Interactive System Productivity Facility (ISPF)

ISPF Panels – Advanced

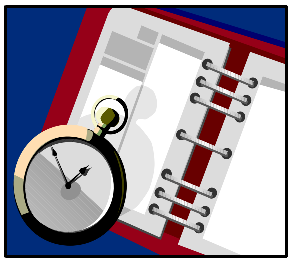
SHARE 118
Session 10615

Peter Van Dyke
IBM Australia
SHARE 118, Winter 2012
pvandyke@au1.ibm.com

Agenda

- Action Bars and Pull-Down Choices
- Point-and-Shoot Fields
- Scrollable Areas
- Dynamic Areas
- Scrollable Fields
- Panel Logic
  - Functions, Other Statements
- Panel REXX
- DTL - Dialog Tag Language
- References

NOTE: A basic knowledge of ISPF panel processing is assumed.



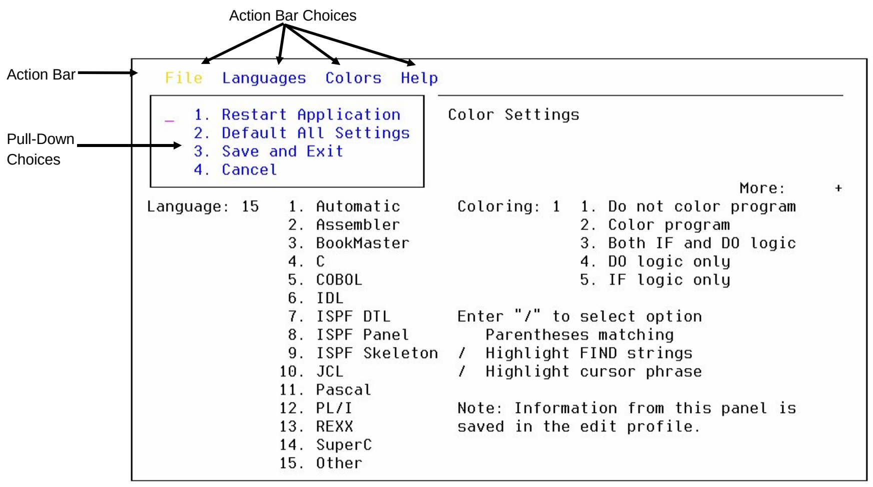
This presentation looks at a number of the more advanced features of ISPF panels, including:

•Action Bar and Pull-Down Choices
•Point and Shoot fields
•Dynamic areas
•Scrollable areas
•Scrollable fields
•Advanced panel logic
•Panel REXX

This session also provides a brief overview of the benefits of using Dialog Tag Language (DTL) to develop ISPF panels.

NOTE: This session assumes you have a basic knowledge of ISPF panel processing.

