ISPF Design Coding
Hints and Tips

Lionel B. Dyck
Kaiser Permanente Information Technology
Session 2646
August 22, 2002   1:30pm
e-mail: Lionel.B.Dyck@kp.org

Welcome to Session S2646. This session has been given for a number of
years and to provide fresh material I have created an application
specifically to demonstrate the coding hints and tips by example.

Objectives

- This session will present ISPF dialog coding techniques utilizing the REXX programming language. ISPF features presented will include: LMDLIST, ISPF Tables (including point-and-shoot and free form text find), BROWSE, EDIT, and VIEW. Use of Field Level Help in panels will be discussed along with popup panels for prompting.

- This session counts towards the ISPF certificate program.

Our objective.

Overview

- Simple ISPF Dialog
- Process a List of Datasets Based on HLQ
- ISPF Panels – 5
- REXX Execs – 1
- Messages – 1
  - Use IBM Provided Message
- Code Available From
  - http://www.lbdsoftware.com/s2646.zip

To demonstrate the coding techniques today I will utilize a simple ISPF dialog that I developed for this session. It utilizes all the basic elements of dialog coding and consists of 5 ISPF Panels that are driven by 1 REXX Exec and uses 1 ISPF Message (the message is an IBM provided message).

The code that you will see today is being made available on my web site at http://www.lbdsoftware.com using the direct link shown in the slide. You can find many real applications on my web site as well, including the popular XMITIP for sending e-mail from the mainframe.

Installation

- Download S2646.ZIP
- Unzip
- Binary upload share.xmit file to OS/390 or z/OS into sequential file with RECFM=FB LRECL=80
- Issue RECEIVE INDS(upload.dsn)
- Review contents of created dataset

To install follow these simple instructions.

- Read $DOC member
- Copy $SHARE member into your EXEC Library
- Invoke:
  - TSO %$SHARE

  or

  - TSO %$SHARE hlq

To use this sample dialog read the $DOC member of the PDS.

Copy member $SHARE into a library in your SYSPROC or SYSEXEC concatenation. Then copy the other members (as noted in the $DOC) into your ISPF Panels Library (ISPPLIB).

Coding

SHARE

- ISPF Panels
- OS/390 REXX

- Use ISPF Edit Models

The coding of this application consisted of creating some ISPF Panels
with the driver application written in REXX. The use of the ISPF Edit
Models aided in the creation of the REXX code and the Panels.

While in ISPF Edit the model selection list can be displayed by entering
the command **model** on the ISPF Edit command line.

If you know the model you want to use then enter the command model
service (e.g. **model vget**).

ISPF Edit Models

```
                            REXX Models
Option ===>

Enter number or service name.
Enter END command to cancel MODEL command.

Variables              Workstation            Library Access
V1  VGET               X1  FILESTAT           L0  LIBACC
V2  VPUT               X2  FILEXFER
V3  VERASE             X3  WSCON              Miscellaneous
                       X4  WSDISCON           M1  SELECT
Display                                       M2  CONTROL
D1  DISPLAY                                   M3  BROWSE
D2  TBDISPL            File Tailoring         M4  EDIT
D3  SETMSG             F1  FTOPEN             M5  LOG
D4  PQUERY             F2  FTINCL             M6  GETMSG
D5  ADDPOP             F3  FTCLOSE            M7  EDREC
D6  REMPOP             F4  FTERASE            M8  LIBDEF
                                              M9  LIST
                                              M10 VIEW
Tables                                        M14 QLIBDEF
T1  TABLES                                    M15 QBASELIB
                                              M16 DSINFO


August 22, 2002        ISPF Design and Coding Hints              7
                              and Tips
```

By entering the word MODEL on the ISPF Edit command line this
selection panel appears.

