDSTORE(ITEMSUB,KITEMXX,KRECXX,KSTAT)
                IF (KSTAT.NE.1) CALL DDFREMB(KTDBKXX,O,KSTAT)
                IF (KSTAT.NE.1) CALL CPABORT
             ENDIF
C........%%% End of manual code.

C         %%% End transaction

          CALL CPETRANS
```

10.8.11. FREE TEXT IN THE PROGRAM

We now wish to call the free text function from the employee lines. We have to use ABM to alter the picture AJBEDR. We introduce a new field into the line. The field consists of one character position, and the data type is TTYPE (TTYPE is reserved by Complete-PG for free text).

After the alteration, the employee lines look like this:

```
Employee :
     Text                                        Code
No.  T     Surname          First name            v Position    Ext.
.... .     ............... ................... . .......... ....
.... .     ............... ................... . .......... ....
.... .     ............... ................... . .......... ....
.... .     ............... ................... . .......... ....
.... .     ............... ................... . .......... ....
```

Now the program has to be generated again. In the 'Use of Program Keys' picture, 'textfunction: Y' will now be shown. PG will generate new code in two places in the program: immediately after CPGET and immediately after CPRSPNS. Below, you can see a section of the program where the new code parts for the free text function are inserted:

After CPGET :

```
          CALL CPGET(EVERYLIN,REFTAB, 2,KIA1NAV,KVA1NAV,
     *               MITEMM2,MRECM2,KITEMA1)


          IF (EXECUTE .AND. IACTCOD.NE.2 .AND. EVERYLIN.EQ.0) THEN
             I = REFTAB(7)
C            %%% Get data from database buffer
             CALL CPINRC(I,KITEMA1,KRECA1)
          ELSEIF (EXECUTE .AND. IACTCOD.NE.2 .AND. EVERYLIN.EQ.1) THEN
             I = 1
C            %%% Get data from database buffer
             CALL CPINRC(I,KITEMA1,KRECA1)
          ENDIF

C         %%% Display text mark

          CALL CPTDISP(REFTAB,MITEMM2,MRECM2,M2TTYPE,"M2TTYPE ",
     +                 KITEMA1,KRECA1,A1TTNR,EVERYLIN)


C         %%% Display on screen

          CITMSUB(1) = '-:*'
          CALL CPDISP(REFTAB,MITEMM2,MRECM2)
C         %%% Read response kode

          CALL CPRSPNS(REFTAB,MITEMM2,MRECM2)
          CALL CPOVER(MITEMM2,MRECM2,KITEMA1,KRECA1)

C         %%%  Text nr. database item name
```

```
      CNAME   = 'A1TTNR  '
C     %%%  Text nr. database item value
      TXTNR   =  A1TTNR
C     %%%  Text indicator field name
      CTYPNAM = 'M2TTYPE '
C     %%%  Text indicator field value
      CTYPE   =  M2TTYPE
C     %%%  User info. used in CPFRTXT
      INFOTXT = ' '

C     %%%  Swap to free text application
      CALL CPFRTXT(REFTAB,TYPNAM,TXTNR,CTYPE,MITEMM2,MRECM2,
     *INAME,MITEMM1,MRECM1,MITEMM2,MRECM2,MAINTAB(4),INFOTXT)

      ENTEXT = .TRUE.

      IF (EXECUTE) THEN
         IF (ENTEXT) THEN
C           %%%  Text nr. database item value in return
            A1TTNR = TXTNR
C           %%%  If text connected, display 'T'
            M2TTYPE = ' '
            IF (TXTNR.GT.0) M2TTYPE = 'T'
            CITMSUB(1) = '+:M2TTYPE *'
         ENDIF
         CALL DDWFLDS(REFTAB,MITEMM2,MRECM2,MSTA)
         IF (MSTA.NE.0) CALL CPABORT
      ENDIF
```

The user may place a value in the variable INFOTXT by using manual code before CPFRTXT. INFOTXT is displayed in the free text screen form (which is delivered with ABM/PG) as an information field. It looks like this: FREE TEXT CONNECTED TO: INFOTEXT (the value of INFOTEXT is displayed here).

10.8.12. SEVERAL FREE TEXTS ON THE SAME RECORD IN THE PROGRAM

It is possible to attach several free text records to one database record. We shall here show an example of this by connecting two free text records to one record in the employee lines. We insert one new field, which consists of one character of data type, eg. T2TYPE, into the line. We can only have one field of type TTYPE per record/region.

The employee lines look like this after the alteration:

```
Employee:
     Text                                        Code
No.  P  S  Surname          First name            v Position    Ext.
.... .  .  ............... ................... . .......... ....
.... .  .  ............... ................... . .......... ....
.... .  .  ............... ................... . .......... ....
.... .  .  ............... ................... . .......... ....
.... .  .  ............... ................... . .......... ....
```

Some manual code is now required to make the program distinguish between the two free text references. After CPGET, we insert a copy of the sequence which was generated when a free text was added (the sequence where MXTTYPE is given a value). In this copy, we must alter the database item, the text number and the data type.

The code after CPGET will then be like this:

```
      CALL CPGET(EVERYLIN,REFTAB,3,KIA1NAV,KVA1NAV,
     *           MITEMM2,MRECM2,KITEMA1)

      IF (EXECUTE .AND. IACTCOD.GT.1) THEN

            IF (EVERYLIN.EQ.0) THEN
               I = REFTAB(7)
            ELSE
               I = 1
            ENDIF

C           %%% Get data from database buffer
            CALL CPINRC(I,KITEMA1,KRECA1)
      ENDIF

