Dynamic ISPF

How to Eliminate All Those Datasets in the LOGON PROC
Part 1

NewEra Software - The zExchange
July 13, 2015

Thomas Conley
Pinnacle Consulting Group, Inc.
59 Applewood Drive
Rochester, NY 14612-3501
P: (585)720-0012
F: (585)723-3713
pinncons@rochester.rr.com

Abstract

Do you have a large number of LOGON procs at your site? Do you ever get tired of adding ISPF application datasets to every LOGON proc? Have you ever wondered if there's a better way to install and maintain your ISPF applications? If you answered 'YES' to any of these questions, this session is for you! Come to this popular session to learn how to invoke ISPF applications dynamically. The knowledge gained will help you to shrink your logon procs to a manageable size, eliminate redundant procs, and get better overall ISPF performance to boot! This session will include an online demonstration of Dynamic ISPF.

Agenda

- "Standard" ISPF vs. Dynamic ISPF
- Coding Rexx Driver EXECs
- ISPF Menu Construction
- Performance Considerations
- LOADLIBs, Dynamic Steplib, TSOLIB
- Solving Common Problems
- Summary
- Coming Attractions

The "Standard" ISPF Environment

- "Standard" ISPF is LOGON PROC with every ISPF dialog dataset allocated to it
- Can also be EXEC or CLIST dynamically allocating ISPF dialog datasets at LOGON
- Many LOGON PROCs or LOGON CLISTs customized with specific dialog datasets
- Selectively allows access to specific ISPF applications
- Many customized menus and submenus required to invoke ISPF applications

“Standard” ISPF - LOGON PROC

```
//TSOTGC2  EXEC PGM=IKJEFT01,
//         DYNAMNBR=50,PARM='%LOGON'
//SYSEXEC  DD DISP=SHR,DSN=ISP.SISPEXEC
//         DD DISP=SHR,DSN=SYS1.SEDGEXE1
//SYSPROC  DD DISP=SHR,DSN=SYS2.ISPCLIB
//         DD DISP=SHR,DSN=ISP.SISPCLIB
//         DD DISP=SHR,DSN=GIM.SGIMCLS0
//         DD DISP=SHR,DSN=SYS1.SEDGEXE1
//         DD DISP=SHR,DSN=SYS1.DGTCLIB
//         DD DISP=SHR,DSN=SYS1.SCBDCLST
//         DD DISP=SHR,DSN=SYS1.HRFCLST
//         DD DISP=SHR,DSN=SYS1.SBLSCLI0
//ISPMLIB  DD DISP=SHR,DSN=SYS2.ISPMLIB
//         DD DISP=SHR,DSN=ISP.SISPMENU
//         DD DISP=SHR,DSN=GIM.SGIMMENU
//         DD DISP=SHR,DSN=SYS1.SEDGMENU
//         DD DISP=SHR,DSN=SYS1.DGTMLIB
//         DD DISP=SHR,DSN=SYS1.DFQMLIB
//         DD DISP=SHR,DSN=SYS1.SCBDMENU
//         DD DISP=SHR,DSN=SYS1.HRFMSG
//         DD DISP=SHR,DSN=SYS1.SBLSMSG0
//ISPPLIB  DD DISP=SHR,DSN=SYS2.ISPPLIB
//         DD DISP=SHR,DSN=ISP.SISPPENU
//         DD DISP=SHR,DSN=GIM.SGIMPENU
//         DD DISP=SHR,DSN=SYS1.SEDGPENU
//         DD DISP=SHR,DSN=SYS1.DGTPLIB
//         DD DISP=SHR,DSN=SYS1.DFQPLIB
//         DD DISP=SHR,DSN=SYS1.SCBDPENU
//         DD DISP=SHR,DSN=SYS1.HRFPANL
//         DD DISP=SHR,DSN=SYS1.SBLSPNL0
. . .
```