ISPF Edit Model: VGET

SHARE

```
EDIT      SYSLBD.LIONEL.EXEC(TESTPAN) - 01.00
Command ===>
****** ***************************************************** Top of Data *********
000100   'VGET' namelist 'ASIS'
=NOTE=
=NOTE=     namelist   - Names of one or more variables to be copied.
=NOTE=     Choose one to indicate the source of the copied variables:
=NOTE=        ASIS     - Default, variables are to be copied from the shared
=NOTE=                   pool or, if not found there, from the profile pool.
=NOTE=        SHARED   - Variables are to be copied from the shared pool.
=NOTE=        PROFILE  - Variables are to be copied from the profile pool.
=NOTE=
=NOTE=     EXAMPLE:  ADDRESS ISPEXEC
=NOTE=                 'VGET (N1 N2 N3) PROFILE'
=NOTE=
000200   If rc ¬= 0 Then              /* Return codes                */
000300     Do                         /* 8  - Variable not found     */
000400     End                        /* 16 - Translation error or data */
000500   Else                         /*      truncation             */
000600                                /* 20 - Severe error           */
****** ***************************************************** Bottom of Data *******
```
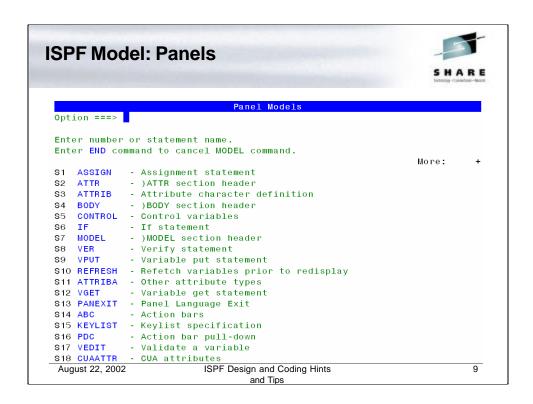
This is an example of the MODEL code generated for the VGET service.

ISPF Model: Panels

```
                                Panel Models
Option ===>

  Enter number or statement name.
  Enter END command to cancel MODEL command.
                                                      More:      +
  S1   ASSIGN   - Assignment statement
  S2   ATTR     - )ATTR section header
  S3   ATTRIB   - Attribute character definition
  S4   BODY     - )BODY section header
  S5   CONTROL  - Control variables
  S6   IF       - If statement
  S7   MODEL    - )MODEL section header
  S8   VER      - Verify statement
  S9   VPUT     - Variable put statement
  S10 REFRESH   - Refetch variables prior to redisplay
  S11 ATTRIBA   - Other attribute types
  S12 VGET      - Variable get statement
  S13 PANEXIT   - Panel Language Exit
  S14 ABC       - Action bars
  S15 KEYLIST   - Keylist specification
  S16 PDC       - Action bar pull-down
  S17 VEDIT     - Validate a variable
  S18 CUAATTR   - CUA attributes
```

Models for ISPF Panels.

This is an example of the ISPF panel model ATTR.

```
/* rexx */
address tso
'altlib activate application(exec)' , 'dataset(share.pds)'
address ispexec
'libdef ispplib dataset id(share.pds) stack'
'select cmd(%share) newappl(isr) passlib'
address tso
'altlib deactivate application(exex)'
address ispexec
'libdef ispplib'
exit 0
```

This sample REXX Exec can not only be used to test this sample
application but also used to demonstrate how to dynamically access
ISPF application libraries such that you do not have to install all the
elements (execs, clists, panels, etc.) into system level libraries.

The ALTLIB is used to define a dynamic addition to the current
SYSEXEC concatenation where REXX programs reside.

The Address statement is used to identify the environment in which the
subsequent commands are to execute in. ALTLIB is a TSO command
while LIBDEF and SELECT are ISPF commands and require the
ISPEXEC environment.

The LIBDEF is an ISPF service for dynamically allocating a temporary
ISPF library, in this case a PANEL library.

The SELECT service is used to invoke a ISPF Service, in this case a
REXX Exec.

```
/* REXX */

arg options

Address ISPExec
"Vget (Zapplid)"
if zapplid <> "SHAR" then do
   "TBCreate sharcmds names(zctverb zcttrunc zctact ,
   zctdesc)",
   "replace share nowrite"
   zctverb = "RFIND"
   zcttrunc = 0
   zctact = "&SHARFIND"
   zctdesc = "RFIND for SHARE Dialog"
   "TBAdd sharcmds"
   "Select CMD(%"sysvar('sysicmd') options ") ,
   "Newappl(SHAR)",
   "Passlib SCRName(SHAREDLG)"
   "TBEnd sharcmds"
   exit 0
end
```

The example code that you will see on the slides is mostly without comments to conserve slide space. For the comments see the actual exec.

All REXX Programs, also called EXECs, should start with a comment, which must contain the word REXX, so that the EXEC can be installed in either a SYSPROC or SYSEXEC concatenation.

The next statement acquires any options passed with the command. The 'arg options' will retrieve all the passed parameters and place them in the REXX variable options, after translating the parameters to upper case. To avoid the upper case translation use 'parse arg options'.

The next section of code is something that I like to use with my dialog applications. It makes sure that the application is executed under the correct ISPF Application ID.

This code:

1. VGETs the current ISPF Application ID

2. Tests to see if it is SHAR and if it is not the following code executes, otherwise the do/end falls through to the mainline code.