C     %%% Display text mark

      IF (EXECUTE) THEN

         IF (NOFOUND.GT.1) REFTAB(7) = 1

C        %%% Loop for every line in region

         DO FOR I = REFTAB(7), REFTAB(7)+NOFOUND-1

C           %%% Get data from database buffer
            IF (EVERYLIN.EQ.0) THEN
               CALL CPINRC(I,KITEMA1,KRECA1)
```

```
                ELSE
                   CALL CPINRC(1,KITEMA1,KRECA1)
                ENDIF

C               %%% Get data from screen buffer
                REFTAB(7) = I
                CALL DDGETRC(REFTAB,MITEMM2,MRECM2,MSTA)
                IF (MSTA.NE.0) CALL CPABORT
C
                IF (A1TTNR.GT.0) THEN
                   M2TTYPE = 'T'
                ELSE
                   M2TTYPE = ' '
                ENDIF

                CALL DDPUTRC(REFTAB,MITEMM2,MRECM2,MSTA)
                IF (MSTA.NE.0) CALL CPABORT
             ENDDO

             IF (NOFOUND.GT.1) REFTAB(7) = 1
          ENDIF

C........%%% Manual code : Display text mark

          IF (EXECUTE) THEN

             IF (NOFOUND.GT.1) REFTAB(7) = 1

C            %%% Loop for every line in region

             DO FOR I = REFTAB(7), REFTAB(7)+NOFOUND-1

C               %%% Get data from database buffer
                IF (EVERYLIN.EQ.0) THEN
                   CALL CPINRC(I,KITEMA1,KRECA1)
                ELSE
                   CALL CPINRC(1,KITEMA1,KRECA1)
                ENDIF

C               %%% Get data from screen buffer
                REFTAB(7) = I
                CALL DDGETRC(REFTAB,MITEMM2,MRECM2,MSTA)
                IF (MSTA.NE.0) CALL CPABORT
C
                IF (A1ATTNR.GT.0) THEN
                   M2T2TYPE = 'T'
                ELSE
                   M2T2TYPE = ' '
                ENDIF

                CALL DDPUTRC(REFTAB,MITEMM2,MRECM2,MSTA)
                IF (MSTA.NE.0) CALL CPABORT
             ENDDO

             IF (NOFOUND.GT.1) REFTAB(7) = 1
          ENDIF
C........End of manual code.

C         %%% Display on screen
```

```
         CITMSUB(1) = '-:*'
         CALL CPDISP(REFTAB,MITEMM2,MRECM2)
```

When the user calls the free text, the application cannot know whether it is free text for person (A1TTNR) or for seniority (A1ATTNR) that should be used. So that the user can give this information, we add the necessary additional code before and after the CPFRTXT call.

The way of doing this, i.e. asking the user a question and reading the user's reply, may vary. The method we have chosen in this example may be used for up to 10 free texts in connection with the same record:

The user chooses the type of free text by typing in a digit from 1 to 9 (in our example 1 and 2). CPFRTXT is then be called with the correct parameters, given by the user. This method involves three manual code sections. Help variables for communication with the user must be declared. This is done using manual code at the beginning of the program:

```
      $INCLUDE (abm-user)CP-PROBOT-COM:INCL

C......%%% Manual code : Manual declaratons
C  CHR  : Response code from user.
C
      INTEGER   CHR
C......%%% End of manual code.

C     %%% ABM interface

      $INCLUDE (CCO-ADM-INCL)DECDDI-PROSAJOU
```

Question to user and reading of reply, as well as assignment of parameter values is all done before CPFRTXT :

```
C        %%% Read response code

         CALL CPRSPNS(REFTAB,MITEMM2,MRECM2)

C        %%%  Text no. database item name
         CNAME = 'A1TTNR  '
C        %%%  Text no. database item value
         TXTNR = A1TTNR
C        %%%  Text indicator field name
         TYPNAM = "M2TTYPE "
C        %%%  Text indicator field value
         CTYPE = M2TTYPE

C........%%%  Manual code : Find which additional text is wanted :
         IF (CPABLED(FLFRTXT,1)) THEN
           CALL DDCMSGE(MSTA)
           IF (MSTA.NE.0) THEN
              CALL CPABORT
```

```
          ELSE
C            %%% Display question for user.
             CALL CPGETMSG(103)
             CALL DDWMSGE(ITEXT,MSTA)
             IF (MSTA.NE.0) THEN
                CALL CPABORT
             ELSE
C               %%% Init. variables.
                CHR   = 0
C               %%% Read reply.
                DO WHILE (CHR.NE.49.AND.CHR.NE.50.AND.MSTA.EQ.0)
                   CALL FCRCHR(CHR,MSTA)
                ENDDO
                IF (MSTA.EQ.0) THEN
C                  %%% Give parameters correct value according to reply.
                   IF (CHR.EQ.49) THEN
C                  %%% Has been given value via generated code.
                   ELSEIF (CHR.EQ.50) THEN
                      CNAME   = 'A1ATTNR '
                      TXTNR   =  A1ATTNR
                      CTYPNAM = 'M2T2TYPE'
                      CTYPE   =  M2T2TYPE
                   ENDIF
                ELSE
                   CALL CPABORT
                ENDIF
             ENDIF
          ENDIF
        ENDIF
C
C........End of manual code.

