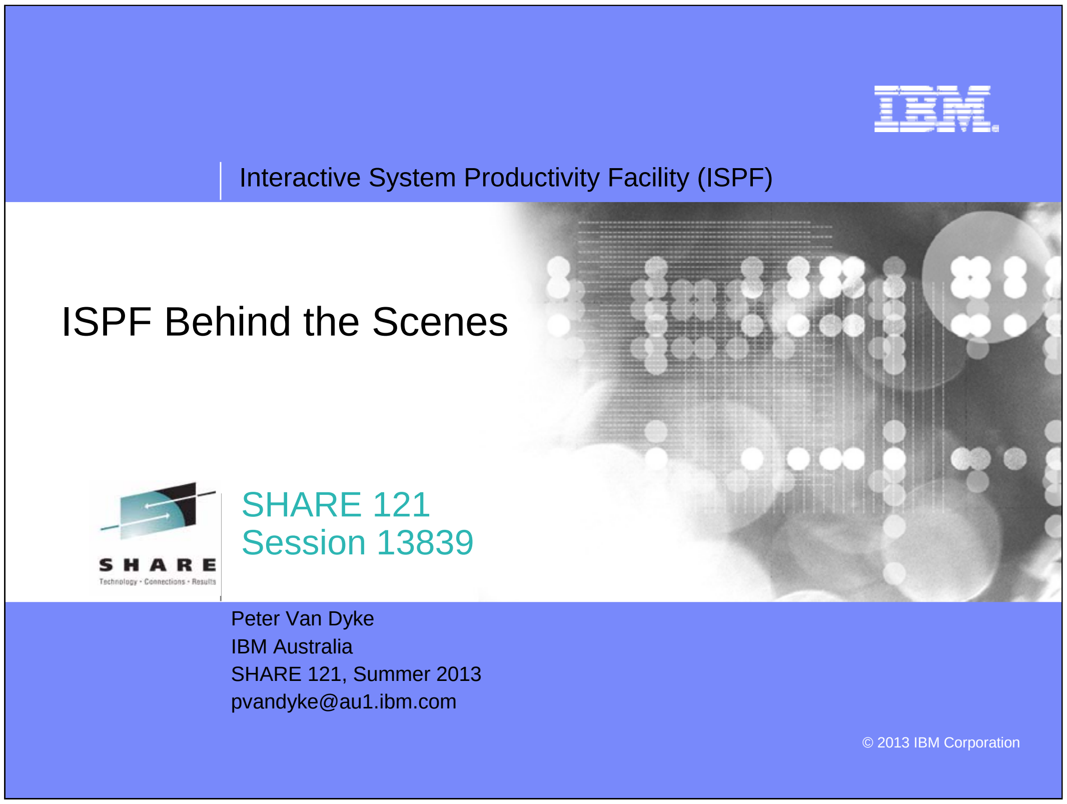
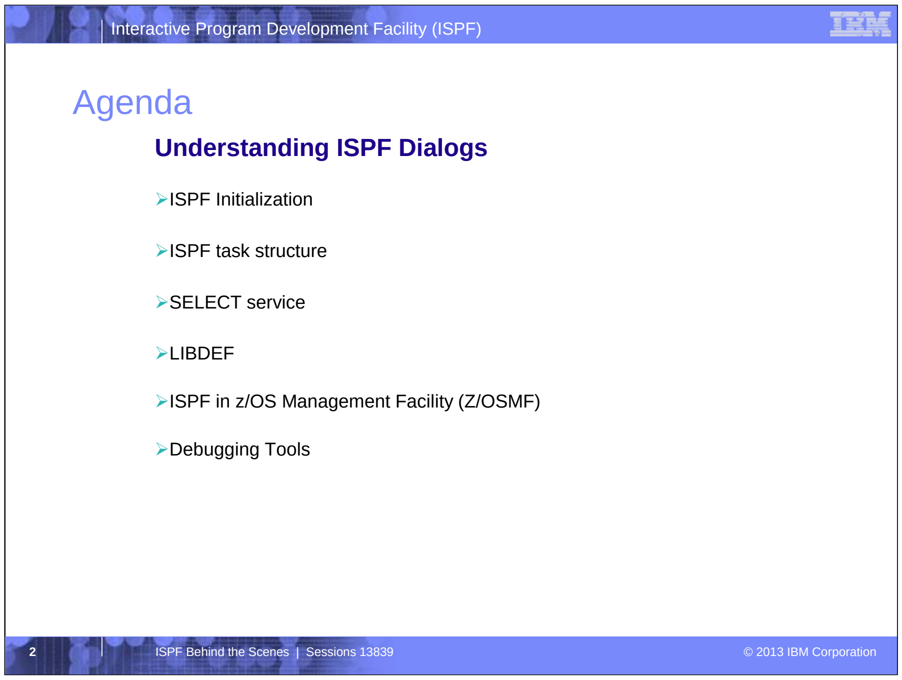


Interactive System Productivity Facility (ISPF)

ISPF Behind the Scenes

SHARE 121
Session 13839

Peter Van Dyke
IBM Australia
SHARE 121, Summer 2013
pvandyke@au1.ibm.com



Agenda

Understanding ISPF Dialogs

- ISPF Initialization
- ISPF task structure
- SELECT service
- LIBDEF
- ISPF in z/OS Management Facility (Z/OSMF)
- Debugging Tools

Our aim with this session is to provide you with an understanding of some of the internal functions and behaviours of ISPF.

We will cover:

•ISPF Initialization - how ISPF is invoked and some of the things that happen during the initialization process

•The ISPF task structure

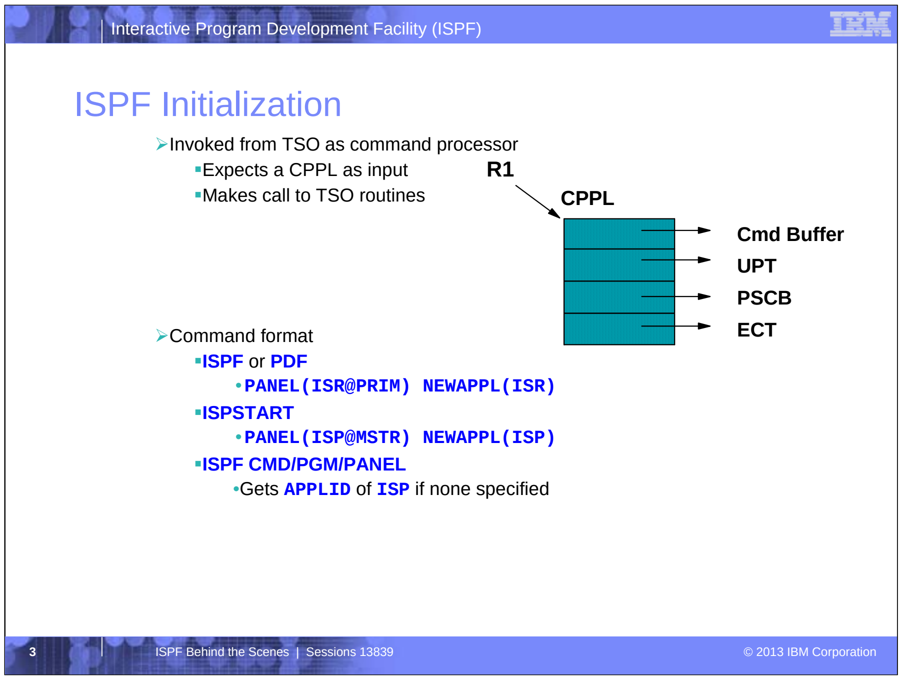
•The ISPF SELECT service and what effect this has on the the ISPF environment

•The LIBDEF service, in particular LIBDEF of ISPLLIB

•How ISPF runs in a web browser using z/OS Management Facility (Z/OSMF)

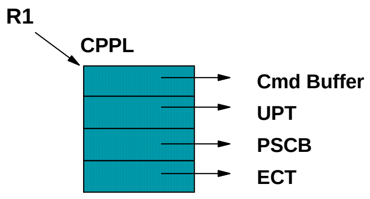
•and finally we will have a brief look at some of the debugging tools available to ISPF users to debug ISPF dialogs

I am happy to have some questions during the presentation or you can save them for the end.



ISPF Initialization

- Invoked from TSO as command processor
  - Expects a CPPL as input
  - Makes call to TSO routines



- Command format
  - **ISPF** or **PDF**
    - `PANEL(ISR@PRIM) NEWAPPL(ISR)`
  - **ISPSTART**
    - `PANEL(ISP@MSTR) NEWAPPL(ISP)`
  - **ISPF CMD/PGM/PANEL**
    - Gets `APPLID` of `ISP` if none specified

ISPF is invoked from the TSO environment

•Its runs as a command processor and therefore expects a Command Processor Parameter List (CPPL)

•The CPPL contains a list of 4 addresses that point to the:

•Command Buffer - The command and parameters as entered by the user

•User Profile Table (UPT - contains information stored in UADS, used by LOGON/LOGOFF, TMP, and command processors)

•Protected Step Control Block (PSCB - contains information from UADS, control bits, and accounting data for the userid)

•Environmental Control Table (ECT - this table provides the communication medium for the TMP, command processors and service routines. It contains the current command/subcommand name, return code, pointers to work areas and message chain, and processing control flags)

•It makes use of TSO routines:

•such as TGET/TPUT (for screen I/O),

•IKJCT441 (for TSO variables)

•allows any application program to examine and manipulate CLIST and REXX variables

•IKJEFTSR (for calling programs) to name a few

•allows a user to invoke functions such as commands, programs, CLISTs, or REXX execs from an application program.

ISPF can be started using either:

•the ISPF or PDF command

•by default this will invoke ISPF, displaying panel ISR@PRIM using application id = ISR

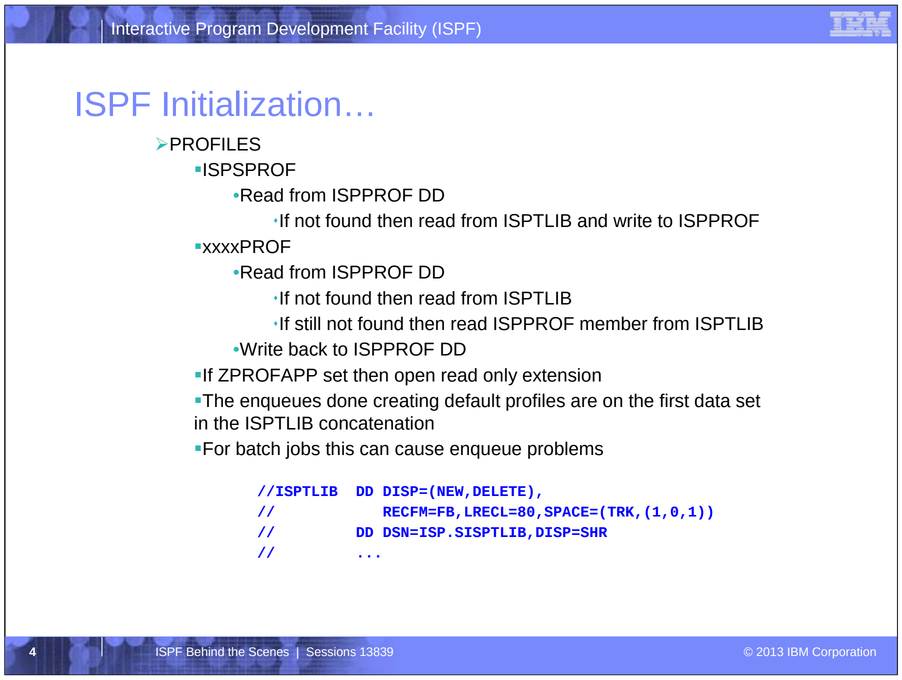
•the ISPSTART command

•by default this will invoke ISPF, displaying panel ISP@MSTR using application id = ISP

•panel ISP@MSTR can be changed using the ISPF Configuration utility (DEFAULT_PRIMARY_PANEL = default primary panel; default is ISP@MSTR)

Specify any of the CMD, PGM or PANEL parameter on the ISPF, PDF or ISPSTART commands will use the default APPLID of ISP, if the NEWAPPL keyword is not specified

Note: ISPF (And PDF) and ISPSTART command support similar command syntax



ISPF Initialization…

- PROFILES
  - ISPSPROF
    - Read from ISPPROF DD
      - If not found then read from ISPTLIB and write to ISPPROF
  - xxxxPROF
    - Read from ISPPROF DD
      - If not found then read from ISPTLIB
      - If still not found then read ISPPROF member from ISPTLIB
    - Write back to ISPPROF DD
  - If ZPROFAPP set then open read only extension
  - The enqueues done creating default profiles are on the first data set in the ISPTLIB concatenation
  - For batch jobs this can cause enqueue problems

```
//ISPTLIB  DD DISP=(NEW,DELETE),
//            RECFM=FB,LRECL=80,SPACE=(TRK,(1,0,1))
//         DD DSN=ISP.SISPTLIB,DISP=SHR
//         ...
```

•ISPF stores information for the user in profiles

•These profiles contain variables and their value settings

•**ISPSPROF** contains system variables owned by ISPF

•**ISPSPROF** is read from ISPPROF DD and if not found then read from ISPTLIB

  •any changes to ISPSPROF are always stored in the data set allocated to ISPPROF

  •private keylists are stored in ISPSPROF

  •This must be partitioned data set - non-concatenated
  (Private KEYLISTs are stored in ISPSPROF)

•Application Profiles are stored in members xxxxPROF, where xxxx is the application id or APPLID.

  •They are also read from ISPPROF and if not found then read from ISPTLIB.

    •Application developer can provide a profile pool in ISPTLIB

  •If still not found the profile is initializes from default profile pool member ISPPROF read from DD=ISPTLIB

  •any changes to an application profile are always stored back to DD=ISPPROF

•If ZPROFAPP is set then the read only extension as defined by ZPROFAPP is read

  •ZPROFAPP holds name of table containing extension variables

  •ISPF provides for a read-only extension of the application profile variable pool. Helps maintain better control over application default profile variables.

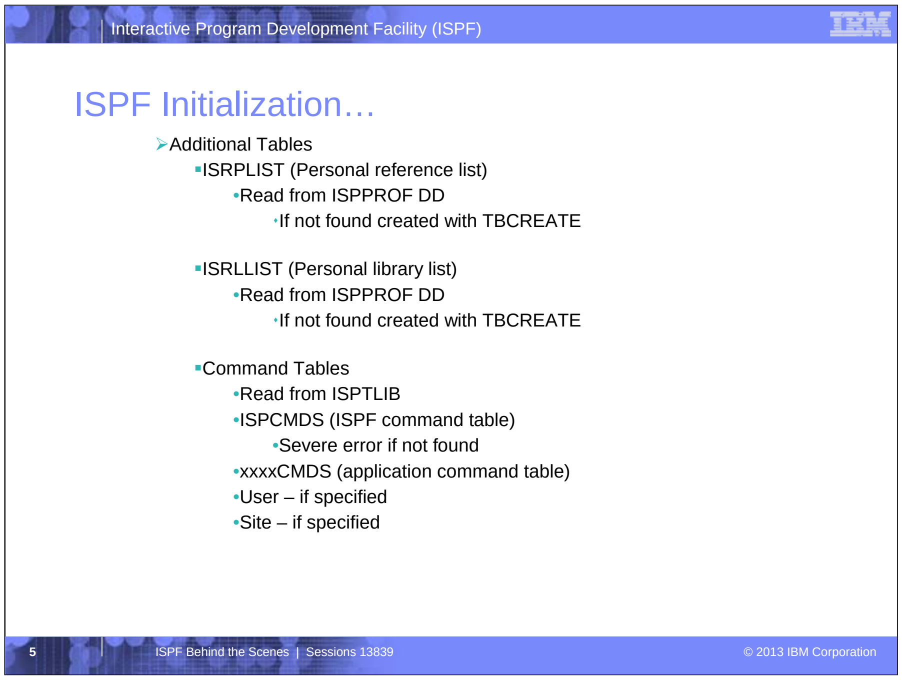
  •Saves space since these variables need not exist in the application profile of every application user.