Action Bars and Pull-Down Choices

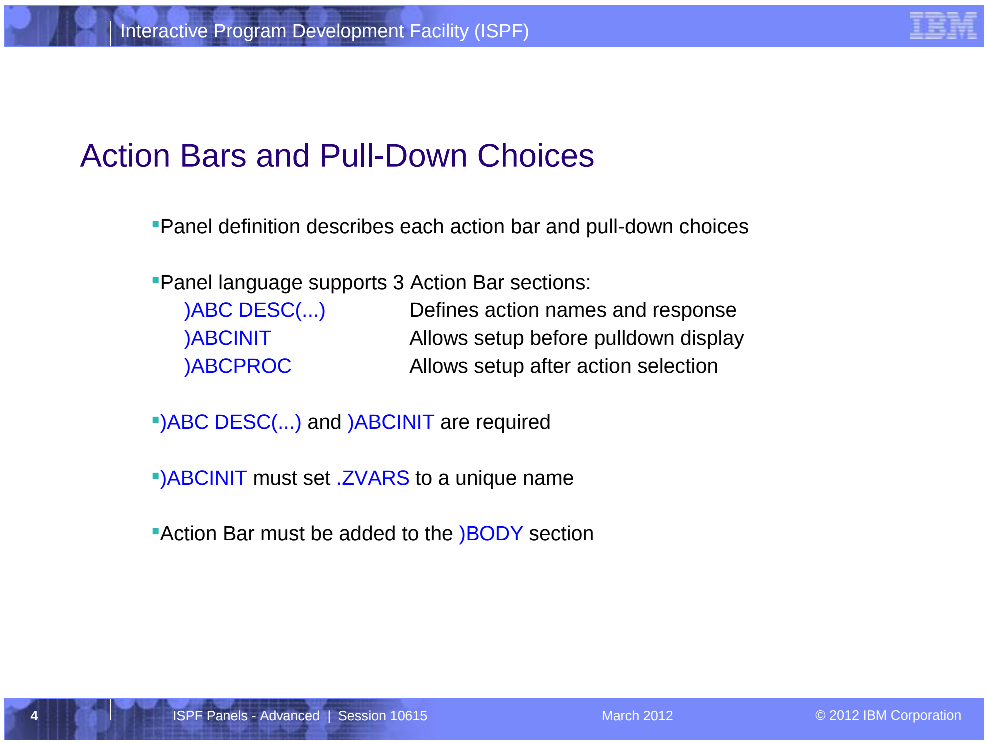


```
 File  Languages  Colors  Help
 ─────────────────────────────────────────────────────────
 _  1. Restart Application    Color Settings
    2. Default All Settings
    3. Save and Exit
    4. Cancel
                                                  More:     +
Language: 15    1. Automatic      Coloring: 1  1. Do not color program
                2. Assembler                   2. Color program
                3. BookMaster                  3. Both IF and DO logic
                4. C                           4. DO logic only
                5. COBOL                       5. IF logic only
                6. IDL
                7. ISPF DTL       Enter "/" to select option
                8. ISPF Panel        Parentheses matching
                9. ISPF Skeleton  /  Highlight FIND strings
               10. JCL            /  Highlight cursor phrase
               11. Pascal
               12. PL/I           Note: Information from this panel is
               13. REXX           saved in the edit profile.
               14. SuperC
               15. Other
```

Action Bars and Pull-Downs Choices support Common User Access (CUA) guidelines. The CUA guidelines define a user interface in terms of common elements, such as the way information appears on a screen, and interaction techniques, such as the way users respond to what appears on a screen. The Action Bar is the area at the top of an application panel that contains action bar choices for the panel.

Pull-Down Choices represents a group of related choices that appear in the pull-down associated with the action bar choice.

When the user selects an action bar choice, the associated pull-down appears directly below the action bar choice.
Pull-Downs contain choices that, when selected by the user, perform actions that apply to the contents of the panel.
In the example shown, the File action bar choice has been selected.



Action Bars and Pull-Down Choices

- Panel definition describes each action bar and pull-down choices
- Panel language supports 3 Action Bar sections:

)ABC DESC(...)	Defines action names and response
)ABCINIT	Allows setup before pulldown display
)ABCPROC	Allows setup after action selection

-)ABC DESC(...) and)ABCINIT are required
-)ABCINIT must set .ZVARS to a unique name
- Action Bar must be added to the)BODY section

A panel defines action bars and pulldown choices using the **)ABC**, **)ABCINIT**, and **)ABCPROC** panel sections.

)ABC DESC(...) - This section is specified after the)ATTR section. It defines the text of the action bar choice, the location of the mnemonic character (the underscored bit), the action names and responses. This is done with keywords and statements:

