7 Introduction to ISPF programming

In the preceding chapters, I you with the operation of ISPF. Likewise, I familiarized you with the programming language REXX. Now that you have the necessary knowledge in REXX programming, I will show you how to create ISPF applications.

7.1 PROGRAMMING LANGUAGES USEABLE IN ISPF

The following programming languages can be used to create ISPF applications:

- Assembler
- REXX
- CLIST
- PL/I
- COBOL
- VS FORTRAN
- C
- PASCAL
- APL2

I marked REXX in the above table. In this book, I will only use REXX as programming language to develop ISPF applications. From the above mentioned languages, REXX is the easiest to use, the fastest and most elegant language. REXX requires the least effort to create ISPF applications.

7.2 7

ISPF is a dialog-oriented online system while CICS and IMS are transaction-oriented online systems in z/OS MVS. Dialogue oriented means in this context that an online program remains in memory as long as the user works in dialogue with it. This has the advantage that you as a programmer of online applications in ISPF do not have to worry about subsequent transactions, storage of data between transactions, etc. At delivery, a toolbox with all the tools needed for programming applications running in ISPF is already included. These tools are described in the following IBM brochures:

ISPF Dialog Developer's Guide and Reference

and

ISPF Services Guide

There are essentially tools for creating or editing the following items:

- Data sets
- Panels
- Messages
- Skeletons
- Tables

We will discuss in detail all of these elements during the further course of the book. For each of the above elements I will give you below a brief introduction followed by examples for each element.

Data sets

REXX normally processes only sequential data sets. This may also involve members of a PDS. There are also functions for editing directories of PDS. VSAM and other types of data sets can be processed with special access programs. For processing DB2 tables See following URL: http://mainframe-tips-and-tricks.blogspot.de/2011/12/sample-db2-rexx-program.html (http://mainframe-tips-and-tricks.blogspot.de/2011/12/sample-db2-rexx-program.html) and http://www-01.ibm.com/support/knowledgecenter/SSEPEK_11.0.0/com.ibm.db2z11.doc.apsg/src/tpc/db2z_codesqlstaterexx.dita (http://www-01.ibm.com/support/knowledgecenter/SSEPEK_11.0.0/com.ibm.db2z11.doc.apsg/src/tpc/db2z_codesqlstaterexx.dita)

Panels

Panels are created using the ISPF editor. They consists of two parts: The first part defines the representation on the screen. The other part performs a certain plausibility check and defines for which fields help panels are available.

Messages

ISPF provides a very simple and user-friendly way to define messages and output them as required. The messages can be outputted from programs and from panels.

Skeletons

Skeletons define a mask for creating data sets, where the variables which are defined in the skeletons will be replaced automatically by the contents of the variables of the same name in the REXX procedure. Skeletons are mostly used for creating JCL lines. The ISPF editor is used to generate skeletons. When a REXX program calls a skeleton, then all variables in the skeleton are replaced by the values that are defined in the REXX program with the same name.

Tables

Tables are by programs created and edited using appropriate ISPF functions. Tables consists of a special ISPF data type. They can only be stored as members in special ISPF table datasets of type PDS. Since they are interspersed with ISPF control data, their content can be viewed with edit and browse.

Variables

Variables play a major role in ISPF. ISPF itself contains a large amount of variables that can be used in some ISPF elements such as panels, messages and programs. ISPF differentiates between system variables, normally starting with Z and user variables that can be anything called. The name must not be longer than **eight** characters. A table of system variables is found in **Appendix D** of the IBM manual **ISPF Dialog Developer's Guide and Reference** found.

Conclusion

Now having introduced you to the theoretical programming elements, I want to look at some examples from my practice. The examples are purposely from practice, because I do not want to bore you with simple things. I ask for your understanding that I cannot explain the commands and procedures for each example in detail as this go too far. All the different elements in the examples are anyway discussed in more detail later in the book.

7.3 7

7.3.1 Example for use of ISPF panels

The program SSS belongs to the SMART ISPF utilities. It is called in front of the name of a data set to perform a SUPER SEARCH in a DSLIST panel. The panel SSSP1 shown below is used by the REXX procedure SSS to read in the text to be searched for.

Program 7.1: Panel definition of panel SSSpl

```
)ATTR
/* DOC: SSSP1    PANEL                                                */
/* DOC: Input panel to perform Super-Search processing                */
/* DOC: Procedure SSS calls this panel.                               */
/* © FRANZ LANZ 2015                                                  */
/*********************************************************************/
+ type(text) intens(low) skip(on)
% type(text) intens(high) skip(on)
# type(output)
$ type(output) intens(low)
? Type(input) caps(on) intens(high)
@ Type(input) caps(on) intens(high) pad(_)
)BODY EXPAND(//)
%SSSP1 /-/+Enter commands for Super-Search execution%/-/
%COMMAND ===>_ZCMD / /
+
+DSN =#dsn                                           % If the data set is a PDS
+                                                    % all members are scanned!
+ 1. =?ssp1
+ 2. =?ssp2
+ 3. =?ssp3
+ 4. =?ssp4
+ 5. =?ssp5
+ 6. =?ssp6
+ 7. =?ssp7
+ 8. =?ssp8
+ 9. =?ssp9
+10. =?ssp10
+
+ PO =@po
%      Fill in here more process options. Press PF1 for help and then select a
%      theme by number or press ENTER to scroll thru pages down continuously.
+MSG1=#msglvl1
+
%Press PF1 for Help, press ENTER to execute the search.
)INIT
   &zcmd    = ' '
   .cursor  = ssp1
   .csrpos  = 2
   .help    = ISR313F1 /* standard help panel of SUPER SEARCH 3.15 */
   &msglvl1 = ' '
)REINIT
 if (.msg NE &z)
    .attr(.cursor)='HILITE(REVERSE)'
    &panrc = 1
)PROC
   &panrc = 0
if (.resp = END)
   &panrc = 1
)END
```