<u>**Note**</u>: The equeues for creating default profiles are done on the first data set in the ISPTLIB concatenation

•ENQ SPFEDIT,rname,E,52,SYSTEMS

•This can cause enqueue problems, particularly in batch jobs where a number of ISPF system tables are opened

•usually resolved by ensuring the first data set of ISPTLIB is unique (not the supplied data set) - eg: a temporary data set



ISPF Initialization…

- Additional Tables
  - ISRPLIST (Personal reference list)
    - Read from ISPPROF DD
      - If not found created with TBCREATE
  - ISRLLIST (Personal library list)
    - Read from ISPPROF DD
      - If not found created with TBCREATE
  - Command Tables
    - Read from ISPTLIB
    - ISPCMDS (ISPF command table)
      - Severe error if not found
    - xxxxCMDS (application command table)
    - User – if specified
    - Site – if specified

•The Personal Reference List Table (ISRPLIST) and Personal Library List (ISRLLIST) tables are read from DD=ISPPROF and created if not found

•ie. they are NOT initialized from an ISPF distributed table

•ISPF Command Tables are also initialized

•Command tables allow a command to be entered directly on the command line of any panel and to be processed by ISPF before an application receives control

•Command Tables are always read from DD=ISPTLIB and they have member names that end with 'CMDS'

•The System Command Table is ISPCMDS and ISPF will terminate with a severe error if it can not locate the system command table

•Application Command Tables are searched first and correspond to the Application Id or APPLID of the current dialog.

•A user can also customize a User and Site command Tables which are included in the command table search (There is no default User or Site Command Table)

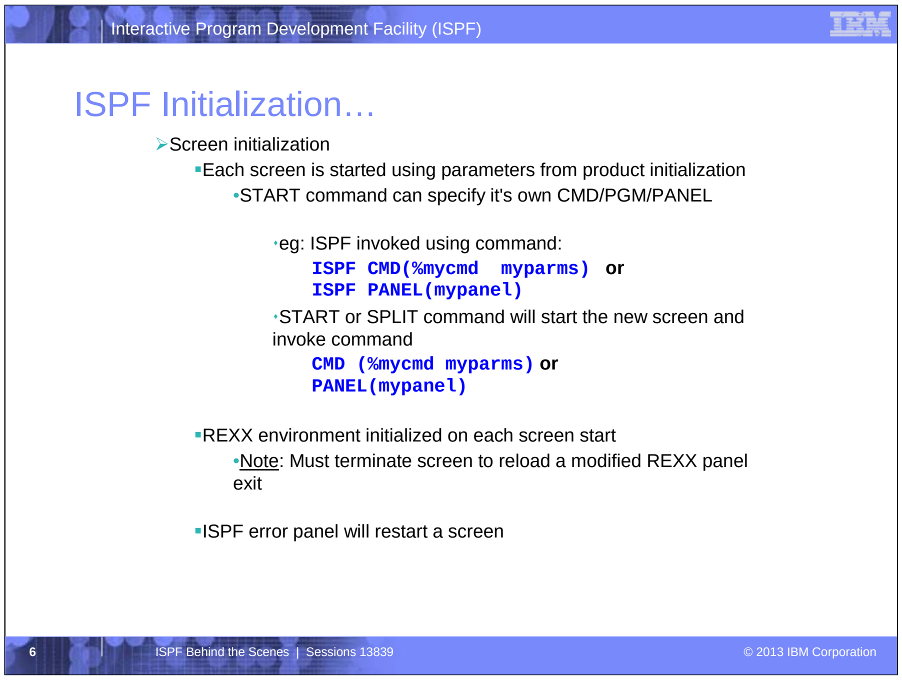
•Command Tables are searched in the order

•Application

•User

•Then either Site or System, depending a configuration option.

•The **default** is **Site** before **System**



ISPF Initialization…

- Screen initialization
  - Each screen is started using parameters from product initialization
    - START command can specify it's own CMD/PGM/PANEL
      - eg: ISPF invoked using command:
        ISPF CMD(%mycmd myparms) **or**
        ISPF PANEL(mypanel)
      - START or SPLIT command will start the new screen and invoke command
        CMD (%mycmd myparms) **or**
        PANEL(mypanel)
  - REXX environment initialized on each screen start
    - Note: Must terminate screen to reload a modified REXX panel exit
  - ISPF error panel will restart a screen

•When ISPF initializes a logical screen it uses the parameters specified when you started ISPF.

•For instance if you specify ISPF CMD(%mycmd myparms), this is invoked to start the inital screen display as well as when a SPLIT or START command is entered

•Use dialog variable ZSPLIT to determine where the command is being invoked

•Initial entry into ISPF or restarting first screen (ZSPLIT=NO)

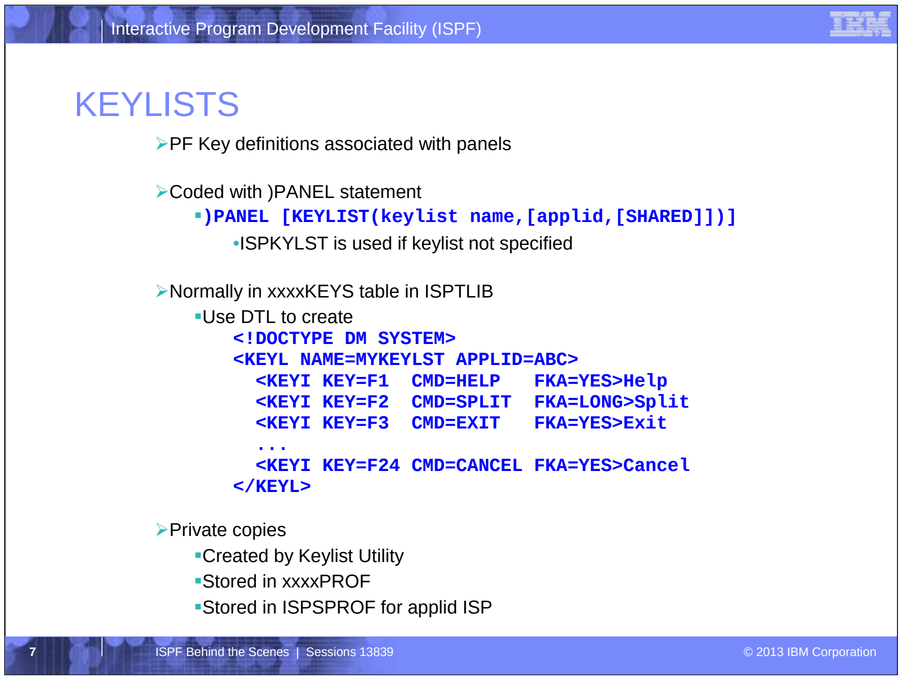
•Starting another screen (ZSPLIT=YES)

•The REXX environment is initialized as each screen is started

•Finally, an ISPF dialog error or abend which causes the ISPF error panel to be displayed will restart at the initial screen (ie. usually the primary options panel)

•So the initial command is re-run, and

•The REXX environment is re-initialized



KEYLISTS

- PF Key definitions associated with panels
- Coded with)PANEL statement
  -)PANEL [KEYLIST(keylist name,[applid,[SHARED]])]
    - ISPKYLST is used if keylist not specified
- Normally in xxxxKEYS table in ISPTLIB
  - Use DTL to create

```
<!DOCTYPE DM SYSTEM>
<KEYL NAME=MYKEYLST APPLID=ABC>
  <KEYI KEY=F1  CMD=HELP   FKA=YES>Help
  <KEYI KEY=F2  CMD=SPLIT  FKA=LONG>Split
  <KEYI KEY=F3  CMD=EXIT   FKA=YES>Exit
  ...
  <KEYI KEY=F24 CMD=CANCEL FKA=YES>Cancel
</KEYL>
```

- Private copies
  - Created by Keylist Utility
  - Stored in xxxxPROF
  - Stored in ISPSPROF for applid ISP

•Keylists define the commands associated with each PFKEY

•Using Keylists, PF Key definitions can be defined on a screen by screen basis, so that only the PF keys that have a function that relates to the panel are defined.

•Keylists are defined on a panel using the)PANEL keyword

  •Keylists are stored in tables in members with 'KEYS' as the suffix, xxxx is the applid

  •They are read from ISPTLIB

  •Keylists can be created using ISPF options 0 ('Function Keys' pull down) or using Dialog Tag Language

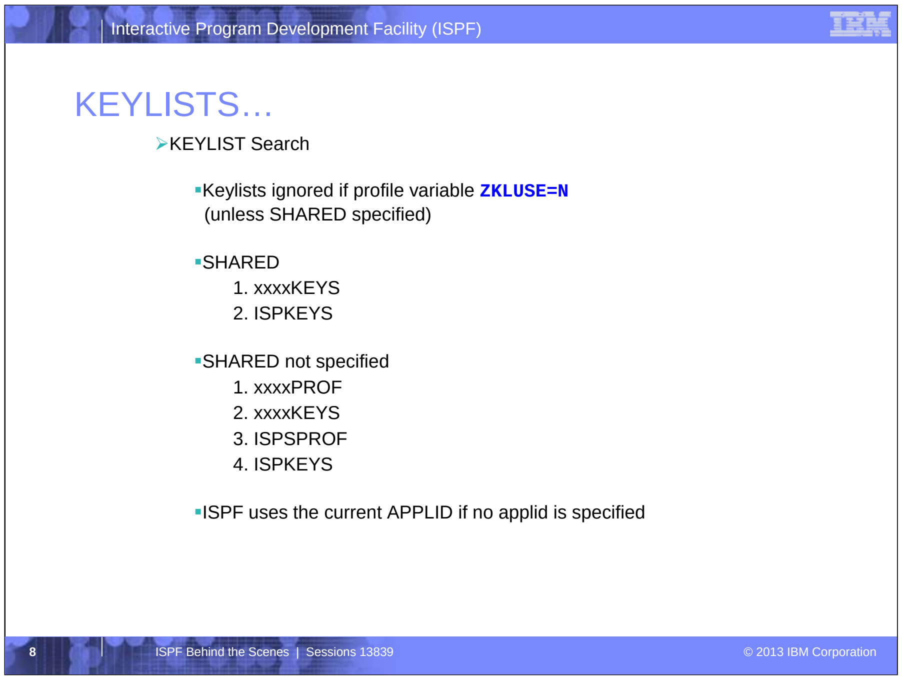
    •The example here shows the DTL syntax to create a keylist MYKEYLST for applid ABC

    •FKA specifies whether the Key assignment is to appear in the function key area at the bottom of the panel - Default is NO, other values are YES, LONG or SHORT. (YES is equivalent to SHORT - provided for compatibility reasons)

•Private copies of Keylists are created using the Keylist Utility (Command = KEYLIST)

  •The resulting keylists are stored in xxxxPROF member in users ISPF profile in DD ISPPROF

  •The private keylists for application Id ISP are stored in ISPSPROF member in users ISPF profile.



KEYLISTS…

- KEYLIST Search
  - Keylists ignored if profile variable **ZKLUSE=N** (unless SHARED specified)
  - SHARED
    1. xxxxKEYS
    2. ISPKEYS
  - SHARED not specified
    1. xxxxPROF
    2. xxxxKEYS
    3. ISPSPROF
    4. ISPKEYS
  - ISPF uses the current APPLID if no applid is specified

•Keylists are ignored if dialog variable ZKLUSE is set to 'N', unless the keyword 'SHARED' is specified on the)PANEL statement

•When SHARED is specified ISPF only looks at the SHARED keylist and not the PRIVATE keylists

•So with SHARED specified, ISPF will only search for the application supplied and ISP keylists (keylists supplied by ISPF)

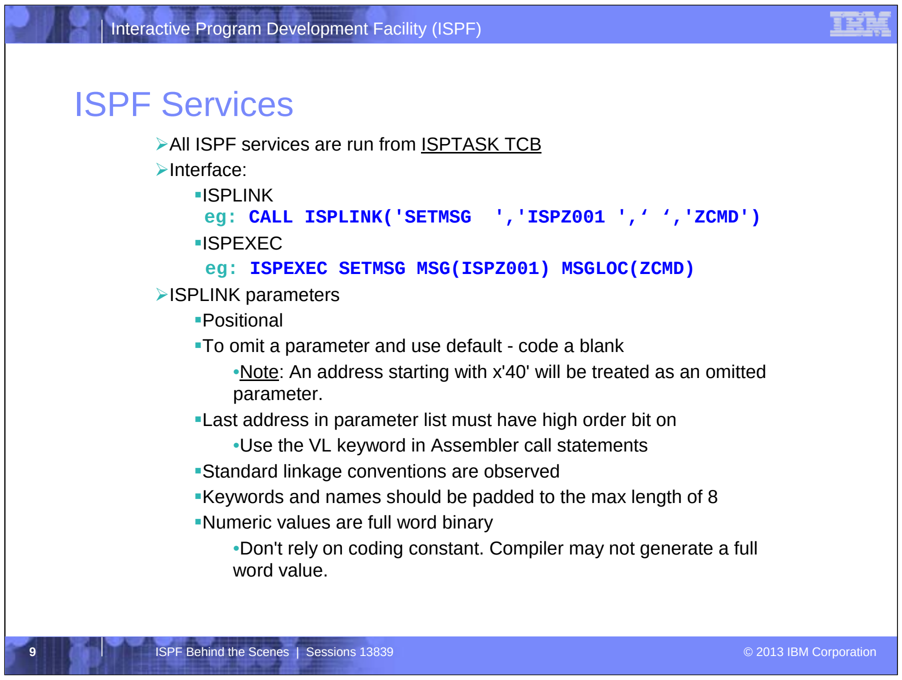
•Without SHARED specified, ISPF will extend the search to include user's profiles and ISPS profiles (to find private copies of the keylist)

•ISPF will use the current APPLID, if no APPLID is specified on the)PANEL statement.

•It is good coding practice to always specify the application id

•avoids confusing

•avoids dialog errors if another application id is used, or not specified, so defaults to ISR/ISP



ISPF Services

- All ISPF services are run from ISPTASK TCB
- Interface:
  - ISPLINK

    eg: `CALL ISPLINK('SETMSG  ','ISPZ001 ',' ','ZCMD')`
  - ISPEXEC

    eg: `ISPEXEC SETMSG MSG(ISPZ001) MSGLOC(ZCMD)`
- ISPLINK parameters
  - Positional
  - To omit a parameter and use default - code a blank
    - Note: An address starting with x'40' will be treated as an omitted parameter.
  - Last address in parameter list must have high order bit on
    - Use the VL keyword in Assembler call statements
  - Standard linkage conventions are observed
  - Keywords and names should be padded to the max length of 8
  - Numeric values are full word binary
    - Don't rely on coding constant. Compiler may not generate a full word value.

•All ISPF services are run from the ISPF **ISPTASK TCB** - we will look at the task structure used by ISPF is a moment

•ISPF provides 2 interfaces to access the ISPF services

•ISPLINK (programs), and

•ISPEXEC (Rexx and CLISTs)

•ISPLINK parameter are positional. They must appear in the order described for each service

•To omit a parameter and use the default, specify a blank - a single blank is sufficient.

•**NOTE**: an address starting with Hex 40 will be treated as an omitted parameter (unlikely to occur)

•The last address in the parameter list MUST have the high order bit set

•This should be done, even if all parameter are specified, as command syntax can change from release to release (as happened to CONTROL ERRORS RETURN/CANCEL - there is now an additional NOSETMSG parameter). Unpredictable errors or Abend0C4's can occur if the High order bit has not been set correctly.

•In assembler use the VL keyword on the CALL macro, most other languages - COBOL, PL1, FORTRAN, PASCAL the High order bit will be set by the compiler.