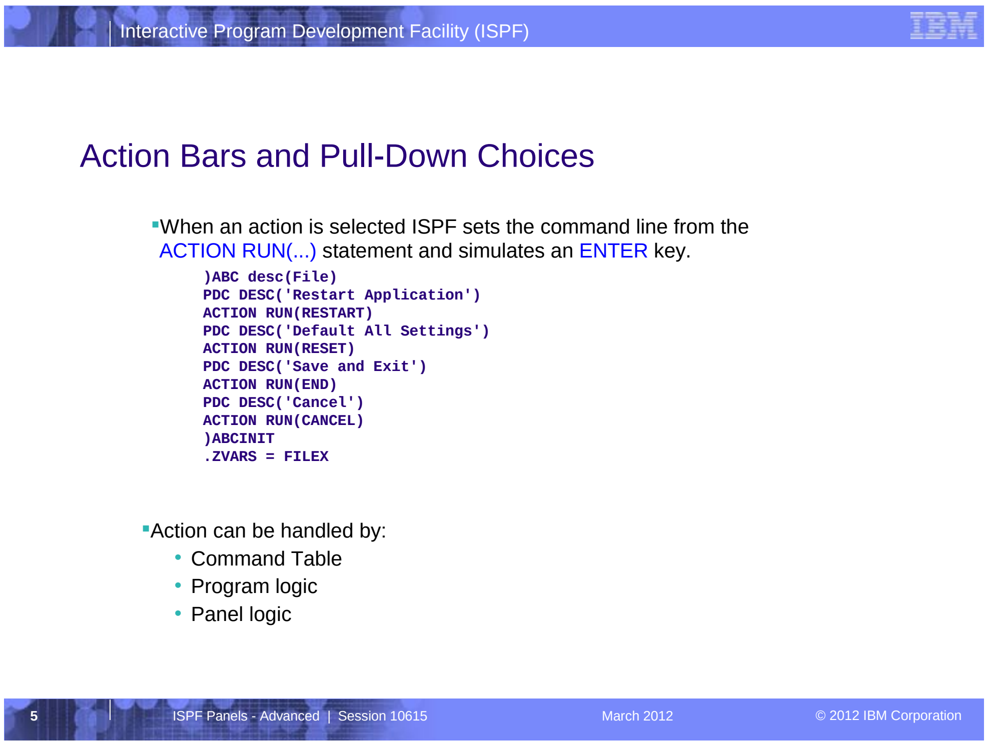
•**DESC(...)** - text of action bar choice. This must match the text in the)BODY section.

•**PDC DESC(...)** - each pull-down choice is identified by the PDC statement.

•**ACTION RUN(...)** - defines command to be run when this choice is selected. Must be 2-8 characters in length.

)ABCINIT - Allows setup before the pull-down display. This allows alteration of the text of the action. For example you may want to verify that some conditions are met and make the text of an action state that the choice is unavailable. Within the)ABCINIT section, you must assign a value to .ZVARS of a unique name, even if you do not use the name for anything else. This is to allow ISPF to create a field on the action bar which will contain the name of the action.

)ABCPROC - Allows setup after action selection. This section is not required and is only needed in special cases (e.g. verification of the pull-down entry field).



Action Bars and Pull-Down Choices

- When an action is selected ISPF sets the command line from the ACTION RUN(...) statement and simulates an ENTER key.

```
)ABC desc(File)
PDC DESC('Restart Application')
ACTION RUN(RESTART)
PDC DESC('Default All Settings')
ACTION RUN(RESET)
PDC DESC('Save and Exit')
ACTION RUN(END)
PDC DESC('Cancel')
ACTION RUN(CANCEL)
)ABCINIT
.ZVARS = FILEX
```

- Action can be handled by:
  - Command Table
  - Program logic
  - Panel logic

This example of an **)ABC** section has 4 actions, defined by the **PDC** (pull down choice) statement. Each **PDC** has an **ACTION** statement which tells ISPF what action to take when the pull-down choice is selected.

When a pull-down choice is selected, ISPF takes the value of the **ACTION RUN(...)** statement (the part in the RUN() section) and logically places it on the command line and simulates an Enter key. This allows the processing to be handled by the command table, program logic, or panel logic.