The variables SSP1 to SSp14 that contain the search texts are written into the ISPF profile and are again read before the panel is recalled. Therefore, I do not have to reenter the search text on every call to SSS. Reading and writing of the variables using VGET and VPUT could also be performed in the panel, but I decided to do this in the SSS procedure. Here is the excerpt from the REXX procedure SSS which calls the panel SSSP1:

```
/* Read search values stored by previous runs of this program        */
"VGET (SSP1,SSP2,SSP3,SSP4,SSP5,SSP6,SSP7,SSP8,SSP9,PO,",
       "SSP10) PROFILE"
/* Call panel to insert new search values.                            */
"DISPLAY PANEL(SSSP1)"
if panrc = 1 then do   /* In SSSP1 was PF3 pressed.                   */
   exit
end
/* Save search values for next call of this program in profile       */
"VPUT (SSP1,SSP2,SSP3,SSP4,SSP5,SSP6,SSP7,SSP8,SSP9,PO,",
       "SSP10) PROFILE"
```

7.3.2 Example for use of skeletons

Skeletons are mostly used to assemble extensive job streams. For such an application, I have taken the following example.

Function:

In a large job step must be commands inserted which perform BIND PACKAGE for DB2. Since this job must be created for different LPARs, all values that relate to the individual LPARs are kept variable.

Here first the skeleton member. The variables in skeletons always begin with an ampersand (&) character. The association of variables with text or other variables will done by a period (.). Since points must separate the individual qualifiers of DSNs, you will have to use two points when creating DSNs by linking variables in skeletons.

JCL 7.3: Skeleton VINTPAC

```
//* DOC: SKEL VINTPAC  ****************************************
//* DOC: STEP TO EXECUTE BIND PACKAGE COMMANDS
//***********************************************************
//&STEPN  EXEC PGM=IKJEFT01,REGION=4M
//STEPLIB   DD DISP=SHR,DSN=&DSNEXIT
//          DD DISP=SHR,DSN=&DSNLOAD
//          DD DISP=SHR,DSN=&VERLOAD
//SYSPROC   DD DISP=SHR,DSN=&VERREXX0
//          DD DISP=SHR,DSN=&VERREXX1
)IM $ISPFDD NT
//DBRMLIB   DD DISP=SHR,DSN=&DBRMLIB
//RUNLIB    DD DISP=SHR,DSN=&DB2RUN
//BGFILE    DD DISP=SHR,DSN=&BGFILE
//STAT      DD DISP=MOD,DSN=&PACKSTAT..&DB2SYS
//ERROR     DD DISP=MOD,FREE=CLOSE,
//             DSN=&DB2FEHL..&DB2SYS..&GRP
//SYSIN     DD UNIT=VIODA,SPACE=(CYL,(10,10)),
//             DSORG=PS,RECFM=FB,LRECL=80,BLKSIZE=27920
//SYSPRINT  DD UNIT=VIODA,SPACE=(CYL,(5,5)),DSORG=PS,
//             RECFM=FBA,LRECL=133,BLKSIZE=27950
//LIST      DD SYSOUT=*
//SYSTSPRT  DD SYSOUT=*
//SYSTSIN   DD *
  PROFILE NOPREFIX NOMSGID
  ISPSTART CMD(%VINTPAC  &DB2SYS &SYST &PACKZEIT)
  END
//INMEM     DD *
```

In the following part of the program variables are defined that are needed in the skeleton VINTPAC to produce the job step. When the statement **ISPEXEC FTINCL VINTPAC** is executed, the skeleton variables are replaced by the actual values.

```
/*-----------------------------------------------------------------*/
/* Read values for skeleton VINTPAC from locations                 */
/*-----------------------------------------------------------------*/
say exname time() "Begin Step   PACEXEC"
dsnexit = read_location("VPROD"syst "DSNEXIT")
dsnload = read_location("VPROD"syst "DSNLOAD")
dbrmlib = read_location("VPROD"syst "DBR")
db2run  = read_location("VPROD"syst "DB2RUNLIB")
bgfile  = bgut
db2sys  = read_location("VPROD"syst "DB2SYS")
```

Note:

Some variables containing in the skeleton are set already earlier in the program. Therefore, they do not appear in this part of the program.

Here is the created step in which the variables with real values have been replaced:

JCL 7.4: Job step build by skeleton VINTPAC

```
//* DOC: SKEL VINTPAC  *******************************************
//* DOC: STEP TO EXECUTE BIND PACKAGE COMMANDS
//***************************************************************
//PACEXEC    EXEC PGM=IKJEFT01,REGION=4M
//STEPLIB    DD DISP=SHR,DSN=DB2P.ALIAS.DSNEXIT
//           DD DISP=SHR,DSN=DB2P.ALIAS.DSNLOAD
//           DD DISP=SHR,DSN=PROX.LOAD
//SYSPROC    DD DISP=SHR,DSN=PROX.CEXEC
//           DD DISP=SHR,DSN=PROX.CEXEC
//* DOC: SKEL $ISPFDD ******************
//ISPMLIB    DD DISP=SHR,DSN=ISP.SISPMENU
//ISPPLIB    DD DISP=SHR,DSN=ISP.SISPPENU
//ISPSLIB    DD DISP=SHR,DSN=ISP.SISPSENU
//ISPPROF    DD DISP=(NEW,PASS),DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120),
//              SPACE=(TRK,(5,5,10),RLSE),UNIT=VIODA
//ISPTLIB    DD DISP=(SHR,PASS),DSN=*.ISPPROF,VOL=REF=*.ISPPROF
//           DD DISP=SHR,DSN=ISP.SISPTENU
//ISPLOG     DD DUMMY
//****** ENDE $ISPFDD ******************
//DBRMLIB    DD DISP=SHR,DSN=PRDXX.DBRMLIB
//RUNLIB     DD DISP=SHR,DSN=DB2P.ALIAS.RUNLIB.LOAD
//BGFILE     DD DISP=SHR,DSN=PRDXX.DB2PACK
//STAT       DD DISP=MOD,DSN=PROX.PACKSTAT.DB2P
//ERROR      DD DISP=MOD,FREE=CLOSE,
//              DSN=PROX.DB2BIND.ERROR.DB2P.PAC
//SYSIN      DD UNIT=VIODA,SPACE=(CYL,(10,10)),
//              DSORG=PS,RECFM=FB,LRECL=80,BLKSIZE=27920
//SYSPRINT   DD UNIT=VIODA,SPACE=(CYL,(5,5)),DSORG=PS,
//              RECFM=FBA,LRECL=133,BLKSIZE=27950
//LIST       DD SYSOUT=*
//SYSTSPRT   DD SYSOUT=*
//SYSTSIN    DD *
  PROFILE NOPREFIX NOMSGID
  ISPSTART CMD(%VINTPAC  DB2P 0 182;182)
  END
//INMEM      DD *
B97158
B97198
```

This is a complex example from practice. Try to understand the process. All important places are grayed out. We will deal in a later chapter in detail concerning the use of skeletons.