3. Otherwise the code creates an ISPF Commands Table, SHARCMDS, as a temporary table (nowrite) and creates a table row for a Repeat Find (RFIND) command that will be used within this dialog for searching a table of datasets.

4. The REXX Exec is then recursively executed using the ISPF Select service using the original command name, sysvar('sysicmd'), and parameters while specifying the desired ISPF Application ID using the NEWAPPL keyword. The PASSLIB is used to tell the SELECT service to use any currently allocated ISPF Libraries (LIBDEF). The Screen name is also specified (SCRNAME) so that when using a SWAP LIST the application will display with the name we specified.

5. The TBEND occurs after the recursive execution completes and closes the temporary ISPF Commands Table.

6. The 'exit 0' statement terminates the execution.

```
/* -------------------- *
 * Define Default Values *
 * -------------------- */
parse value '' with null dsn
zerralrm = "NO"
table_name = "SHARE"random(999)
zerrhm = "sharehlp"
```

Default values must be set and these are set in this application at the
beginning of the mainline code.

In this case two variables are set to null, the variable named **null** and the
dsn variable.

The **zerralrm** is set to **NO** – could be set to **YES** if you want to hear a
beep when a message is issued.

The **zerrhm** variable is set to the help panel for the application so if the
user presses **F1** (Help) when they see the messages, the first **F1** will
show the long message and the second **F1** will display this panel.

The table name is also set using a random number.

```
/* --------------------------------- *
 * Display a prompting panel for the HLQ *
 * --------------------------------- */
if length(options) = 0 then do forever
   "Display Panel(Sharep)"
   options = hlq
   if zcmd = "CANCEL" then exit 4
   if rc = 0 then
      if hlq <> null
         then leave
   if rc > 0 then
      if hlq = null
         then exit 4
   end
```

This code will test for any passed parameters, which in our case would be a high-level-qualifier or hlq. If there are none then display the panel.

The Display service is used to display the prompting panel. The HLQ field from the panel is placed into the options variable for use in our code.

A test is made to see if the user entered the word 'CANCEL' in the command field. Note that the command field is defined as upper case so the test must be for an upper case word. If CANCEL then the dialog is ended with the exit statement.

Next the return code variable (rc) is tested and if zero (0) a test is made for an hlq and if there is one then this 'do forever' loop is ended with the leave statement.

If that isn't the case then the return code is again tested for greater than zero indicating a PF3 or PF4 was entered, a test is made for hlq being null and if null then the dialog is ended with the exit.

Initial Panel

```
-------------------------- SHARE Dialog Example ------------------------
Command ===>

Enter a valid high level qualifier (1 to n levels):

HLQ:===>

   This is a sample ISPF Dialog for demonstration purposes. Press F1
   for a full tutorial or move the cursor to the HLQ entry field for
   field level help.

   Press Enter after entering a valid HLQ or Use the PF3 with no HLQ to Quit.
```

August 22, 2002   ISPF Design and Coding Hints and Tips   15

This is the initial panel. The user would enter the desired HLQ to process and the press Enter to register it and proceed. To quit the process the user would enter the word CANCEL in the Command field or PF3 with no HLQ.

A tutorial is available via the standard PF1 key.

```
)attr default(%+_)
  % type(text) intens(high)
  ~ type(text) intens(high) caps(off) just(asis )color(turq)
  + type(text) color(turq)
```

The first part of the panel is the Attributes section. The statement)**attr**
defines the section. An attribute is defined on the panel using a special
character. In this case I am using the %, ~, and + characters. The
default attributes of %, + and _ are referenced but we are changing the
attributes for the % and + symbols.

```
)Body Expand(\\)
%-\-\- ~SHARE Dialog Example%-\-\-
%Command ===>_zcmd
+
+Enter a valid high level qualifier (1 to n levels):
+
%HLQ:===>_hlq
+
   This is a sample ISPF Dialog for demonstration purposes. Press F1
   for a full tutorial or move the cursor to the HLQ entry field for
   field level help.
+
+ Press%Enter+after entering a valid HLQ or Use the%PF3+with no HLQ to Quit
```

The **)Body** statement includes the **Expand(\\)** field. The Expand is used to specify the characters used for delimiting the expansion and is used for panels where you want the text centered or aligned. The panel text is then coded using the attribute characters to define the attributes (color, intensity, etc.) for the text.

```
)Init
 .help = sharehlp
 .cursor = hlq
)Proc
 if (&zcmd EQ &z)
     ver (&hlq,nb,dsname)
)Help
 Field(hlq)   Panel(sharehh)
)End
```

After the)BODY statement is the)INIT which in this example defines the ISPF Help Panel for this Panel. The help panel is named SHAREHLP. The cursor is also positioned to the HLQ input field.