C       %%%  Swap to free text application
        CALL CPFRTXT(REFTAB,TYPNAM,TXTNR,CTYPE,MITEMM2,
     *  MRECM2,INAME,MITEMM1,MRECM1,MITEMM2,MRECM2,MAINTAB(4))
```

Immediately after CPFRTXT, the text number must be saved:

```
C       %%%  Swap to free text application
        CALL CPFRTXT(REFTAB,TYPNAM,TXTNR,CTYPE,MITEMM2,
     *  MRECM2,INAME,MITEMM1,MRECM1,MITEMM2,MRECM2,MAINTAB(4))

        ENTEXT = .TRUE.

C........%%%  Manual code :  Return text no. and display 'T' if necessary:
        IF (EXECUTE) THEN
           IF (CHR.EQ.49) THEN
C                 %%% Will be given value via generated code.
           ELSEIF (CHR.EQ.50) THEN
               ENTEXT = .FALSE.
               A1ATTNR = TXTNR
               M2T2TYPE = ' '
               IF (A1ATTNR.NE.0) M2T2TYPE = 'T'
               CITMSUB(1) = '+:M2T2TYPE*'
           ENDIF
        ENDIF
```

```
C.........Manual code ends.

          IF (EXECUTE) THEN
             IF (ENTEXT) THEN
C               %%%
                A1TTNR = TXTNR
C               %%% If text connected, display 'T'
                M2TTYPE = ' '
                IF (TXTNR.GT.0) M2TTYPE = 'T'
                CITMSUB(1) = '+:M2TTYPE *'
             ENDIF
             CALL DDWFLDS(REFTAB,MITEMM2,MRECM2,MSTA)
             IF (MSTA.NE.0) CALL CPABORT
          ENDIF
```

10.8.13. SELECTION OF RECORDS

When listing from the database to a screen picture, it is possible to select the records you want displayed. CPGET either fetches one record at a time (EVERYLIN=1), or as many records as there are lines in the screen picture (EVERYLIN=0). This is controlled by the parameter EVERYLIN. Complete-PG sets EVERYLIN to 0 at the beginning.

In our example, during querying, we now want to list all the employees with a code equal to the firm's code, as long as it is different from 1. If the firm's code is equal to 1, all employees are to be listed. While a user is modifying or deleting, all employees should be listed. This may be solved by inserting manual code before and after CPGET.

Part of the manual code where the change is made (in FORTRAN):

```
C               %%% Get record

         LA1BED1 = M1BEDNR
         LA1BNR1 = M1BEDNR
         HA1BED1 = M1BEDNR
         HA1BNR1 = M1BEDNR
         IF (CPKEYNK().EQ. 1) THEN
            LA1BED2 = M2ANSNR
         ENDIF
         IF (CPKEYNK().EQ. 2) THEN
            LA1BNR2 = M2ENAVN
         ENDIF

*........Manual code before CPGET, select if criteria fulfilled

         IF (IACTCOD.EQ.1 .AND. B1KODE.NE.1) EVERYLIN = 1

         CALL CPGET(EVERYLIN,REFTAB, 1,KIA1BED,KVA1BED,
     *              MITEMM2,MRECM2,KITEMA1)

         CALL CPGET(EVERYLIN,REFTAB, 2,KIA1NAV,KVA1NAV,
     *              MITEMM2,MRECM2,KITEMA1)


         IF (EXECUTE .AND. IACTCOD.NE.2 .AND. EVERYLIN.EQ.0) THEN
            I = REFTAB(7)
C           %%% Get data from database buffer
            CALL CPINRC(I,KITEMA1,KRECA1)
         ELSEIF (EXECUTE .AND. IACTCOD.NE.2 .AND. EVERYLIN.EQ.1) THEN
            I = 1
C           %%% Get data from database buffer
            CALL CPINRC(I,KITEMA1,KRECA1)
         ENDIF

*........Manual code after CPGET
*        Only employees with the same code as firm are to be listed

         IF (EXECUTE .AND. EVERYLIN.EQ.1) THEN
            IF (A1KODE.EQ.B1KODE) THEN
               CALL CPENABLE(FLOK)
            ELSE
```

```
                CALL CPDISABL(FLOK)
             ENDIF
          ENDIF

C         %%% Display on screen

          CITMSUB(1) = '-:*'
          CALL CPDISP(REFTAB,MITEMM2,MRECM2)
       *....Manual code after cpget.
       *    Only employees with same code as firm are to be listed.
```

Part of the manual code where the change is made (in COBOL):

```
*         %%% Get record

     MOVE SCV-M1-B1BEDNR       TO DBKV-A1ANSAT-A1BEDAN-LOW-1.
     MOVE SCV-M1-B1BEDNR       TO DBKV-A1ANSAT-A1BNRNA-LOW-1.
     MOVE SCV-M1-B1BEDNR       TO DBKV-A1ANSAT-A1BEDAN-HIGH-1.
     MOVE SCV-M1-B1BEDNR       TO DBKV-A1ANSAT-A1BNRNA-HIGH-1.
     CALL 'CPKEYNC' USING KEY-NO.
     IF KEY-NO =  1
         MOVE SCV-M2-A1ANSNR    TO DBKV-A1ANSAT-A1BEDAN-LOW-2.
     CALL 'CPKEYNC' USING KEY-NO.
     IF KEY-NO =  2
         MOVE SCV-M2-A1ENAVN    TO DBKV-A1ANSAT-A1BNRNA-LOW-2.

*....Manual code before CPGET. Select if criteria fulfilled

     IF MAINTAB(5) EQUAL 1
         AND DBV-B1BEDR-B1KODE EQUAL 1

         MOVE 1 TO EVERYLIN.