7.3.3 Example for use of tables

ISPF tables are data objects that are normally processed within programs using ISPF. ISPF tables can be viewed and manipulated directly using the ISPF option 7.4. This tool is actually only useful for testing purposes.

Example:

As an example, I want to take the table that is used in the program SLE of the SMART ISPF utilities. It contains the names of the recently edited data sets, along with other information such as date and time of use. First, here is the program section of the edit macro #IMACRO2 that is called when an edit session ends. Here, the Table $SLETAB is opened and the DSN of recently edited data set is inserted in the table. When a member of a PDS was edited the member name will be inserted in the DSN. If the table does not exist, it is created and written after processing into the ISPF library ISPPROF. The following part of the program SLE shows the insertion of a DSN in the table $SLETAB.

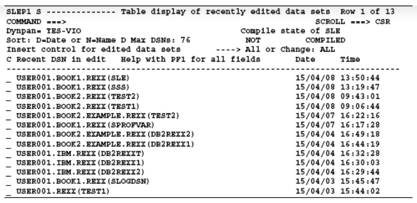
Screentext 7.1: Program part from SLE – Insert a DSN into table $SLETAB

```
/*****************************************************************/
/* Insert the name of the data set into the $SLETAB table.      */
/*****************************************************************/
in_tab_ins:
if zedtmem <> "" then dsn = strip(zedtdsn)"("strip(zedtmem)")"
                 else dsn = strip(zedtdsn)
"VGET (ZSYSID) SHARED"
tabname = "$SLETAB"
"TBOPEN "tabname" LIBRARY(ISPTABL) WRITE SHARE"
openrc = rc
FUNCTION = ""; DATE = date("O"); TIME = time();
select
when openrc = 0 then do /* Table $SLETAB does exist */
   "TBMOD    "tabname" ORDER"
   "TBSORT   "tabname" FIELDS(DATE,C,D,TIME,C,D)"
   "TBCLOSE  "tabname" LIBRARY(ISPTABL) REPLCOPY PAD(100)"
end
when openrc = 8 then do /* Table $SLETAB does NOT exist */
   "TBCREATE "tabname" KEYS(DSN) NAMES(FUNCTION,DATE,TIME)"
   "TBADD    "tabname" ORDER"
   "TBCLOSE  "tabname" LIBRARY(ISPTABL) REPLCOPY PAD(100)"
end
when openrc > 8 then do
   zedlmsg = exname":" zerrlm
   "SETMSG MSG(ISRZ001)"
end
otherwise nop
end /* select */
return
```

Here is the part in the program SLE which displays the panel SLEP1 containing the contents of table $SLETAB.

```
/*---------------------------------------------------------------*/
/* Display table                                                 */
/*---------------------------------------------------------------*/
"TBTOP   "tabname
"TBDISPL "tabname" PANEL(SLEP1) AUTOSEL(NO)"
/* Start error handling */
if rc > 8 then call ispf_error rc "//"||,
"*TBDISPL *tabname* PANEL(SLEP1) AUTOSEL(NO)*//"||,
"TBDISPL "tabname" PANEL(SLEP1) AUTOSEL(NO)"
/* End   error handling */
```

Moreover, here the panel:



```
SLEP1 S -------------- Table display of recently edited data sets  Row 1 of 13
COMMAND ===>                                                   SCROLL ===> CSR
Dynpan= YES-VIO                                 Compile state of SLE
Sort: D=Date or N=Name D Max DSNs: 76            NOT        COMPILED
Insert control for edited data sets       ----> All or Change: ALL
C Recent DSN in edit   Help with PF1 for all fields       Date     Time
-----------------------------------------------------------------------------
_ USER001.BOOK1.REXX(SLE)                                 15/04/08 13:50:44
_ USER001.BOOK1.REXX(SSS)                                 15/04/08 13:19:47
_ USER001.BOOK2.REXX(TEST2)                               15/04/08 09:43:01
_ USER001.BOOK2.REXX(TEST1)                               15/04/08 09:06:44
_ USER001.BOOK2.EXAMPLE.REXX(TEST2)                       15/04/07 16:22:16
_ USER001.BOOK1.REXX(SPROFVAR)                            15/04/07 16:17:28
_ USER001.BOOK2.EXAMPLE.REXX(DB2REXX2)                    15/04/04 16:49:18
_ USER001.BOOK2.EXAMPLE.REXX(DB2REXX1)                    15/04/04 16:44:19
_ USER001.IBM.REXX(DB2REXXT)                              15/04/04 16:32:28
_ USER001.IBM.REXX(DB2REXX1)                              15/04/04 16:30:03
_ USER001.IBM.REXX(DB2REXX2)                              15/04/04 16:29:44
_ USER001.BOOK1.REXX(SLOGDSN)                             15/04/03 15:45:47
_ USER001.REXX(TEST1)                                     15/04/03 15:44:02
```

When I put the cursor in front of a DSN positioned in this display, I can enter selections for edit, view, browse etc. This panel appears again at the end of each action and I can make a new selection. A detailed description of the application is in section 12.5 Example of working with tables on page 218.