The)PROC is the process section.

A test is made for the command field to verify that it is equal to &Z, which is the ISPF Variable for null. If null then the HLQ variable is verified to be non-blank (nb) and to conform to the syntax of a dataset name.

The)HELP section is for field level help. The FIELD keyword defines the input field on the panel to which the field level help applies. The PANEL keyword defines the ISPF Help Panel.

The)END statement ends the panel definition statements.

Field Level Help Panel

```
----------------------- SHARE Dialog Example -----------------------
Command ===>

Enter a valid high level qualifier (1 to n levels):

HLQ:===>

 This  | -------------- SHARE HLQ Specification --------------
 for a |
 field | Enter a valid high level qualifier of 1 to n levels.
       |
 Press | e.g. SYS1                                              to Quit.
       |      SYS1.ABC
       |
       |
```

This is a field level help panel being displayed. Notice that it is a popup
and situated below and to the right of the field to which it is associated.

```
)attr default(%+_)
` type(text) intens(high) caps(off) just(asis ) color(yellow)
~ type(text) intens(high) caps(off) just(asis ) color(turq)
+ type(text) color(turq)
% type(text) intens(high)
)Body Window(55,8) Expand(\\)
%-\-\- ~SHARE HLQ Specification%-\-\-
+
+Enter a valid high level qualifier of 1 to n levels.
`
+e.g.`SYS1
`     SYS1.ABC
)Init
)Proc
    &zup = sharehlp
)End
```

This is an example of a Field Level Help Panel. It is no different from any other ISPF Panel, which should be no surprise.

In this example the)BODY statement uses a WINDOW keyword to define the area of the panel to 55 characters wide and 8 lines deep.

We've already seen the EXPAND keyword.

Primary Help Panel

```
------------------------  SHARE Sample ISPF Dialog  ------------------------
Selection ===> █

  This ISPF Dialog provides a sample ISPF application for demonstration
  purposes. It is intended to be used as a learning vehicle to learn some
  of the many ISPF services along with useful REXX coding techniques.

  This dialog presents a list of datasets based on a supplied high level
  qualifier.  From this list the user may Browse, Edit, Migrate, Recall
  or View the dataset.

  The following topics are presented in sequence, or may be selected by
  number:

   1  High Level Qualifier specification
   2  Dataset Selection
```

The primary help panel is shown here. It has two selections which are displayed automatically by just pressing the ENTER key or the selection may be made manually by entering 1 or 2.

Primary Help Panel
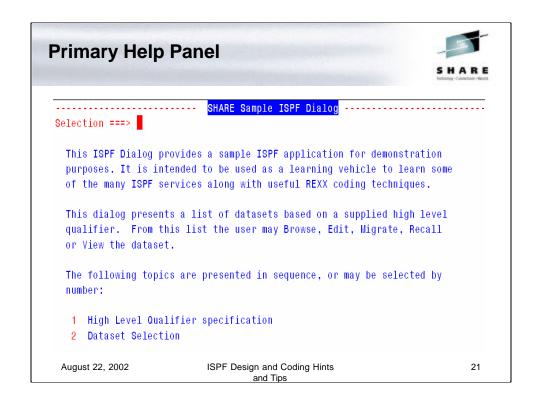
```
)attr default(%+_)
` type(text) intens(high) caps(off) just(asis ) color(turq)
~ type(text) intens(high) caps(off) just(asis ) color(turq) hilite(reverse)
% type(text) intens(high) color(red)
+ type(text) color(turq)
)Body Expand(\\)
%-\-\- ~SHARE Sample ISPF Dialog%-\-\-
%Selection ===>_ZCMD                                                          +
%
`   This ISPF Dialog provides a sample ISPF application for demonstration
`   purposes. It is intended to be used as a learning vehicle to learn some
`   of the many ISPF services along with useful REXX coding techniques.
`
`   This dialog presents a list of datasets based on a supplied high level
`   qualifier.  From this list the user may Browse, Edit, Migrate, Recall
`   or View the dataset.
`
`   The following topics are presented in sequence, or may be selected by
`   number:
`
   %1+ High Level Qualifier specification
   %2+ Dataset Selection
+
)Proc
     &zsel = trans( &zcmd
                1,sharehh
                2,shared1
                *,'?'
                )
     &zup = sharehlp
)end
```

Here is the code for the primary help panel for this application.

```
/* ------------------------------------ *
 * Do Library Dataset list initialization *
 * ------------------------------------ */
"Lmdinit Listid(LMD) Level("options")"


/* ------------------------------ *
 * Create the temporary ISPF Table *
 * ------------------------------ */
"TBCreate" table_name "keys(dsn)" ,
   "names(act zdlcdate zdlvol)" ,
   "Share Replace Nowrite"
```