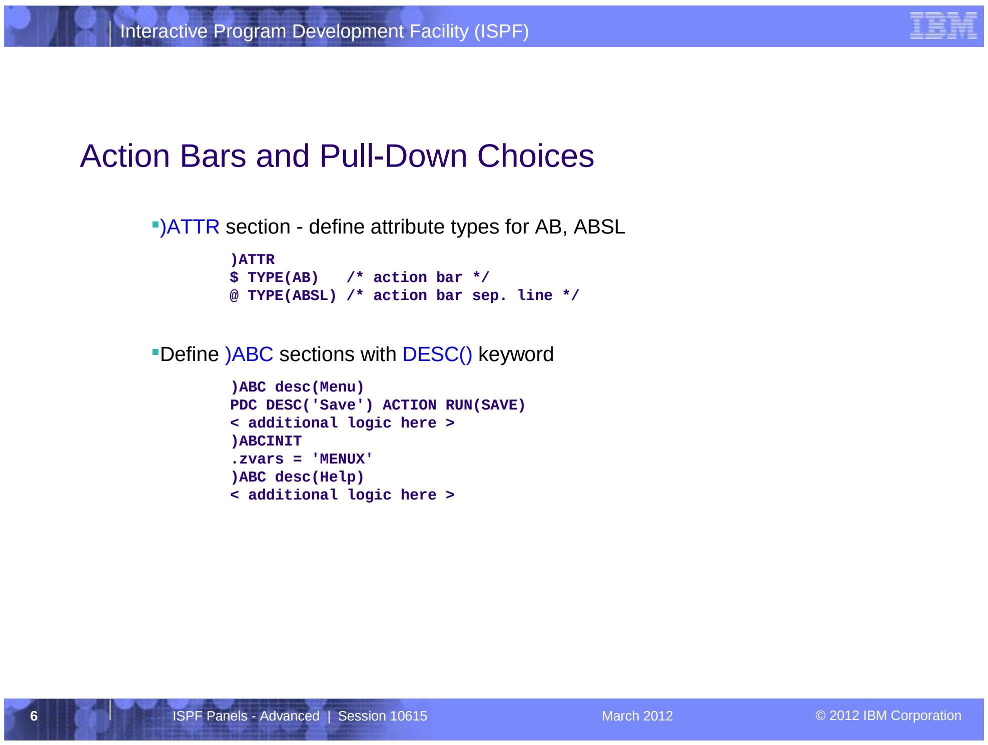
You can use the **ISRROUTE** command to invoke the **ISPF SELECT** service (ISRROUTE is a command defined in the ISPF command table ISPCMDS).

```
e.g. ACTION RUN(ISRROUTE) PARM('SELECT PGM(USERLIST)
NEWAPPL(USR)')
```

You can use the **PARM** keyword to pass parameters.

For example, you can specify **ACTION RUN(CANCEL)** and have the ISPF command table handle the CANCEL command. Or you could say **ACTION RUN(XYZ)** and have your panel logic and/or program handle the XYZ command (assuming it is not in the active command table).

NOTE: The **)ABCINIT** section must contain a **.ZVARS** control variable assignment statement to associate a variable name with the pull-down entry field.



Action Bars and Pull-Down Choices

-)ATTR section - define attribute types for AB, ABSL

```
)ATTR
$ TYPE(AB)   /* action bar */
@ TYPE(ABSL) /* action bar sep. line */
```

- Define)ABC sections with DESC() keyword

```
)ABC desc(Menu)
PDC DESC('Save') ACTION RUN(SAVE)
< additional logic here >
)ABCINIT
.zvars = 'MENUX'
)ABC desc(Help)
< additional logic here >
```

To create an action bar on a panel, you need to:

1. Add at least the TYPE(AB) to the)ATTR section.

2. Add)ABC and)ABCINIT sections to describe the action bar choices.

Remember to:

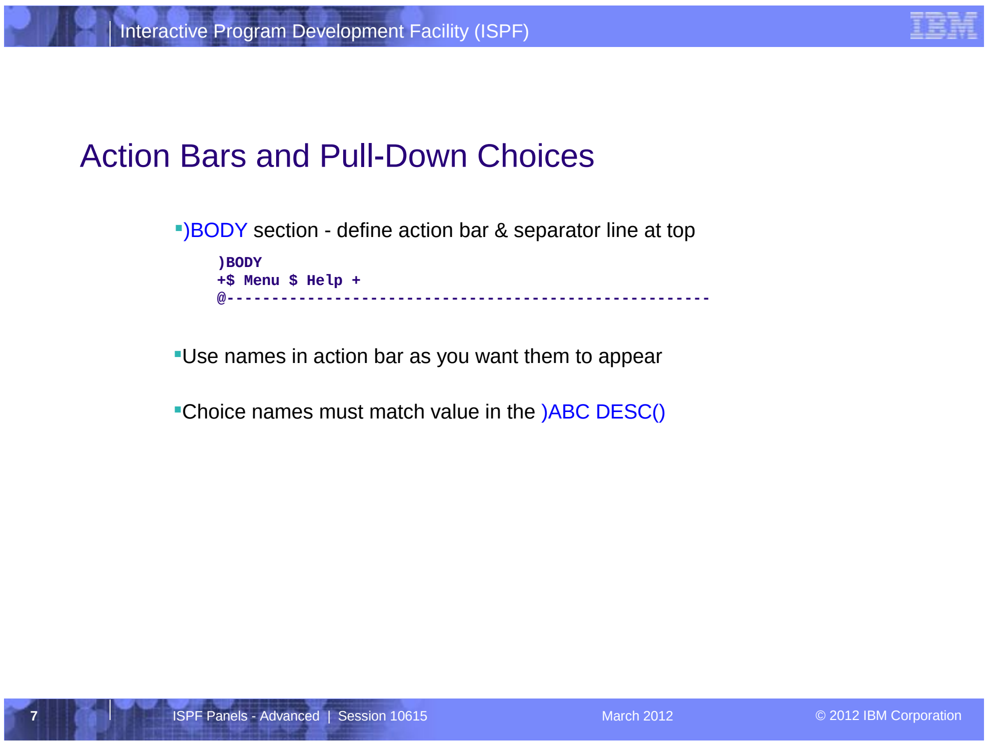
•set the **DESC** value in)ABC to the corresponding text in the)BODY section

•assign **.ZVARS** in the)ABCINIT section

Notes:

•The "additional logic here" shown in the example refers to other pull-down choices, not processing logic.

•In the example the **TYPE(ABSL)** attribute character is used to define the action bar separator line.



Action Bars and Pull-Down Choices

-)BODY section - define action bar & separator line at top

```
)BODY
+$ Menu $ Help +
@-------------------------------------------------------
```

- Use names in action bar as you want them to appear

- Choice names must match value in the)ABC DESC()

3. Add an action bar to the panel using an **AB** attribute character ($) for each pull down. After the attribute, place the name of the pull-down from the description in the)ABC section.

Notes:

•there must be a blank between the AB attribute and the text.

•the '+' attribute character represents normal text.

Program or panel logic, or entries in a command table are required to process the action bar selections.

Action Bars and Pull-Down Choices - Example

```
)ATTR
. TYPE(PT)   /* Panel title              */
% TYPE(ET)   /* Emphasized text          */
+ TYPE(NT)   /* Normal text              */
_ TYPE(NEF)  /* Normal entry field       */
$ TYPE(AB)   /* Action bar               */
@ TYPE(ABSL) /* Action bar separator line */
)ABC desc(Menu)
PDC DESC('Save') ACTION RUN(SAVE)
PDC DESC('End') ACTION RUN(END)
PDC DESC('Cancel') ACTION RUN(CANCEL)
)ABCINIT
.zvars = 'MENUX'
)ABC DESC(Help)
PDC DESC('Extended Help...')
ACTION RUN(HELP)
)ABCINIT
.ZVARS = HELPX
.RESP = ENTER   /* Don't even show choice */
                /* This is an example and */
                /* is NOT CUA..           */
)BODY WINDOW(48,6)
+$ Menu $ Help +
@-----------------------------------------------+
.               Panel Title
%===>_ZCMD                                      +
+This is normal text. Right?_ANS+ (Yes, No)
)END
```

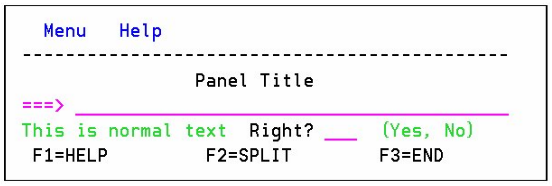
This is a working example of a panel which will appear in a window (if an ADDPOP is issued). It has 2 pull-downs, **Menu** and **Help**. It shows the use of some CUA attributes (e.g. TYPE(NEF)), overriding the defaults (&, + , and _) and it shows an example of adding processing to the **)ABCINIT** section for the **Help** pulldown.