7.3.4 Example for use of ISPF variables

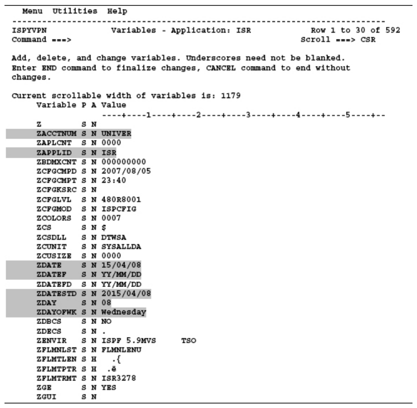
There are two types of ISPF variables:

– System variables

– User variables

System variables are, as its name implies, created and managed by ISPF. You can use them in all ISPF functions. In some, you can change the content. See ZEDLMSG above. Some system variables are assigned to specific ISPF services. Since I have used in the shown examples so far some variables of both types, I will refrain from further examples. However, I want to show you how you can view the currently defined variables and their contents in your ISPF. To do this, I proceed as follows: I choose the ISPF option 7.3 and get the following panel, which is showed shortened below:

Screen 7.2: ISPF variables display using ISPF menu 7.3



```
   Menu  Utilities  Help
 --------------------------------------------------------------------------
 ISPYVPN            Variables - Application: ISR           Row 1 to 30 of 592
 Command ===>                                             Scroll ===> CSR

 Add, delete, and change variables. Underscores need not be blanked.
 Enter END command to finalize changes, CANCEL command to end without
 changes.

 Current scrollable width of variables is: 1179
      Variable P A Value
                   ----+----1----+----2----+----3----+----4----+----5----+--
      Z        S N
      ZACCTNUM S N UNIVER
      ZAPLCNT  S N 0000
      ZAPPLID  S N ISR
      ZBDMXCNT S N 000000000
      ZCFGCMPD S N 2007/08/05
      ZCFGCMPT S N 23:40
      ZCFGKSRC S N
      ZCFGLVL  S N 480R8001
      ZCFGMOD  S N ISPCFIG
      ZCOLORS  S N 0007
      ZCS      S N $
      ZCSDLL   S N DTWSA
      ZCUNIT   S N SYSALLDA
      ZCUSIZE  S N 0000
      ZDATE    S N 15/04/08
      ZDATEF   S N YY/MM/DD
      ZDATEFD  S N YY/MM/DD
      ZDATESTD S N 2015/04/08
      ZDAY     S N 08
      ZDAYOFWK S N Wednesday
      ZDBCS    S N NO
      ZDECS    S N .
      ZENVIR   S N ISPF 5.9MVS     TSO
      ZFLMNLST S N FLMNLENU
      ZFLMTLEN S H   .{
      ZFLMTPTR S H  .ë
      ZFLMTRMT S N ISR3278
      ZGE      S N YES
      ZGUI     S N
```

If permitted by ISPF, you can change the contents of variables, delete them entirely or create new variables in this panel. I grayed some of the Z-variable. These contain values that are often needed in daily operations. To get the contents of ISPF variables, read them using the command:

"ISPEXEC VGET (ZDATESTD ZACCTNUM ZAPPLID ZDAY ZDAYOFWK ZJDATE ZJ4DATE", "ZDATE ZDATEF ZTIME ZTIMEL)"

7.3.5 Example for data processing with TSO and ISPF

For the processing of data sets, you have two options in an ISPF environment and in the use of the language REXX:

– Use the REXX command EXECIO that only needs the support of TSO.

– Use ISPF data set management services. Then you additionally need ISPF data set services.

This distinction is important because it decides whether the procedure to perform I/O operations requires only the TSO level or additionally the ISPF level. This definition is particularly important too, when the intention is to use the procedure in batch jobs. The ISPF environment is in the first case not required but always in the second. This is important for preparation of the batch jobs, as we can see in the chapter 6 Execute REXX programs on page 105.

Example of data set processing with TSO and ISPF:

To explain the difference between I/O operations using the TSO command EXECIO and the ISPF LM services I wrote two programs, which each perform the same work. They read a flight log and summarize the number of flights conducted and hours of flight time for each aircraft type found in the records.

```
/* DOC: IOEXMPL1 REXX MAIN                                           */
/* DOC: Use the EXECIO method to perform I/O operations in REXX.     */
/* DOC: Function of this program:                                    */
/* DOC: Calculate the number of flights per aircraft type and the    */
/* DOC: related total flight time in hours and put the result in the */
/* DOC: member RESULT4 into the same data set.                       */
/* © FRANZ LANZ 2015                                                 */
/********************************************************************/
/*------------------------------------------------------------------*/
/* Part one: Read the flight log. The records are in the stem flights*/
/*------------------------------------------------------------------*/
"alloc dd(in1) dsn('essmstr.flight.data(flights)') reuse shr"
"execio * diskr in1(stem flights. finis"
"free dd(in1)"
/*------------------------------------------------------------------*/
/* Part two: Calculate the results                                   */
/*------------------------------------------------------------------*/
types = ""  /* String for collecting the airplane type names         */
ft_nbr. = 0 /* Stem for adding the number of flights                 */
ft_hrs. = 0 /* Stem for adding the summary of flight hours           */
do i = 1 to flights.0 /* Loop over all data set records read         */
   ft = word(flights.i,9) /* The airplane type is in the ninth word  */
   if wordpos(ft,types) = 0 then types = types ft
   /* This IF checks whether an airplane type is still in string     */
   /* and if not, the new name is added to the existing types        */
   ft_nbr.ft = ft_nbr.ft + 1  /* Count the number of flights/type    */
   ft_hrs.ft = ft_hrs.ft + translate(word(flights.i,6),'.',',')
   /* Add the flight times for this type                             */
end i
/*------------------------------------------------------------------*/
/* Part three: Print the results                                     */
/*------------------------------------------------------------------*/
do i = 1 to words(types)
   ft = word(types,i)
   out.i = left(ft,5) right(ft_nbr.ft,5) right(ft_hrs.ft,8)
   say out.i
end i
/*------------------------------------------------------------------*/
/* Part four: Write the results into a data set member               */
/*------------------------------------------------------------------*/
"alloc dd(ut1) dsn('essmstr.flight.data(result4)') reuse shr"
"execio "words(types)" diskw ut1(stem out. finis"
"free dd(ut1)"
exit
```