“Standard” ISPF - LOGON PROC (cont'd)

```
//ISPSLIB  DD DISP=SHR,DSN=ISP.SISPSENU
//         DD DISP=SHR,DSN=ISP.SISPSLIB
//         DD DISP=SHR,DSN=ISP.SISPSLIB
//         DD DISP=SHR,DSN=GIM.SGIMSENU
//         DD DISP=SHR,DSN=SYS1.SEDGSKL1
//         DD DISP=SHR,DSN=SYS1.DGTSLIB
//         DD DISP=SHR,DSN=SYS1.HRFSKEL
//         DD DISP=SHR,DSN=SYS1.SBLSKEL0
//ISPTLIB  DD DISP=SHR,DSN=SYS2.ISPTLIB
//         DD DISP=SHR,DSN=ISP.SISPTENU
//         DD DISP=SHR,DSN=GIM.SGIMTENU
//         DD DISP=SHR,DSN=SYS1.SMP.SMPTABL
//         DD DISP=SHR,DSN=SYS1.SEDGTBL1
//         DD DISP=SHR,DSN=SYS1.DGTTLIB
//         DD DISP=SHR,DSN=SYS1.SCBDTENU
//         DD DISP=SHR,DSN=SYS1.SBLSTBL0
//CIDTABL  DD DISP=SHR,DSN=SYS1.SMP.SMPTABL
//SMPTABL  DD DISP=SHR,DSN=SYS1.SMP.SMPTABL
//SYSHELP  DD DISP=SHR,DSN=ISP.SISPHELP
//         DD DISP=SHR,DSN=SYS1.HELP
//         DD DISP=SHR,DSN=SYS1.SEDGHLP1
//         DD DISP=SHR,DSN=TCPIP.SEZAHELP
//SYSUADS  DD DISP=SHR,DSN=SYS1.UADS
//SYSLBC   DD DISP=SHR,DSN=SYS1.BRODCAST
//SYSPRINT DD TERM=TS,SYSOUT=Z
//SYSTERM  DD TERM=TS,SYSOUT=Z
//SYSIN    DD TERM=TS
```

“Standard” ISPF - LOGON CLIST

```
PROC 0
CONTROL MAIN NOMSG
SET PROFDSN = &STR(&SYSUID..ISPF.ISPPROF)
ALLOC FI(ISPPROF) DA('&PROFDSN') OLD
IF &LASTCC NE 0 THEN +
   DO
      ALLOC DA('&PROFDSN') NEW CATALOG SPACE(1 1) CYLINDERS DIR(45) +
         DSORG(PO) RECFM(F B) LRECL(80) BLKSIZE(0) FI(ISPPROF)
      IF &LASTCC NE 0 THEN +
         WRITE UNABLE TO ALLOCATE ISPF PROFILE DATASET &PROFDSN
      ELSE +
         WRITE ISPF PROFILE DATA SET &PROFDSN CREATED
   END
/**********************************************************************/
/* REALLOCATE USER DATASETS.                                          */
/**********************************************************************/
IF &SUBSTR(1:3,&SYSUID) = &STR(USR) THEN +
   DO
      FREE FI(SYSPROC)
      ALLOC FI(SYSPROC) DA('SYS1.ISPCLIB','ISP.SISPEXEC') SHR
      FREE FI(ISPMLIB)
      ALLOC FI(ISPMLIB) DA('SYS1.ISPMLIB','ISP.SISPMENU') SHR
      FREE FI(ISPPLIB)
      ALLOC FI(ISPPLIB) DA('SYS1.ISPPLIB','ISP.SISPPENU') SHR
      FREE FI(ISPSLIB)
      ALLOC FI(ISPSLIB) DA('SYS1.ISPSLIB','ISP.SISPSENU') SHR
      FREE FI(ISPTLIB)
      ALLOC FI(ISPTLIB) DA('SYS1.ISPTLIB','ISP.SISPTENU') SHR
   END
ISPF
```

“Standard” ISPF - Advantages

- “Standard” ISPF install method used by nearly all ISPF applications
- No extra coding required since dialog is invoked as specified in install doc
- Requires no additional knowledge or training in ISPF