•Standard linkage conventions are used

•R1 points to a list of addresses for each parameter,

•R13 contains the savearea pointer

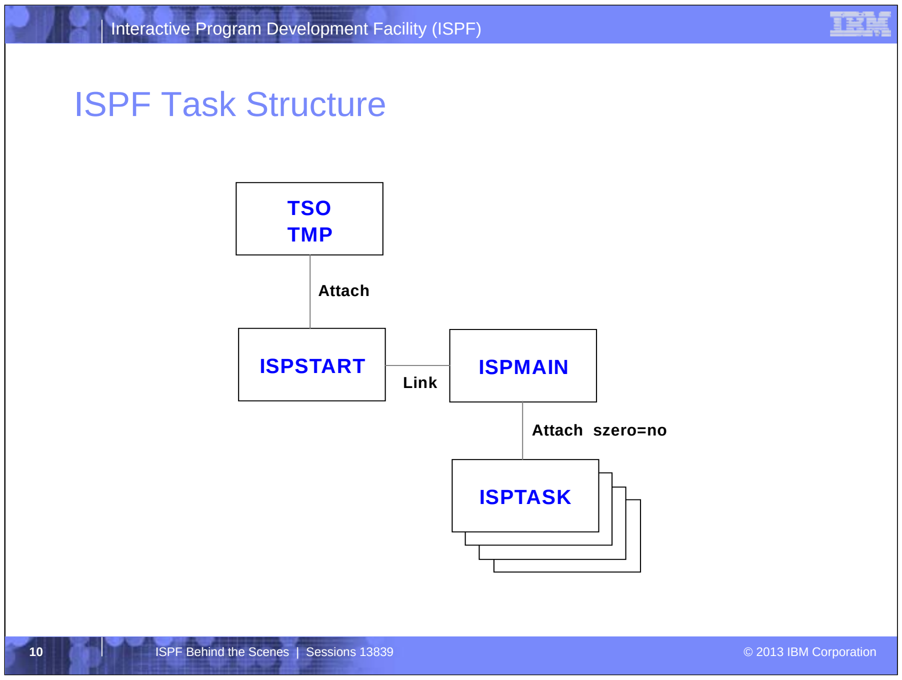
•R14 contains the return address, and

•R15 contains the entry address.

•Keywords and names should be padded to the max length of 8 characters

•**NOTE**: - If literal at end of page boundary, may get Abend 0C4 if length < 8

•Numeric values are always a fullword binary number


Interactive Program Development Facility (ISPF)

ISPF Task Structure

•As we have already said, ISPF runs as a TSO command processor, so ISPF must run under the TSO Terminal Monitor Program - IKJEFT01

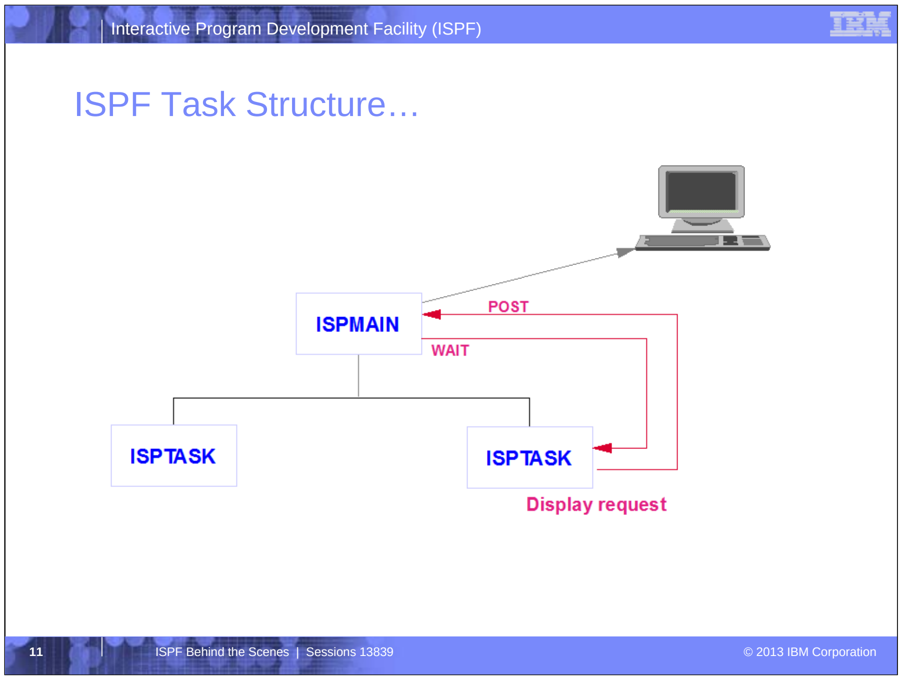
- •(A TSO LOGON proc or BATCH ISPF job will have EXEC PGM=IKJEFT01 in the JCL)

•When ISPF, PDF or the ISPSTART command is issued, like all TSO command processor, it runs in a task of its own. This is the main ISPF task shown here as ISPMAIN and runs as a sub-task of IKJEFT01

•ISPMAIN starts each of the screen tasks as a separate subtask under ISPMAIN

- •The first screen task is initialized before the ISPF Primary Menu is displayed
- •szero=no indicates that subpool 0 is NOT shared between the subtasks – that is storage and hence variable pools are not shared between the different screens

- •**NOTE**: szero=no is not supported by VSAM - receive ABEND0C4 opening a VSAM KSDS on more than 1 screen (see APAR OZ66402 closed PRS in 1982) (The VSAM BLDVRP macro uses subpool 0 for some control blocks. When a task ends, the system frees subpool 0 unless it is shared with another task (ie. SZERO=YES). If vsam dataset1 is opened in one tso session, and then another tso session is establish thru split screen, vsam will connect into the existing control block structure(built for open of ds in tso session 1) for the second open of the file. When the split screen occurs tso or ispf does an attach svc42 (x'2a') with szero=no which means no sharing of subpool zero. If session 1 is terminated then subpool zero is freed, therefore when session 2 is terminated (which closes the vsamds1 again) it abends because the structure that was setup in subpool zero is no longer there.



ISPF Task Structure…

•All panel displays are handled by the ISPMAIN task (ISPMAIN is always in a WAIT state on screen tasks)

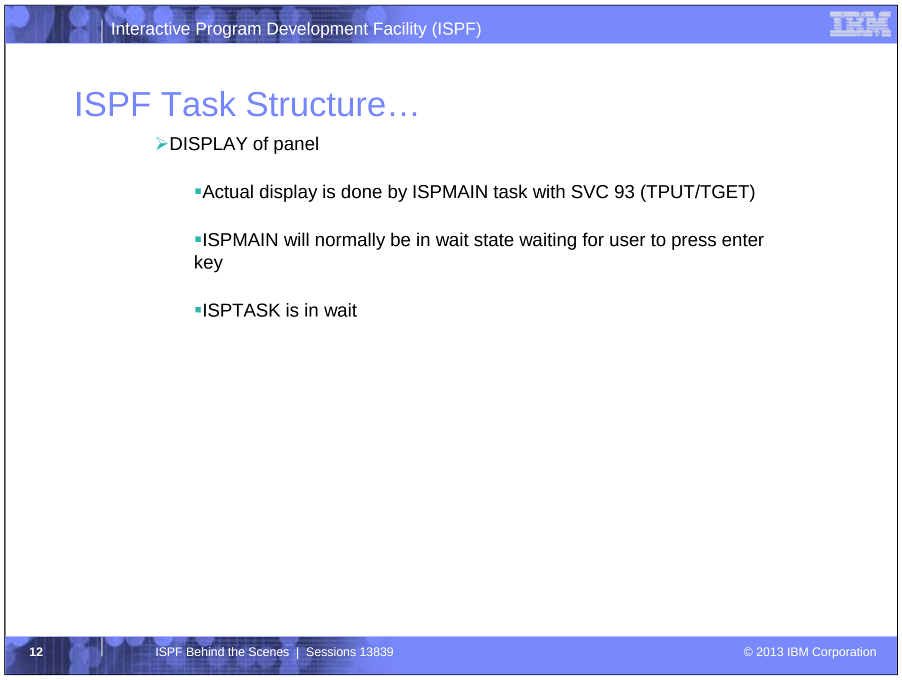
•A subtask that issues a DISPLAY request will POST to the MAIN task to handle the display

•The Sub task will then wait on the Main task

•Once the user responds at the terminal, the Main task will post back to the Sub Task and go back into a wait state for the next display request

•Why is it done this way

  •ISPF split screen means that the physical display may comprise of information from 2 logical screens (4 on a 3290), so the main task becomes responsible for building the composite display



ISPF Task Structure…

- DISPLAY of panel
  - Actual display is done by ISPMAIN task with SVC 93 (TPUT/TGET)
  - ISPMAIN will normally be in wait state waiting for user to press enter key
  - ISPTASK is in wait

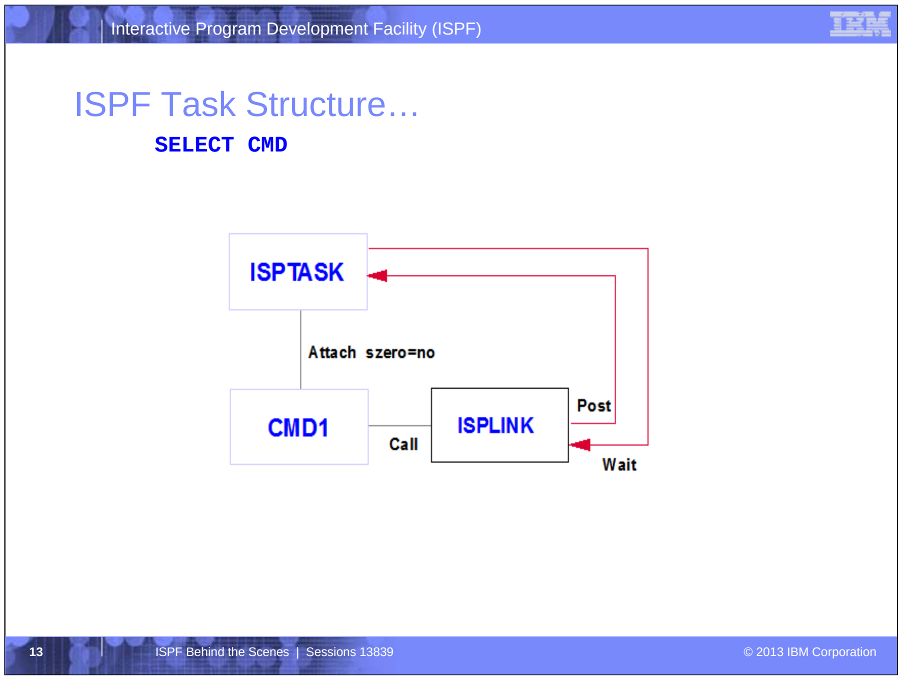
•All panel displays are handled by the ISPMAIN task using SVC93 which interfaces to the TSO TPUT and TGET services

•ISPMAIN will build the display based on the active screen(s)

  •This is required as the terminal display may contain data from 1 or 2 (or 4 for a 3290 terminal) depending on the position of the split line

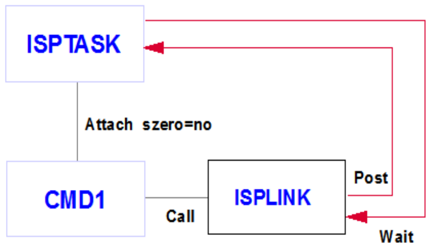
•ISPMAIN will normally be in a wait state waiting for you to press 'Enter', and

•ISPTASK will be in a wait state, waiting for ISPMAIN to post a result



ISPF Task Structure…

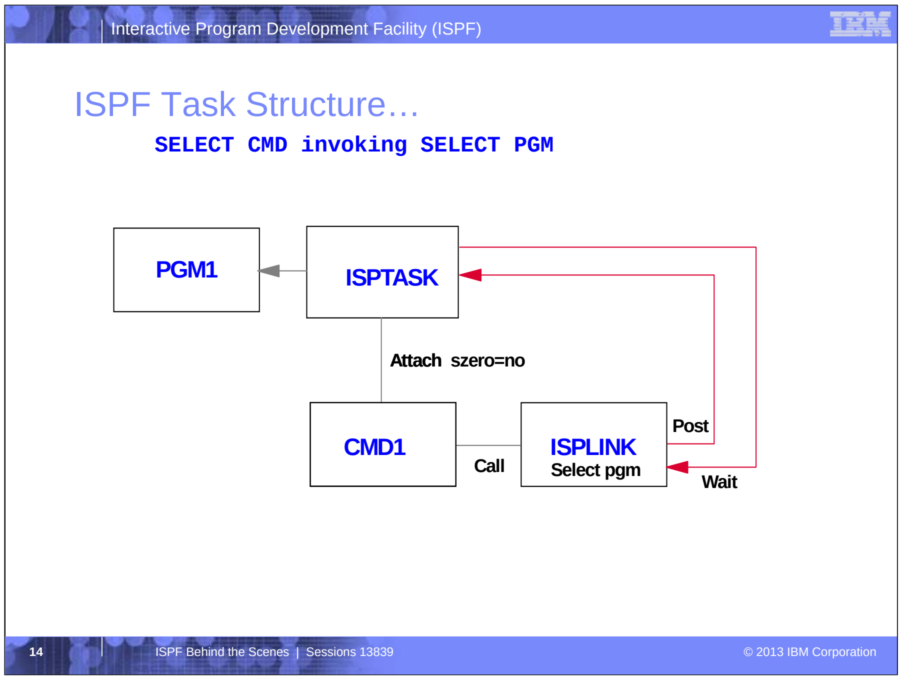
SELECT CMD



•We will now look at the effect on the ISPF task structure depending on the way a program or command in invoked.

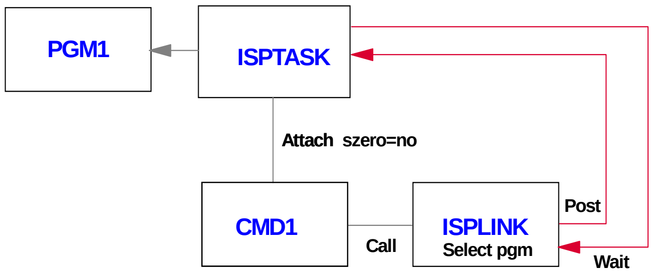
•A Command invoked via the SELECT CMD service will attach CMD1 as a subtask of the screen Task - ISPTASK - where the command is invoked (ISPTASK in a WAIT state waiting for a ISPF service call from CMD1)

•A call to ISPLINK (or ISPEXEC) to invoke an ISPF service will be posted to run by ISPTASK. When complete control will be posted back and CMD1 will continue processing.



ISPF Task Structure…

SELECT CMD invoking SELECT PGM



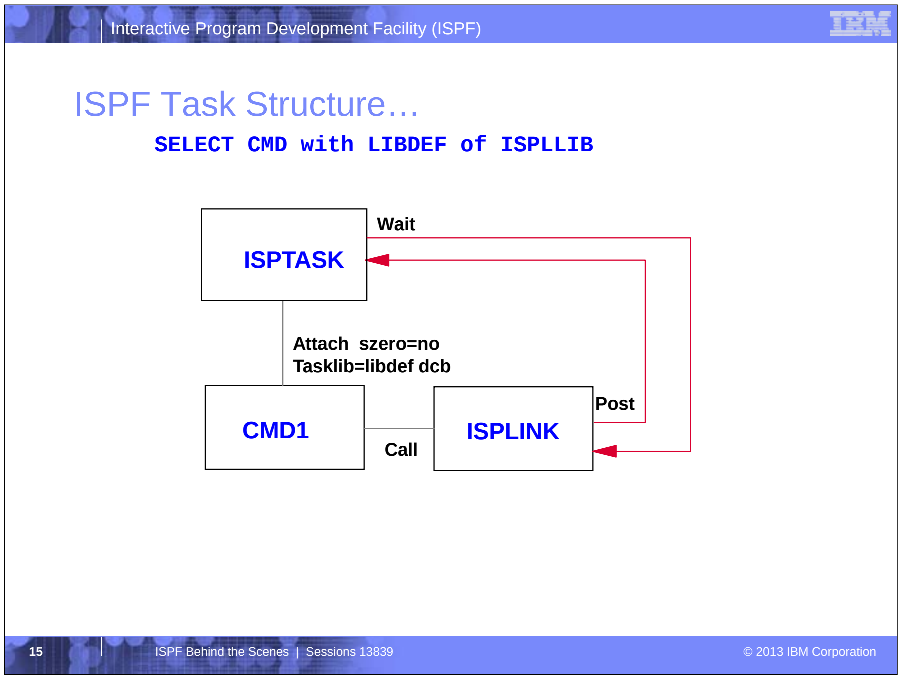
•If CMD1 now invokes PGM1 via the ISPF SELECT PGM service,

•then as before CMD1 will run as a separate task under ISPTASK

•The SELECT PGM will be posted to ISPTASK and PGM1 will run within ISPTASK

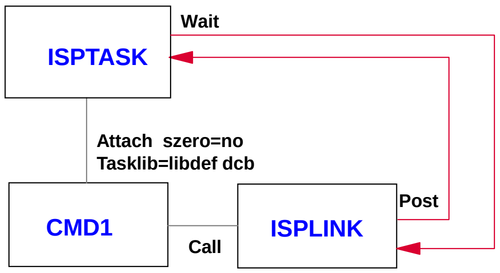
•When PGM1 finishes control will be returned via ISPLINK to CMD1

•So far, this should be simple to understand ...