*
     CALL 'CPGET' USING EVERYLIN, DDC-REF-TABLE,  1,
                        DBKI-A1ANSAT-A1BEDAN ,
                        DBKV-A1ANSAT-A1BEDAN , DDS-M2-SUBSCHEMA,
                        SCV-M2, DDB-A1ANSAT-SUBSCHEMA.

*
     CALL 'CPGET' USING EVERYLIN, DDC-REF-TABLE,  2,
                        DBKI-A1ANSAT-A1BNRNA ,
                        DBKV-A1ANSAT-A1BNRNA , DDS-M2-SUBSCHEMA,
                        SCV-M2, DDB-A1ANSAT-SUBSCHEMA.

     IF EXECUTE = 1
         AND EVERYLIN = 0
         AND MAINTAB(5) NOT = 2
         MOVE SCC-START-RW-LINE TO INDX
*        Get data from databasebuffer
         CALL 'CPINRC' USING INDX, DDB-A1ANSAT-SUBSCHEMA,
                             DBV-A1ANSAT
     ELSE
         IF EXECUTE = 1
            AND EVERYLIN = 1
            AND MAINTAB(5) NOT = 2
            MOVE 1 TO INDX
*           Get data from databasebuffer
            CALL 'CPINRC' USING INDX, DDB-A1ANSAT-SUBSCHEMA,
                                DBV-A1ANSAT.

*....Manual code after CPGET
*    Only employees with the same code as firm are to be listed
     IF EXECUTE EQUAL 1
         AND EVERYLIN EQUAL 1

         IF DBV-B1BEDR-B1KODE EQUAL DBV-A1ANSAT-A1KODE
             CALL 'CPENABLE' USING TRIGGER-OK
         ELSE
             CALL 'CPDISABL' USING TRIGGER-OK.
```

```
*
      CALL 'CPDISP' USING  DDC-REF-TABLE,
                           DDS-M2-SUBSCHEMA,
                           SCV-M2.
```

10.8.14. READING OF KEY IN SEVERAL READ CALLS

Sometimes you may want to read the key in several separate READ calls. You might want to display some information in between, or you do not want all key fields to be read, depending on what is given in the preceding key fields.

Read the key in several separate READ calls by using the generated CPKEY call for the field that is to be read first. Afterwards, use the ABM DDRFLDS for the remaining fields. In order to terminate the DDRFLDS call in the same way as the reading of fields in Complete-PG's subroutines, you can apply the subroutine CPTERMCH.

Example:

In the picture, the key consists of firm number + department + employee number.

First, the firm number is to be read. Then, the name of the firm is to be fetched from the firm realm and displayed in the picture before department and employee number is read. If the firm number you have typed in does not exist, a message is to be displayed, and you have to type in the firm number once more.

```
C........Manual code before CPKEY
         CITMSUB(1) = '+:M1BEDNR*'

C........End of manual code for CPKEY

         CALL CPKEY(REFTAB,MITEMM1,MRECM1,FLACTCOD,FLREAD)

C........Manual code after CPKEY

         IF (CPABLED(FLREAD,1)) THEN; % i.e. firm number is read

             <Fetch firm name from firm realm>

             IF (firm not found) THEN
                 <display message>
                 CALL CPJUMP(FLKEY)
             ELSE
C............firm found, display name and
C............read rest of key
             CITMSUB(1) = '+M1BNAVN*'
             CALL DDWFLDS(REFTAB,MITEMM1,MREM1,MSTA)
                   IF (MSTA.NE.0) THEN
                       CALL CPABORT
                   ELSE
                       CITMSUB(1) = '+:M1AVDNR M1ANSNR*'
                       CALL DDRFLDS(REFTAB,MITEMM1,MREM1,MSTA)
                           IF (MSTA.NE.0) THEN
                              CALL CPABORT
                           ELSE
                              CALL CPTERMCH(RETAB,MITEMM1,MREM1,FLKEY,FLREAD)
                           ENDIF
                   ENDIF
             ENDIF
         ENDIF
C........End of manual code after CPKEY
```

CHAPTER 11

INTERFACE TO MENU CONTROL SYSTEM

11. INTERFACE TO MENU CONTROL SYSTEM

All programs generated by Complete-PG are subroutines, and have to be started by a main program or menu control system. This menu control system takes care of the following tasks:

- opens databases
- initiates FOCUS
- presents menus on the screen and read menu choice
- calls chosen program
- closes databases
- terminates FOCUS

> NOTE:
> When you have started the generating of a program by giving the command 'X', you are asked whether you want a main program to be generated. If you reply 'Y', a main program will be generated to take care of the above tasks.

There are also some other tasks that have to be solved through cooperation between the generated program and the menu control system. These tasks are:

- Find out whether the user has access to the chosen program, and if so, what kind of access (only querying, or full access).
- Fetch a free code connected to each user. This free code may for instance be the administration unit the user belongs to.
- Reserve flags and check against SIBAS before updating, and release flags after updating.
- Take care of direct transfer to a new program, and transfer data area (main key).

11.1. SUBROUTINES ON CP-DUMMY-LIB

A dummy library called CP-DUMMY-LIB:SYMB comes with Complete-PG. Here you find the subroutines that will execute tasks in cooperation with the menu control system.

The subroutines are empty, because you have to adapt them to your menu control system. If there is a feature that you do not want in your system, simply leave the subroutine empty.

Here follows a description of the routines on CP-DUMMY-LIB:

FRIKODE (CFRIKODE) FORTRAN
CFRIKODE (IFRIKODE) COBOL

Parameter list:

```
Character*40     CFRIKODE
INTEGER*2        IFRIKODE(20)
```

Routine description: This routine fetches a text string or code of 40 characters which may be connected to each user defined in the menu control system. This text string may contain general information about the user, such as administrative unit and what kind of access the user has.

CHCKACS (IACS)

Parameter list: INTEGER IACS

Routine description: Checks whether the user has access to the program that s/he has chosen.