The **LMDINIT** is used to define a dataset list ID for a Level or a dataset.
In this case it is for a dataset Level. The value in the **LISTID** is a literal
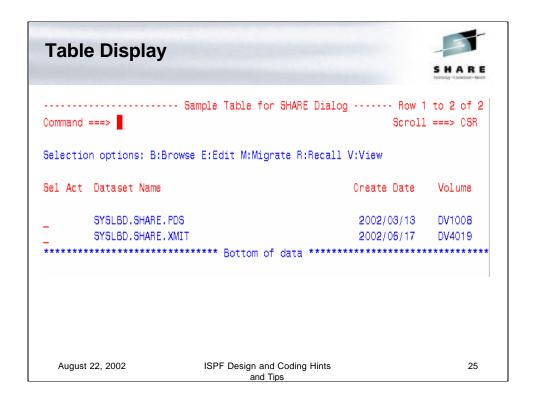in this statement and is referenced later as a variable.

The **TBCREATE** is the Table Create service. It is used to create an
ISPF Table. The **keys** field defines any variable key names to be used
for the table with the **names** field defining the other, non-key, variables
that will be stored in the table rows. The **Nowrite** indicates that this table
is to be kept in memory and not written to a table library.

```
/* ------------------------------------------------------ *
 * Now loop thru the LMDList for each dataset and add the *
 * results to the temporary ISPF table.                   *
 * ------------------------------------------------------ */
do forever
   "Lmdlist Listid("lmd") Stats(YES) Dataset(dsn)" ,
           "Option(LIST)"
   trc = rc
   if trc > 0 then do
      "Lmdfree listid("lmd")"
      leave
      end
   "TBadd" table_name
end
```

The **LMDLIST** service will return the datasets for a high level qualifier, one at a time, starting with the first call which is just the hlq. Each subsequent call will return the next dataset in the list.

The return code is tested and if non-zero the **LMDFREE** is issued to close out that service and the **leave** statement terminates the **do forever** loop.

If the return code is zero, meaning a successful **LMDLIST**, then the information is added to the ISPF table we created on the prior slide using the **TBADD** service.

Page24

Table Display

```
---------------------- Sample Table for SHARE Dialog ------- Row 1 to 2 of 2
Command ===> █                                              Scroll ===> CSR

Selection options: B:Browse E:Edit M:Migrate R:Recall V:View

Sel Act  Dataset Name                         Create Date    Volume

_        SYSLBD.SHARE.PDS                     2002/03/13    DV1008
_        SYSLBD.SHARE.XMIT                    2002/05/17    DV4019
***************************** Bottom of data ******************************
```

This is the table display.

The Sel column is where the selection option is entered. The Act field is updated by the application to indicate the last action performed on a dataset.

```
    )Attr Default(%+_)
       ! type( input) intens(high) caps(on ) just(left ) pad('_')
       ¬ type(output) intens(low ) caps(off) just(left )
       + type(text) color(turq)
    )Body  Expand(//)
    %-/-/- Sample Table for SHARE Dialog -/-/-
    %Command ===>_zcmd                                 / /%Scroll ===>_amt +
    %
    +Selection options: B:Browse E:Edit M:Migrate R:Recall V:View

    %Sel Act  Dataset Name                             Create Date    Volume
    +
    )Model
    !z+    ¬z+¬z                                       ¬z          +  ¬z
    )Init
      .help = SHAREHLP
      .zvars = '(sel act dsn zdlcdate zdlvol)'
      &amt = csr
    )Reinit
    )Proc
     ver (&sel,list,B,E,M,R,V)
     if (&ztdsels = 0000)
        &row = .csrrow
        if (&row NE 0)
            if (&sel = &z)
                &sel = B
     if (&ztdsels NE 0000)
        &row = &z
    )Help
     Field(sel)   Panel(shared1)
    )End
```

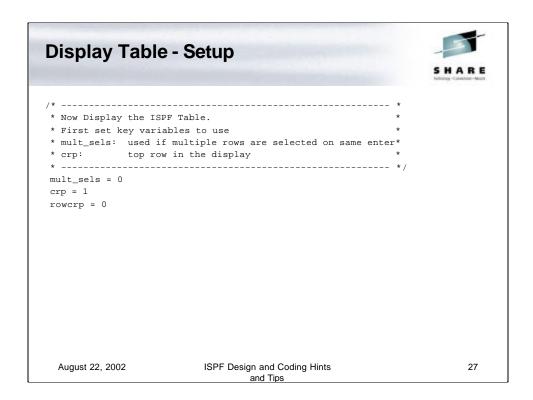This is the panel code for the table display.