“Standard” ISPF - Disadvantages

- Every LOGON PROC with an ISPF application must be tested when adding, changing, or deleting that ISPF application
- Users must LOGOFF and LOGON to access new or changed ISPF applications
- Datasets unavailable for maintenance due to ENQ’s for many different users, even when not using specific ISPF application
- Users must LOGOFF to free datasets

“Standard” ISPF - Disadvantages

- Significant BLDL overhead with large concatenations (BLDL is MVS macro that locates a member within library or concatenation)
- Working set size increase due to large number of datasets allocated to LOGON PROC negatively impacts performance
- Slower LOGON time compared to Dynamic ISPF

“Standard” ISPF - Disadvantages

- For disaster recovery, “Standard” ISPF requires restores for all datasets in LOGON PROC before anyone can logon to TSO
- Multiple failure points, since each LOGON PROC is potential point of failure

Dynamic ISPF

- Dynamic ISPF uses minimal LOGON PROC with only ISPF product and site-specific datasets allocated in LOGON PROC
- ISPF applications invoked by Rexx driver EXECs from site-wide SYSPROC library allocated to every TSO LOGON PROC
- ISPF application datasets only allocated when ISPF application invoked by Rexx driver EXEC

Dynamic ISPF - LOGON PROC

```
//TSOTGC1  EXEC PGM=IKJEFT01,
//         DYNAMNBR=99,PARM='%LOGON'
//ISPSLIB  DD DSN=ISP.SISPSENU,DISP=SHR
//         DD DSN=ISP.SISPSLIB,DISP=SHR
//ISPMLIB  DD DSN=SYS2.ISPMLIB,DISP=SHR
//         DD DSN=ISP.SISPMENU,DISP=SHR
//ISPPLIB  DD DSN=SYS2.ISPPLIB,DISP=SHR
//         DD DSN=ISP.SISPPENU,DISP=SHR
//ISPTLIB  DD DSN=SYS2.ISPTLIB,DISP=SHR
//         DD DSN=ISP.SISPTENU,DISP=SHR
//SYSEXEC  DD DSN=ISP.SISPEXEC,DISP=SHR
//* SYS2.ISPCLIB IS THE REXX DRIVER LIBRARY
//SYSPROC  DD DSN=SYS2.ISPCLIB,DISP=SHR
//         DD DSN=ISP.SISPCLIB,DISP=SHR
//SYSUADS  DD DSN=SYS1.UADS,DISP=SHR
//SYSLBC   DD DSN=SYS1.BRODCAST,DISP=SHR
//SYSPRINT DD TERM=TS,SYSOUT=Z
//SYSTERM  DD TERM=TS,SYSOUT=Z
//SYSIN    DD TERM=TS
```

Dynamic ISPF - Advantages

- No changes or testing of LOGON PROCs required
- Quick prototyping and immediate availability of ISPF applications after coding Rexx driver EXEC
- Datasets only allocated when user actively using ISPF application, so datasets usually available for maintenance
- If not, users just leave specific ISPF application instead of logging off

Dynamic ISPF - Advantages

- BLDL overhead reduced since ISPF application datasets "first" in concatenation
- Typically only one library to search with fewer members than "Standard" TSO concatenation
- Faster TSO logon time than "Standard" ISPF, especially when "Standard" ISPF uses LOGON CLIST/EXEC to deallocate and reallocate each ISPF dataset

Dynamic ISPF - Advantages

- Smaller TSO working set size compared to "Standard" ISPF, since fewer datasets are allocated to the TSO LOGON PROC
- For disaster recovery, Dynamic ISPF ensures TSO availability as soon as ISPF product and site-specific datasets are recovered
- No need to wait for every program product to be restored

Dynamic ISPF - Advantages

- Rexx driver EXEC is single point of failure
- Dynamic ISPF easier to maintain and more reliable than “Standard” ISPF

Dynamic ISPF - Disadvantages

- Effort required to code and test Rexx driver EXECs for each Dynamic ISPF application
- ISPF systems programmer will need additional knowledge in ISPF and Rexx
- Invocation for Dynamic ISPF application is slightly longer, due to dynamic allocation of ISPF datasets