```
Output: IACS = 0 : no access.
             = 1 : querying only.
             = 2 : full access.
```

SMRESRV (IFLAG)

Parameter list: INTEGER IFLAG

Routine description: The routine reserves flags and takes checkpoint.

```
Input:  IFLAG = 0 : conditional checkpoint.
              = 1 : unconditional checkpoint.
```

SMRELES

Parameter list: None.

Routine description: The routine releases flags.

Direct transfer from a program: When there is a direct transfer from one program to another, the calling program builds up a buffer with key values that are to be sent from the calling program and received by the called program.

Here follows a description of the subroutines with parameters that take care of this transfer.

CPSEND (NUMBER,ARRAY,STATUS)

Parameter list:

```
INTEGER  NUMBER , STATUS
INTEGER*2 ARRAY(*)
```

Routine description: The subroutine is called in the calling program, and transfers a buffer from the calling program to the called program.

NUMBER : number of 16-bit words that may be sent (today a maximum of 200).

ARRAY : buffer with the transferred key items and key values.

STATUS = 0 : means OK.

CPRECEIVE (NUMBER,ARRAY,STATUS)

Parameter list:

```
INTEGER  NUMBER , STATUS
INTEGER*2 ARRAY(*)
```

Routine description: The subroutine is started from the called program, and receives a buffer from the calling program. The subroutine makes the buffer available to the calling program.

NUMBER : number of 16-bits words that are transferred (today a maximum of 200).

ARRAY : buffer with the key items and key values that are transferred.

STATUS = 1 : data may be received.
= 0 : no data to receive.

Status is set to 0 when transferred data is received.

AUTOFUNK (DIRECTION,STATUS)

Parameter list: INTEGER DIRECTION , STATUS

Routine description: The subroutine is called in the calling program when the command to start the next program or to start the previous program is given. The routine checks whether this current program has any following or preceding program defined.

If the program has a following or preceding program defined, the program is ended. If not, a message is displayed on the screeen.

```
DIRECTION =  1 : check whether a following
                 program exists (NAPL given)
          = -1 : check whether a preceding
                 program exists (PAPL given)

