

ISPF Programmer's Guide

PREV

14 Edit macros – Create and apply

NEXT

List of programs

15 The SMART ISPF utilities

Since the early 1980ies, I have been working with ISPF. Over the time, I often missed some practical functions in ISPF. In the early 1980ies, no internet was available. Therefore, it was very difficult to find appropriate functions to fulfil my needs. In this situation, I began to develop these functions from time to time myself. Now a whole series of programs have accumulated that I use in my daily work in ISPF. They are all REXX programs in which I very often use the functions provided by ISPF. Often there are functions consisting of one program only. Sometimes it happens, however, that in order to fulfill the functions, more programs have to work together. Although there are many utilities for ISPF available on the internet, I think that my programs so far are still unique.

In my two books on **Smart Practices for ISPF**, I make the SMART ISPF utilities available for you to download, so that these utilities provide good services.

15.1 NAMING CONVENTIONS

All programs are written in REXX language. There are generally two types of programs:

Edit macros:

These program names all begin with character # (hash). Using these names rule, the macros are easily distinguishable from the other programs.

Direct executable:

These program names all begin with character S (stands for smart).

15.2 THE DYNAMIC PANEL CONCEPT

All panels of SMART ISPF utilities are embedded in the programs REXX code and are automatically loaded in a temporary ISPLLIB of DD Name SSDYNPAN when a program executes.

This technique has the advantage that you do not need a separate ISPLLIB version of ISPF panels to run SMART ISPF utilities. Another advantage of this procedure is that REXX programs using ISPF panels are easily distributable because the simultaneous distribution of a panel data set is not necessary.

The following excerpt from a program of the SMART ISPF utilities shows the part where the decision is made to load the panels:

```
/*-----*/
/* Check loading dynamic panels */
/*-----*/
*VGET (DYNPAN) PROFILE*
if rc > 0 then dynpan = "NO"
if dynpan = "YES" then call store_panels /* Load embedded panels */
/*-----*/
```

To decide whether the panels are to be loaded dynamically, the variable DYNPAN from the ISPF profile will be read in. The program SPROFVAR loads this variable together with other variables needed by some SMART utility programs.

Urgent recommendation:

Before you can use the SMART ISPF utilities, you must edit the program SPROFVAR, the values contained therein, customize them and run the edit macro ##. This edit macro is executable without having to load the ISPF profile variables for the SMART ISPF utilities. See the following excerpt from program

Find answers on the fly, or master something new. Subscribe today. See pricing options.

https://learning.oreilly.com/library/view/ispf-programmers-guide/9783110407655/xhtml/Ch15.xhtml

1/19

```
jobclass = 'A'          /* Job Run class      */
msgclass = 'E'          /* Job message class */
dynpan = 'YES'          /* Dynamic panel store indicator */
dynunit = 'VIO'         /* Unit to alloc dynamic panels */
                        /* data set.          */
```

See also the description of the installation process on page 286.

15.3 LIST OF EXECUTABLE PROGRAMS

The following table shows all programs and their functions:

Table 15.1: List of the SMART ISPF utilities programs

Name	Usage	Description
#	stand alone	Edit macro to execute a currently edited REXX procedure immediately.
#ALTXT	stand alone	Realign documentation texts within given borders.
#EDMEM	stand alone	Edit a member which name is the first word in a line of a data currently being edited.
#IMACROA	assist SLE	General ISPF edit macro. Automatically executed by ISPF when an edit session starts even if no initial macro is set for the data set currently being edited.
#IMACRO1	assist SDOC	Initial macro to insert DOC: lines into a data set, which is called for editing.
#IMACRO2	assist SLE	This macro inserts the DSN of an edited data set into the table \$SLETAB.
#ISPFB	stand alone	Edit macro to submit a batch job to execute the REXX procedure currently being edited within an ISPF environment.
#LCH	stand alone	Offers the possibility to perform an edit change command with a long parameter list.
#SSS	assist SSS	Purges unnecessary parts of a SRCHFOR list.
#SSSCH	assist SSS	This macro can be used when a super search list currently being edited to restore the displayed lines back into the members in which the searched lines were found.
#SPLJ	stand alone	This macro performs SPLIT and JOIN operations within an edit session.
#SU	stand alone	Submit a JCL that does not contain a JOB statement.
#TSOB	stand alone	Edit macro to submit a batch job that executes the currently edited REXX procedure within a TSO environment.
#VERASE	stand alone	Erase an ISPF variable from the profile and the shared pool.



Find answers on the fly, or master something new. Subscribe today. [See pricing options.](#)

		or DSINFO. The DSN must appear on an ISPF panel.
SDOC	stand alone	This program reads all members of a PDS and produces the \$DOC member in the same PDS.
SHOSTPAC	stand alone	This macro grabs all members of a PDS and saves the result under the name of the data sets LLQ in the same PDS.
SHOSTUNP	stand alone	Unpack a packed member coming as an upload from a PC which was originally packed at a z/OS by the REXX program SHOSTPAC.
SICMD	stand alone	Maintaining ISPF command tables.
SIDCAMS	stand alone	Produce an IDCAMS LISTCAT of a data set and browse the list. Call in front of a DSN in a DSLIST display.
SJOBSUFF	function	Provide job suffix character for dynamic job assembly.
SLE	stand alone	Display recently edited data sets.
SLIBOFF	stand alone	Reset a LIBDEF.
SLISTC	stand alone	Produce an IDCAMS LISTCAT of a data set and browse the list. Call in front of a DSN in a DSLIST display.
SLISTRAC	stand alone	Produce a RACF LISTDSD of a data set and browse the list. Call in front of a DSN in a DSLIST display.
SLOGON	stand alone	SMART user logon procedure.
SPROFEDT	stand alone	Execute program SPROFVAR to store users ISPF profile environment variables into ISPF profile.
SPROFVAR	stand alone	Store the user's environment variables into ISPF profile.
SSC	stand alone	SMART super clone function for data sets.
SSCo1	assist SSC	Messages for panel SSCPPo1 called by function SSC.
SSS	stand alone	Perform a super-search operation by entering SSS in front of a DSN in a DSLIST panel.

15.4 PROGRAM DESCRIPTIONS

This chapter contains individual descriptions of the programs of SMART ISPF utilities. Most programs include extensive help descriptions displayable using the PF1 key. When calling up some programs, a question mark can be entered as parameter. In this case, the program will not run with its functionality. It will only display a description of its functions in a help panel. The following description of



Find answers on the fly, or master something new. Subscribe today. See pricing options.

15.4.1 Edit macro ## - Execute a currently edited REXX procedure

Edit macro to execute a currently edited REXX procedure. After typing ##, enter a parameter string which is transferred to the REXX program for execution. This program is executable without having to load the ISPF profile variables for the SMART ISPF utilities. Help display with ## ?

15.4.2 Edit macro #ALTXT - Realign line parts

Edit macro to realign line parts within given columns. Help display with #ALTXT ?

15.4.3 Edit macro #EDMEM - Edit of a member

Edit of a member which name is the first word in a line of a data set currently being edited.

Description:

The edited list can come from following sources:

- A list output of the procedure SSS and processed by the #SSS macro.
- A list output produced by ISPF function SRCHFOR in the resulting display list.
- The member SDOC from a PDS.

At call, the cursor must be in a line in which the member name occurs as first word. No further help available.



Note:

For an optimal use of this macro, its call should be set on a PF key. E.g. PF22.