Program 7.3: IOEXMPL2 – Read records using LM services of ISPF

```
/* DOC: IOEXMPL2 REXX MAIN                                           */
/* DOC: Use the ISPF LM services to perform I/O operations in REXX.  */
/* DOC: Function of this program:                                    */
/* DOC: Calculate the number of flights per aircraft type and the    */
/* DOC: related total flight time in hours and put the result in the */
/* DOC: member RESULT4 into the same data set.                       */
/* © FRANZ LANZ 2015                                                 */
/********************************************************************/
/*------------------------------------------------------------------*/
/* Part one: Read the flight log. The records are in the stem flights*/
/*------------------------------------------------------------------*/
indsn = "'essmstr.flight.data'"
"ISPEXEC LMINIT DATAID(IN) DATASET("indsn") ENQ(SHR)"
"ISPEXEC LMOPEN DATAID("in") OPTION(INPUT)"
"ISPEXEC LMMFIND DATAID("IN") MEMBER(flights) STATS(NO)"
il = 0 /* Counter for records read */
do iline = 1
   "ISPEXEC LMGET DATAID("in") MODE(INVAR) DATALOC(LINE) DATALEN(INL)",
            "MAXLEN(80)"
   if rc = 8 then leave iline /* EOF of member flights */
   il = il + 1
   flights.il = line /* Put line into stem */
end iline
flights.0 = il-1 /* Put number of read lines into flights.0 */
"ISPEXEC LMCLOSE DATAID("in")"
"ISPEXEC LMFREE  DATAID("in")"
/*------------------------------------------------------------------*/
/* Part two: Calculate the results                                   */
/*------------------------------------------------------------------*/
types = ""  /* String for collecting the airplane type names         */
ft_nbr. = 0 /* Stem for adding the number of flights                 */
ft_hrs. = 0 /* Stem for adding the summary of flight hours           */
do i = 1 to flights.0 /* Loop over all data set records read         */
   ft = word(flights.i,9) /* The airplane type is in the ninth word  */
   if wordpos(ft,types) = 0 then types = types ft
   /* This IF checks whether a airplane type is still in string      */
   /* and if not, the new name is added to the existing types        */
   ft_nbr.ft = ft_nbr.ft + 1  /* Count the number of flights/type    */
   ft_hrs.ft = ft_hrs.ft + translate(word(flights.i,6),'.',',')
   /* Add the flight times for this type                             */
end i
/*------------------------------------------------------------------*/
/* Part three: Print the results                                     */
/*------------------------------------------------------------------*/
do i = 1 to words(types)
   ft = word(types,i)
   out.i = left(ft,5) right(ft_nbr.ft,5) right(ft_hrs.ft,8)
   say out.i
end i
/*------------------------------------------------------------------*/
/* Part four: Write the results into a data set member               */
/*------------------------------------------------------------------*/
"ISPEXEC LMINIT DATAID(UT) DATASET("indsn") ENQ(SHRW)"
"ISPEXEC LMOPEN DATAID("ut") OPTION(OUTPUT)"
do i = 1 to words(types)
   ft = word(types,i)
   out.i = left(ft,5) right(ft_nbr.ft,5) right(ft_hrs.ft,8)
   record = out.i
   "ISPEXEC LMPUT DATAID("ut") MODE(INVAR) DATALOC(RECORD) DATALEN(80)"
end i
"ISPEXEC LMMREP  DATAID("ut") MEMBER(RESULT4)"
"ISPEXEC LMCLOSE DATAID("ut")"
"ISPEXEC LMFREE  DATAID("ut")"
exit
```

These both programs produce exactly the same result. I have grayed the statements used for each I/O method. It is obvious that the method with EXECIO by far requires the lower programming costs. The actually necessary error query and error analysis after each ISPF command has been deliberately omitted to improve clarity of the code.

Notes:

The processing of data sets using ISPF LM services is much faster than the method using TSO EXECIO. In addition the error checking and error analysis that ISPF provides is much better than TSO offers. Sometimes the higher programming costs are worth.

7.3.6 Output of messages with ISPF

For the issuance of messages in the ISPF, a standard procedure is available. In its use, ISPF system variables will be filled with contents. A standard ISPF service dis- plays these special variables on the screen. This service is only usable for outputting messages online on a screen. It is not usable to print the messages in a batch job. If a REXX procedure shall be run online as well as in batch the output of messages must be controlled accordingly. See section 9.1.2 Output of error messages on page 178. The following excerpt from the **Appendix D Dialog variables** of the IBM manual **ISPF Dialog Developer's Guide and Reference** shows the ISPF system variables concerning error messages:

Table 7.1: ISPF system variables concerning error messages

		alarm was specified in the message definition otherwise, the value NO. Set when ISPF services issue a return code of 8 or greater.
ZERRHM	Char 8	The name of a Help panel, if one was specified in the message definition. Set when ISPF services issue a return code of 8 or greater.
ZERRLM	Char 512	Long-message text in which variables have been resolved. Set when ISPF services issue a return code of 8 or greater.
ZERRMSG	Char 8	Message ID. Set when ISPF services issue a return code of 8 or greater.
ZERRSM	Char 24	Short-message text in which variables have been resolved. Set when ISPF services issue a return code of 8 or greater.

How messages are displayed?

ISPF offers a standard method to display messages when a program runs online. These messages are displayed on the ISPF screen. The ISPF service **SETMSG** is used to display the messages. The SETMSG service expects the error messages in the two special variables:

ZEDSMSG

ZEDLMSG

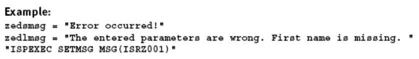
S and L stand for SHORT and LONG. This means that you can put a message up to **24** characters in ZEDSMSG and in ZEDLMSG a long message up to **512** characters. See Screentext 7.1: Program part from SLE – Insert a DSN into table $SLETAB on page 115.