STATUS    =  1 : following/preceding program
                 exists.
          =  0 : following/preceding program
                 does not exist.
```

CHAPTER 12

FREE TEXT FUNCTION

12. FREE TEXT FUNCTION

The free text function in Complete-PG gives you the possibility of connecting a number of text lines to records in the database.

The free text function works like a standard Complete-PG line-oriented function, with standard Complete-PG commands and function keys.

Here we shall have a closer look at what is required of the database when the free text function is used.

12.1. DATABASE REQUIREMENTS

Free text item

- Each realm with records you want to connect free text to, must contain a free text item. This item must be called:

  XXTNR in FORTRAN where XX=realm prefix
  TNR in COBOL

- The item must be defined as INTEGER*4, and be 9 characters long, i.e. PIC 9(9).

- TTYPE must only be defined for lines or records belonging to realms with TNR/XXTNR items in the database. See page 20.

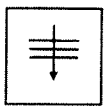
- The TTYPE field is an indicator field which displays a 'T' if there is free text in the record. Otherwise, the field will be blank. The field is only used for output.

12.2. USE OF FREE TEXT FUNCTION

Calling free text:

The free text function can be called from all programs that are defined with free text. Call the free text function either by

- giving the command 'TEXT' in the command field, or



- pressing this function key

The cursor must be in the part of the picture/record where TTYPE is defined.

Querying: If there is free text belonging to a record (shown by TTYPE field = T), the free text function may be called during a query. If it is called, and there does not exist any free text in the record (the TTYPE field is blank), a message will be displayed saying that no free text is registered.

Storing/ modifying/ deleting: The record you want to connect free text to must be registered before the free text is added. In order to store, modify or delete free text, you must enter the record that the text is going to be connected to, with 'modification' access. Then call the free text function.

CHAPTER 13

THE HELP FUNCTION IN COMPLETE-PG

13. THE HELP FUNCTION IN COMPLETE-PG

HELP on several levels

The HELP function in Complete-PG is flexible, and designed to give the user of the Complete-PG generated application, help on most levels. Via the HELP function, the user may get help information about:

1. the application he or she is using.
2. each field in the screen picture.
3. each message the application displays.
4. all legal command words which can be used in the application.
5. all legal function keys which can be used in the application.

Dynamic help information

The help information is dynamic. Authorized users may modify/register/delete help information. The amount of help text that can be stored is limited only by the storage space in the SIBAS database.

13.1. DATABASE REQUIREMENTS

Separate realm

The Complete-PG HELP function stores all help information on a separate realm in the database. This realm, D7HELP, must be inserted in the database before the PG help function can be used.

CP-REDEF-HELP

The insertion of the realm is performed by running SIB-DRL with the file CP-REDEF-HELP:SYMB.

This file first has to be adapted by adding the database name, OS file name and system realm name.

13.2. PROGRAMMING WITH THE PG HELP FUNCTION

During the generating of programs with the Complete-PG 2C version, the code for calling the PG help function will be generated automatically for all applications. The programmer need not think about the PG help function when programming.

13.3. "STAND-ALONE"

Subroutine

Used stand-alone, the PG help function may be called in a menu system by calling the subroutine AJHELP.

Independent program

The PG help function may also be run as an independent program or domain. In that case, a main program that calls AJHELP must be written:

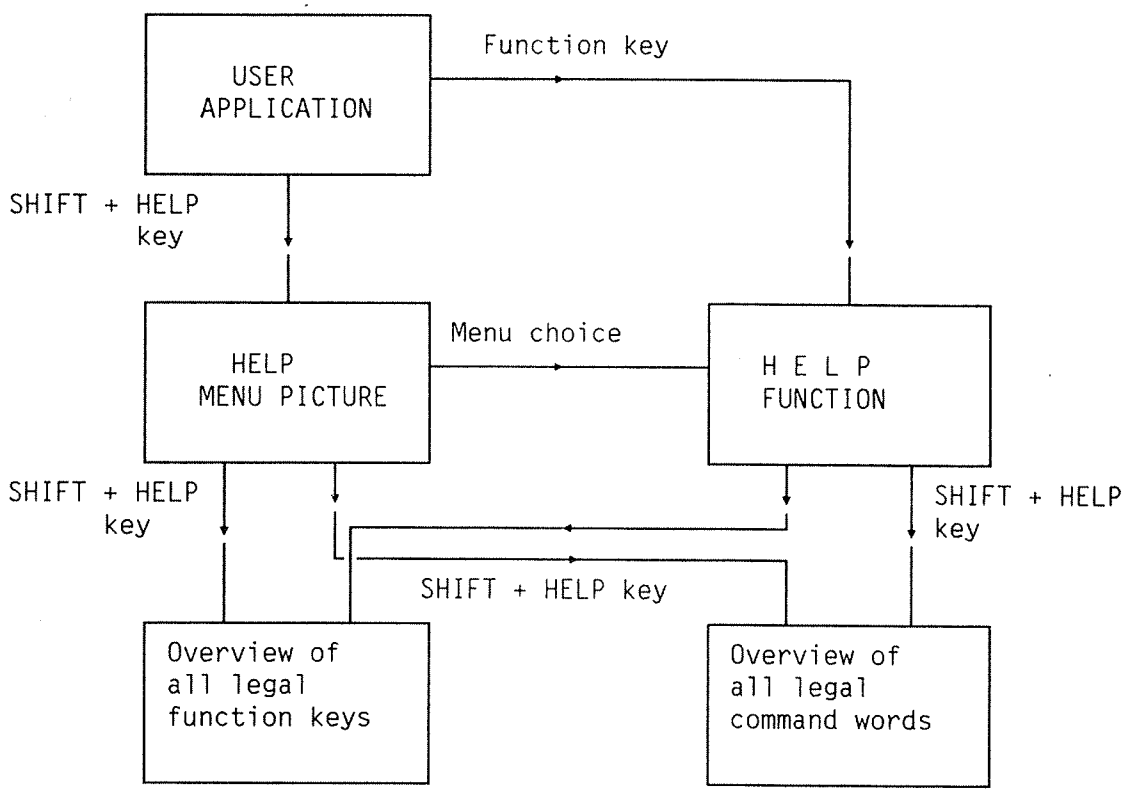
```
        PROGRAM  HELP
C----------------------------- Main prog. for AJHELP.
        CALL AJHELP
        END
```

This main program is compiled and loaded in the same way as a standard PG application.

13.4. FROM A USER APPLICATION

The PG help function may be called from the user applications in several ways. Depending on the type of help that is wanted, the function may be called by means of five different function keys, or it may be called via the help menu, which is activated by pressing SHIFT + HELP.

13.4.1. OVERVIEW OF THE HELP FUNCTION



THE FUNCTION KEYS ARE:

[FIELD]	: Call to the help function for help information about fields.
[PARA]	: Call to the help function for help information about the user application.
[SENT]	: Call to the help function for help information about the last message.
[WORD]	: Call to the help function for help information about command words.
[FUNC] + [T]	: Call to the help function for help information about function keys.
SHIFT + [HELP]	: Call for the help menu.

HELP MENU FOR COMPLETE-PG:

```
                    Complete PG   HELP menu.

    Help information about:          Called directly from function:

  . 1.Field.                                   [FIELD]

  . 2.Picture.                                          [PARA]

  . 3.Messages.                                [SENT]

  . 4.Function key.                                          FUNC + T
      HELP gives an overview of all legal function keys.

  . 5.Command word.                                                  [WORD]
      SHIFT + HELP gives an overview of all legal command words.

  Navigate with [↓] / [↑] and choose with [↵] / (ENTER key).

  Help may also be chosen by          Leave help with [EXIT]
  pressing 1-5 directly!
```

HELP FUNCTION WITH QUERY ACCESS

Help concerning field, application, message	If the help function is called when you have query access, the user in the cases 1, 2, and 3 (cf. the menu picture) will only get help information about the given field/given application or last message. The user will only be able to page through the help information registered about this field, function or message.
Help concerning command word or function key	In cases 4 and 5 (cf. the menu picture) the user will be able to type in the command word or press the function key and get a display of registered help information about this command word or function key. A new command word or function key will be read until the EXIT key is pressed.

HELP FUNCTION PICTURE:

```
                    Complete-PG  HELP FUNCTION
____________________________________________________________________
..... ..... ................ .....      Leave help, return to
.......... .......  .......... ........  function, press:        [EXIT]
.............................................. . ....

                    HELP INFORMATION :

.......................................................................