ISPF Task Structure…

SELECT CMD with LIBDEF of ISPLLIB



•What happens when we now have an active LIBDEF for ISPLLIB?

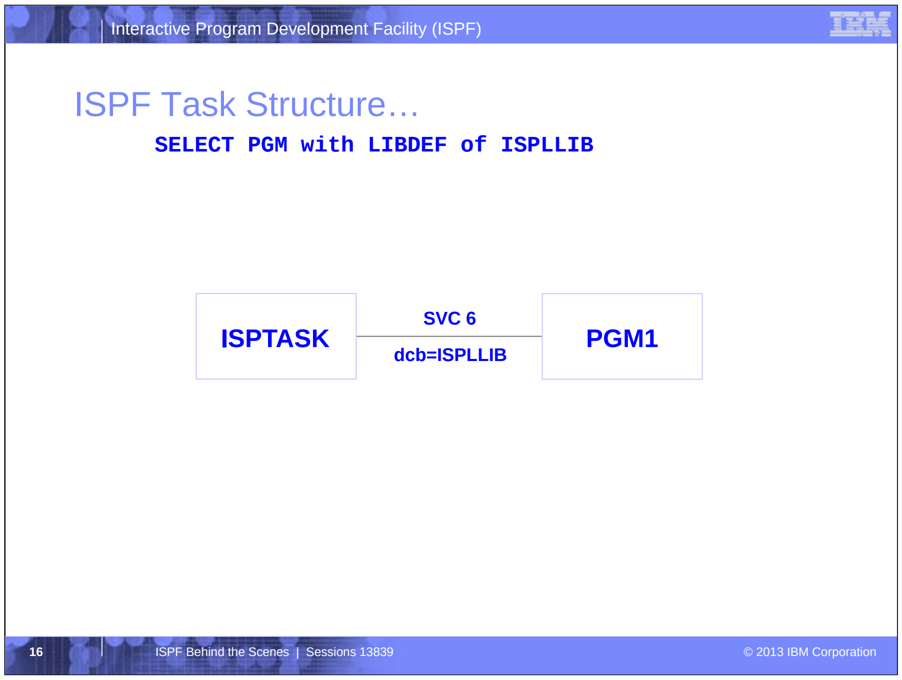
•Remember as the LIBDEF is an ISPF service, ISPF will establish the LIBDEF environment at the ISPTASK level

•When CMD1 is now invoked with the LIBDEF for ISPLLIB active, ISPTASK will attach the subtask for CMD1 and pass the LIBDEF DCB as the TASKLIB

<u>Additional Notes - This is discussed in LIBDEF / ISPLLIB section</u>

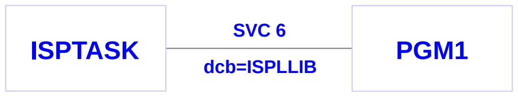
•We will see shortly that ...
this means that any programs called from CMD1 will be able to be loaded from the ISPLLIB LIBDEF.

•Of course if CMD1 issues a SELECT CMD or SELECT PGM, as this will run within the ISPTASK subtask, they too will be able to use the ISPLLIB LIBDEF.



ISPF Task Structure…

SELECT PGM with LIBDEF of ISPLLIB



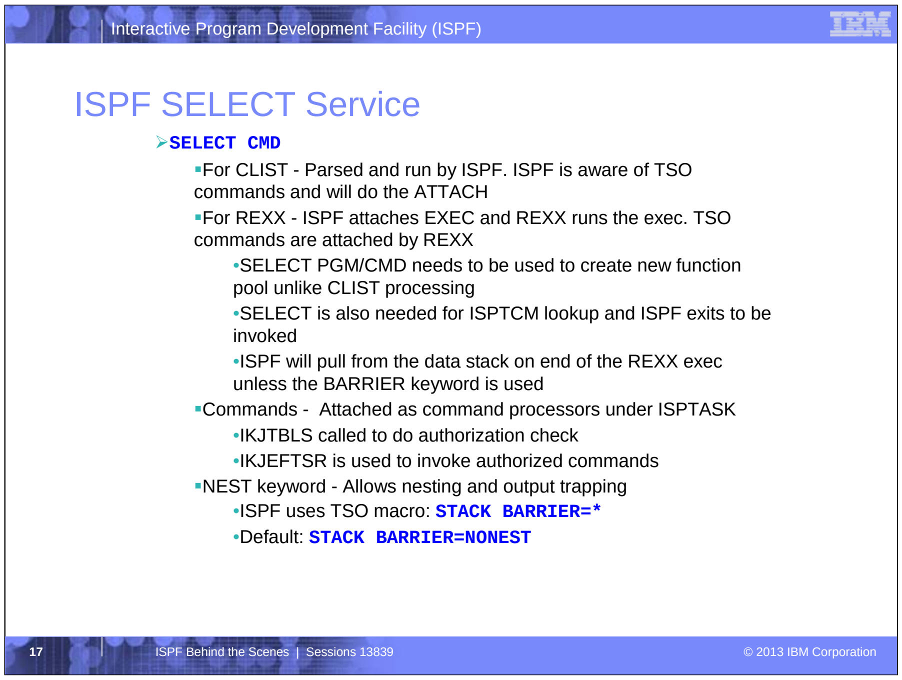
•If we invoke PGM1 via a SELECT PGM with a LIBDEF for ISPLLIB active

•The LINK to PGM1(via SVC 6) will use the DCB for ISPLLIB LIBDEF
NOTE: For LINK, the DCB is for the PDS containing the load module for the specified EP.

Additional Notes - This is discussed in LIBDEF / ISPLLIB section

•But we will see shortly that while PGM1 can be loaded via the ISPLLIB LIBDEF, any programs that PGM1 calls (via BALR R14,R15) will not be able to use the ISPLLIB LIBDEF

•Any called programs must be loaded from LPALIB / LINKLIST / STEPLIB / TASKLIB or an ISPLLIB that is allocated before ISPF is initialized



ISPF SELECT Service

- **SELECT CMD**
  - For CLIST - Parsed and run by ISPF. ISPF is aware of TSO commands and will do the ATTACH
  - For REXX - ISPF attaches EXEC and REXX runs the exec. TSO commands are attached by REXX
    - SELECT PGM/CMD needs to be used to create new function pool unlike CLIST processing
    - SELECT is also needed for ISPTCM lookup and ISPF exits to be invoked
    - ISPF will pull from the data stack on end of the REXX exec unless the BARRIER keyword is used
  - Commands - Attached as command processors under ISPTASK
    - IKJTBLS called to do authorization check
    - IKJEFTSR is used to invoke authorized commands
  - NEST keyword - Allows nesting and output trapping
    - ISPF uses TSO macro: **STACK BARRIER=***
    - Default: **STACK BARRIER=NONEST**

•The ISPF SELECT CMD service allows ISPF to attach and run:

•CLISTS,

•REXX Execs, and

•Compiled programs (Command Processors)

•ISPF is aware of TSO CLISTS and will manage the attach of TSO commands

•However, REXX execs are attached by ISPF and run by REXX,

•As part of the SELECT CMD processing ISPF will

•create the new function pool for dialog variables,

•Check the TSO Command Module (ISPTCM), and

•invoke any ISPF exits

•IKJTBLS - table look-up service to search the lists of authorized commands

•IKJEFTSR - allows a user to invoke functions such as commands, programs, CLISTs, or REXX execs from an application program.

•PUSH - puts one item of data on the top of the data stack.

•QUEUE - puts one item of data on the bottom of the data stack.

•PULL and PARSE PULL - remove one element from the top of the data stack.

•Types of input that can be stored on the data stack are:

•Data for the PULL and PARSE PULL

•Information the EXECIO command reads from and writes to data sets when performing I/O

•Responses to TSO commands

•TSO commands to be issued after the exec ends

•The NEST keyword on the SELECT CMD causes commands invoked with SELECT to be nested and allows:

•for trapping of command output, and

•communication through global variables

•NEST keyword causes ISPF to issue the TSO STACK macro

•BARRIER: creates a barrier element, which divides the input stack into substacks, on top of the input stack

•**BARRIER=*** CLISTs and REXX execs on opposing sides of this barrier are nested. They are able to use command output trapping and can communicate through global variables. Command processors can use routines IKJCT441 and IRXEXCOM to access variables on the opposing side of the barrier.

•**BARRIER=NONEST** CLISTs and REXX execs on opposing sides of the barrier are not nested. This type of barrier halts the effect of command output trapping and halts the use of the routines IKJCT441 and IRXEXCOM to access variables on the opposing side of the barrier. While CLIST global variables are not communicated across this barrier, CLISTs on top of this barrier can begin using global variables and communicate with further nested CLISTs through global variables.

•CLIST Output trapping - &SYSOUTTRAP and &SYSOUTLINE variables save output from TSO/E commands and allow a CLIST or application to process the output.

•REXX output trapping - OUTTRAP function puts lines of command output into a series of numbered variables, each with the same prefix. These variables save the command output and allow an exec to process the output.

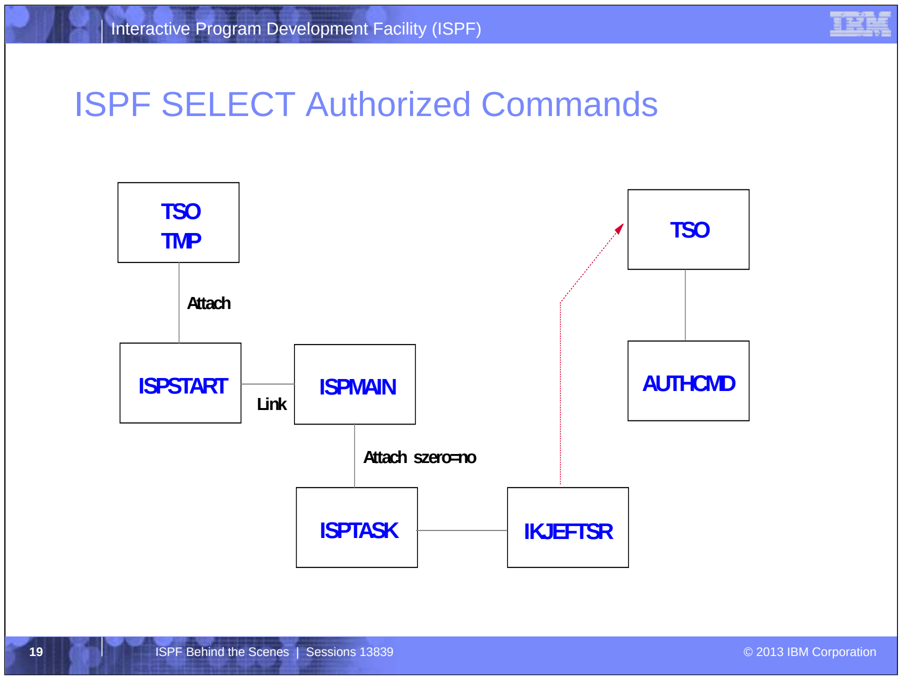
•CLIST global variables - defined on a GLOBAL statement and allow communication between nested CLISTs.

•**NOTE**: The BARRIER keyword stops commands from the REXX data stack being pulled upon completion of a command invoked with the SELECT service.

•ISPF uses STACK DELETE=BARRIER

ISPF SELECT Service…

- **SELECT PGM**
  - LINK (SVC 06) macro used to invoke program
  - Authorization check done with call to IKJTBLS
  - LIBDEF only affects selected pgm
  - PARM - half word length followed by data
- **NEWAPPL**
  - Opens the following ISPF tables
    - xxxxPROF
    - xxxxCMDS
  - Edit will open xxxxEDIT
  - LIBDEF of ISPTLIB must be done prior to SELECT to affect xxxxCMDS

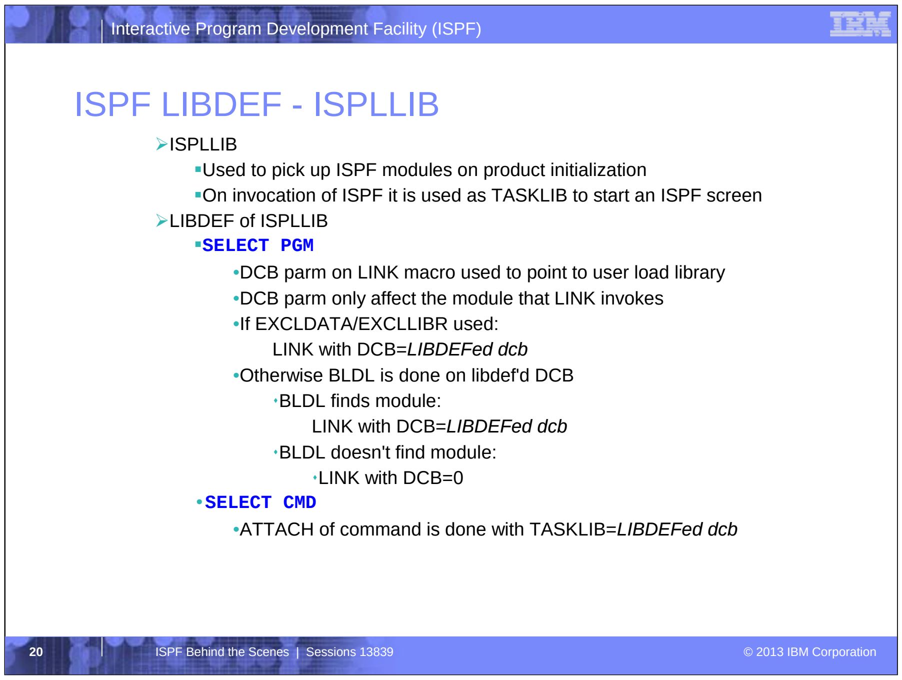


ISPF SELECT Authorized Commands

•When the ISPTASK detects via the table look up service IKJTBLS that an authorized command is being called it invokes the TSO Service Routine IKJEFTSR to invoke the command as authorized

•This will post to a parallel TSO TMP task (isolated environment) to attach and run the authorized command.

  •As this command is not running under any of the ISPF Tasks, the Authorized command can NOT invoke any ISPF services



ISPF LIBDEF - ISPLLIB

- ISPLLIB
  - Used to pick up ISPF modules on product initialization
  - On invocation of ISPF it is used as TASKLIB to start an ISPF screen
- LIBDEF of ISPLLIB
  - **`SELECT PGM`**
    - DCB parm on LINK macro used to point to user load library
    - DCB parm only affect the module that LINK invokes
    - If EXCLDATA/EXCLLIBR used:
      LINK with DCB=*LIBDEFed dcb*
    - Otherwise BLDL is done on libdef'd DCB
      - BLDL finds module:
        LINK with DCB=*LIBDEFed dcb*
      - BLDL doesn't find module:
        - LINK with DCB=0
  - **`SELECT CMD`**
    - ATTACH of command is done with TASKLIB=*LIBDEFed dcb*

•When ISPLLIB is allocated before ISPF is initialized,

•ISPF will use the ISPLLIB during product initialization to load the ISPF modules,

•It also will use the ISPLLIB DCB as the TASKLIB as it attaches the subtask to start each ISPF screen (ISPTASK in the previous diagrams)

•This means that ANY program that we call that is loaded via either the LOAD, XCTL or LINK macros can be loaded from the LIBDEF'ed ISPLLIB

•The behaviour of ISPLLIB differs when it is allocated using the ISPF LIBDEF sevice

•When a SELECT PGM is issued,

•the ISPLLIB DCB is passed as the DCB parameter on the LINK macro.

•This parameter only affects the module that LINK is invoking - not any subsequent modules

•When SELECT CMD is issued,

•The ISPLLIB DCB is passed as the TASKLIB parameter on the ATTACH macro

•This means that the ISPLLIB is available for all LOAD, XCTL, LINK macro calls that are made within the subtask

•An example of this.