15.4.4 Edit macro #IMACROA - General ISPF edit macro

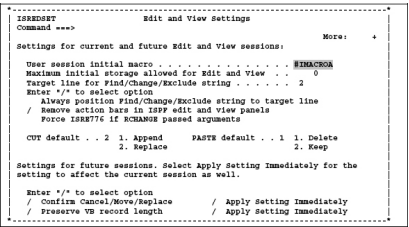
General ISPF edit macro. Whenever you start an edit session, this macro will execute, regardless of whether an initial macro is defined for this data set or not.

Activating of this macro:

1. Start an edit session in an ISPF environment.
2. Enter in line command == => EDITSET or EDSET
3. Make the necessary entries in the appearing panel. See following panel:

No further help available.

Screen 15.1: Set activation of #IMACROA



15.4.5 Edit macro #IMACRO1 - Initial edit macro

Initial macro to insert DOC: lines into edited members of partitioned data sets (PDS).

When starting an edit session of a PDS member and no DOC lines are found in the edited member within the first 20 lines, this macro then displays a panel that offers the opportunity to insert DOC lines at the beginning of the member currently being edited.

Help can be displayed via PF1 when the panel IMACRO11 appears. The following screen shows an example for inserting DOC and REM lines into a member.

Screen 15.2: Panel IMACRO11 for edit macro #IMACRO1



Find answers on the fly, or master something new. Subscribe today. [See pricing options.](#)

```
IMACRO1 ----- Insert DOC lines into edited data set -----
Command ---->
Display = TRS-VID      Sysid = TESTEX  Sysid = TESTMF  Mode = 2019
Data-set-name = LAMSTR.SSCH.REXX(AAA)
Comment-Prefix = #      Comment-Suffix = #
Cape      = OFF         set CAPE ON or OFF
Data set type = REXX     s.g. REXX, JCL, PARMLIB and so on
Function type = MAIN     s.g. MACRO, MAIN, COPY, SUBR
Filling unit = #         s.g. (asterisk), (hyphen)
These DOC and REM lines are inserted at beginning of data set
Author = " Frank Laus 2015
1. DOC - This is an example
   DOC - how DOC lines are inserted
   DOC - in a new
   DOC - edited member
   DOC - of a PDS
2. REM - Remarks
   REM - can also
   REM - be inserted
   REM - These remarks are not inserted in the $DOC member of the PDS.
   REM - The $DOC member is produced using the program $DOC.
Compile status of IMACRO1: NOT COMPILED
```

The greyed fields show the standard options inserted here by #IMACRO1 depending on LLQ. These options are changeable. The following screen shows the resulting contents of the new member after ENTER is pressed:

```
IRREDC04 LAMSTR.SSCH.REXX(AAA) - 01.00      Column 0001 0002
Command ---->      Scroll ----> CSR
000001 /* DOC: AAA      REXX MAIN
000002 /* DOC: This is an example
000003 /* DOC: how DOC lines are inserted
000004 /* DOC: in a new
000005 /* DOC: edited member
000006 /* DOC: of a PDS
000007 /*-----
000008 /* " Frank Laus 2015
000009 /******
000010 /* REM: Remarks
000011 /* REM: can also
000012 /* REM: be inserted
000013 /* REM: These remarks are not inserted in the $DOC member of the PDS.
000014 /* REM: The $DOC member is produced using the program $DOC.
000015 /******
***** Bottom of Data *****
```

15.4.6 Edit macro #IMACRO2 - Edit session end macro

This macro inserts the DSN of an edited data set into the table \$SLETAB. This macro is executed when an edit session ends in which the macro #IMACRO1 was executed. No further help available.

15.4.7 Edit macro #ISPFB - Submit an ISPF batch job

Edit macro to submit a **batch job** to execute the REXX procedure currently being edited within an ISPF environment. When the macro is called, a parameter text can be added in the call statement. This text is as a parameter transferred to the executed REXX procedure. Help display with #ISPFB ?

15.4.8 Edit macro #LCH - Perform long edit change commands

Perform edit change with long parameter texts.

Function:

When the "Command ==>" line in the normal edit panel is too short to contain the whole CHANGE command, you can use this program to solve the problem. This program opens a panel for input of two lines each with max 70 characters. The first line is the SEARCH argument for the change and the second line is the replacement value. Additionally, you can enter parameters for the CHANGE in a third line. E.g. 1, ALL, WORD...

15.4.9 Edit macro #SPLJ - SPLIT and JOIN lines

This macro performs SPLIT and JOIN operations within an edit session.

Functional description:

When this macro is called, further processing of the position of the cursor depends on the following:

- 1. If the cursor is within a line and after this position, characters are on the line, these characters are moved to a **new** line after the current one. This is the split function.
- 2. If after the position of the cursor only **blanks** are on the line, the line after the current line moves to the current line after the cursors position. This is the join function.

There is no other help available.

15.4.10 Edit macro #SSS - Clear SCHFOR lists

Remove unnecessary parts from SCHFOR list.

Functional description:

Delete all unnecessary lines from a list produced by the Super Search Function of ISPF. The procedure SSS calls this macro as initial macro. If you have performed a manual search, you can also use this macro to revise the list output. There is no other help available.

15.4.11 Edit macro #SSSCH - Mass update of members

This macro offers a very smart functionality:

When you have performed a super search and you make some changes in the edited list, you can write back the displayed lines completely to its members.



Find answers on the fly, or master something new. Subscribe today. See pricing options.

Purpose:

This macro can be used, when a super search list is displayed in edit session, to restore the displayed lines back into the members in which the searched lines were found. There is no other help available.

**Remarks:**

You can remove rows and members from the data set currently being edited. If you delete a member, all lines belonging to this member must also be deleted bulky. New rows cannot be inserted because the row numbers which are included in the displayed list, contain the relative position of these lines in the member. These line numbers will be used when the lines are written back to the member. The existing lines with the same number will be overwritten. If you do not want to write back changes in individual lines, you can of course delete these lines before calling #SSSCH.

15.4.12 Edit macro #SU - Submit JCL without a JOB statement

Submit a JCL that does not contain a JOB statement. Parameters: MSGLEVEL in the form: 0,0 / 1,0 / 2,0 / 0,1 / 1,1 / 2,1. This MSGLEVEL is in the JOB statement set.

Function description:

The program assembles the job statement using user-defined variables from ISPF profile pool. Use the program SPROFVAR to store these variables into the ISPF profile pool.

#SU is usable at the following positions:

- As an edit macro.
- In front of a member in a DLIST/M display.

Special function:

EXCLUDED lines of the edited data set are NOT inserted in the submitted job stream.

**Attention:**

Due to using the TSO SUBMIT command, all lines of the edited data set are translated to UPPER CASE during SUBMIT. This is a regular effect of SUBMIT.

**Author's remark:**

During my years of consultant activity, I always had all my job statements adapted to varying conditions in the data centers. To have to not always do this, I developed the REXX program SPROFVAR. There, I have all the values that apply to a data center inserted as variables and stored them in the ISPF profile. When I installed the SMART ISPF utilities at a new customer, I adapted and executed only the member SPROFVAR. In all the jobs I had to run at the customer's site, I was able to leave the job statement and start jobs with #SU.

No further help available.

15.4.13 Edit macro #TSOB - Submit a TSO batch job

Edit macro to submit a batch job which executes the REXX procedure currently being edited within a **TSO batch** environment. A parameter text can be added to the macro call. This text is transferred to the executed REXX procedure.

Attention:

This macro does not save the source code of the data set currently being edited. Hence, the UNDO status of the edit session is **not changed** by this macro.