If the **Help** pull-down is selected, ISPF will automatically simulate an extra enter key because of the **.RESP=ENTER** in the)ABCINIT section. Thus the pull-down will never be shown and since the default is for ISPF to select the first pull-down choice when just the enter key is pressed and the pull-down is selected for the 1st time, ISPF will run the HELP command and will never actually show the pull-down.

This is a method you can use to have action bar choices automatically take some action.

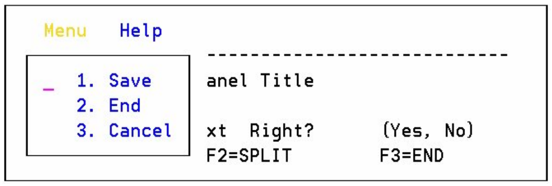
Action Bars and Pull-Down Choices - Example

The panel looks like this when displayed:



```
  Menu   Help
---------------------------------------------
              Panel Title
===> ________________________________________
This is normal text  Right? ___   (Yes, No)
 F1=HELP         F2=SPLIT        F3=END
```

When 'Menu' is selected, a pulldown is shown:



```
  Menu   Help
 ---------------- ----------------------------
 | _  1. Save   | anel Title
 |    2. End    |
 |    3. Cancel | xt  Right?      (Yes, No)
 ---------------- F2=SPLIT        F3=END
```

Shown here is the format of the panel when initially displayed and when the **Menu** choice from the action bar is selected.

Point-and-Shoot Fields

- Point-and-shoot fields are fields with an associated action that results when the ENTER key is pressed with the cursor on the field

- Define attribute
  - PAS(ON|OFF) - Input and output fields
  - TYPE(PS) - Text

- Define FIELD entries in)PNTS section
  - FIELD(*field_name*|ZPS*xxyyy*) VAR(*variable*) VAL(*value*)
    - *field_name* is the corresponding point-and-shoot field
    - *xx*: 00 if field in)BODY section; 01-99 for)AREA number in which field is defined
    - *yyy*: 001-999 for relative position of field within)BODY or)AREA
    - *variable* is set to *value* when point-and-shoot field is selected

- Dialog should take an action based on the *value* in the *variable*

Point-and-Shoot fields are areas of a panel that cause an action to take place when the cursor is placed on the area and the **ENTER** key is pressed. To create a Point-and-Shoot field you need to:

1) In the)ATTR section, define the attribute byte used for point-and-shoot input/output fields and text

(a) PAS(ON|OFF) – defines the attribute byte for input and output point-and-shoot field. It is not a CUA type, therefore, it can have the color, intensity, and highlight changed.

(b) TYPE(PS) – defines the attribute byte for point-and-shoot text. It is a CUA type, so it does not allow color, intensity, and highlight changes.

2) For each panel field defined with a point-and-shoot attribute, there must be a corresponding **FIELD** entry in the **)PNTS** section. If there is no corresponding entry in the)PNTS section no action is taken when the field is selected. The)PNTS section is specified after the)BODY and)AREA sections. The name specified in the FIELD statement is *field_name* for input/output fields, and *ZPSxxyyy* for text fields. When a point-and-shoot field is selected, ISPF sets *variable* to *value*. The dialog should take an appropriate action based on the value found in variable.