The ISPF service SETMSG writes the error message texts contained in ZEDSMSG and ZEDLMSG on the currently displayed screen. However, the following rules are to be considered:

– If the variable ZEDSMSG contains a text, then initially only this text is displayed on the right top corner of the screen and nothing else happens. First, when the user presses PF1, then the related ZEDLMSG message will also be displayed on the bottom of the screen.

– If the variable ZEDSMSG contains nothing, only the ZEDLMSG message is immediately displayed on the bottom of the screen.

Example:



Example:

```
zedsmsg = "Error occurred!"
zedlmsg = "The entered parameters are wrong. First name is missing. "
"ISPEXEC SETMSG MSG(ISRZ001)"
```

Notes:

The SETMSG service is usable for every display of messages. You can generate an error message in ZEDLMSG by combining the content of ZERRLM together with your own texts. You can fill the variables with a text of your choice and display them. It is not necessary that a program error situation is present.

Program example:

The following program shows two typical applications of the ISPF message services:

– In the first use of the service are the names of data sets, which have an ENQ reservation displayed. The function QUERYENQ detects such ENQs. These are the statements 202 to 212.

– The second application displays an error message when a data set should be edited for which the editor is unable to edit.

Screentext 7.2: Example of the ISPF messages service

```
when vcactnx <= 3 then do
   uttyp "DATASET("dsn")" /* uttyp -> EDIT, VIEW, BROWSE     */
   select
      when rc <= 4 then nop
      when rc = 14 then do /* When edit & RC=14 data set is enqued */
         if words(s) > 1 then enqt = left(word(s,1),44)word(s,2)
                         else enqt = left(word(s,1),44)
         s = queryenq(enqt)
         zedlmsg =,
           "Data set "dsn" is ENQUEUED by following users:"
         do i = 1 to enqjob.0
            zedlmsg = zedlmsg word(enqjob.i,1)
         end i
         "SETMSG MSG(ISRZ001)"
      end
      otherwise do
         zedlmsg = "Error at invocation of "uttyp":" strip(zerrlm)
         "SETMSG MSG(ISRZ001)"
      end
   end /* select */
end
otherwise do
```

There are two possible messages displayed:

1. An attempt was made to edit an ENQUED data set.

```
*---------------------------------------------------------------------------*
| Data set 'USER001.BOOK2.REXX(TEST2)' is ENQUEUED by following users: USER001 |
*---------------------------------------------------------------------------*
```

2. This error message is self-explanatory.

```
| Error at invokation of: PROX.SA.SIROS.VSTAF is a VSAM data set. An attempt |
| was made to invoke a VSAM editor, viewer, or browser, but it is not allowed |
| due to configuration table settings.                                       |
```

7.4 LIBDEF – DYNAMIC LINKING OF ISPF LIBRARIES

We have already discussed the ISPF libraries earlier in this book. And we have also seen that some library chains must be assigned before ISPF starts. However, if you want to run an application of ISPF objects stored in own ISPF libraries; you have to copy all these objects in the appropriate permanently assigned libraries. As this would be very costly and impractical, ISPF contains a command to allocate additional libraries dynamically before the existing permanently allocated libraries. This is the LIBDEF service.

Function:

Allocate additional data sets to the existing ISPF libraries dynamically and remove such assignments again.

Format:

```
ISPEXEC LIBDEF  lib-type
                [DATASET|EXCLDATA|LIBRARY|EXCLLIBR]
                [ID(dataset-list)|ID(libname)]
                [COND|UNCOND|STACK|STKADD]
```

`lib-type`	This indicates the library type for the allocation to be done. For the following types of libraries a LIBDEF allocation can be performed: `ISPMLIB` Message library `ISPPLIB` Panel library `ISPSLIB` Skeleton library `ISPTLIB` Table input library `ISPTABL` Table output library `ISPFILE` File-tailoring output data set `ISPLLIB` Load module library

`[DATASET|EXCLDATA|LIBRARY|EXCLLIBR]`

`DATASET`	Specifies that the data in ID identify a number of DSNs.
`EXCLDATA`	Specifies in a LIBDEF ISPLLIB command that the data in ID identify a number of DSNs.
`LIBRARY`	Specifies that the data in ID identify a DD name.
`EXCLLIBR`	Specifies in a LIBDEF ISPLLIB command that the information in ID identify a DD name.

`[ID(dataset-list)|ID(libname)]`

`dataset-list`	Up to 16 DSNs can be specified here. Define the dataset-list as follows: 'DSN1','DSN2','DSN3' etc. See the example below.
`libname`	A DD name can be specified here.

`[COND|UNCOND|STACK|STKADD]`

With these options, you can define how the LIBDEF chains shall be arranged.

Advice:

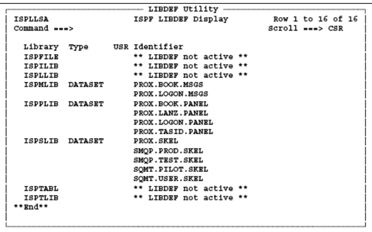
Always use the STACK option. If this option is missing when creating the LIBDEF and you reset this LIBDEF assignment, then all pre-existing LIBDEFs also cancels. If you have called a new application from another application which also has LIBDEFs, their LIBDEFs are also withdrawn. This means that you can no longer work with the previous application after return.

Program 7.4: LIBDEF example

```
/* DOC: REXX LIBDEF   ******************************************/
/* DOC: LIBDEF example.                                        */
/* © FRANZ LANZ 2015                                           */
/***************************************************************/
address ISPEXEC
"CONTROL ERRORS RETURN"
"LIBDEF ISPSLIB DATASET ID('PEVP.SKEL',",
                          "'SMQP.PROD.SKEL',",
                          "'SMQP.TEST.SKEL',",
                          "'SQMT.PILOT.SKEL'",
                          "'SQMT.USER.SKEL') STACK"
if rc > 0 then call ispf_error
"LIBDEF ISPPLIB DATASET ID('PROX.BOOK.PANEL',",
                          "'PROX.LANZ.PANEL',",
                          "'PROX.LOGON.PANEL',",
```

Use the ISPF command ISPLIBD to display the current LIBDEF assignment. See the following screen. It shows the currently assigned LIBDEFs within this logical ISPF screen.