Remarks:

To avoid the SAVE command, the currently edited REXX procedure is stored to a work data set. The batch job will use this work data set to execute the REXX procedure. The description of the 'SUBMIT *' command in the manual **TSO/E Command Reference** contains the following note: The SUBMIT command converts all characters of the job stream to **uppercase** before the stream is submitted to the system.

**Note:**

To avoid setting all JCL lines in uppercase by command SUBMIT * and thus



Find answers on the fly, or master something new. Subscribe today. [See pricing options.](#)

member by using the command: "SUBMIT ""workfile"(JOB)". In this case, the lines of the member JOB will not be transferred to uppercase.

Help is available with #TSOB ?

15.4.14 Edit macro #VERASE – Erase ISPF profile variables

With this edit macro, you can delete an ISPF variable from the profile and the shared pool. At call, you can enter only one variable name.

15.4.15 Program SCURSOR – Calling a data set from an ISPF screen

With this program you can call a data set which **name is somewhere** in an ISPF screen. When calling, a small panel appears which allows you to select the type of file processing.

Description:

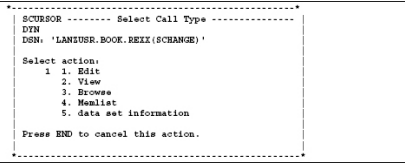
If a DSN of a data set appears on **any** ISPF screen, you can get access to this data set by positioning the cursor within the DSN and call SCURSOR. For a comfortable use of this function, I recommend to program a PF KEY with TSO %\$SCURSOR. If you have positioned the cursor at any position within a DSN and then pushed the PF key a panel appears to offer a range of access possibilities to the data set. There are some restrictions for displaying GDGs and VSAM data sets.

This function can be used in many displays. Some examples:

- In an EDIT panel
- In a BROWSE panel
- In the SDSF display
- In a data set display
- In a POPUP panel

See the selection panel below:

Screen 15.3: SCURSOR selection panel



15.4.16 Program SDOC – Produce documentation members

This program reads all members of a PDS and produces the **SDOC** member in the same PDS.

Function:

The program scans all members of the PDS for lines with keyword DOC: as the second word in a line. Only the first 200 lines in each member will be scanned. When during this scan no DOC: lines are found, then only the member name will be stored in the output. When the first DOC: line is found and there are more than five lines following the last found DOC: line without a DOC: the scan of this member ends.

15.4.17 Program SLE – Display last edited data sets

Purpose:

The program starts by input of SLE in the ISPF command line. It displays an ISPF table containing names of recently edited data sets. The program displays the ISPF table \$SLETAB. The EDIT initial macro #IMACRO2 updates this table with DSNs of recently edited data sets. The display panel offers the ability to perform some functions on the data sets displayed. See the help panels available in the programs screen. Smart Last Edit offers the following functions:

Screentext 15.1: Description of SLE functionality

1. Col. C --> D --> Display the **DELETE** panel (like 3.4) for this data set.
--> R --> Display **MENULIST** of a **PGM** and **NAME** selection columns.
--> E --> Edit this data set or member.
--> B --> Browse this data set or member.
--> Y --> View this data set or member.
--> D --> Display **EDIT** member list for this PDS.
--> R --> Remove this line from panel (multiple R are allowed).
--> I --> Display data set information.
2. The field **MAX lines shown** shows the current lines in the table \$SLETAB. By entering a lower number here the lines in the table are reduced to this Note: The program itself **NEVER** reduces the number of lines held in the table.
3. In the **COMMAND** line the following commands may be used:
a) f name --> A find for name in column DSN is started.
name must always be entered without apostrophe.
b) sort DSN --> The column DSN is temporarily sorted by name.
This sorted display is reset to standard at next display.
A subsequent change of selection set is caused by entering



Find answers on the fly, or master something new. Subscribe today. See pricing options.

The following screen shows the above-mentioned input fields:

Screen 15.4: SLE working panel

SLEP1 8 ----- Table display of recently edited data sets Row 1 of 73		
COMMAND ---->	Utilities	SCROLL ----> CUR
Display: FIELDS	Compile state of all	
Sort: D-Date or M-Name # Max lines shown: 73	BY	COMPILED
Direct control for edited data sets ----> All or Change ALL		
C Recent DSN in edit	Help with PF1 for all fields	Date

LANGUOR.BOGG.REXX (SLU)		15/01/23 08:55:55
LANGUOR.BOGG.REXX (FIMACRO)		15/01/23 08:48:25
LANGUOR.BOGG.REXX (FUSOCC)		15/01/23 08:46:46
LANGUOR.BOGG.REXX (FPAP)		15/01/23 08:43:03
LANGUOR.BOGG.REXX (F)		15/01/23 08:40:03
LANGUOR.BOGG.REXX (FALTTT)		15/01/23 08:36:50
LANGUOR.BOGG.REXX (FROX)		15/01/23 08:36:25
LANGUOR.BOGG.REXX (FHS)		15/01/23 08:30:30

i Remark:

Help screens are available for all greyed fields. Position the cursor in such a field and press PF1 to display the help. The general help screen appears, when the cursor is on any other position on the screen and PF1 is pressed.

i Tip:

Insert the following command in the ISPF command table to call the last edit function very fast when you type in SL in the ISPF command line.

_SLE 2 SELECT CMD(%SLE)

REXX SLE CALL THE LAST EDIT FUNCTION

15.4.18 SLOGDSN – Data member containing DSNs for logon

The member SLOGDSN contains DD names and the corresponding DSNs. The logon procedure SLOGON reads this member, looks for the already existing allocations by the same DDs and assembles a new structure in which the members referred to in SLOGDSN DSNs are respectively arranged in front of the existing DSNs of each affected DD. **The program SLOGON then allocates this new structure using the REUSE option.** This technique ensures that the DSNs specified by the user allocates in the member SLOGDSN BEFORE the previously existing DSNs in each affected DSN chain. DDs that are not found in the existing allocations will additionally allocated.

Program 15.1: Member SLOGDSN

```
* DOC: SLOGON Control member used by the procedure SLOGON of SMART
* DOC: Utilities
* © FRANK LANG 2015
*****
* Description:
*
* This member contains DD names and DSNs assigned to these DD names.
* The DD names and their assigned DSNs are used by the logon
* procedure SLOGON to allocate these elements in front of existing
* library allocations. If this member is missing there are no
* additional allocations possible.
* Lines beginning with an asterisk are skipped by the SLOGON program.
*****
* The definitions have the following structure:
*****
*
* CONAME#1 DSN# ONLY
*
*      DSN#
*
* CONAME#2 DSN#
*
*      DSN#
*
*      DSN#
*
*****
* Special cases:
* If the third word in a DD line is ONLY, then all existing
* allocations for this DD are discarded and only the here defined DD
* and its DSNs are allocated.
* If the HLQ is @USERID the logon procedure replaces this with the
* real user ID.
*****
*STUBJOB SYSTEM LOGON CLIST
*
*STUBJOB: AMNL15.BOGG.REXX
*          AMNL15.REXX
*          SYSTEM.REXX
*          AMNL15.BOGG.PANEL#
*          AMNL15.PANEL#
*          SYSTEM.PANEL#
*          AMNL15.MISS#
*          AMNL15.BOGG.MISS#
*          SYSTEM.MISS#
*          AMNL15.LOAD
*          SYSTEM.LOAD
*          AMNL15.PANEL#
```

15.4.19 Program SLOGON – Personal logon procedure

This is a personal logon procedure. It works as follows:

1. It reads all existing allocations of the TSO user and caches them.
2. It reads the control member SLOGDSN containing the DSNs that must allocated before the already existing DSNs. The new DSN chains are assembled in such a way that the DSNs read from the member SLOGDSN stay before the existing DSNs.
3. When in the member SLOGDSN new DDs and their DSNs are contained that are not present in the previous allocations, these chains will be added to the allocation stream.
4. SLOGON now performs the new allocations using the new allocation stream and the TSO ALLOC option REUSE. The REUSE option replaces the old allocations.