Point-and-Shoot Fields - Example

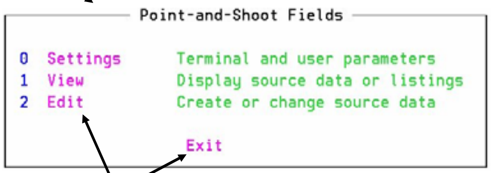
Panel Definition

```
)ATTR
 !  TYPE(SAC)
 %  TYPE(NT)
 +  TYPE(PS)
 $  TYPE(OUTPUT) CAPS(OFF) PAS(ON)
)BODY WINDOW(50,6)
%
!0 +Settings    %Terminal and user parameters%
!1 +View        %Display source data or listings%
!2 +Edit        %Create or change source data%

                 $EXITPB%
)INIT
&ZWINTTL = 'Point-and-Shoot Fields'
&EXITPB = 'Exit'
)PROC
&ZCMDWRK = &Z
IF (&ZCMD ¬= &Z)
  &ZCMDWRK = TRUNC(&ZCMD,'.')
  &ZTRAIL=.TRAIL
  IF (&ZCMDWRK = &Z)
    .MSG = ISRU000
&ZSEL = TRANS (TRUNC (&ZCMD,'.')
  0,'PGM(ISPISM) SCRNAME(SETTINGS)'
  1,'PGM(ISRBRO) PARM(ISRBRO01) SCRNAME(VIEW)'
  2,'PGM(ISREDIT) PARM(P,ISREDM01) SCRNAME(EDIT)'
   *,'?')
)PNTS
FIELD(ZPS00001) VAR(ZCMD) VAL(0)
FIELD(ZPS00002) VAR(ZCMD) VAL(1)
FIELD(ZPS00003) VAR(ZCMD) VAL(2)
FIELD(EXITPB) VAR(ZCMD) VAL(EXIT)
)END
```

Rexx Driver

```
/* REXX */
ADDRESS 'ISPEXEC'
"ADDPOP"
"DISPLAY PANEL(PSFIELDS)"
"REMPOP"
if zsel ¬= '?' then do
  "SELECT " zsel
end
```

Panel Display



```
Point-and-Shoot Fields

0  Settings     Terminal and user parameters
1  View         Display source data or listings
2  Edit         Create or change source data

                 Exit
```

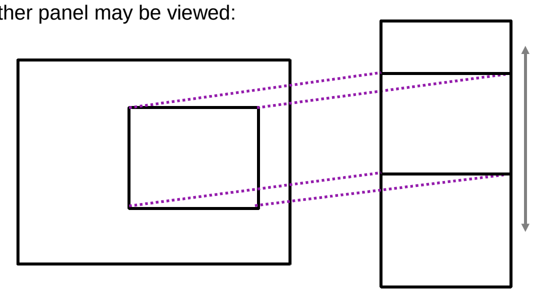
Point-and-Shoot Fields

This is a working example of point-and-shoot fields on a selection panel displayed in a popup panel. The menu options "Settings", "View" and "Edit" have the PS attribute. When the cursor is placed on the text with a PS attribute and Enter is pressed, the corresponding value specified in the)PNTS section is assigned to the ZCMD variable. Logic in the)PROC section of the panel then causes the appropriate action to be taken.

The variable EXITPB is assigned to an output point-and-shoot field by using the $ attribute character which has PAS(ON) specified. When the cursor is placed on this field the value EXIT is assigned to the variable ZCMD.

)BODY - Scrollable Areas

- Scrollable areas allow the application developer to define a portion of the panel that ISPF manages for scrolling
- A scrollable area can be thought of as a hole in a panel through which another panel may be viewed:



- There may be multiple scrollable areas on one panel.
  - When there are, scrolling is cursor sensitive.

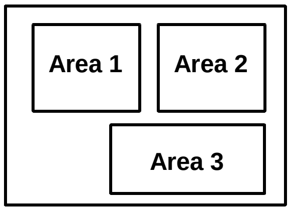
ISPF allows a section of an ISPF panel to be defined as a scrollable area. A scrollable area can be thought of as a window (or “hole”) in a panel through which a larger section of data may be viewed. ISPF handles any requests by the user to scroll the data.
ISPF supports multiple scrollable area on one ISPF panel, in which case scrolling is cursor sensitive.