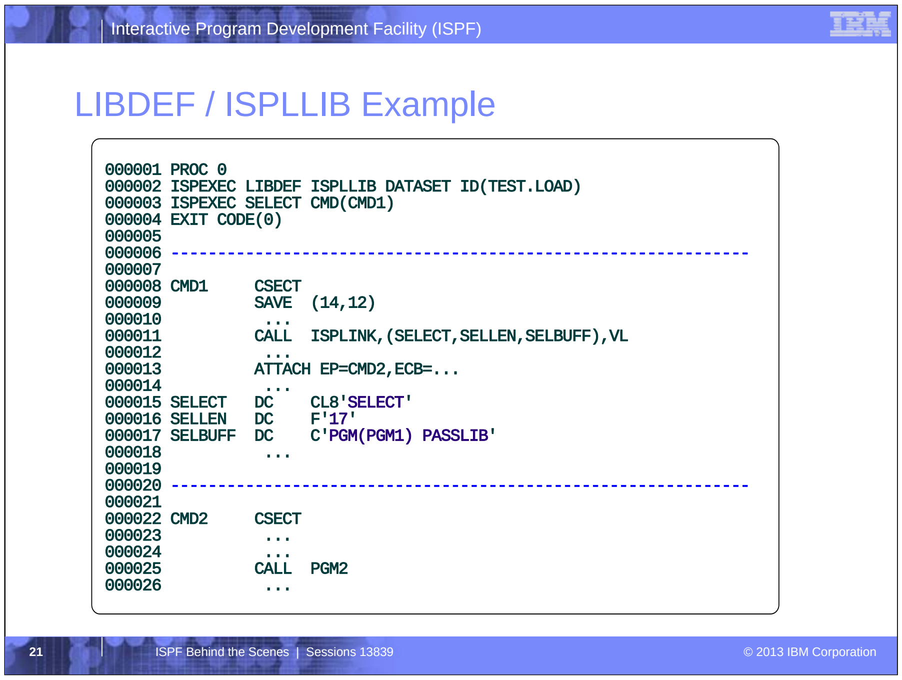
•We have an application program that requires access to DB2.

•DB2 has a stub module - DSNALI with several entry points (DSNALI, DSNHLI, DSNHLI2, DSNWLI, and DSNWLI2) that is normally linkedited into our application module

•As part of our application we can issue a LIBDEF for ISPLLIB and then invoke our application program that will access DB2

•If we use the SELECT PGM to invoke our application program then the DB2 routine will not be able to make use of the ISPLLIB, - there is no way of passing the ISPLLIB DCB into the DB2 routines.

•However, if we use SELECT CMD, then the DB2 routines will be able to access the ISPLLIB via the TASKLIB and locate its program modules



LIBDEF / ISPLLIB Example

```
PROC 0
ISPEXEC LIBDEF ISPLLIB DATASET ID(TEST.LOAD)
ISPEXEC SELECT CMD(CMD1)
EXIT CODE(0)

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

CMD1     CSECT
         SAVE  (14,12)
          ...
         CALL  ISPLINK,(SELECT,SELLEN,SELBUFF),VL
          ...
         ATTACH EP=CMD2,ECB=...
          ...
SELECT   DC    CL8'SELECT'
SELLEN   DC    F'17'
SELBUFF  DC    C'PGM(PGM1) PASSLIB'
          ...

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

CMD2     CSECT
          ...
          ...
         CALL  PGM2
          ...
```

•As an example, if we have a CLIST that:

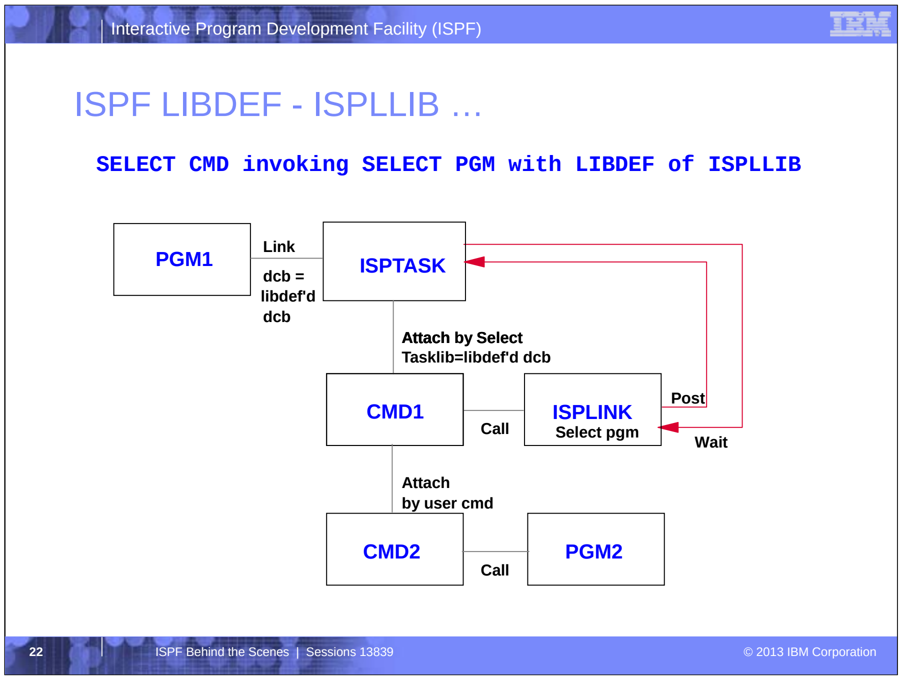
- •Issue a LIBDEF for ISPLLIB, and
- •Then issues a SELECT CMD for CMD1

•CMD1 is an assembler program which:

- •Issues a SELECT PGM for PGM1, and
- •Then issues an ATTACH for CMD2

•CMD2 then calls PGM2

- •(we would need to issue a LOAD macro to obtain the entry address for PGM2)



ISPF LIBDEF - ISPLLIB …

SELECT CMD invoking SELECT PGM with LIBDEF of ISPLLIB

This is how the task structure would look for our example:

•CMD1 is attached as a subtask with a TASKLIB for our LIBDEF DCB

•The SELECT PGM for PGM1 will be posted to ISPTASK and PGM1 will run within the ISPTASK subtask

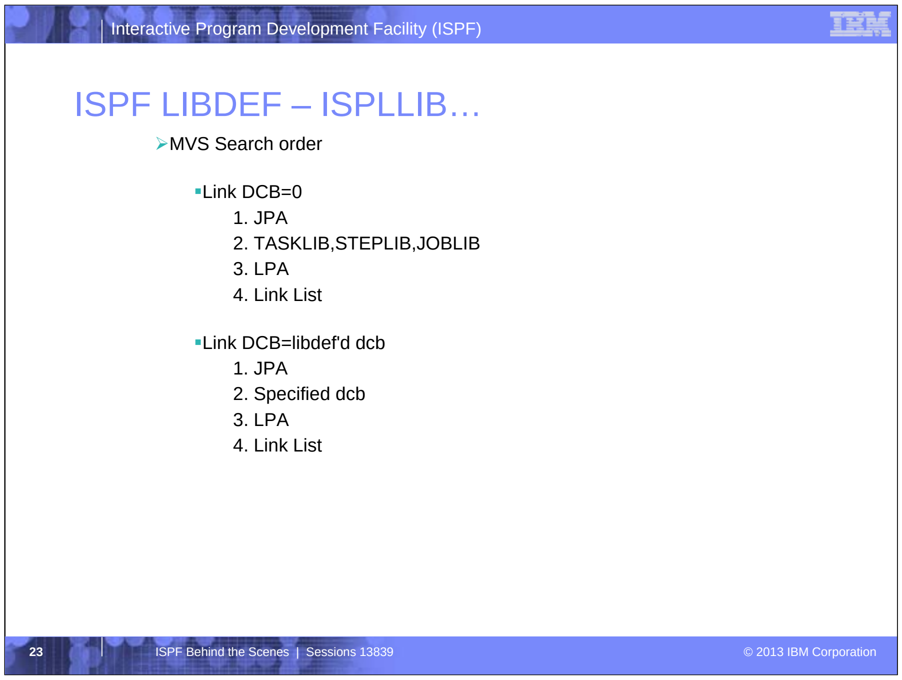
•PGM1 terminates and returns control back to the CMD1 subtask

•CMD1 then attaches CMD2 as a new subtask

 •The ISPLLIB DCB is not passed as a TASKLIB

•CMD2 then calls PGM2. ATTACH by default gets the TASKLIB from the invoking subtask (ie. ISPLLIB will be searched for PGM2).

Just for completeness and its not shown on your handouts, CLIST1 that issues the LIBDEF will be running in its own subtask under ISPTASK



ISPF LIBDEF – ISPLLIB…

- MVS Search order
  - Link DCB=0
    1. JPA
    2. TASKLIB,STEPLIB,JOBLIB
    3. LPA
    4. Link List
  - Link DCB=libdef'd dcb
    1. JPA
    2. Specified dcb
    3. LPA
    4. Link List

What is the MVS (or z/OS) search order for load modules?

•Without the LIBDEF (ie DCB=0 on the LINK), it searches:

•The Job Pack Area (programs that have already been loaded)

•The TASKLIB (could have one supplied by TSO via ALTLIB),

•then the STELIB and JOBLIB

•Then it searches the Link Pack Area (or LPA), and

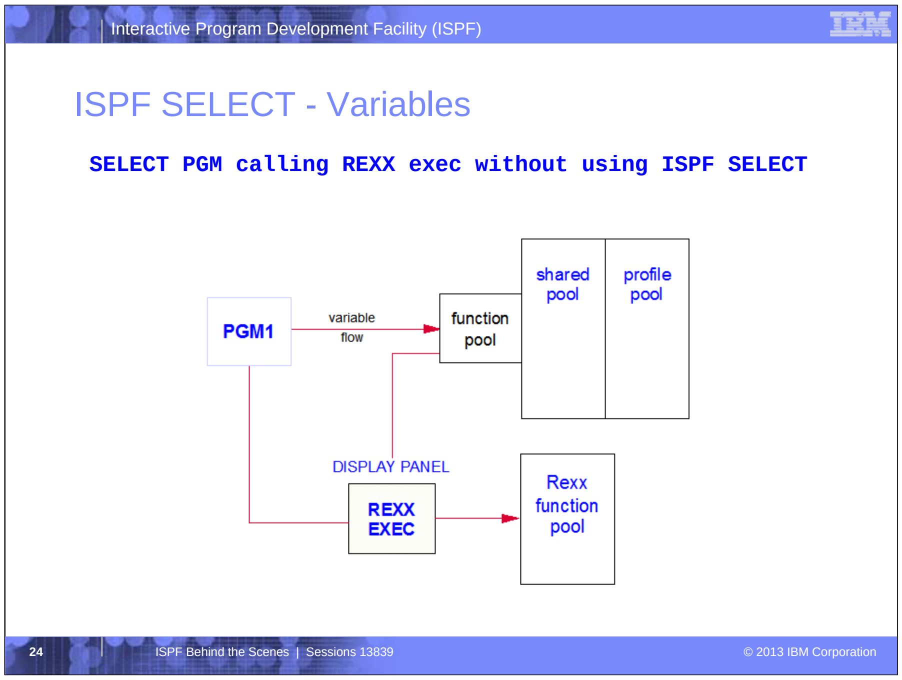
•then finally the Link List

•With a LIBDEF for ISPLLIB, in which case there is a DCB parameter on the LINK macro, MVS will search:

•the JPA first,

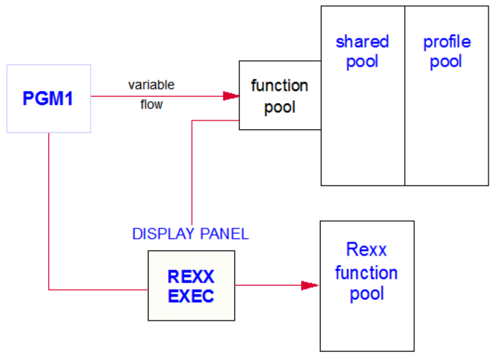
•then the LIBDEF DCB,

•followed by the LPA and the Link List



ISPF SELECT - Variables

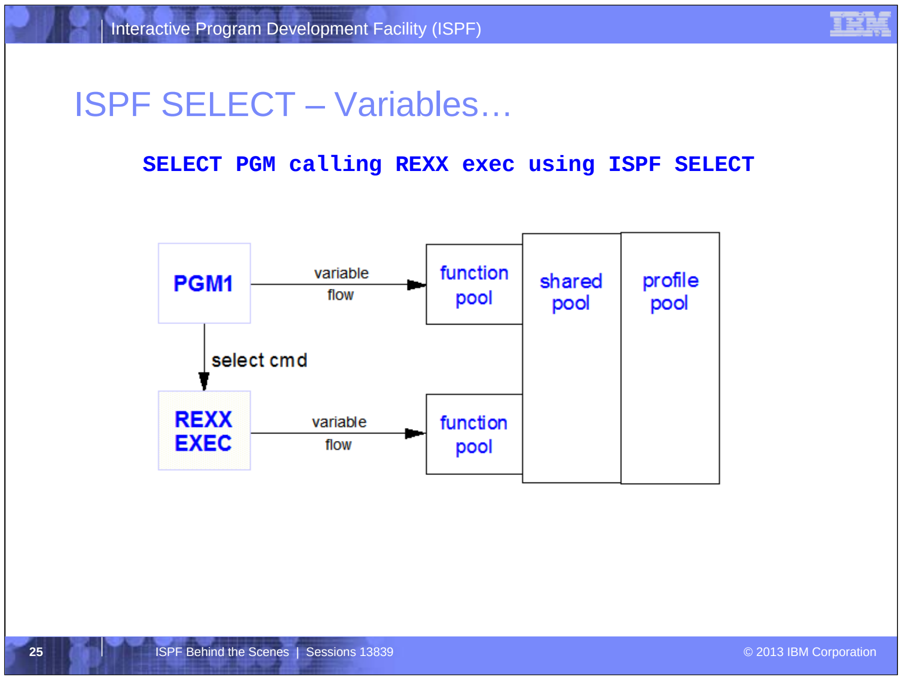
SELECT PGM calling REXX exec without using ISPF SELECT



•A program running under ISPF that invokes a Rexx Exec will result in the Rexx Exec being attached as a subtask of the TSO Terminal Monitor Program task

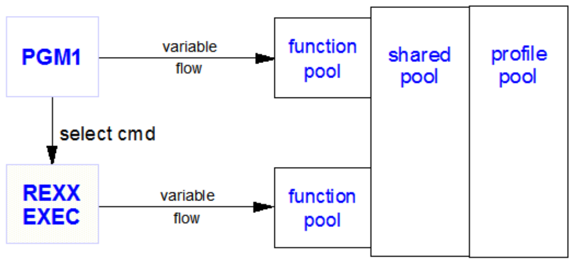
•As a result of this the Rexx Exec will have its Rexx function pool, but will have no way of communicating to either of the ISPF shared or profile pools.

If the Rexx Displays a panel via the DISPLAY service, this will be run by the ISPTASK subtask and will have access to the variable pools



ISPF SELECT – Variables…

SELECT PGM calling REXX exec using ISPF SELECT



When a Rexx Exec is invoked using the ISPF SELECT CMD service

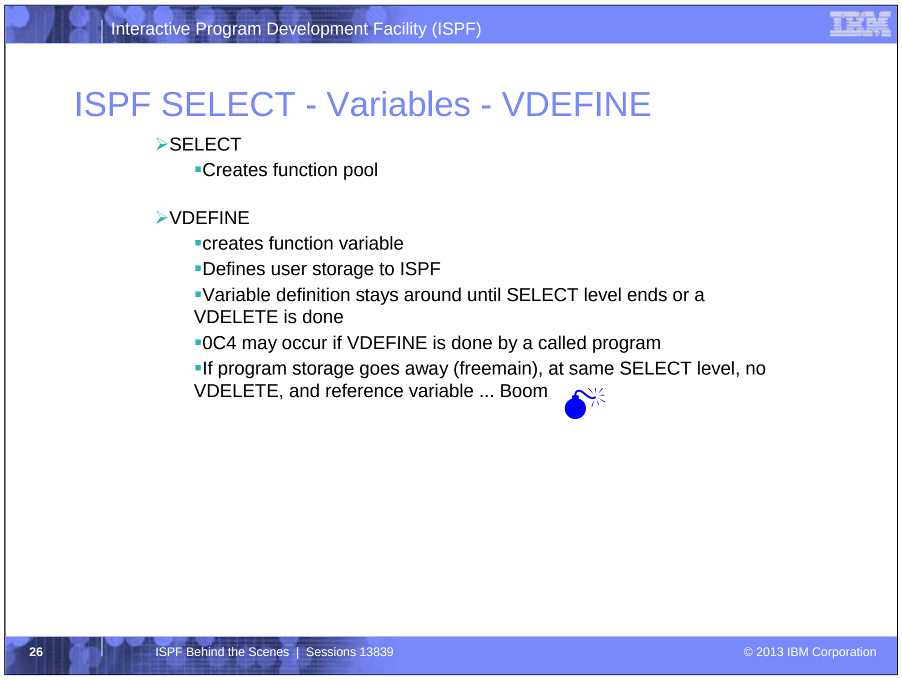
•PGM1 will be executing within the ISPTASK subtask, and

•the REXX Exec will run in a Subtask of the ISPTASK

•Each will have storage for their own local variables,
and the ISPF function pool,

•but they will share both the ISPF shared and profile variable pools

•This will allow variables to be easily passed between the 2 programs



ISPF SELECT - Variables - VDEFINE

- SELECT
  - Creates function pool
- VDEFINE
  - creates function variable
  - Defines user storage to ISPF
  - Variable definition stays around until SELECT level ends or a VDELETE is done
  - 0C4 may occur if VDEFINE is done by a called program
  - If program storage goes away (freemain), at same SELECT level, no VDELETE, and reference variable ... Boom

•Each time the select service is issued a new function pool is created

•This isolates local variables for each application

•The VDEFINE service allows our application programs to create variables in the function pool

•The VDEFINE service is only applicable to the ISPLINK interface.