Coding Rexx Driver EXECs

- Dynamic ISPF uses Rexx driver EXECs to setup environment for each ISPF application
- Rexx driver EXECs invoked from site-wide EXEC library allocated to SYSPROC in every TSO LOGON PROC

Coding Rexx Driver EXECs

- Basic logic for any Rexx driver EXEC:
  - ALTLIB CLIST or Rexx EXEC libraries
  - LIBDEF ISPF libraries
  - ALLOC files not handled by ALTLIB or LIBDEF (be sure to include “REUSE” parameter on ALLOC statement, just in case previous invocation failed and left DD allocated)
  - Invoke ISPF application
  - Free ALTLIB, LIBDEF, and ALLOC datasets

Rexx Driver EXECs - The Tools

- ALTLIB is TSO service modifying standard TSO search order for CLISTs and Rexx EXECs

```
ALTLIB ACT APPL(CLIST) DA('SYS1.DGTCLIB')
```

- LIBDEF is ISPF service modifying normal ISPF search order for file tailoring output, images, load libraries, messages, panels, skeletons, and tables

```
LIBDEF ISPPLIB DATASET ID('SYS1.DGTPLIB') STACK
```

Rexx Driver EXECs - Examples

```
/* Rexx */
/*trace i*/
/*********************************************************************/
/* This exec invokes IBM's ServerPac Installation Dialog.  Change    */
/* 'order' to your ServerPac order number.                           */
/*********************************************************************/
parse arg ztrail
address tso "ALTLIB ACT APPL(CLIST) DA('CPAC.order.SCPPCENU')"
address ispexec "LIBDEF ISPLLIB DATASET ID('CPAC.order.SCPPLOAD') STACK"
address ispexec "LIBDEF ISPMLIB DATASET ID('CPAC.order.SCPPMENU') STACK"
address ispexec "LIBDEF ISPPLIB DATASET ID('CPAC.order.SCPPPENU') STACK"
address ispexec "LIBDEF ISPSLIB DATASET ID('CPAC.order.SCPPSENU') STACK"
address ispexec "LIBDEF ISPTLIB DATASET ID('CPAC.order.SCPPTENU') STACK"
address ispexec "SELECT CMD(%CPPCISPF CPAC) NEWAPPL(CPP) PASSLIB",
                "SCRNAME(SERVPAC)"
address ispexec "LIBDEF ISPLLIB"
address ispexec "LIBDEF ISPMLIB"
address ispexec "LIBDEF ISPPLIB"
address ispexec "LIBDEF ISPSLIB"
address ispexec "LIBDEF ISPTLIB"
address tso "ALTLIB DEACT APPL(CLIST)"
```

Rexx Driver EXECs - Examples

```
/* Rexx */
/*********************************************************************/
/* This exec invokes IBM's RACF dialog.                              */
/*********************************************************************/
parse arg ztrail
address tso "ALTLIB ACT APPL(CLIST) DA('SYS1.HRFCLST')"
address ispexec "LIBDEF ISPPLIB DATASET ID('SYS1.HRFPANL') STACK"
address ispexec "LIBDEF ISPMLIB DATASET ID('SYS1.HRFMSG') STACK"
address ispexec "LIBDEF ISPSLIB DATASET ID('SYS1.HRFSKEL') STACK"
address ispexec "CONTROL ERRORS RETURN"
address ispexec "SELECT PANEL(ICHP00) NEWAPPL(RACF) OPT("ztrail")",
                "PASSLIB SCRNAME(RACF)"
address ispexec "LIBDEF ISPPLIB"
address ispexec "LIBDEF ISPMLIB"
address ispexec "LIBDEF ISPSLIB"
address tso "ALTLIB DEACT APPL(CLIST)"
```