Screen 7.3: ISPLIBD – Display the current active LIBDEF assignment



```
------------------------------ LIBDEF Utility --------------------------------
ISPLLSA                   ISPF LIBDEF Display           Row 1 to 16 of 16
Command ===>                                           Scroll ===> CSR

  Library  Type     USR Identifier
  ISPFILE               ** LIBDEF not active **
  ISPILIB               ** LIBDEF not active **
  ISPLLIB               ** LIBDEF not active **
  ISPMLIB  DATASET      PROX.BOOK.MSGS
                        PROX.LOGON.MSGS
  ISPPLIB  DATASET      PROX.BOOK.PANEL
                        PROX.LANZ.PANEL
                        PROX.LOGON.PANEL
                        PROX.TASID.PANEL
  ISPSLIB  DATASET      PROX.SKEL
                        SMQP.PROD.SKEL
                        SMQP.TEST.SKEL
                        SQMT.PILOT.SKEL
                        SQMT.USER.SKEL
  ISPTABL               ** LIBDEF not active **
  ISPTLIB               ** LIBDEF not active **
**End**
```

Advice:

The LIBDEF definitions apply only to the logical ISPF screen in which they are executed. This is necessary because otherwise you could not even run different ISPF applications simultaneously on several logical ISPF screens.

7.5 ALTLIB – DYNAMIC LINKING OF EXEC LIBRARIES

With LIBDEF, only ISPF libraries can be set before the default associated libraries. However, when a REXX program executes, the system searches for the program only in the libraries that are allocated under SYSEXEC or SYSPROC. With the ALTLIB command other data sets can dynamically be assigned before the SYSEXEC and SYSPROC libraries. The ALTLIB command has to be executed as a TSO command. As we will see below, the ALTLIB command works slightly different in an ISPF environment as in a pure TSO environment.

The ALTLIB command can achieve the following:

– Define alternative libraries for procedures calls.

– Define the user, application and system-level libraries that are used for searching programs.

– Exclude one or more libraries level from the search.

– Reset the search order to the system level.

– View the current search order.

Normally, the TSO is set so that when a procedure is called, then the SYSEXEC library is first searched and afterwards the program is searched in the SYSPROC library. This is of course only valid if both library types are assigned.

The command TSO ALTLIB DISPLAY displays the current ALTLIB setting:

Example:

Current search order (by DDNAME) is:

System-level EXE CDDNAME=SYSEXEC

System-level CLIST DDNAME=SYSPROC

7.5.1 Search sequence in the procedures libraries

The following table shows the search order for procedures libraries. The corresponding DD names will also displayed. These DD names can either be created dynamically with the ALLOCATE command or have been created even by the login procedure.

Table 7.2: Search sequence in the procedures libraries

Nbr.	Library Level	Type	Assigned DD name
1	User	REXX Exec	SYSUEXEC
2	User	CLIST	SYSUPROC
3	Application	REXX Exec	Defined by DD name or DATASET option
4	Application	CLIST	Defined by DD name or DATASET option
5	SYSTEM	REXX Exec	SYSEXEC
6	SYSTEM	CLIST	SYSPROC

Example 1:

Temporary activation of a library for executing a REXX procedure from that data set.

"ALTLIB ACTIVATE APPLICATION(EXEC) DDNAME(##DD)"

"ISPEXEC SELECT CMD(%"mem strip(pp)")"

Here is the data set that is already allocated under the DD name **##DD,** provided for the call of the procedure named in the SELECT statement.

Example 2:

Disable the CLIST system level.

TSO ALTLIB DEACTIVATE SYSTEM(CLIST)

When I now execute the command TSO ALTLIB DISPLAY, the following display occurs:

Current search order (by DDNAME) is: System-level
EXEC DDNAME=SYSEXEC

This means that procedures in the SYSPROC allocation will not be found regardless whether the allocation exists or not.

Example 3:

I have written a small REXX procedure that executes some ALTLIB commands.

Program 7.5: ALTLIB1: Example of the application of the command ALTLIB

```
/* DOC: REXX ALTLIB1                                               */
/* DOC: Example 1 for the ALTLIB command under TSO                 */
/* © FRANZ LANZ 2015                                               */
/*****************************************************************/
"altlib display"
"alloc dd(sysuexec) dsn(PROX.BOOK.REXX,",
                     !!"PROX.UTIL.REXX) shr reuse"
"altlib act user(exec)"
say
"altlib display"
"altlib reset"
say
"altlib display"
"ispexec qbaselib sysuexec id(dsns)"
say
say "DSNS for DD Name SYSUEXEC: "dsns
"free dd(sysuexec)"
```

When the procedure executes, the following display appears:

```
Current search order (by DDNAME) is:
System-level EXEC        DDNAME=SYSEXEC
System-level CLIST       DDNAME=SYSPROC

Current search order (by DDNAME) is:
User-level EXEC          DDNAME=SYSUEXEC
System-level EXEC        DDNAME=SYSEXEC
System-level CLIST       DDNAME=SYSPROC

Current search order (by DDNAME) is:
System-level EXEC        DDNAME=SYSEXEC
System-level CLIST       DDNAME=SYSPROC

DSNS for DD Name SYSUEXEC: 'PROX.BOOK.REXX','PROX.UTIL.REXX'
```

7.5.2 The ALTLIB command in ISPF

As long as ALTLIB commands are used only under TSO, the once by ALTLIB set search sequences remain active until they are changed with another ALTLIB command. On the other hand, when the ALTLIB command is called under the ISPF environment, this rule changes as follows:

Rules when ALTLIB is used in the ISPF environment:

– If you use ALTLIB while ISPF is active, you can issue this command on the application level. Libraries that are allocated by a running application remain active as long as this application is running. When the application ends, the state is the same as before calling the application.

– When an application is started in the ISPF split screen mode which settles ALTLIB commands, then their defined search order applies only to this split screen panel.

– Libraries that are allocated at the start of an application in the ISPF will not be deactivated by the NEWAPPL and PASSLIB parameters in the SELECT command.