The new thing here is the)**MODEL** statement and the **z** variables.

The)**MODEL** indicates that the next statement(s) are used to display the rows of the table. The **z** variables are defined in the)**INIT** section using the **.zvars** statement.

In the)**PROC** section the **ver** statement is used to limit the values the user can use in the **sel** field.

To support point-and-shoot for row selection the **ztdsels** is tested for 0000 and if so then the **row** is set to the **.csrrow** value. If **row** is not 0 then the **sel** variable is tested and if null (**&z**) then the **sel** value is set to **B** for Browse

If the **ztdsels** is not equal to 0000 then the **row** value is set to null (**&z**).

Display Table - Setup

SHARE

```
/* --------------------------------------------------------- *
 * Now Display the ISPF Table.                               *
 * First set key variables to use                            *
 * mult_sels:  used if multiple rows are selected on same enter*
 * crp:        top row in the display                        *
 * --------------------------------------------------------- */
mult_sels = 0
crp = 1
rowcrp = 0
```

This routine sets some default values for our table display routine.

The **mult_sels** is used to contain the number of rows selected. The **crp** is the current top row pointer and the **rowcrp** is the cursor position row value.

```
disp:
do forever
   sharfind = "PASSTHRU"
   sel = null
   if mult_sels = 0 then do
     "TBTop" table_name
     "TBSkip" table_name "Number("crp") "
     "TBDispl" table_name "Panel(sharetbl)" ,
               "Csrrow("rowcrp") AutoSel(No)"
     end
   else
     "TBDispl" table_name
   t_rc = rc
```

This is the table display routine which is a **do forever** loop.

The first thing is to set the **sharfind** variable to PASSTHRU. This variable was defined in the command table we created when we started this exec. **PASSTHRU** informs ISPF to pass the **RFIND** (repeat find) command to this application and not to attempt to process it. This is done so that we can process the RFIND commands.

Next the **sel** variable is set to null. This is the variable used to indicate the selection options for each row.

Next the **mult_sels** is tested and if zero then we position the table by using the **TBTOP** to go to the very top of the table, then the **TBSKIP** to skip down to the last row that was referenced. The **TBDISPL** service is then called to display the table.

If the **mult_sels** was not zero then the **TBDISPL** service is called with just the table name. This allows the additional rows to be processed.

```
crp = ztdtop
mult_sels = ztdsels
if row <> null then
   if row > 0 then do
      "TBTop" table_name
      "TBSkip" table_name "NUMBER("row")"
      end
if t_rc > 7 then do
   "TBEnd" table_name
    return
    end
sharfind = null
call do_it
```

After the table is displayed we have to process any actions.

First the top row value (**ztdtop**) is saved in the **crp** variable. Then the number of rows selected (**ztdsels**) is saved in **mult_sels**.

Then the **row** variable is tested. If not null, then if greater than zero, then we find the selected row by going to the top using **TBTOP** and then skipping down to the selected row using **TBSKIP**. This gets the information from the row for our use.

If the return code is greater than 7 then the table is closed using **TBEND** and the routine returns to the caller, which in this case is the ISPF **Select** service.

Next the **sharfind** variable is set to null and the **do_it** routine is called.

```
/* --------------------------------------------- *
  * Place selection into act field and update the row *
  * --------------------------------------------- */
  act = sel
  "TBPut" table_name
end
```

After processing the row the **act** variable is set from the last action requested (**sel**), and the row in the table is updated using the **TBPut** service. The **do forever** loop then continues.

Page30

```
Do_It:
  "Control Display Save"
  Select
    When abbrev("FIND",word(zcmd,1),1)  = 1
         then call do_find
    When abbrev("RFIND",word(zcmd,1),1) = 1
         then call do_find
    When sel = "B" then "Browse Dataset('"dsn"')"
    When sel = "E" then "Edit   Dataset('"dsn"')"
    When sel = "M" then do
         Address TSO,
              "Hmig '"dsn"'"
         zdlvol = "MIGRAT"
         end
```

Once a selection is made the selection needs to be analyzed and
processed. This routine (this slide and the next 2) do that.

The **"Control Display Save"** is an ISPF service that saves the current
display environment. This is required in case other displays are done by
the selection action (e.g. Browse).

A REXX Select/When/Otherwise/End clause is used to test the user
selection against the supported selections.

Notice the Browse selection. This is how to invoke the ISPF Browse
service.