For installing this logon procedure, see [section 15.6.2](#) Installation on page 286.

Find answers on the fly, or master something new. Subscribe today. [See pricing options.](#)

If SLOGDSN contains DSNs of not existing data sets, a warning message is displayed during SLOGON, but the logon process continues anyway. This ensures that the logon process is successfully performed and a working user environment is created. **This is a very important advantage of the procedure SLOGON.**

15.4.20 Program SPROFEDT – Store users ISPF profile variables

Execute program SPROFVAR to store users ISPF profile environment variables into ISPF Profile. This program is called by ISPF command SPROFVAR.

15.4.21 SPROFVAR – Load user ISPF variables

SPROFVAR defines and stores variables needed by the SMART ISPF utilities in the ISPF profile pool. Several SMART utility programs use these profile variables. The PF keys are also set. Each user can add his own profile variables as needed!

15.4.22 Program SSC – Super clone for data sets Purpose:

This program uses a model data set to perform some functions on this data set or to create a new data set based on the model data set.

Smart Super Clone offers the following functions:

- Allocate a new data set with the same or different attributes as the model data set has with optionally copying the data.
- Restructure the model data set. Some attributes may be changed.
- Copy the model data set into another data set. The target data set must exist. During this operation, the data set attributes cannot be changed.
- The source data set can be a PDS and the target data set can be sequential. In this case, the program continuously writes the members into the sequential data set.
- Only the following data set types can be processed with this program:
Generally data sets on DASD and Sequential, PDS, PDSE and VSAM.

i Note:

A sequential data set cannot be copied into a PDS member.

Screen 15.5: SSC working screen

```
SSCP01 ----- SSC: data set Manager under ISPF / LANG ENGB -----
COMMAND ===>
Display = TEL-VIO  Sysplex = TRUPLX  Sysid = TESTSYS  Mode = 2019
Model = LANGUSR.B00E.B00E  Compile state of SSC
New = LANGUSR.B00E.B00E  NOT  COMPILED
PUNCH = NO  NO TO IS EA NO BR CO CR BE  (see Help with PF1)
SECTN = PS  Reference Date = 2015/01/23  Creation Date = 2011/02/03
DIR = 8  Alloc. Dir. Bks = N/A  Creator = IBMSSVS2
LIBCL = 80  Allocated TrkCyl = 24016  Number of Mems = 15
BLKSIZE = 6140  Blocks / Track = N/A  Used Dir. Bks = N/A
PDSN = 16  UMSA = 1.44 MB  Space Used = 104
SEC = 10  DSOB = P-R  Unit Type = 3190
UNIT = CYLINDER  Ext. all/used = 1/0  Compressed = NO
DSNTYPE = LIBRARY  Trks for Dir = N/A  Tracks used for directory
VOLSER = DMT002
SPORCLAS = TUGRAME  DATACLAS =  MEMTLAS = TUGRAME
SSM = TEL  TEL = use SSM, NO = do not use SSM
Exp. data = *****Data or TTY/MS/DO blank means *****
Model =
Model =
Reason =
Reason = LANGUSR
----- Yellow fields are input fields. -----
```

i Remark:

Help screens are available for all greyed fields. Position the cursor in such a field and press PF1 to display the help. The general help screen appears when the cursor is on any other position on the screen when PF1 is pressed.

15.4.23 Program SSS – Perform a Super-Search

Purpose:

Perform a Super-Search operation by entering SSS in front of a DSN in a DSLIST panel.

Two types of calling SSS (Smart Super Search) are available:

- Enter SSS <search string> in a line in front of a DSN on a DSLIST panel. The search operation then performs with only **one** string. Enter the search string without quotes and it can only consist of **one** word.
- Enter SSS in front of a DSN on a DSLIST panel. In this case, the panel SSSP1 appears. This panel allows the input of many search terms.

Screen 15.6: SSS input panel



```
ISPF ----- Edit commands for Super-Search execution -----
COMMAND ==>
DDB = 'LANDER.BOCF.CREX'          If the data set is a PDS
                                  all members are scanned!
1. = 'VANDAT',N
2. =
3. =
4. =
5. =
6. =
7. =
8. =
9. =
10. =
PO = MISS
Fill in data base process options. Press PF1 for help and then select a
choice by number or press ENTER to scroll thru pages down continuously.
MSG=
Press PF1 for Help, press ENTER to execute the search.
```

- Remarks:
- The program SSS calls the ISPF standard super search utility **ISRSUPC** to perform the search.
 - The standard ISPF help facility for super search appears when PF1 is pressed on the SSS screen.
 - SSS saves the entered search parameters in lines 1 to 10 automatically in the ISPF profile and redisplayes them at the next call of SSS.
 - The edit macro **#SSS** removes unnecessary output contents from the standard output list of the search before the list appears on the screen. The macro **#SSS** is indicated as initial macro when the editor is invoked for displaying the search results.

i SSS special update function:

SSS stores the optimized search results into a temporary data set and then displays this data set in the editor. This data set can be edited by using the normal editor functions. If individual lines of the data set are changed, these changes can **restored** into the individual members using the edit macro **#SSSCH**. See description of **#SSSCH** on page 264. **This is one of the most important advantages of the program SSS.**

15.5 PROGRAMMING AIDS

This chapter includes program elements which are intended for programmers who develop programs using the REXX language. The following table shows these edit macros, programs and subroutines, which are contained in the programming aid package.

Table 15.2: Programming aids Program Function

Program	Function
#C	This edit macro compiles the REXX program currently being edited and executes the compiled program immediately after the compile. One parameter can be added in the #c call statement. This parameter string will be passed to the compiled program to be used at execution time. To get an online tutorial description enter: #c ?
#RO	Online compile of the REXX program currently being edited. This procedure can be called as an edit macro or in front of a member name in a DSLIST/M display panel. Params: When #RO is used as a macro additional compiler options may be passed as parameters.
#RC	Use this procedure to assemble and submit a batch job to compile a REXX procedure.
#RCLOAD	Assembly and SUBMIT a job to compile a REXX procedure and generate a LOAD module or a CALL module.
#IE	Edit macro to insert in an application program the call lines to the internal ISPF_ERROR subroutine. The inserted statements calls the ISPF error handling subroutine



Find answers on the fly, or master something new. Subscribe today. See pricing options.

ISPFERR	ISPF error handling routine This subroutine is called when a call to an ISPF service function ends with an error.
#PAN	The screen whose program code is currently being edited will be displayed immediately.
DAYDIFF	Calculates the number of days between two DATE values. DATE values are entered in the form dd.mm.yyyy.
ENDDAY	Calculates the target date for a period of days starting at a given date plus or minus a number of days.
JULDATE	This subroutine calculates for a standard date the JULDATE or vise versa.
LEAPYEAR	Calculates whether the entered year is a leap year or not. This function returns a 1 if the entered year is a leap year, otherwise it returns 0. Caution: No plausibility checking of the entered year parameter is performed.
SCHANGE	This REXX subroutine performs changes in any strings. See section 4.6.3 SCHANGE – Change of texts on page 86.
SDYNPAN	This program converts the source code of an ISPF panel so that the code can be embedded in a REXX procedure to be dynamically loaded from there into a temporary ISPLLIB for execution.