•Clists and Rexx Execs have function pools automatically created

•The VDEFINE associates user storage with a ISPF variable, and

•the variable definition stays around until either:

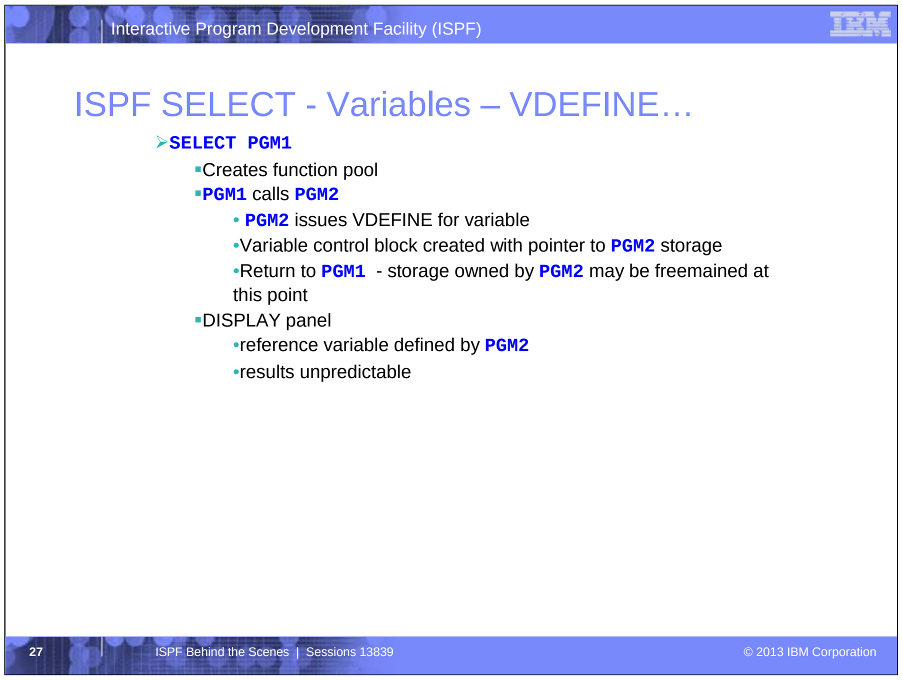
•the SELECT level end (Subtask terminates), or

•a VDELETE is issued

•An Abend S0C4 may occur if a VDEFINE for a variable is done by a called program.

•If the program fails to issue the VDELETE and the program storage is freed (as when a called program terminates), then any reference to the variable may address storage that is no longer owned by the application ... and ... BOOM

•The classic symptoms are an Abend S0C4 in module ISPDVCGT of ISPSUBS



ISPF SELECT - Variables – VDEFINE…

- **SELECT PGM1**
  - Creates function pool
  - **PGM1** calls **PGM2**
    - **PGM2** issues VDEFINE for variable
    - Variable control block created with pointer to **PGM2** storage
    - Return to **PGM1** - storage owned by **PGM2** may be freemained at this point
  - DISPLAY panel
    - reference variable defined by **PGM2**
    - results unpredictable

•If we Issue a SELECT PGM for PGM1

•ISPF will create the function pool for it

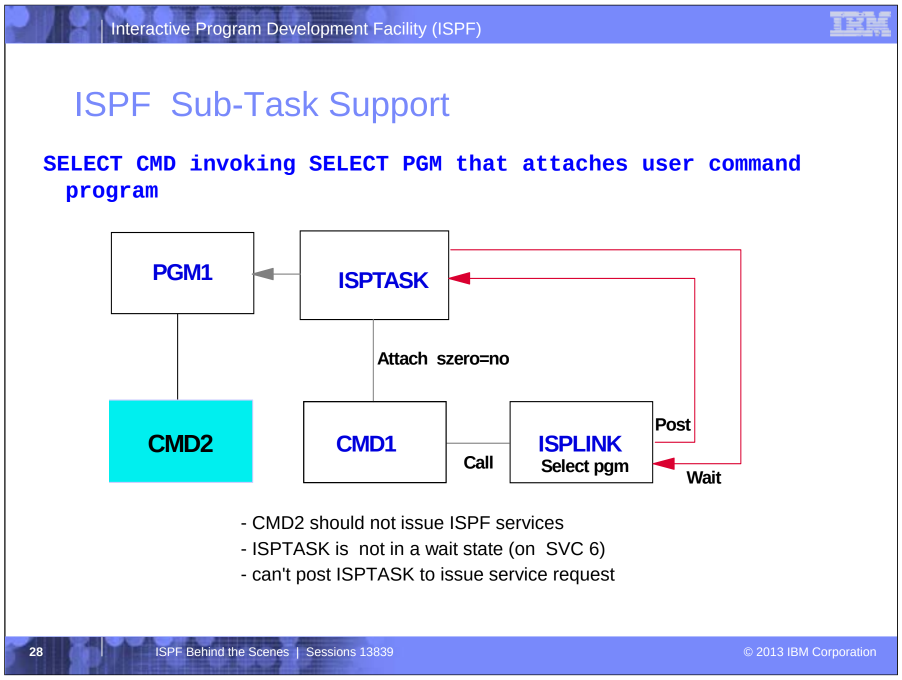
•If PGM1 then calls PGM2 (via a BALR), and
PGM2 issues a VDEFINE for a variable with the variable located in PGM2 storage

•When PGM2 end, the storage owned by PGM2 may be freemained

•If we then reference the variable, such as in a DISPLAY of a panel, the results are unpredictable

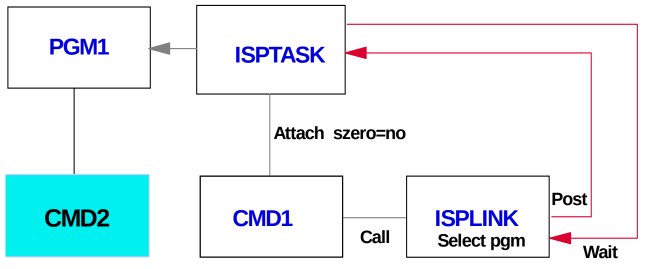
•We could either get an Abend S0C4, or
even worse (or harder to trace) a storage overlay, if something else has acquired the storage

<u>It is very important to cleanup all VDEFINED variables correctly with a VDELETE</u>



ISPF Sub-Task Support

SELECT CMD invoking SELECT PGM that attaches user command program



- CMD2 should not issue ISPF services
- ISPTASK is not in a wait state (on SVC 6)
- can't post ISPTASK to issue service request

One final issue relating to ISPF subtasks

•When writing application programs we need to consider the subtask structure

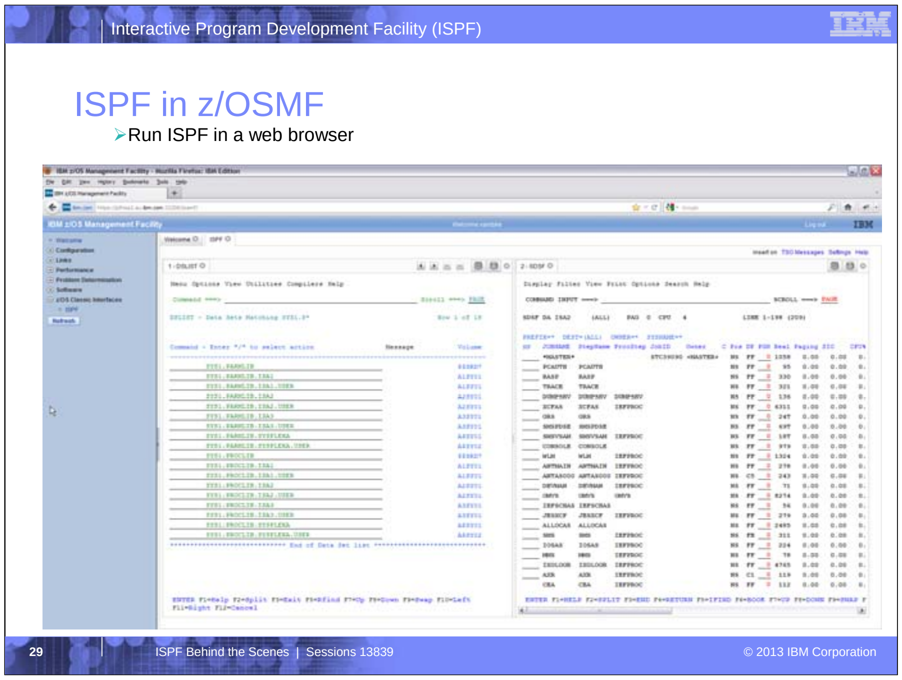
•In this example we have a SELECT CMD that invokes a SELECT PGM that attaches a user Command Program

•In this case CMD2 is unable to issue any ISPF services

•The ISPTASK is NOT in a wait state on the LINK SVC for PGM1

•Using the attach, PGM1 can still be running, so PGM1 can continue to issue ISPF requests,

•If CMD2 attempts to invoke an ISPF service it will be put into an indefinite wait, waiting on an ECB that will never be posted



Interactive Program Development Facility (ISPF)

ISPF in z/OSMF

- ➢Run ISPF in a web browser

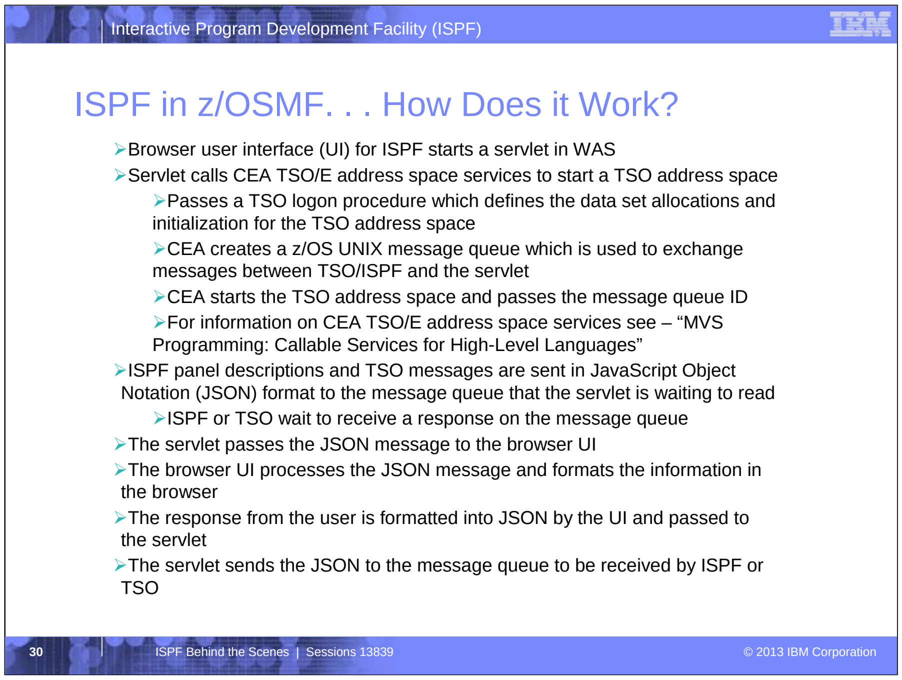
```
1-DSLIST
Menu  Options  View  Utilities  Compilers  Help

Command ===> ________________________________________  Scroll ===> PAGE

DSLIST - Data Sets Matching SYS1.P*                          Row 1 of 18

Command - Enter "/" to select action                  Message          Volume
-------------------------------------------------------------------------------
         SYS1.PARMLIB                                                  SRSRD7
         SYS1.PARMLIB.ISA1                                             A1SYS1
         SYS1.PARMLIB.ISA1.USER                                        A1SYS1
         SYS1.PARMLIB.ISA2                                             A2SYS1
         SYS1.PARMLIB.ISA2.USER                                        A2SYS1
         SYS1.PARMLIB.ISA3                                             A3SYS1
         SYS1.PARMLIB.ISA3.USER                                        A3SYS1
         SYS1.PARMLIB.SYSPLEXA                                         AASYS1
         SYS1.PARMLIB.SYSPLEXA.USER                                    AASYS2
         SYS1.PROCLIB                                                  SRSRD7
         SYS1.PROCLIB.ISA1                                             A1SYS1
         SYS1.PROCLIB.ISA1.USER                                        A1SYS1
         SYS1.PROCLIB.ISA2                                             A2SYS1
         SYS1.PROCLIB.ISA2.USER                                        A2SYS1
         SYS1.PROCLIB.ISA3                                             A3SYS1
         SYS1.PROCLIB.ISA3.USER                                        A3SYS1
         SYS1.PROCLIB.SYSPLEXA                                         AASYS1
         SYS1.PROCLIB.SYSPLEXA.USER                                    AASYS2
***************************** End of Data Set list ****************************

ENTER F1=Help F2=Split F3=Exit F5=Rfind F7=Up F8=Down F9=Swap F10=Left
F11=Right F12=Cancel
```

```
2-SDSF
Display  Filter  View  Print  Options  Search  Help

COMMAND INPUT ===> ________________________________________  SCROLL ===> PAGE

SDSF DA ISA2     (ALL)     PAG  0  CPU   4               LINE 1-198 (209)

PREFIX=*  DEST=(ALL)  OWNER=*  SYSNAME=*
NP   JOBNAME  StepName ProcStep JobID    Owner     C Pos DP PGN Real Paging SIO    CPU%
     *MASTER*                   STC39090 +MASTER+    NS  FF   0 1358  0.00  0.00   0.
     PCAUTH   PCAUTH                                 NS  FF   0   95  0.00  0.00   0.
     RASP     RASP                                   NS  FF   0  330  0.00  0.00   0.
     TRACE    TRACE                                  NS  FF   0  321  0.00  0.00   0.
     DUMPSRV  DUMPSRV  DUMPSRV                       NS  FF   0  136  0.00  0.00   0.
     XCFAS    XCFAS    IEFPROC                       NS  FF   0 6311  0.00  0.00   0.
     GRS      GRS                                    NS  FF   0  247  0.00  0.00   0.
     SMSPDSE  SMSPDSE                                NS  FF   0  497  0.00  0.00   0.
     SMSVSAM  SMSVSAM  IEFPROC                       NS  FF   0  187  0.00  0.00   0.
     CONSOLE  CONSOLE                                NS  FF   0  979  0.00  0.00   0.
     WLM      WLM      IEFPROC                       NS  FF   0 1324  0.00  0.00   0.
     ANTMAIN  ANTMAIN  IEFPROC                       NS  FF   0  278  0.00  0.00   0.
     ANTAS000 ANTAS000 IEFPROC                       NS  C5   0  243  0.00  0.00   0.
     DEVMAN   DEVMAN   IEFPROC                       NS  FF   0   71  0.00  0.00   0.
     OMVS     OMVS     OMVS                          NS  FF   0 8274  0.00  0.00   0.
     IEFSCHAS IEFSCHAS                               NS  FF   0   54  0.00  0.00   0.
     JESXCF   JESXCF   IEFPROC                       NS  FF   0  279  0.00  0.00   0.
     ALLOCAS  ALLOCAS                                NS  FF   0 2495  0.00  0.00   0.
     SMS      SMS      IEFPROC                       NS  FE   0  311  0.00  0.00   0.
     IOSAS    IOSAS    IEFPROC                       NS  FF   0  224  0.00  0.00   0.
     HWS      HWS      IEFPROC                       NS  FF   0   78  0.00  0.00   0.
     IXGLOGR  IXGLOGR  IEFPROC                       NS  FF   0 4765  0.00  0.00   0.
     AXR      AXR      IEFPROC                       NS  C1   0  119  0.00  0.00   0.
     CEA      CEA      IEFPROC                       NS  FF   0  112  0.00  0.00   0.

ENTER F1=HELP F2=SPLIT F3=END F4=RETURN F5=IFIND F6=BOOK F7=UP F8=DOWN F9=SWAP F
```

29 ISPF Behind the Scenes | Sessions 13839 © 2013 IBM Corporation

With z/OS V1R13, ISPF has been available in IBM z/OS Management Facility (z/OSMF). z/OSMF provides a framework for managing various aspects of a z/OS system through a web browser interface. So this effectively means ISPF applications can be run from a web browser.

z/OSMF uses IBM WebSphere® Application Server OEM Edition to provide the native application server runtime environment. Technologies such as JavaScript and a Dojo framework are used for serving the web browser interface.