ISPF Menu Construction

- ISPF menus use different construction in Dynamic ISPF environment
- Menu invokes Rexx driver EXECs instead of “Standard” ISPF invocations for dialogs
- Proper menu construction necessary to ensure Dynamic ISPF applications operate transparently, providing same function and usability as “Standard” ISPF interface

ISPF Menu Construction

- Nested menu options must be invoked correctly under Dynamic ISPF, just as they are in "Standard" ISPF
- If "S" invokes SDSF, then "S.DA" should invoke the SDSF Display Active panel, not "S;DA"
- Without transparent operation, user community won't accept Dynamic ISPF

“Standard” ISPF Menu Construction

- “Standard” ISPF menu uses this)PROC:

```
)PROC
&ZSEL = TRANS (TRUNC (&ZCMD,'.')
  0,'PGM(ISPISM) SCRNAME(SETTINGS)'
  1,'PGM(ISRBRO) PARM(ISRBRO01) SCRNAME(VIEW)'
  2,'PGM(ISREDIT) PARM(P,ISREDM01) SCRNAME(EDIT)'
  3,'PANEL(ISRUTIL) SCRNAME(UTIL)'
  4,'PANEL(ISRFPA) SCRNAME(FOREGRND)'
  5,'PGM(ISRJB1) PARM(ISRJPA) SCRNAME(BATCH) NOCHECK'
  6,'PGM(ISRPTC) SCRNAME(CMD)'
  7,'PGM(ISPYXDR) PARM(&ZTAPPLID) SCRNAME(DTEST) NOCHECK'
  9,'PANEL(ISRDIIS) ADDPOP'
 10,'PGM(ISRSCLM) SCRNAME(SCLM) NOCHECK'
 11,'PGM(ISRUDA) PARM(ISRWORK) SCRNAME(WORK)'
  X,EXIT
 SP,'PGM(ISPSAM) PARM(PNS)'
 ' ',' '
   *,'?' )
&ZTRAIL=.TRAIL
```

- “Standard” ISPF menu fails since &ZTRAIL set too late for &ZSEL substitution

Dynamic ISPF Menu Construction

- Dynamic ISPF menu uses this)PROC

```
)PROC
&ZQ = &Z
IF (&ZCMD ¬= ' ')
   &ZQ = TRUNC(&ZCMD,'.')
   &ZTRAIL = .TRAIL
   IF (&ZQ = ' ')
      .MSG = ISRU000
&ZSEL = TRANS ( &ZQ
  F,'CMD(%@FFST)'
  H,'CMD(%@HCD)'
  I,'CMD(%@ISMF &ZCMD) NOCHECK'
 IP,'CMD(%@IPCS &ZTRAIL) NOCHECK'
  S,'CMD(%@SMPE &ZCMD) NOCHECK'
  X,EXIT
 SP,'PGM(ISPSAM) PARM(PNS)'
 ' ',' '
   *,'?' )
```

- &ZTRAIL must be resolved before &ZSEL TRANS to pass lower-level menu options to Rexx driver EXEC

Dynamic ISPF Menu Construction

- OW45623 documented changes necessary to set &ZTRAIL prior to &ZSEL TRANS statement for releases prior to z/OS V1R2
- At z/OS V1R2+, new ISPF selection menus and DTL compiler changed to set &ZTRAIL for Dynamic ISPF
- ISPF automatically checks submenu options for SELECT unless NOCHECK used
- NOCHECK required when passing either &ZCMD or &ZTRAIL on SELECT command

Dynamic ISPF Menu Construction

- ISMF and SMP/E are exceptions that require entire &ZCMD string because they perform their own truncation processing
- If "I" invokes ISMF, and user enters "I.1" to display the ISMF Data Set option (&ZCMD is "I.1" and &ZTRAIL is "1"), pass &ZCMD "I.1" to ISMF; passing just &ZTRAIL "1" displays ISMF Primary Option Menu)

Performance Considerations

- LOGON time minimized by Dynamic ISPF, especially compared to LOGON CLIST or EXEC freeing/reallocating ISPF datasets
- Dynamic ISPF application may take longer to invoke, but it executes more quickly due to decreased BLDL overhead
- Smaller TSO working set size compared to "Standard" ISPF results in less real storage per user, as well as reduced paging