15.5.1 Edit macro #C – Compile and execute a REXX program

If the REXX compiler is available in your installation of z/OS systems, you can use this macro to compile your REXX programs very easily.

Program description:

Edit macro #c:

Edit macro to compile and execute the REXX procedure currently being edited. After the characters #c you can enter a string, which will be passed as parameter to the program for execution.

Examples:

```
#c      --> 0 Parameter
#c xx 3 4 --> 1 Parameter
```

Attention:

The entered parameter string MUST NOT contain the ISPF delimiter character. This character is used for ISPF command stacking and your parameter will possibly be truncated at the delimiter character position. The ISPF delimiter character can be found in ISPF menu 0 in the line Command delimiter.

Processing flow:

The REXX program currently being edited is written to a work data set. This data set is used for input to the REXXCOMP program. If compilation is successful, the CEXEC module is written to the online work data set with DDNAME ##DD. If the ##DD data set is not allocated, it will be allocated. If the compilation ends with a RC > 0, the compiler listing is immediately browsed. The compiled program will be executed using the ISPF SELECT CMD service.

Remark:

Because the REXX program currently being edited is not stored to its data set for the purpose of compilation, the UNDO command can still be used.



Find answers on the fly, or master something new. Subscribe today. See pricing options.

15.5.2 Edit macro #RO – Online compile of a REXX program

This procedure can be called as an edit macro or in front of a member name in a DSLIST/M display panel. If called as an edit macro the following rules exist.

1. The source code currently being edited is stored into a temporary data set and the compiler uses this data set as SYSIN file. When using this technique, the editors save command will not be performed and the change status of the edited file remains unchanged. Hence, the UNDO command can still be used in the editor.
2. After #RO the user can add some additional parameters which will be transferred to REXX compiler as compiler options.
3. If the REXX source code contains TRACE statements, the compiler produces no XREF listing and the compilers output list will be written to a data set.

If called in front of a member name in a DSLIST/M display:

1. The compiler reads the REXX source from this member.
2. No additional compiler options may be set. This is because at this point parameters cannot be added to the procedure call statement.
3. If the REXX source code contains TRACE statements, NO CEEXEC output is produced and the compile ends with an error.

The name of the output data set for the CEEXEC code is set up as follows: The last qualifier of the input DSN is substituted by CEEXEC

Example: Input DSN → LANZ.LOGON.REXX
Output DSN → LANZ.LOGON.CEEXEC

15.5.3 Edit macro #RC – Compile a REXX procedure with a batch job

Functional description:

Use this procedure to assemble and submit a batch job to compile a REXX procedure. This procedure can be called as follows:

1. As an edit macro. In this case will the currently being edited REXX procedure be saved and the batch job to compile the REXX procedure will be submitted.
2. In a DSLIST/M in front of a REXX procedure member.
3. In a DSLIST/M display in the command line to compile all or a selected number of members, e.g.: **sei ab* #rc** --> all members which names are beginning with characters **ab** are compiled. At this type of call no NOTIFY=userid() parameter is inserted into the JOB statement.

The name of the output data set for the CEEXEC code is set up as follows: The last qualifier of the input DSN is substituted by CEEXEC

Example: Input DSN → LANZ.LOGON.REXX
Output DSN → LANZ.LOGON.CEEXEC

15.5.4 Edit macro #RCLOAD – Produce a load or a call module

Assembly and SUBMIT a job to compile a REXX procedure and generate a load module or a CALL module. This procedure can only be called as edit macro.

Parns: none

Attention:

There are two things necessary in order to use this macro:

1. The REXX compiler REXXCOMP must be available in your system. You can check this using the DDLIST utility.
2. The REXXCL.jcl procedure must be available in your system. Ask your system programming department where it is.

Two different types of load modules can be generated:

1. A load module that can be executed in a batch job using the EXEC PGM=statement. This type is called EXEC type. In this case the following comment line must be found in the source code: /* STUB: MVS */
2. A load module which was added to a Function Package Lib needs to be called with a CALL command. (CALL type). In this case the following comment line must be found in the source code: /* STUB: EFPL */
After generating the load module of this call type, it has to be implemented in a Function Package. See the corresponding manual **TSO/E REXX Reference**. If you try to execute such a program directly, then the CALL ends with a return code 0C4. This means the program ends with a hard crash.

Link library:



Find answers on the fly, or master something new. Subscribe today. [See pricing options.](#)

```
/* LNK: DSN of the load library */
```

Example:

```
/* LNK: PEVX.LANZLOAD */
```

If no such row is found, then NO compile is performed.

15.5.5 Edit macro #IE – Insert a call to ISPF_ERROR

Preface:

I have been looking for a method a long time to report errors when executing ISPF services elegantly and detailed. I think I have now found this method. This method consists of two parts which must be inserted into each program:

First part of error handling:

After each call to an ISPF service function, the return code RC will be checked. If this code signals an error, the error handling subroutine ISPF_ERROR will be called. The parameters in this call statement contain all information to report the error in detail.

After each call to an ISPF service, the following five lines of code are inserted:

```
Line 1: /* Start error handling */
Line 2: if rc > 0 then call ispf_error rc "/*||",
Line 3: /*LINIT DATAID(ID) DATASET('dataset') ENQ(SHR)/*||",
Line 4: /*LINIT DATAID(ID) DATASET('dataset') ENQ(SHR)*/
Line 5: /* End error handling */
```

The five lines must be manually inserted in the source code of a program as follows:

- 1. Enter #IE in the ISPF command line or program a PF key with #IE.
- 2. Move the cursor to the line containing the call to an ISPF service.
- 3. Press enter or the appropriate PF key which contains the command #IE. The above mentioned five lines will be inserted immediately behind the ISPF command line.

Description of the lines:

Lines 1 and 5 contain comments and are self-explanatory.

Line	Explanation
2	The RC returned by the ISPF service is passed as first parameter. Depending on the RC that was returned by the ISPF service, a call will be executed to the error handling subroutine ISPF_ERROR.
3	This line is passed as the second parameter to the subroutine ISPF_ERROR. It contains the original ISPF command line without the variables contained in the command line being resolved. This text is used by the subroutine ISPF_ERROR to find the original command line in the source code of the program by using the REXX command SOURCELINE().
4	This line is passed to ISPF_ERROR as third parameter. It contains the original ISPF command line with the variables contained in the command line being resolved.

These three parameters are used by the subroutine ISPF_ERROR to print a comprehensive error report.

Second part of error handling:

Insert the source code of the subroutine ISPF_ERROR manually into each program that contains calls to ISPF_ERROR. This is necessary because an externally called subroutine does not know the source code of the calling program. Therefore, the command SOURCELINE is not applicable to determine the line number of the line in the source code at which the error occurs.



In large programs, it can happen that a certain ISPF command line at several points occurs with the same content. In such a case, it is important to know the number of the program line in which the error occurred.



When using this method, errors in executing ISPF services are displayed as follows:

```
.....
ISPF execution error in source line 284, RC = 20 ISPF statement:
  INCHIT DATASET(10) DATASET('faked dsn') MSG(000)
ISPF error message: A list of names was found where a list was not expected.
.....
```

I replaced in the above example the contents of the variable DATASET by the text **faked DSN** to manually enforce the error. The statement where the error occurs is in line number **284** in the source code of the program.