Interactive Program Development Facility (ISPF)

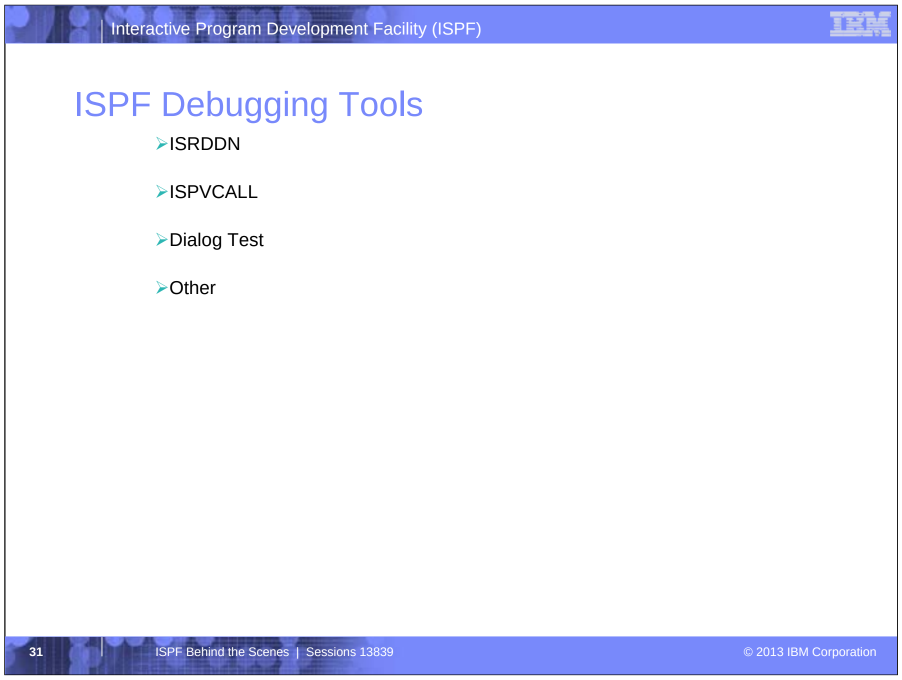
ISPF in z/OSMF. . . How Does it Work?

- Browser user interface (UI) for ISPF starts a servlet in WAS
- Servlet calls CEA TSO/E address space services to start a TSO address space
  - Passes a TSO logon procedure which defines the data set allocations and initialization for the TSO address space
  - CEA creates a z/OS UNIX message queue which is used to exchange messages between TSO/ISPF and the servlet
  - CEA starts the TSO address space and passes the message queue ID
  - For information on CEA TSO/E address space services see – “MVS Programming: Callable Services for High-Level Languages”
- ISPF panel descriptions and TSO messages are sent in JavaScript Object Notation (JSON) format to the message queue that the servlet is waiting to read
  - ISPF or TSO wait to receive a response on the message queue
- The servlet passes the JSON message to the browser UI
- The browser UI processes the JSON message and formats the information in the browser
- The response from the user is formatted into JSON by the UI and passed to the servlet
- The servlet sends the JSON to the message queue to be received by ISPF or TSO

30 ISPF Behind the Scenes | Sessions 13839 © 2013 IBM Corporation

So how was it possible to get ISPF to run in a web browser under z/OSMF? Here’s the view from 30,000 feet.

Z/OSMF starts the ISPF user interface (UI) in the user’s browser. The UI starts a servlet in WAS which calls CEA TSO/E address space services to start a TSO address space for the user. Among the information passed to CEA is the name of a TSO logon procedure. CEA creates a z/OS UNIX message queue to be used to exchange messages between TSO/ISPF and the servlet. CEA then creates the TSO address using the logon procedure to initialize the TSO environment and allocate the data sets for the address space. The message queue ID is returned to the servlet which waits to receive a message on the queue. For a panel display, ISPF creates a panel description in JavaScript Object Notation (JSON) format and sends it to the message queue. Similarly TSO messages are put into JSON format and sent to the message queue. ISPF or TSO then wait to receive a response on the queue. The servlet receives the TSO/ISPF JSON message from the queue and passes it on to the browser UI which processes the JSON and formats the information in the browser. A response entered into an ISPF panel or for a TSO message is formatted by the UI into a JSON message and passed to the servlet which then sends the JSON to the message queue. ISPF or TSO extract the response from the JSON and processing continues in basically the same way as when the user is at a 3270 screen.



ISPF Debugging Tools

- ISRDDN
- ISPVCALL
- Dialog Test
- Other

There are a number of ways we can debug ISPF applications.

We will have a quick look at: -

•ISDDDN is an ISPF command that not only shows data set allocation information, it has a number of other useful displays.

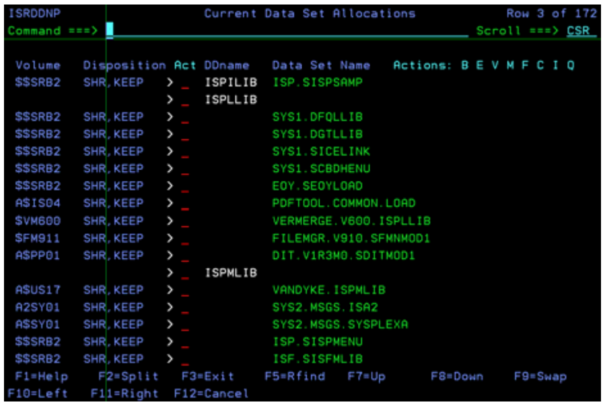
•ISPVCALL traces internal ISPF module flow and can display other information. It is usually requested from ISPF support to assist with problem tracking, but users are finding it provides useful debugging information.

•ISPF Dialog Test provides facilities for testing both complete ISPF applications and ISPF dialog parts, including functions, panels, variables, messages, tables, and skeletons.

•Some other useful debugging techniques

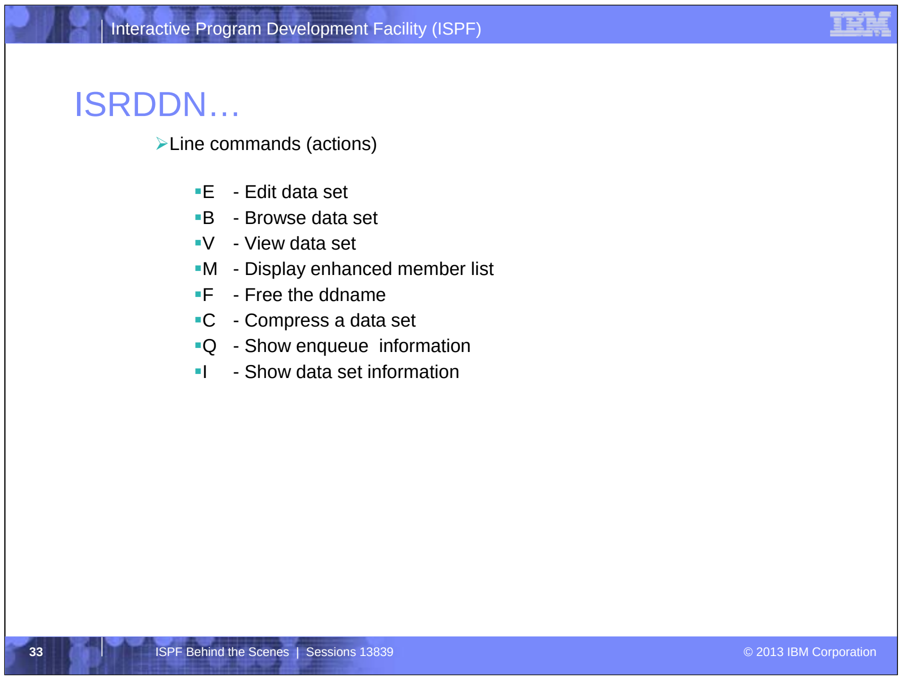
ISRDDN

- Scrollable list of allocated DD's and associated data set names
- Invoked using TSO ISRDDN or DDLIST commands
- Documented in the ISPF User's Guide Volume I



```
ISRDDNP                    Current Data Set Allocations           Row 3 of 172
Command ===>                                                    Scroll ===> CSR

 Volume   Disposition Act DDname   Data Set Name   Actions: B E V M F C I Q
 $$SRB2   SHR,KEEP    > _ ISPILIB  ISP.SISPSAMP
                      > _ ISPLLIB
 $$SRB2   SHR,KEEP    > _          SYS1.DFQLLIB
 $$SRB2   SHR,KEEP    > _          SYS1.DGTLLIB
 $$SRB2   SHR,KEEP    > _          SYS1.SICELINK
 $$SRB2   SHR,KEEP    > _          SYS1.SCBDHENU
 $$SRB2   SHR,KEEP    > _          EOY.SEOYLOAD
 A$IS04   SHR,KEEP    > _          PDFTOOL.COMMON.LOAD
 $VM600   SHR,KEEP    > _          VERMERGE.V600.ISPLLIB
 $FM911   SHR,KEEP    > _          FILEMGR.V910.SFMNMOD1
 A$PP01   SHR,KEEP    > _          DIT.V1R3M0.SDITMOD1
                      > _ ISPMLIB
 A$US17   SHR,KEEP    > _          VANDYKE.ISPMLIB
 A2SY01   SHR,KEEP    > _          SYS2.MSGS.ISA2
 A$SY01   SHR,KEEP    > _          SYS2.MSGS.SYSPLEXA
 $$SRB2   SHR,KEEP    > _          ISP.SISPMENU
 $$SRB2   SHR,KEEP    > _          ISF.SISFMLIB
 F1=Help     F2=Split    F3=Exit     F5=Rfind    F7=Up       F8=Down     F9=Swap
F10=Left    F11=Right   F12=Cancel
```

•ISRDDN displays a list of allocated DDnames and their associated data sets



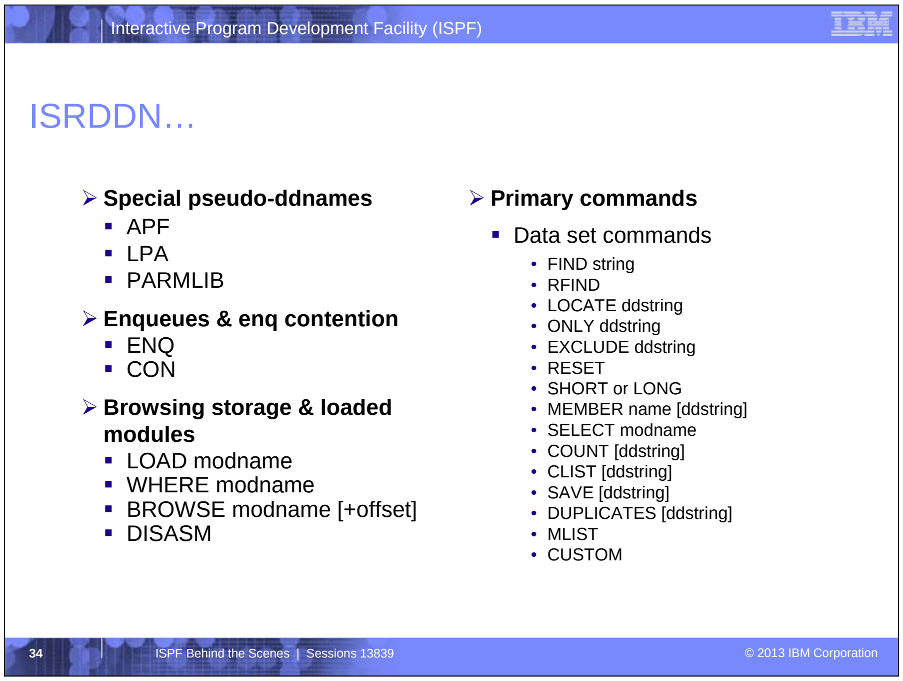
ISRDDN…

- Line commands (actions)
  - E - Edit data set
  - B - Browse data set
  - V - View data set
  - M - Display enhanced member list
  - F - Free the ddname
  - C - Compress a data set
  - Q - Show enqueue information
  - I - Show data set information

ISRDDN has grown to include a number of useful displays that have grown out of the unsupported TASID (Developed by Doug Nadel - previous ISPF developer)

•The available line commands that can be issued against the data set list are:

- •E to edit a data set,
- •B to browse, or
- •V to view.
- •M to display the member list.
- •F to Free the DDname
- •C to compress a data set
- •Q to show data set enqueue information, and
- •I to show data set information

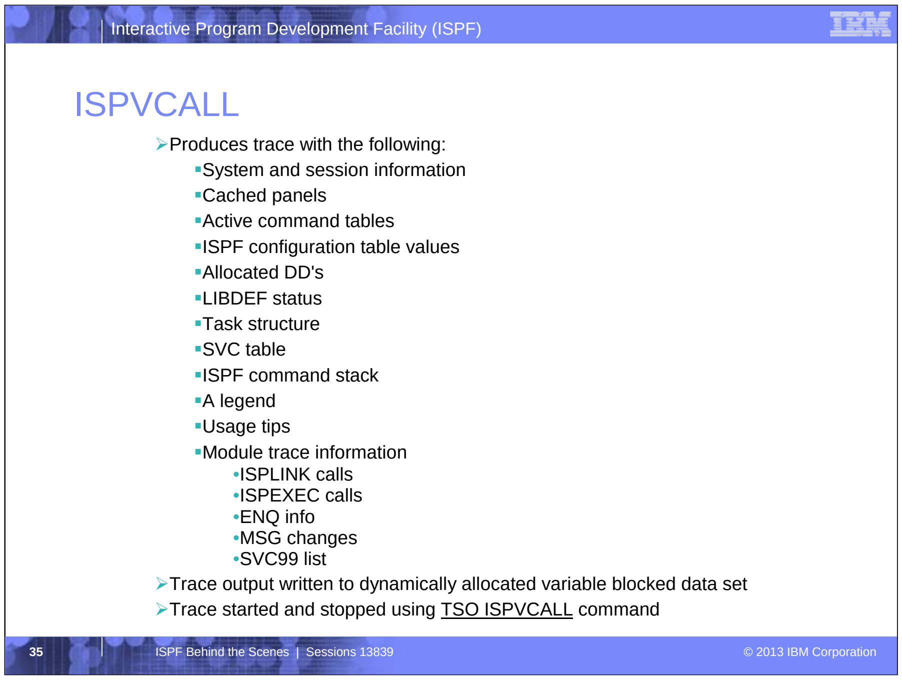


ISRDDN…

- **Special pseudo-ddnames**
  - APF
  - LPA
  - PARMLIB
- **Enqueues & enq contention**
  - ENQ
  - CON
- **Browsing storage & loaded modules**
  - LOAD modname
  - WHERE modname
  - BROWSE modname [+offset]
  - DISASM
- **Primary commands**
  - Data set commands
    - FIND string
    - RFIND
    - LOCATE ddstring
    - ONLY ddstring
    - EXCLUDE ddstring
    - RESET
    - SHORT or LONG
    - MEMBER name [ddstring]
    - SELECT modname
    - COUNT [ddstring]
    - CLIST [ddstring]
    - SAVE [ddstring]
    - DUPLICATES [ddstring]
    - MLIST
    - CUSTOM

•There are a number of Primary commands supported by ISRDDN - some fairly standard ones in Find, Repeat Find, Locate, etc.
•Some of the other more useful commands are:
•Member - Scans the allocated ddnames for a particular member. (Search of ddnames can be restricted by specifying a ddstring parameter - ddstring can be any part of the ddname)
•Select - Search for loaded modules with out searching any allocated data sets. It searches the JPA (Job Pack Area) and LPA (Link Pack Area)
•CList and SAve commands create a CLIST which contains ALLOCATE statements for all the current data set allocations
•DUPlicate - scans all ddnames in the current list and alto the Link Pack Directory for duplicate members
•MList - shows the eyecatchers for some major ISPF and PDF programs
•CUstom - shows the ISPF DEFAULTS and TCM settings and the TSO IKJTSOxx settings
•APF, LPA and Parmlib will add to the displayed list an entry showing the APF, LPA of PARMLIB data sets
•For the purposes of debugging the LOAD, WHERE and Browse commands are very useful
•The browse command has a number of additional subcommands for viewing storage
•These are documented in the ISPF User's Guide Volume 1 (SC34-4822)

For speaker - other descriptions if required -
•FIND - Finds 'string' in the current displayed list
•RFIND - Repeats the last find command
•LOCATE - Locates next ddname containing 'ddstring'
•ONLY - Display ddnames containing ONLY 'ddstring'
•EXCLUDE - Exclude all ddnames containing 'ddstring'
•RESET - Rebuild the displayed list (REFRESH)
•SHORT and LONG - Alter the format of the list. Short places the ddname for a concatenation on the same line; long places it on a separate line
•COUNT - Count the number of members in each Partitioned Data Set
•ENQ - Display data set Enqueues
•CON - Display data set Enqueue Contention



ISPVCALL

- Produces trace with the following:
  - System and session information
  - Cached panels
  - Active command tables
  - ISPF configuration table values
  - Allocated DD's
  - LIBDEF status
  - Task structure
  - SVC table
  - ISPF command stack
  - A legend
  - Usage tips
  - Module trace information
    - ISPLINK calls
    - ISPEXEC calls
    - ENQ info
    - MSG changes
    - SVC99 list
- Trace output written to dynamically allocated variable blocked data set
- Trace started and stopped using TSO ISPVCALL command

•ISPVCALL provides some useful information for debugging ISPF Dialogs.