– If an application is launched while NEWAPPL is specified, but not PASSLIB, you will not have all LIBDEF and ALTLIB pre-allocated libraries in this application available. They are only available again when this application ends.

– If an application is started using NEWAPPL and PASSLIB, then all library allocations previously created with LIBDEF and ALTLIB remain active. However, LIBDEF and ALTLIB allocations that are made in this application are only active as long as the application is running.

– If neither NEWAPPL nor PASSLIB are specified when starting an application, then all previously with LIBDEF and ALTLIB created allocations are active in this application. Any changes made using LIBDEF and ALTLIB in this application remain active even after the end of the application.

– ALTLIB allocations that were made before starting of ISPF are unknown after the call to ISPF.

Conclusion:

7.5.3 Stacking of the APPLICATION level ALTLIBs

If multiple ALTLIB commands are issued on the APPLICATION level within an ap- plication, they will be stacked up to a depth of eight levels. This stacking takes place so that always the last created allocation is at the top.

Example: stacking APPLICATION Level ALTLIBs.

Program 7.6: Stacking ALTLIB commands

```
/* DOC: REXX ALTLIB2                                  */
/* DOC: Example: Executing of ALTLIB in the ISPF      */
/* © FRANZ LANZ 2015                                  */
/*****************************************************/
"altlib display"
say
"altlib act application(exec) dataset(PROX.BOOK.REXX)"
"altlib display"
say
"altlib act application(exec) dataset(PROX.UTIL.REXX)"
"altlib display"
say
"altlib act application(exec) dataset(PROX.LANZ.REXX)"
"altlib display"
say
"altlib act application(exec) dataset(PROX.LOGON.REXX)"
"altlib display"
say
"altlib reset"
```

Here is the output that generates the above procedure:

```
Current search order (by DDNAME) is:
System-level EXEC       DDNAME=SYSEXEC
System-level CLIST      DDNAME=SYSPROC

Current search order (by DDNAME) is:
Application-level EXEC  DDNAME=SYS00002
System-level EXEC       DDNAME=SYSEXEC
System-level CLIST      DDNAME=SYSPROC

Current search order (by DDNAME) is:
Application-level EXEC  DDNAME=SYS00003
                Stacked DDNAME=SYS00002
System-level EXEC       DDNAME=SYSEXEC
System-level CLIST      DDNAME=SYSPROC

Current search order (by DDNAME) is:
Application-level EXEC  DDNAME=SYS00004
                Stacked DDNAME=SYS00003
                Stacked DDNAME=SYS00002
System-level EXEC       DDNAME=SYSEXEC
System-level CLIST      DDNAME=SYSPROC

Current search order (by DDNAME) is:
Application-level EXEC  DDNAME=SYS00005
                Stacked DDNAME=SYS00004
                Stacked DDNAME=SYS00003
                Stacked DDNAME=SYS00002
System-level EXEC       DDNAME=SYSEXEC
System-level CLIST      DDNAME=SYSPROC
```

As you can see, only the DD names of allocated data sets are displayed. If you want to see the associated DSNs, you can view them using the ISPF command DDLIST.

7.5.4 The QUIET operand of the ALTLIB DISPLAY command

If you use the ALTLIB DISPLAY command in ISPF, you can specify the QUIET option. This option ensures that the ALTLIB DISPLAY command output information will not be outputted via SYSTSPRT but rather written in ISPF variables in the SHARED pool. You can then use these ISPF variables to use the information contained therein in the program.

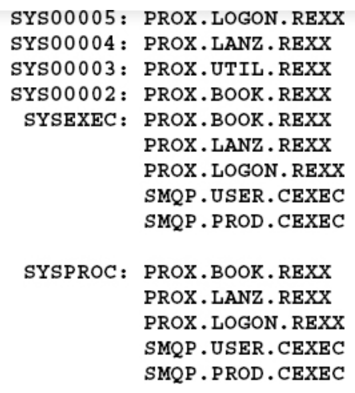
The following ISPF variables will be filled:

– IKJADM contains the number of rows that were written in ISPF variables.

– The variables IKJADM1 to IKJADM99 contain the individual lines. The following example shows how you can use these service in a program:

Program 7.7: ALTLIB3 – Using of ALTLIB DISPLAY QUIET command

```
/* DOC: REXX ALTLIB3 ***************************************/
/* DOC: Using the command ALTLIB DISPLAY QUIET under ISPF */
/* © FRANZ LANZ 2015                                      */
/*********************************************************/
"altlib act application(exec) dataset(PROX.BOOK.REXX)"
"altlib act application(exec) dataset(PROX.UTIL.REXX)"
"altlib act application(exec) dataset(PROX.LANZ.REXX)"
"altlib act application(exec) dataset(PROX.LOGON.REXX)"
"altlib display quiet"
"ispexec vget (ikjadm) shared
do i = 1 to ikjadm
   "ispexec vget (ikjadm"i") shared"
   z = value("ikjadm"i)
   parse var z x "DDNAME=" ddname .
   if ddname <> "" then do
      "ispexec qbaselib "ddname" id(dsns)"
      dsns = translate(dsns,"  ","'",")
      say right(ddname":",9) word(dsns,1)
      do k = 2 to words(dsns)
         say "          " word(dsns,k)
      end k
   end
   say
end i
"altlib reset"
```



```
SYS00005: PROX.LOGON.REXX
SYS00004: PROX.LANZ.REXX
SYS00003: PROX.UTIL.REXX
SYS00002: PROX.BOOK.REXX
  SYSEXEC: PROX.BOOK.REXX
           PROX.LANZ.REXX
           PROX.LOGON.REXX
           SMQP.USER.CEXEC
           SMQP.PROD.CEXEC

  SYSPROC: PROX.BOOK.REXX
           PROX.LANZ.REXX
           PROX.LOGON.REXX
           SMQP.USER.CEXEC
           SMQP.PROD.CEXEC
```

Recommended / Playlists / History / Topics / Settings / Get the App / Sign Out

PREV
6 Execute REXX programs

NEXT
8 Data set processing using ISPF