15.5.6 Subroutine ISPF_ERROR - Display ISPF error messages

I described above how the call to the subroutine ISPF_ERROR is performed. To complete the method, here is the source code of the subroutine ISPF_ERROR:

Program 15.2: ISPF_ERROR subroutine

```
/* DOC: ISPFERR REXX INTERNAL SUBROUTINE */
/* © FRANK LANG 2015 */
/*-----*/
ispf_error: procedure expose zerrin
/*-----*/
/* ISPF Error handling routine */
/* This subroutine is called up when a call to an ISPF function */
/* service ends with an error. */
/* To display the statement in error and program line where the */
/* statement in error resides, a tricky programming is inserted into */
/* this subroutine. This trick works as follows: */
/* 1. The call to this subroutine contains a parameter consisting of */
/* three parts: */
/* a) The RC from the failing ISPF function call. */
/* b) A text string which is used by the subroutine to localize */
/* the failing ISPF function call within the main program. */
/* c) A text string containing the original ISPF function call */
/* statement. */
/* The parameters are each separated by //. */
/* 2. The subroutine localizes the line of the CALL using the text */
/* string and the REXX SOURCELINE function. */
/* The line before the CALL line is the failing ISPF function */
/* call. */
/* 3. An error message is assembled und displayed when ISPF displays */
/* the next screen. */
/* */
/* Caution: */
/* This program is only usable as an internal subroutine because an */
/* external subroutine cannot find a command line within the calling */
/* program. */
/* If you want to use this subroutine in your program, copy this */
/* code into your program. */
/*-----*/
parse arg rc//org//new
do i = 1 to sourceline ( ) /* Search the failing program line */
  if pos(org,sourceline(i)) > 0 then do
    zerrin = "ISPF execution error in source line "i", RC = "rc,
    "ISPF statement:" copies(" ",60) new copies(" ",60)
    if zerrin <> "ERRR" then do
      zerrin = zerrin copies(" ",60) "ISPF error message:" zerrin
    end
    "STMSG MSG(1582001)"
  end
end i
end i
exit
```

15.5.7 Edit macro #PAN - Execute a panel source code

When developing ISPF panels it is very tedious to test this using the ISPF menu 7.2. Therefore, I wrote the macro #PAN to test a panel which source code is currently being edited. Please read the comment lines at the beginning of the source code. It will describe how to use the macro.

Program 15.3: Edit macro #PAN to test macros directly

```
/* DOC: #PAN REXX MACRO */
/* DOC: This edit macro tests a panel currently being edited directly */
/* © FRANK LANG 2015 */
/*-----*/
/* PARM: none */
/* */
/* Program flow: */
/* To perform the display of the panel the ISPF function 7.2 is used. */
/* This means the program which is used in ISPF menu 7.2 of name */
/* ISPFERR is called by ISPF SELECT service. */
/* Necessary variables to impact the program are set and written to */
/* ISPF profile pool. */
/* */
/* To optimize the using of this function you should set a PF-Key */
/* with this command: #PAN(11, ( )) is the ISPF delimiter character. */
/* If you are currently being editing a ISPF panel and you would like */
/* to test the panel then you press only this PF-Key and the panel */
/* will be immediately displayed. */
/* If the panel definition contains a WINDOW parameter in the /BODY */
/* statement the variable ISPTPOP is set to YES to force the panel */
/* display in a window. */
/*-----*/
address ISPTPOP
"MACRO"
"DATA"
"MENU" = MENUSER
"DSN" = DATASET
ISPTPOP = "" /* panel name in profile pool */
ISPTPOP = "NO" /* set display option for POPUP panel off */
"END BODY FIRST" /* If the panel is a POPUP panel */
"END WINDOW( .scr .scr)" /* set display option to force display */
if rc = 0 then ISPTPOP = "YES" /* to POPUP panel
address ISPTPOP
"CONTROL REPOS REPOS"
"VSUT (ISPTPOP ISPTPOP) PROFILE" /* Write to Profile Pool */
SAYVUT = "DATA"
"VSUT (SAYVUT) PROFILE"
"VSUT (SAYVUT) SHARED"
"SELECT PGM(ISPTPOP) PARM('sappid') NOCHECK"
"ENDP ISPTPOP"
"VSUT (ISPTPOP) SHARED"
address ISPTPOP "REPOS"
exit 1
```

15.5.8 Subroutine DAYDIFF - Calculate number of days

Please see the source code to recognize the function of this subroutine.

Program 15.4: Subroutine DAYDIFF

```
/* DOC: DAYDIFF REXX USSO ENGLISH VERSION */
/* DOC: Calculates number of days between two DATE values. */
/* DOC: DATE values are entered in the form dd.mm.yyyy. */
/* © FRANK LANG 2015 */
/*-----*/
DAYDIFF: procedure
  set DATE DATE
  DAY1 = DATE("B",right(DAY1,4) || substr(DAY1,4,2) || left(DAY1,2),"B")
  DAY2 = DATE("B",right(DAY2,4) || substr(DAY2,4,2) || left(DAY2,2),"B")
  return abs(DAY2-DAY1)
```

Example:



The following call to DAYDIFF returns 365:

Find answers on the fly, or master something new. Subscribe today. See pricing options.

Please see the source code to recognize the function of this subroutine.

Program 15.5: Subroutine ENDDAY

```
/* DOC: ENDDAY REXX MAIN ENGLISH VERSION */
/* DOC: calculates the target date for a period of days starting at */
/* DOC: a given date plus or minus a number of days */
/* DOC: The start date must be entered in the form dd.mm.yyyy. */
/* DOC: Enter the number of days with a negative sign when an earlier */
/* DOC: date shall be calculated */
/* DOC: For positive values is no plus sign necessary. */
/* DOC: If * (asterisk) is entered as first argument the current */
/* DOC: date is used as first argument. */
/* DOC: Examples: */
/* DOC: Input = '16.12.2009 90' --> Output = 16 Mar 2010 */
/* DOC: Input = '20.01.2009 -50' --> Output = 1 Dec 2008 */
/* DOC: Input = '*' -50 --> Output = 21 Dec 2014 */
/* © FRANK LANG 2015 */
/*****
parse arg dat1 days
if dat1 = '*' | days = '' then do
say; say "Number of parameters is wrong"; say
call wrongarg
return
end
if dat1 <> '' then do
if datumok(dat1) <> "OK" then do
say; say "1st parameter has a wrong format"; say
call wrongarg
return
end
end
if datatype(days) <> "NUM" then do
say; say "2nd parameter is not numeric"; say
call wrongarg
return
end
if dat1 = '' then dat1 = date("B")
else
DAT1 = date("B",right(DAT1,4) || substr(DAT1,4,2) || left(DAT1,2),".B")
eday = dat1 + days
DAT1 = date("B",eday,"B") /* Output form: tt Mon yyyy */
return(dat1)
wrongarg
say "Wrong arguments entered: ">dat1 days"<
say "Examples:"
say "16.12.2009 90 --> calculate 16.12.2009 plus 90 days"
say "20.01.2009 -50 --> calculate 16.12.2009 minus 50 days"
say "*" -50 --> calculate current date minus 50 days"
return
*****/
```

15.5.10 Subroutine JULDATE – Translate a date to Julian and vise versa

Please see the source code to recognize the function of this subroutine.