Performance Considerations

- Higher overhead for dynamic allocation more than offset by decreased BLDL overhead, as well as decreased paging due to smaller TSO working set

Performance Considerations - VLF

- Use VLF to improve performance of all CLISTs and EXECs system-wide
- In COFVLFxx parmlib member, use IKJEXEC VLF class to specify every CLIST and EXEC library in EDSN parm
- Only SYSPROC concatenation or ALTLIB APPL(CLIST) datasets eligible for VLF caching
- SYSEXEC not eligible for VLF caching

Performance Considerations - VLF

- Here’s an example of COFVLFxx PARMLIB member syntax for CLISTs and EXECs allocating 4 Meg for MAXVIRT:

```
CLASS NAME(IKJEXEC)
      EDSN(SYS1.SBLSCLI0)
      EDSN(SYS1.SEDGEXE1)
      EDSN(SYS1.SCBDCLST)
      EDSN(SYS1.DGTCLIB)
      EDSN(CAI.CAICLIB)
      EDSN(SYS2.ISPCLIB)
      EDSN(GIM.SGIMCLS0)
      EDSN(ISP.SISPCLIB)
      EDSN(SYS1.HRFCLST)
      EDSN(SYS1.SERBCLS)
      MAXVIRT(1024)
```

Performance Considerations - VLF

- Specify enough MAXVIRT to cache all CLISTs and EXECs; MAXVIRT is number of 4K blocks allocated to VLF class
- SMF Type 41, subtype 3 records hold statistics such as MAXVIRT specified, current storage, objects trimmed, etc.
- Determine optimal MAXVIRT for each VLF class from SMF records

Performance Considerations - VLF

- TSO VLFNOTE command must be used when updating member in VLF-managed library shared by multiple systems
- Issue TSO VLFNOTE command on each other sharing system, otherwise old member will continue to execute on other sharing systems:

```
TSO VLFNOTE UPDATE DATASET('SYS2.ISPCLIB(@RACF)')
```

LOADLIBs

- LOADLIBs can be very confusing - should they be in ISPLLIB, STEPLIB, LINKLIST?
- LIBDEF'd ISPLLIB typically only works for programs invoked with ISPEXEC SELECT
- Dialogs using CALL/LINK/ATTACH/XCTL interface don't work with ISPLLIB LIBDEF
- The ISPF Behind the Scenes session is the bible for LOADLIBs in ISPF:

ISPF Behind the Scenes

LOADLIBs

- LOADLIBs for ISPF dialog can be satisfied by following list from best to worst:
- Add LOADLIB to LINKLIST (least overhead, best performance, requires master catalog entry for LNKLSTxx/PROGxx unless using VOLUME)
- Use Dynamic STEPLIB product (good performance if used in driver EXEC, possible cost issue)

LOADLIBs

- Allocate LOADLIB to ISPLLIB from READY before invoking ISPF (reasonable performance, user input or LOGON PROC or LOGON EXEC changes required)
- Use TSOLIB before invoking ISPF (poor performance, not recommended, user input or LOGON EXEC changes required, but it is useful for testing purposes)

LOADLIBs

- STEPLIB LOADLIB in TSO LOGON PROC (poor performance, not recommended, significant TSO impact)
- See Cheryl Watson’s January 1991 Tuning Letter, available at (Thank you, Cheryl!):

http://www.watsonwalker.com/JAN91.pdf

Dynamic STEPLIB

- Dynamic STEPLIB products can create or modify STEPLIB concatenation for TSO PROC
- Using Dynamic STEPLIB in Rexx Driver EXECs saves LINKLIST extents by removing ISPF LOADLIBs from LINKLIST
- Dynamic STEPLIB is essential for QMF prior to QMF V6; QMF V6 added DSQLLIB to remove need for Dynamic STEPLIB

Dynamic STEPLIB

- Dynamic STEPLIB also useful for running two or more releases of ISPF application in parallel if application requires STEPLIB