•ISPVCALL was developed to allow level2 and 3 support to trace internal flow, however it provides users with a lot of extra information.

•ISPVCALL has a number of additional parameters that can be supplied to aid with the tracing. These are provided to a customer by IBM support depending on the nature of a reported problem.

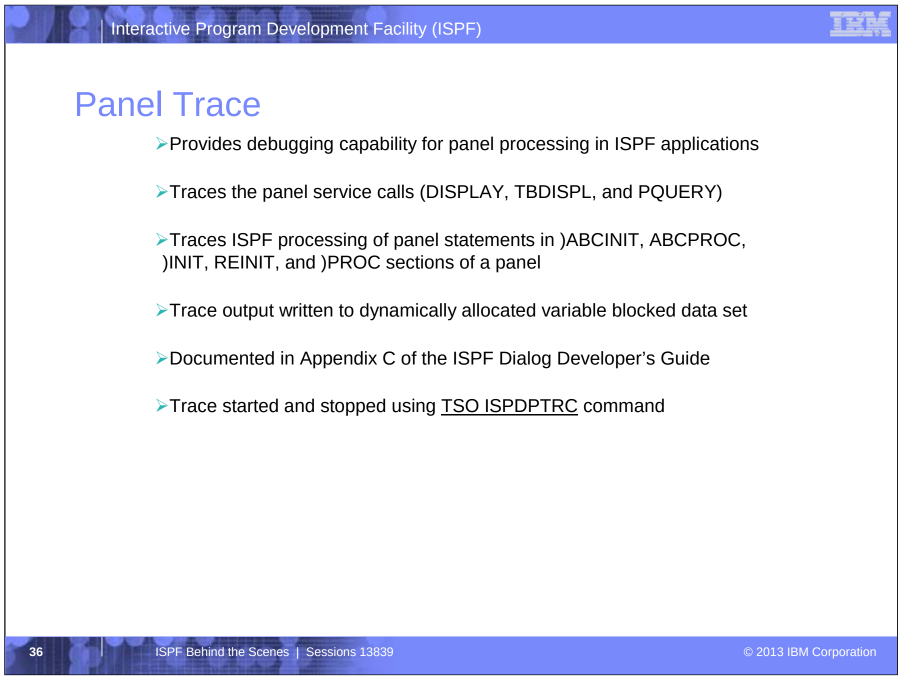
•The standard ISPVCALL will generate a trace data set - named *userid*.ISPVCALL.TRACE

•ISPVCALL is invoked from the command line from any panel (with the command TSO ISPVCALL)

•The first time ISPVCALL is invoked the tracing is started, the next invocation of ISPVCALL will terminate the trace and view the trace data set.

•ISPVCALL provides a ot of useful information, It contains

- •basic System and ISPF information - product versions, system name, etc
- •The names of all panels in the ISPF cache
- •The ISPF configuration options including where the data set from where the options are loaded
- •A list of all the allocated DDnames and there data sets
- •Status of the LIBDEFed data sets
- •The ISPF TASK structure
- •The SVC table,
- •and command stack
- •A legend and some usage tips, and
- •finally the module trace information, showing entry and exit into each module.
- •It also shows the ISPLINK and ISPEXEC calls, including the parameters,
- •shows ENQ information,
- •shows all message information,
- •which can be useful when more than one message is issued , but ISPF only displays the last message issued.
- •and it show the SVC99 - dynamic allocation parameter



Panel Trace

- Provides debugging capability for panel processing in ISPF applications
- Traces the panel service calls (DISPLAY, TBDISPL, and PQUERY)
- Traces ISPF processing of panel statements in)ABCINIT, ABCPROC,)INIT, REINIT, and)PROC sections of a panel
- Trace output written to dynamically allocated variable blocked data set
- Documented in Appendix C of the ISPF Dialog Developer's Guide
- Trace started and stopped using TSO ISPDPTRC command

Support added (z/OS 1.7) to the ISPF Dialog Manager to trace the Panel Service calls (DISPLAY, TBDISPL, and PQUERY) and the processing that occurs within the Dialog Manager Panel code, including the processing of the each statements within the)ABCINIT, ABCPROC,)INIT, REINIT, and)PROC sections of the panel.

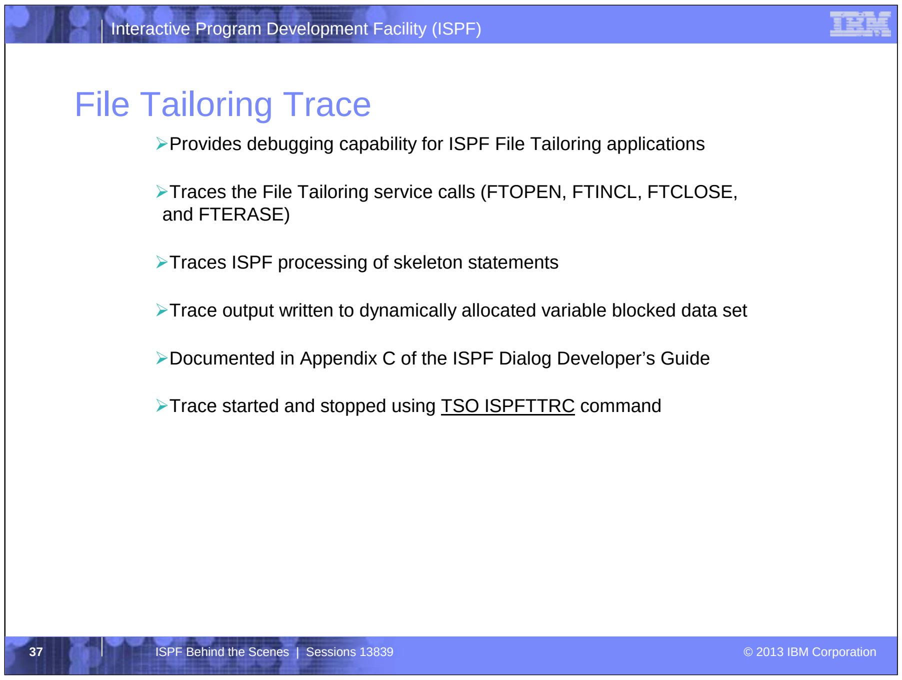
The trace is started and stopped using the new ISPDPTRC command that will either

•start the trace if not active, or

•stop and optionally view or edit the trace output where the trace is active.

The output from the trace is written to a dynamically allocated variable blocked data set that has a record length of 255. If the ddname ISPDPTRC is preallocated this data set will be used, provided it refers to a sequential, variable blocked data set with a record length that is at least 255.

A new appendix in the ISPF User's Guide Volume I describes the new ISPDPTRC command and the contents of the panel trace.



File Tailoring Trace

- Provides debugging capability for ISPF File Tailoring applications
- Traces the File Tailoring service calls (FTOPEN, FTINCL, FTCLOSE, and FTERASE)
- Traces ISPF processing of skeleton statements
- Trace output written to dynamically allocated variable blocked data set
- Documented in Appendix C of the ISPF Dialog Developer's Guide
- Trace started and stopped using TSO ISPFTTRC command

Support added (z/OS 1.7) to ISPF File Tailoring to trace the File Tailoring Service calls (FTOPEN, FTINCL, FTCLOSE, and FTERASE) and the processing that occurs within the File Tailoring code and processing of each skeleton statement.

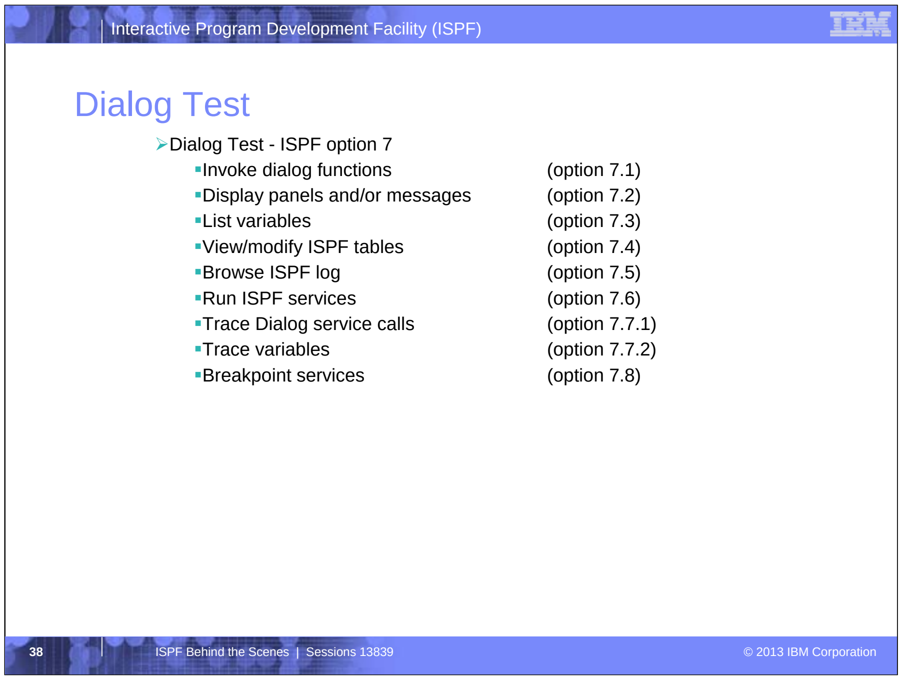
The trace is started and stopped using the new ISPFTTRC command that will either

•start the trace if not active, or

•stop and optionally view or edit the trace output where the trace is active.

The output from the trace is written to a dynamically allocated variable blocked data set that has a record length of 255. If the ddname ISPFTTRC is preallocated this data set will be used, provided it refers to a sequential, variable blocked data set with a record length that is at least 255.

A new appendix in the ISPF User's Guide Volume I describes the new ISPFTTRC command and the contents of the File Tailoring trace.



Dialog Test

- Dialog Test - ISPF option 7

Invoke dialog functions	(option 7.1)
Display panels and/or messages	(option 7.2)
List variables	(option 7.3)
View/modify ISPF tables	(option 7.4)
Browse ISPF log	(option 7.5)
Run ISPF services	(option 7.6)
Trace Dialog service calls	(option 7.7.1)
Trace variables	(option 7.7.2)
Breakpoint services	(option 7.8)

•The ISPF Dialog Test facility is found from option 7 from the ISPF primary menu

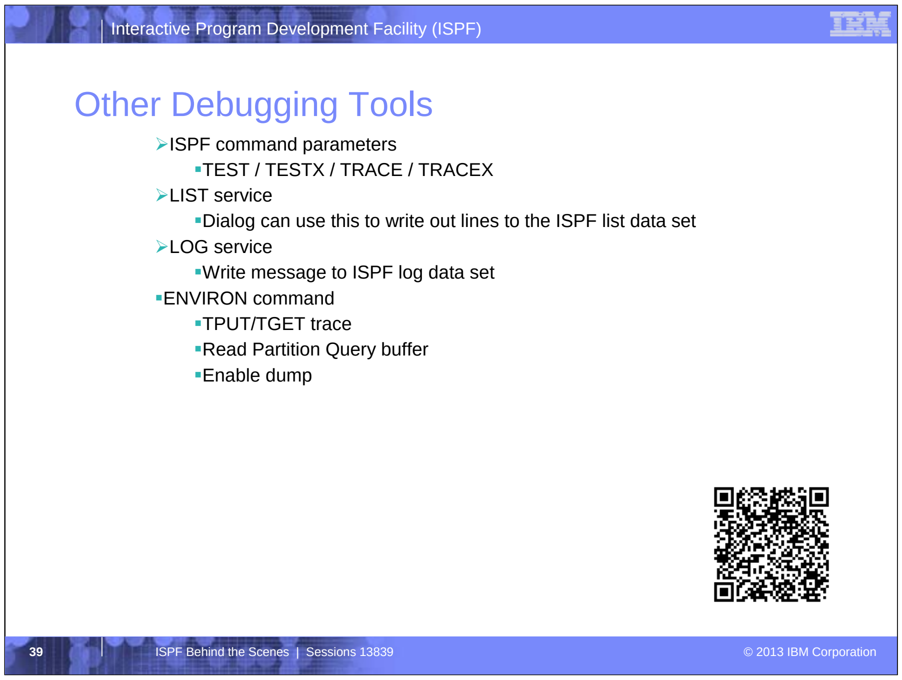
•Most people have used this at some time to display the ISPF dialog variables (option 7.3) or to browse the Log (option 7.5)

•It has many other useful features that allow us to set breakpoints (7.8) and to enable us to define ISPF services (7.7.1) and variables(7.7.2) to be traced.

•We can then use Option 7.1 to invoke our dialog that we want to test

•Dialog test also has options to-

- •Display panels or messages (7.2),
- •View and modify ISPF tables (7.4), and
- •to run any of the ISPF services (7.6)



Other Debugging Tools

- ISPF command parameters
  - TEST / TESTX / TRACE / TRACEX
- LIST service
  - Dialog can use this to write out lines to the ISPF list data set
- LOG service
  - Write message to ISPF log data set
- ENVIRON command
  - TPUT/TGET trace
  - Read Partition Query buffer
  - Enable dump

Other ways of debugging ISPF dialogs:

•The TEST or TESTX invokes ISPF in Test mode

•panels, messages not cached

•tutorial panels show current panel ID, previous panel ID, and previous msg ID

•screen printouts show line numbers, current panel name, and message ID

•Option 3.1 index listing for PDS shows TTR for each member

•Dialog operating in CANCEL error mode (default) panel displayed on an error allows user to force the dialog to continue

•ISPF-detected error, ABEND, or program interrupt of SELECTed command causes ISPF to ABEND

•PA1 causes immediate exit from ISPF

•With TRACE or TRACEX all ISPF service calls are logged

•in both cases the X is an extended mode that displays all logged messages on the terminal.

•The ISPF LIST service allows a dialog to write information to the ISPF LIST data set,
and the ISPF LOG service write information to the ISPF LOG.

•The ENVIRON command traces

•all terminal input and output data done by the TSO TPUT, TGET, and PUTLINE services

•**NOTE**: ISPSNAP or other dd as specified has DCB: VBA lrecl=125, blksize=1632 - This is a snap restriction

•Query terminal characteristics

•Can enable a dump for a subtask abend