Program 15.6: Subroutine JULDATE

```
/* DOC: JULDATE REXX SUBROUTINE */
/* DOC: Julian date calculations. */
/* DOC: Function: */
/* DOC: This subroutine calculates to a specified standard date the */
/* DOC: JULDATE or vise versa. */
/* © FRANK LANG 2015 */
/*****
/* Input: JULDATE = 'YYYY/MM' output: DATE = YYYY/MM/TT */
/* Input: DATE = 'YYYY/MM/TT' output: JULDATE = YYYY/TTT */
/* If the input date is incorrect then 999 is returned. */
/* Caution: */
/* When the input is a literal, it must be enclosed in quotes. */
/* This is because REXX interprets the slash (/) as a division */
/* operator. */
/*****
arg indat1
indate = strip(indat1)
select
when substr(indate,5,1) = '/' & /* Normal date entered */
length(indate) = 10 then do
parse var indate yyyy/"mm"/"tt"
if datatype(yyyy,"M") < datatype(mm,"M") &
datatype(tt,"M") = 0 then return("999")
if mm > 12 | tt > 31 | (yyyy * mm * tt = 0) then return("999")
ddd = date("B",date("B",yyyy||mm||tt,"B"),"B") + 1
retv = yyyy/18262.5/ddd
end
when substr(indate,5,1) = '/' & length(indate) = 8 then do /* Juldate */
parse var indate yyyy/"ddd"
if datatype(yyyy,"M") <
datatype(ddd,"M") = 0 then return("999")
leap = (yyyy // 4 = 0) - (yyyy // 100 = 0) + (yyyy // 400 = 0)
if ddd > (1826leap) | (yyyy * ddd = 0) then return("999")
retv = date("B",substr(yyyy,3,2)||ddd,"B")
retv = substr(retv,1,4)/substr(retv,5,2)/substr(retv,7,2)
end
otherwise return("999")
end /* select */
return retv
*****/
```

15.5.11 Subroutine Return – LEAPYEAR the leap year information

Please see the source code to recognize the function of this subroutine.

Program 15.7: REXX function LEAPYEAR

```
/* DOC: LEAPYEAR REXX FUNCTION */
/* DOC: Calculates whether the entered year is a leap year or not. */
/* DOC: This function returns 1 if the entered year is a leap year; */
/* DOC: Returns 0 if not. */
/* DOC: Caution: No plausibility checking of the year characters will */
/* DOC: be performed. */
/* © FRANK LANG 2015 */
/*****
ARG year
RETURN ((year // 4 = 0) - (year // 100 = 0) + (year // 400 = 0))
*****/
```

15.5.12 REXX subroutine SCHANG – REXX change function

Please see the source code to recognize the function of this subroutine.

Program 15.8: REXX subroutine SCHANG

```
/* DOC: SCHANG REXX SUBR */
/* DOC: This REXX subroutine performs changes in any strings. */
/* © FRANK LANG 2015 */
/*****
/* Call: result = change(input,target,new,only) */
/* result: Resulting string. */
/* input : Input string. */
/* target: Search string. */
/* new : String which replaces the TARGET string. */
/* only : The change will only take place if the text to be */
/* searched also contains this text. */
/* Remarks: */
/* The replacing text may be longer or shorter than the replaced */
/* text. */
/* If no replacement text is entered, the search text will be */
/* removed from the scanned string. */
/* All found search strings are replaced in the searched text by the */
/* replacement text. */
/*****
schange:
parse arg input,target,new,only
if length(only) > 0 & pos(only,input) = 0 then return(input)
la = length(target)
la = length(new)
ip = 1
do i = 1
if pos(target,input,ip)
if la = 0 then return(input)
if la < 0 then input = delete(input,1,la)
*****/
```




Find answers on the fly, or master something new. Subscribe today. See pricing options.

15.5.13 Program SDYNPAN - Convert a panel source code

To save space, I will refrain from the expressions of the entire source code. The comment lines at the beginning of the program include a detailed description of the function. The source code of SDYNPAN is very long. Therefore, only the functional description of the program is inserted below.

```
Program 15.9: Program SDYNPAN


/* DOC: SDYNPAN REXX MAIN */
/* DOC: This program converts an ISPF panel such that its contents */
/* DOC: can be embedded in a REXX procedure to be dynamically */
/* DOC: loaded from there into a temporary ISPLIB for the execution. */
/* * PRANS LANS 2015 */
/*-----*/
/* Call this program by entering its name in front of a panel name. */
/* in a DELIST display of a panels data set. */
/*-----*/
/* This technique, panels dynamically during the run of a REXX */
/* program to load, has the advantage that you do not need a */
/* separate ISPLIB version of the ISPF panels, where the panels are */
/* stored. Therefore, a simultaneous distribution of a panel data */
/* set is not necessary. The converted panel definitions are stored */
/* in a sequential data set. The Lsq of this data set is the panel */
/* name. This procedure uses the panel SHOWNAM. */
/*-----*/
```

 Remark:

I used this program to convert all panels of the SMART ISPF utilities to load them dynamically when a program is started, which contains panels.

15.6 INSTALLATION OF SMART ISPF UTILITIES

To use the smart utilities, you must install them in your TSO/ISPF system. Before you start transferring programs and panels, you need to make an important decision:

 Note:

The REXX procedures of the SMART ISPF utilities contain all required panel definitions in their source code. Therefore, you do not have to transfer the panels to your ISPLIB. The installation program SPROFVAR that stores the ISPF environment variables in the ISPF PROFILE POOL, contains the variable **dynpan**. If you set dynpan to YES before you execute SPROFVAR, then the REXX procedures uses imbedded panels when executed.

15.6.1 Download and unzip

Perform the following steps to install the SMART ISPF utilities:

1. Download the SMART ISPF utilities ZIP file from the DeGruyter website www.degruyter.com (<http://www.degruyter.com>).
2. Unzip the file to a folder in your PC. After unzipping, the following data sets are in the folder:

REXX	The REXX program package.
PANELS	The ISPF panel package.
SHOSTUNP	REXX program to unpack REXX programs and ISPF PANELS into their libraries on z/OS host.

The two files REXX and PANELS contain all procedures and all panels of the SMART ISPF utilities in packed form. After transferring into the appropriate libraries on the z/OS/ system, they must be unpacked using the REXX program SHOSTUNP. This REXX program is the **only** program that is usable to unpack the data sets because the packed data sets have a proprietary format.

Upload and unpack the members to z/OS host:

1. Use the 3270 emulation software you have on your PC to upload the three members to the z/OS host into a PDS or PDS-E with RECFM=FB and LRECL=80.
2. Then use the program SHOSTUNP to extract the packed members into the appropriate libraries. Use the **Command Shell (menu 6)** of ISPF to execute the program SHOSTUNP. The following screen shows these commands:

Screen 15.7: Examples of unpacking SMART ISPF utilities members

```
Menu List Mode Functions Utilities Help
-----
Enter TSO or Workstation commands below:
ISPF Command Shell
****
Place cursor on choice and press enter to retrieve command
```



Find answers on the fly, or master something new. Subscribe today. See pricing options.

**Note:**

The last three lines in the above panel show the commands automatically executed under TSO while the upload was running.

15.6.2 Installation

For the installation of the SMART ISPF utilities, multiple scenarios concerning your TSO/ISPF user environment are possible:

1. You still have an own ISPF library environment with an own logon procedure. This is the ideal case. You can copy the REXX procedures of the SMART ISPF utilities directly into your SYSEXEC or SYSPROC library and unpack them there. If you want to use a panel library for the panels, you can copy the panels to the ISPLIB chain. Additionally, you can decide whether you want to use the SLOGON procedure of the SMART ISPF utilities together with the member SLOGDSN to make your logon process more variable and convenient.
2. You get the **TSO READY prompt** when you leave ISPF.
In this case, you can call the logon procedure SLOGON at the READY PROMPT using the command **EX 'dsn(SLOGON)'**. First, you must only adjust the member SLOGDSN so that there the correct data set names are included.
3. Your TSO user is completely logged off when you leave ISPF and you have no chance to start an own logon procedure.
In this case, you can still use the SMART ISPF utilities. However, installation of the call environment is somewhat more complex. You can use the command **ALTLIB**. While there is a restriction that makes the process somewhat inconvenient: If you have performed an **ALTLIB** command, the calling environment thus created is always only available in the logical screen in which the **ALTLIB** command was executed. See the description of the **ALTLIB** command below.