If calling a native TSO service then the **Address TSO** statement needs
to be used, as we do for the **HMigrate** action.

```
 When sel = "R" then do
     if zdlvol <> "MIGRAT" then do
          "Control Display Restore"
          sel = null
           zerrsm = "Error"
           zerrlm = "HRecall is not valid for a non",
                    "migrated dataset."
          "Setmsg Msg(isrz002)"
          return
        end
     Address TSO,
         "Hrecall '"dsn"'"
     end
 When sel = "V" then "View  Dataset('"dsn"')"
 otherwise do
          "Control Display Restore"
          rowcrp = 0
          return
          end
   end
 "Control Display Restore"
```

The **R** (HRecall) option requires that we verify that the volser is
MIGRAT and if it isn't then we tell the user via the **SETMSG** service.

The Otherwise clause does the **"Control Display Restore"** and then
returns to the display routine.

If the Otherwise clause is not processed, because there was a valid
selection option, then the **"Control Display Restore"** is issued after the
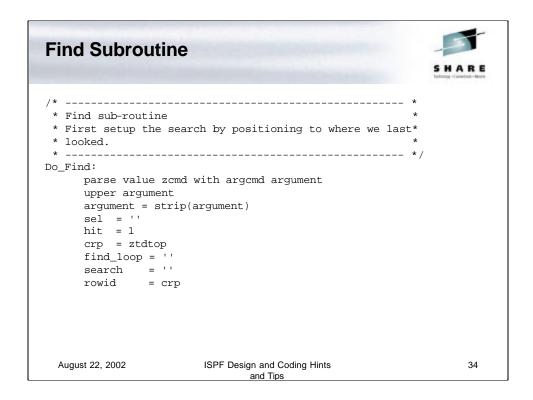Select/When/Otherwise/End falls through.

```
/* ------------------------------------------------------- *
 * Test if original volume was MIGRAT and if not then return *
 * else use LISTDSI to get current volser and creation date. *
 * ------------------------------------------------------- *
 Select
   When length(sel) = 0 then return
   When pos(sel,"BVER") = 0 then return
   When zdlvol <> "MIGRAT" then return
   Otherwise nop
   end
 call listdsi "'"dsn"'"
if sysvolume = "MIGRAT" then return
 parse var syscreate year"/"day
 jdate = right(year,2)day
 sdate = date('s',jdate,'j')
 zdlcdate = left(sdate,4)"/"substr(sdate,5,2)"/"right(sdate,2)
 zdlvol   = sysvolume
 return
```

The last action after processing a valid selection action is to test the volser to determine if the table row value needs to be updated.

First test to see if the selection action character is BVER and if not then return.

Then test the volser for MIGRAT and if not then return.

If we haven't returned then use the **listdsi** REXX function to get the current volser and creation date and update the table row variables with these values. Remember that the return to the display routine will update the row with the last action along with these updates using the **TBPUT**.

Find Subroutine

```
/* -------------------------------------------------- *
 * Find sub-routine                                   *
 * First setup the search by positioning to where we last*
 * looked.                                            *
 * -------------------------------------------------- */
Do_Find:
     parse value zcmd with argcmd argument
     upper argument
     argument = strip(argument)
     sel  = ''
     hit  = 1
     crp  = ztdtop
     find_loop = ''
     search    = ''
     rowid     = crp
```

This is the local, non-ISPF, FIND routine. This differs from the standard ISPF routine in that you can tailor it to look in any field in the table and anywhere in a field.

This routine takes two parameters. FIND or RFIND and the search value.

The search value is translated to upper case and leading and trailing blanks are removed.

Then some default values are set.

```
if argcmd = "RFIND" then do
   argument = save_arg
   last_find = last_find + 1
   "TBTOP " table_name
   "TBSKIP" table_name "Position(ROWID)" ,
           "Number("Last_find")"
   end
   else do
       if rowid > 1 then
       "TBSKIP" table_name "Position(rowid)"
       end
```

The **argcmd** option is tested for **RFIND** and if so then we set up for the Repeat Find by:

- setting **argument** to the last used search value

- incrementing the **last_find** (row number) by 1

- positioning to the new row (**last_find**) by using **TBTOP** and then **TBSKIP** services

If the **argcmd** is not **RFIND** (it must be **FIND**) we position down 1 row for the search test.

```
if rc = 8 then do
              "TBTop" table_name
              "TBSKIP" table_name "Position(ROWID)"
              s_smsg = "Wrapped"
              end
         else s_smsg = "Found"
```

If the **TBSKIP** return code is an 8 then we have reached the end of the
table so we need to start over at the top. To do this we use the **TBTOP**
service followed by the **TBSKIP** and set the short message to
'wrapped'.