Dynamic STEPLIB

- I do not endorse or recommend any of these products; they are listed for your information only:

Dan Dalby's Dynamic Steplib CBT file 452

Tone Software DYNA-STEP

PIE/TSO Dynamic STEPLIB for z/OS

TSOLIB

- TSOLIB modifies standard TSO search order for commands and programs
- TSOLIB only works from READY, so it can’t change loadlib search order in ISPF

```
TSOLIB ACT DA(‘SYS1.DGTLLIB’)
```

- TSOLIB not suitable for production, but it is useful for testing ISPF applications with LOADLIB dependencies without requiring LINKLIST or LOGON PROC changes

TSOLIB

- To test with TSOLIB, exit ISPF, enter TSOLIB at READY prompt, then re-enter ISPF to execute Dynamic ISPF driver EXEC
- TSOLIB can only be invoked from READY prompt or by CLIST executing from READY
- Rexx EXEC executing from READY must QUEUE or PUSH TSOLIB, which then runs AFTER Rexx EXEC

Solving Common Problems

- ***<u>ALWAYS use STACK with LIBDEF!!</u>***
- z/OS V1R7+ provides ISPF Config option to make LIBDEF STACK default - ***DO IT!!***
- After omitting STACK on LIBDEF, most common problem is omitting PASSLIB on SELECT statement when using NEWAPPL
- If vendors don't support LIBDEF, they don't code PASSLIB with NEWAPPL
- If passing &ZTRAIL or &ZCMD to Rexx Driver EXEC, don't forget NOCHECK

Solving Common Problems

- If passing menu items to program with ISPF SELECT PGM(…) isn’t working, use PARM operand to pass &ZCMD or &ZTRAIL
- If ISPF applications have problems with Dynamic ISPF, open PMR to fix problem
- If passing invalid initial option to non-primary option menu (&ZPRIM=NO):

```
address ispexec “SELECT PANEL(zprimno) OPT(invalid)”
```

you will get a nasty surprise…

Solving Common Problems

```
                          ISPF Dialog Error
Command ===>

******************************************************************************
* ISPD221                                                                    *
*                                                                            *
* Specified option invalid                                                   *
* 'PANEL(zprimno) OPT(invalid)' - contains invalid OPT selection.            *
*                                                                            *
*                                                                            *
*                                                                            *
*                                                                            *
*                                                                            *
*                                                                            *
* Current dialog statement:                                                  *
* SELECT PANEL(zprimno) OPT(invalid)                                         *
*                                                                            *
*  Enter HELP command for further information regarding this error.          *
*  Press ENTER key to terminate the dialog.                                  *
*                                                                            *
*                                                                            *
*                                                                            *
*                                                                            *
******************************************************************************
```

Solving Common Problems

- Following Rexx code fixes this problem by trapping error and issuing ISPF “Invalid option” message instead of dialog termination panel:

```
address ispexec "CONTROL ERRORS RETURN"
do forever
   address ispexec "SELECT PANEL(zprimno) OPT("ztrail")
        SCRNAME(scrname)"
   if rc <= 4 then
      leave
   else
      do
         address ispexec "SETMSG MSG("zerrmsg")"
         if zerrmsg <> 'ISPD221' then
            leave
      end
   ztrail = ''
end
```

Solving Common Problems

- Another common problem is ISPF dialog requiring keylists, but doesn't set ISPF keylist variable ZKLUSE to "Y"
- Dialogs requiring keylists often run into problems at sites that turn off keylists by default
- The HCD dialog set standard by requiring ZKLUSE, since HCD is not usable if KEYLIST is OFF

Solving Common Problems

- Some IBM components recognize need to set ZKLUSE to "Y" (WLM APAR OW53981, SDSF APARs PQ57763, PQ60751, IPCS APAR OW50398, TCP/IP APAR PQ74283) but others have not (MQSeries APARs PQ24700, PQ25929)
- Don't let IBM or OEM vendors force you to set KEYLIST ON by default
- Tell them to use correct ISPF dialog coding techniques instead!