...  ...................................................................
...  ...................................................................
...  ...................................................................
...  ...................................................................
...  ...................................................................
...  ...................................................................
...  ...................................................................
...  ...................................................................
...  ...................................................................
...  ...................................................................
...  ...................................................................
```

The help function picture will vary according to which types of help information is sought. On the 11 lower lines, the help information will be displayed. This is the part you may scroll through, if more than 11 lines of help information is registered. A maximum of 999 lines of help information may be registered.

The upper part of the picture will look as follows, where XXXXXX is the field name in ABM, YYYYYYY is the function name of the calling function, and 999 is the message number.

PG HELP FUNCTION CALLED FOR HELP ABOUT FIELDS:

```
                    Complete-PG  HELP FUNCTION
____________________________________________________________________
                                        Leave help, return to
                                        function, press:        [EXIT]


                    HELP INFORMATION:

Help information about field: XXXXXXXX . In function : YYYYYY .

     ..................<.Help information about the field.>...............
```

HELP FUNCTION CALLED FOR HELP ABOUT FUNCTION :

```
                    Complete-PG  HELP FUNCTION
____________________________________________________________________
                                        Leave help, return to
                                        function, press:        [EXIT]


                    HELP INFORMATION:

Help information about function: YYYYYYYY .

     ..................<.Help information about function .>...............
```

PG HELP FUNCTION CALLED FOR HELP ABOUT MESSAGE:

```
                    Complete-PG  HELP FUNCTION
____________________________________________________________________
               Message number :   999   Leave help, return to
                                        function, press:        [EXIT]


                    HELP INFORMATION:

999 : < last message        >

     ..................<.Help information about last message.>.............
```

PG HELP FUNCTION CALLED FOR HELP ABOUT COMMAND WORD:

```
                        Complete-PG  HELP FUNCTION
------------------------------------------------------------------------
                                          Leave help, return to
                                          function, press:          EXIT
Give command you want help information about:  ....


                        HELP INFORMATION:

.......................................................................

     ..................<.Help information about command word.>.............
```

PG HELP FUNCTION CALLED FOR HELP ABOUT FUNCTION KEY:

```
                        Complete-PG  HELP FUNCTION
------------------------------------------------------------------------
                                          Leave help, return to
                                          function, press:          EXIT
Press function key you want help information about: .


                        HELP INFORMATION:

.......................................................................

     ..................<.Help information about function key.>.............
```

HELP FUNCTION WITH UPDATE ACCESS:

If the help function is called with update access, the help picture will look as follows:

```
                    Complete-PG  HELP FUNCTION
------------------------------------------------------------------------
Code : ........ Message number : .....      Leave help, return to
Function : ........ Field name : ........   function, press:          [EXIT]
Press function key you want help info. about: .


                        HELP INFORMATION:

.......................................................................

...  ...................................................................
...  ...................................................................
...  ...................................................................
...  ...................................................................
...  ...................................................................
...  ...................................................................
...  ...................................................................
...  ...................................................................
...  ...................................................................
...  ...................................................................
...  ...................................................................
```

The user will now be able to update all five types of help information. The keys for the five types of help information are:

Type of help:		Key:
1. Help on field	:	Code + function name + field name.
2. Help on function	:	Code + function name.
3. Help on message	:	Code + message number.
4. Help on function key	:	Code + function key.
5. Help on command word	:	Code + command word.

FIELD EXPLANATION:

Code	This field may be left open if you do not use a TP monitor such as TRUE.
Function name	The name of the function. The name is set automatically when generating functions in Complete-PG.
Field name	Is the name of the field in the screen picture in the user application. The name is fetched from ABM.
Message number	The number of the message, from the message file.
Function key	The FOCUS code for function keys.
Command word	The command words in PG applications.

In standard PG functions, the user may alternate between the five keys in the function, and ask for, register, modify and remove help information.

APPENDIX A

OTHER AUXILIARY ROUTINES

OTHER AUXILIARY ROUTINES

CP-SERVICE

A separate service library, CP-SERVICE, contains subroutines that may profitably be used for manual programming. Some service routines are made especially for setting, resetting or testing the flags for the various routines.

Below, you will find a description of the various service routines.

LOGICAL FUNCTION CPABLED (FLXXXX, WORD)

Parameter list: INTEGER FLXXXX, WORD

Routine description: The function tests whether the specified flag is set. TRUE is returned if this is the case.

CPABORT

Routine description: Aborts/terminates the program. Must be called when errors occur. CPABORT should be called if an error occurs in a SIBAS file or a FOCUS file.

CPABORT resets all flags so that the DO loop is ended, and CPEND is called. CPEND will, if MSTAf0 or KSTATf1, write an error message to the error message file.

> NOTE:
> An error message is written by CPEND only if MSTA≠0 or KSTAT≠1.

CPBYTE (IUNIT,CLINE)

Parameter list:

```
CHARACTER*(*) CLINE
INTEGER       IUNIT
```

Routine description: Writes a text string to a specified unit number without using FORTRAN I/O.

FUNCTION CPCAVD (INTEG)

Parameter list:

```
CHARACTER CPCAVD*10
INTEGER*4 INTEG
```

Routine description: Converts a double integer to a character string.

FUNCTION CPCAVINT (INTEG)

Parameter list:

```
CHARACTER CPCAVINT*5
INTEGER*2 INTEG
```

Routine description: Converts an integer to a character string.

CPDATUM (CDATUM,DATE)

Parameter list:

```
CHARACTER*(*) CDATUM
INTEGER*4     DATE
```

Routine description: Fetches the current date and time. Moves this information to a text string.

CPDELTXT (TTNR,KSTAT)

Parameter list:

```
INTEGER*4      TTNR     I
INTEGER        KSTAT    O
```

Routine description: Deletes all existing text lines on text number TTNR from the D3TEXT realm in the database. If everything is OK, KSTAT will be returned with a value of 1.

If KSTAT is different to 1, a SIBAS error has occurred.

KSTAT must be tested after the routine call.

CPDISABL (FLXXXX)

Parameter list:

```
INTEGER  FLxxxx       (FORTRAN)
INTEGER  TRIGGER-xxxx (COBOL)
```

Routine description: FLxxxx = flag of a PG routine. Resets the flag of a routine (i.e. the routine will not be executed).

CPENABLE (FLXXXX)

Parameter list:

```
INTEGER  FLxxxx       (FORTRAN)
INTEGER  TRIGGER-xxxx (COBOL)
```

Routine description: FLxxxx = flag of a PG routine you want executed. The routine sets this flag.

CPGETMSG (MSGNO)

Parameter list: INTEGER MSGNO

Routine description: Gets the message text corresponding to the given message number from the message file, and moves the message text to CTEXT.

CPIENABL (TRIGGER-XXXX,RESULT)

Parameter list:

```
INTEGER  TRIGGER-xxxx (Input)
INTEGER  RESULT       (Output)
```

Routine description: The routine tests whether the specified flag is set.

RESULT = 1 if the flag of the PG routine is set, otherwise RESULT = 0.

CPIN (ISUB)

Parameter list: INTEGER*2 ISUB(4)

Routine description: The routine must be called at the beginning of each subroutine, in order to get the current subroutine name written to the error message file if an error occurs in the routine.

CPINITEM (REFTAB,"FIELDNAME") <= FORTRAN
CPINITEM (REFTAB,FIELDNAME) <= COBOL

Parameter list: INTEGER*2 REFTAB(*), fieldname*(4)

Routine description: The routine puts field names into REFTAB. CPINITEM builds a REFTAB list for use in, for example, CPINVER.

CPINVER (REFTAB,MITEM,MREC)

Parameter list: INTEGER*2 REFTAB(*), MITEM(*), MREC(*)

Routine description: The routine sets all screen picture fields defined in REFTAB, into inverse video, and displays a message for the fields having an illegal value. The last executed CPREAD call is then executed again.

CPJUMP (FLAG)

Parameter list: INTEGER FLAG : (Input) The name of the the routine you want to activate (= FLxxxx).

Routine description: This routine activates the desired routine, and skips intermediate routines (resets all flags) and intermediate manual code.

CPMESS

Parameter list: None.

Routine description: Displays the message in CTEXT. If the last subroutine call was CPREAD, this call is activated once more, whereas intermediate routines and manual code are skipped.

CPOUT (ISUB)

Parameter list: INTEGER*2 ISUB(4)

Routine description: If CPIN has been called at the beginning of a subroutine, CPOUT must be called at the end. The routine checks whether an error has occurred (NOERR=.FALSE.).

CPEND

Parameter list: None.

Routine description: Terminates the program.

CPSWAP (INTEG1, INTEG2)

Parameter list: INTEGER INTEG1, INTEG2

Routine description: Swaps the contents of INTEG1 and INTEG2.

CPTERMCH (REFTAB, MITEM, MREC, PREV, NEXT)

Parameter list:
```
INTEGER*2  REFTAB(*), MITEM(*), MREC(*)
INTEGER    PREV, NEXT
```

Routine description: PREV contains the flag of the routine to be executed when the user presses the left arrow. NEXT contains the flag of the next routine to be executed.

The routine controls the termination after a read call in the same way as the read calls in CPREAD. (See description of function keys.)

CPTOKEY

Parameter list: None.

Routine description: Activates the CPKEY routine, so that a new read of the key field is permitted. Skips all intermediate routines and intermediate manual code.

CPTRNSFR (ARRAY1,POS1,ANTALL,ARRAY2,POS2)

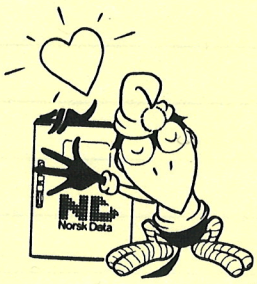
Parameter list:

```
INTEGER*2  ARRAY1(*), ARRAY2(*)
INTEGER    POS1, NUMBER, POS2
```

Routine description: Transfers data from ARRAY1 position POS1 to ARRAY2 position POS2.

NUMBER = Number of words to be transferred.

INDEX LIST



SEND US YOUR COMMENTS!

Are you frustrated because of unclear information in our manuals? Do you have trouble finding things?

Please let us know if you:
- find errors
- cannot understand information
- cannot find information
- find needless information.

Do you think we could improve our manuals by rearranging the contents? You could also tell us if you like the manual.

Send to:
Norsk Data A.S
Documentation Department
P.O. Box 25 BOGERUD
N - 0621 OSLO 6 - Norway

NOTE!

This form is primarily for documentation errors. Software and system errors should be reported on Customer System Reports.

COMPLETE PROGRAM GENERATOR

Manual Name: User Manual _____ Manual number: ND-60.219.1 EN

Which version of the product are you using? _____

What problems do you have? (use extra pages if needed) _____

_____

_____

_____

_____

_____

Do you have suggestions for improving this manual? _____

_____

_____

_____

_____

_____

_____

Your name: _____ Date: _____

Company: _____ Position: _____

Address: _____

_____

What are you using this manual for? _____

_____

Answer from Norsk Data _____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Answered by _____ Date _____

Norsk Data A.S
Documentation Department
P.O. Box 25, Bogerud
0621 Oslo6, Norway