If the return from **TBSKIP** was not 8 then the short message is set to
'found'.

```
     /* --------------------- *
      * Now perform the Search *
      * --------------------- */
     save_arg = argument
     do forever
        search = dsn zdlvol
        if pos(argument,search) > 0 then do
           crp = rowid + 0
           rowcrp = crp
           last_find = crp
               zerrsm = s_smsg  /* "Found" */
               zerrlm = argument "found during search in row:" crp
              "Setmsg Msg(isrz002)"
           leave
           end
```

Now we do the actual find by using the **pos** function to determine if the search argument is in the search fields.

The search fields are set by setting the **search** variable to contain the row values we want to search – **dsn** and **zdlvol** in this case.

If so then we have a match and we set the short message variable **zerrsm** to the short message value (**s_smsg**). The long message variable **zerrlm** is set to the search string plus **'found during search in row:'** and the row number. The routine then leaves the compare loop which falls though to the return statement on the next slide.
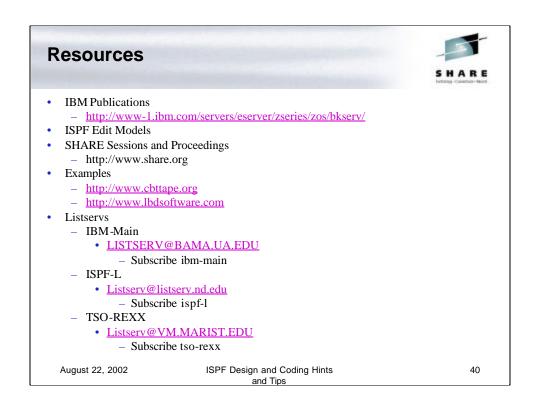
```
"TBSKIP" table_name "POSITION(Rowid)"
if rc = 8 then do
     "TBTOP" table_name
      s_smsg = "Wrapped"
   if find_loop = "on" then do
      zerrsm = "Not Found"
      zerrlm = argument "Not found during search"
      rowid = crp
      "Setmsg Msg(isrz002)"
      leave
      end
      else find_loop = "on"
   end
sel = ''
end
return
```

If the compare fails then the **TBSKIP** service is used to jump to the next row. If the return code from the **TBSKIP** is 8 then we start again at the top of the table.
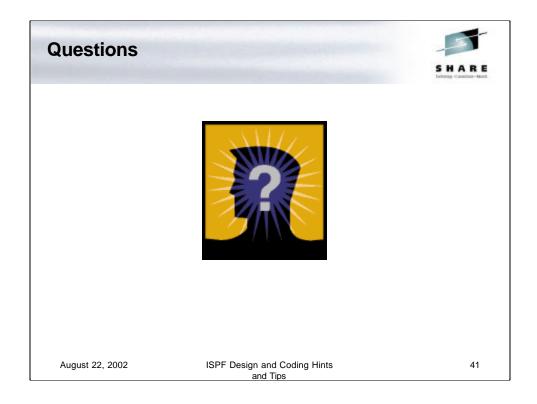
The **find_loop** variable is tested to determine if we have already been here (at the end of the table) before and if so informs the users via the short and long message that the search value could not be found and the **leave** statement ends the search loop.

```
ISRZ000  '&ZEDSMSG                  ' .ALARM = NO .HELP = ISR2MACR NOKANA
'&ZEDLMSG'
ISRZ001 '&ZEDSMSG' .ALARM = YES .HELP = ISR2MACR NOKANA
'&ZEDLMSG'
ISRZ002 '&ZERRSM' .ALARM = &ZERRALRM .HELP = &ZERRHM NOKANA
'&ZERRLM'
ISRZ003 '&ZERRSM' .A=&ZERRALRM .H=&ZERRHM .T=&ZERRTP .W=&ZERRWN NOKANA
'&ZERRLM'
```

IBM has provides some generalized messages that can be used. These are ISRZ000 to ISRZ003.

Resources

- IBM Publications
  - http://www-1.ibm.com/servers/eserver/zseries/zos/bkserv/
- ISPF Edit Models
- SHARE Sessions and Proceedings
  - http://www.share.org
- Examples
  - http://www.cbttape.org
  - http://www.lbdsoftware.com
- Listservs
  - IBM-Main
    - LISTSERV@BAMA.UA.EDU
      - Subscribe ibm-main
  - ISPF-L
    - Listserv@listserv.nd.edu
      - Subscribe ispf-l
  - TSO-REXX
    - Listserv@VM.MARIST.EDU
      - Subscribe tso-rexx

These are just some of the available resources that you should check out to learn more.

Questions

Now it is your turn to ask any questions you didn't ask during the presentation.