15.6.3 ALTLIB command

Excerpt from the IBM manual **TSO-E Reference** concerning **ALTLIB**:

Use the **ALTLIB** command to:

- Define alternative application-level libraries of REXX execs or CLISTS.
- Indicate that user-, application-, and system-level libraries of REXX execs and CLISTS will be searched.
- Exclude one or more library levels (user, application, system) from being searched.
- Reset the search order to the system level.
- Obtain a display of the search order that is in effect.

Using ALTLIB in ISPF:

When using **ALTLIB** when ISPF is active, you can define the libraries (user, application, and system) that are active for each **application**. Libraries that you define while running an application are in effect only while that application has control. When the application completes, the previous libraries (user, application, and system) will automatically be reactivated.

If you are in split-screen mode in ISPF and you issue the **ALTLIB** command from a one-screen session, the changes affect only that screen session. The **ALTLIB** search order is not valid across split screens.

To execute the **ALTLIB** command, enter the following command in an ISPF command line:

```
TSO ALTLIB ACT APPLICATION(EXEC) DSNNAME('dsn')
```

dsn is the data set where your REXX procedures reside. If you have installed the REXX procedures of SMART ISPF utilities into this dsn, the utilities are also executable.

**Recommendation:**

Program a PF key with this **ALTLIB** call. When you open a new logical screen in ISPF, then you can execute the command immediately by pressing the PF key.

Example:

The following screen shows the assignment of keys PF18 and PF19 for execution of the **ALTLIB** command.

Screen 15.8: PF key setting to execute **ALTLIB**



Find answers on the fly, or master something new. Subscribe today. [See pricing options.](#)

```
Command ----> PF Key Definitions and labels - Alternate Keys

Note: Definitions and labels below apply only to terminals with 24 PF keys.

PF13 . . . HELP
PF14 . . . SNAP LIST
PF15 . . . END
PF16 . . . RETURN
PF17 . . . TWO ALTSEQ
PF18 . . . two altlib set application(exec) ddname('userid.rexx')
PF19 . . . two altlib display
PF20 . . . DOWN
PF21 . . . SUPPL
PF22 . . . BEGINMENU
PF23 . . . RIGHT
PF24 . . . TWO ASCCURSOR

PF13 label . . . PF14 label . . . PF15 label . . .
PF16 label . . . PF17 label . . . PF18 label . . .
PF19 label . . . PF20 label . . . PF21 label . . .
PF22 label . . . PF23 label . . . PF24 label . . .

Press ENTER key to display primary keys. Enter END command to exit.
```

The command ALTLIB DISPLAY shows the current ALTLIB chain. After pressing PF18 and then PF19, the following display appears:

```
Current search order (by DDNAME) is:
Application-level EXEC DDNAME=SYS00278
System-level EXEC DDNAME=SYSEXEC
System-level CLIST DDNAME=SYSPROC
***
```

I used DDLIST to find the DDNAME SYS00278 and found it in the following screen:

```
Command ----> Current Data Set Allocations Row 111 of 148
Scroll ----> CSR

Volume Disposition Act DDname Data set Name Actions: R E V W P C I Q
DMTCAT SHR,KEEP > STSTCPD CENTER.PARMLIB (DATA00)
NEW,DEL > SYSTEM ----- Allocated to the terminal -----
DMTU08 SHR,KEEP > SYS00278 USERID.SHAFT1.REXX
----- End of Allocation List -----
```


Remarks concerning the ALTLIB command:

- ALTLIB commands can occur in a stacked manner.
- Only the procedures of the first level in a chain of stacked ALTLIB chains are reachable.
- Use the command `altlib deact application(exec)` to remove the top level of the chain.

Example:

I pressed PF18 four times and then I pressed PF19. The following chain appears:

```
Current search order (by DDNAME) is:
Application-level EXEC DDNAME=SYS00284
Stacked DDNAME=SYS00283
Stacked DDNAME=SYS00282
Stacked DDNAME=SYS00281
System-level EXEC DDNAME=SYSEXEC
System-level CLIST DDNAME=SYSPROC
***
```

 Note:

The command ALTLIB DISPLAY shows in opposition to the command ISPLIBD only the DD names of the allocated data sets. If you want to see the corresponding DSNs, you can view them with the utility DDLIST as shown below.

```
NEW,DEL > SYSTEM ----- Allocated to the terminal -----
DMTU08 SHR,KEEP > SYS00281 USERID.SHAFT1.REXX
DMTU08 SHR,KEEP > SYS00283 USERID.SHAFT1.REXX
DMTU08 SHR,KEEP > SYS00284 USERID.SHAFT1.REXX
----- End of Allocation List -----
```

15.6.4 Make the SMART ISPF utilities ready to run

- Before you can use the SMART ISPF utilities, there are still some steps necessary:
1. If you can fully use an own logon procedure, you must adapt the members SLOGON and SLOGDSN and perhaps copy the procedure SLOGON into an appropriate library. See section 15.6.2 Installation on page 286.
 2. Edit the member SPROFVAR, perform the necessary changes and execute it using the edit macro ##. This program is executable without having to load the ISPF profile variables for the SMART ISPF utilities.

The following profile variables are important to have appropriate values:



```
account = "UNIVER"      /* Account number for job statement */
jobclass = "A"          /* Job Run class */
msgclass = "H"          /* Job message class */
dynpan = "YES"          /* Dynamic panel store indicator */
dynunit = "VIO"         /* Unit to allow dynamic panels */
                        /* data set. SYSDA or VIO */
sortunit = "VIO"        /* SORT Workfile Unit */
maclib = 2              /* Number of Maclibs */
sysproc = 0             /* Number of SYSPROCS */
sysexec = 2             /* Number of SYSEXECs */
steplib = 1             /* Number of STEPLibs */
isplib = 0              /* Number of Skel-Libs */
db2sys = "*"            /* Standard DB2 Server */
cexec = "NO"            /* EXEC compile YES/NO */
storclas = "USERBASE"   /* SMS:Standard Storage Class */
mgmtclas = "USERMONT"    /* SMS:Standard Management Class */
dataclas = " "          /* SMS:Standard Data Class */
maclib1 = "SYS1.MACLIB" /* Assembler Maclib No 1 */
maclib2 = "SYS1.MODGEN" /* Assembler Maclib No 2 */
sortlib = "SYS1.LINKLIB" /* SORT Linklib */
userlink = USERID()".LOAD" /* User Linklib */
rexxload = userid()".CEXEC" /* CEXEC library */
sysexec1 = userid()".REXX" /* Execs */
sysexec2 = "ALLUSER.REX" /* Execs */
steplib1 = "ALLUSER.LOAD" /* Steplibs */
AMT = "CSE"
```



Conclusion:

After having performed all mentioned steps above, the SMART ISPF utilities should work fine and I wish you much success with it.

[Recommended](#) / [Playlists](#) / [History](#) / [Topics](#) / [Settings](#) / [Get the App](#) / [Sign Out](#)



PREV

14 Edit macros – Create and apply

NEXT



List of programs



Find answers on the fly, or master something new. Subscribe today. [See pricing options.](#)