Solving Common Problems

- Following code fixes ZKLUSE “Y” issue:

```
/* rexx */
/*trace i*/
/*********************************************************************/
/* This exec invokes IBM's GRS Monitor dialog.  This exec will call  */
/* @GRSMON1, a stub exec which fixes PQ60751 by turning on PFSHOW    */
/* and setting ZKLUSE to "Y" to enable keylists.                     */
/*********************************************************************/
parse arg ztrail
address tso "ALTLIB ACT APPL(CLIST) DATASET('IBMUSER.DYNISPF.EXEC',",
                                           "'SYS1.SBLSCLI0')"
address ispexec "LIBDEF ISPLLIB DATASET ID('SYS1.MIGLIB') STACK"
address ispexec "LIBDEF ISPMLIB DATASET ID('SYS1.SBLSMSG0') STACK"
address ispexec "LIBDEF ISPPLIB DATASET ID('SYS1.SBLSPNL0') STACK"
address ispexec "LIBDEF ISPTLIB DATASET ID('SYS1.SBLSTBL0') STACK"
address ispexec "SELECT CMD(%@GRSMON1 "ztrail") NEWAPPL(ISGA) PASSLIB",
                "NOCHECK SCRNAME(GRSMON)"
address ispexec "LIBDEF ISPLLIB"
address ispexec "LIBDEF ISPMLIB"
address ispexec "LIBDEF ISPPLIB"
address ispexec "LIBDEF ISPTLIB"
address tso "ALTLIB DEACT APPL(CLIST)"
```

Solving Common Problems

- Driver EXEC invokes this EXEC instead of ISGAMIDM, then this EXEC sets ZKLUSE, turns on PFSHOW, and invokes ISGAMIDM:

```
/* rexx */
/*trace i*/
/*********************************************************************/
/* This exec invokes IBM's GRS Monitor dialog.  It also fixes        */
/* PQ60751 by turning on PFSHOW and setting ZKLUSE to "Y" to enable  */
/* keylists.                                                         */
/*********************************************************************/
parse arg ztrail
address ispexec "VGET (ZPFSHOW)"
pfshowsv = zpfshow
address ispexec "SELECT PGM(ISPOPF) PARM(PFK,ON)"
zkluse = 'Y'
address ispexec "VPUT (ZKLUSE) PROFILE"
address ispexec "SELECT CMD(ISGAMIDM) PARM("ztrail") SCRNAME(GRSMON)"
address ispexec "SELECT PGM(ISPOPF) PARM(PFK,"pfshowsv")"
```

Summary

- Dynamic ISPF is methodology using Rexx EXECs to dynamically invoke ISPF applications
- Dynamic ISPF uses ALTLIB and LIBDEF to dynamically allocate ISPF application datasets, instead of allocating each ISPF dataset in the TSO LOGON PROC
- Properly implemented, Dynamic ISPF provides significant maintenance, storage, and performance benefits

Summary

- ISPF menu construction is key in Dynamic ISPF environment to ensure submenu options correctly passed through Rexx driver EXECs
- Dynamic ISPF performs better than "Standard" ISPF; overhead of dynamically allocating ISPF datasets balanced by lower BLDL and paging overhead

Summary

- Use of LOADLIBs in Dynamic ISPF vary with each application, depending on whether application uses ISPF SELECT or CALL/ATTACH, etc. to invoke its programs
- LOADLIB options are LINKLIST, Dynamic STEPLIB, TSOLIB, ISPLLIB, and STEPLIB
- Common problems can be solved by using specific techniques for menus or opening PMR's if necessary

Coming Attractions

- Part 2 of this presentation will cover:
  - Miscellaneous Recommendations
  - Sell Dynamic ISPF to Management
  - ServerPac - “The Dark Side”
  - Conversion to Dynamic ISPF
  - Dynamic ISPF Starter Set (DISS)
  - Setting Up TSO and ISPF (WAC)
  - Summary
  - Finally...