Defining a Scrollable Area

- Creating a scrollable area requires three actions:
  - Define the attribute character for the scrollable area
  - Define the "hole" in the)BODY section where the scrollable area appears using the scrollable area attribute character
  - Define the area to be displayed in the "hole" in the)AREA section
- The name on the first attribute defines the name of the scrollable area.

It must match the name of the area on the)AREA section.

- ISPF references scrollable areas from left to right, top to bottom:



Defining a scrollable area requires 3 actions:

1. Define the attribute character for the scrollable area
2. Define the “hole” in the)BODY section using the attribute character to delimit the area
3. Define the format of the area to be displayed in the hole - This is the **) AREA** section

The name specified after the first attribute byte of the scrollable area (in the)BODY section) is the **NAME** of the area and corresponds with the name specified on the)AREA section.

ISPF references scrollable area from left to right, top to bottom.

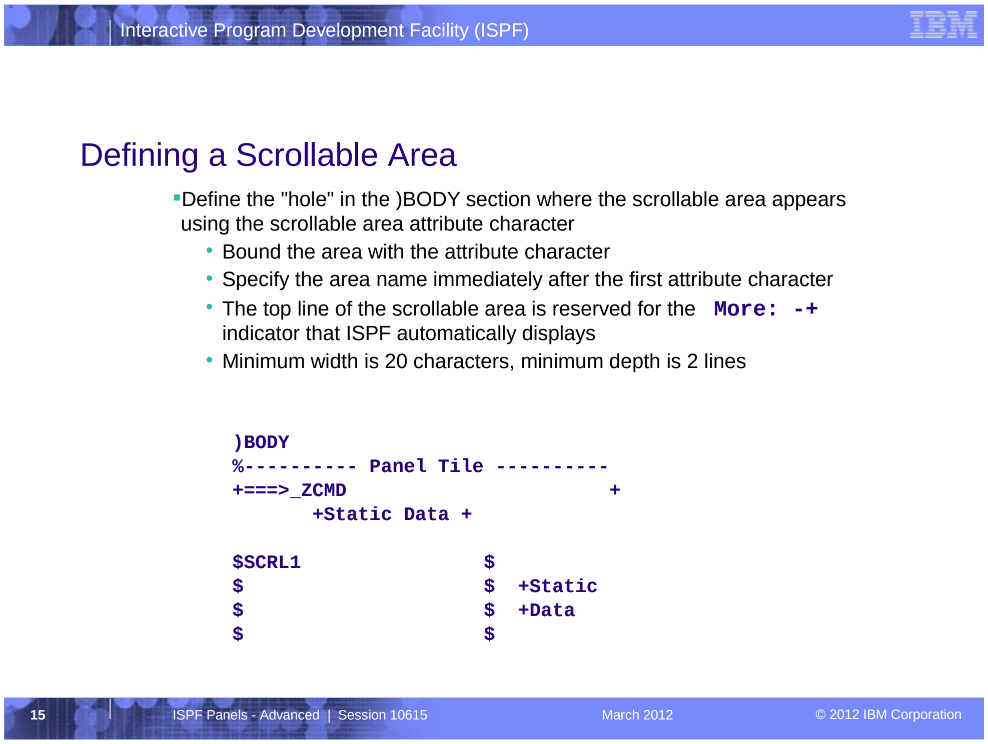
Defining a Scrollable Area

- Define the attribute character for the scrollable area:

```
$ AREA(SCRL) EXTEND(ON|OFF)
```

- EXTEND(ON) specifies that the depth of an area can be automatically increased, if required, so that the depth of the entire body of the panel matches the depth of the physical screen on which it is being displayed.
- Only one extendable area can be specified in a panel definition
- The value for EXTEND cannot be specified as a dialog variable

The attribute byte for a scrollable area is defined with **AREA(SCRL)**, and optionally **EXTEND(ON|OFF)**. This character is used to define the borders of the scrollable area in the)BODY section.
EXTEND(ON) specifies that the depth of the AREA automatically increases to the physical screen size. There can only be ONE extendable area on a panel and the value for the EXTEND option cannot be a dialog variable.



Defining a Scrollable Area

- Define the "hole" in the)BODY section where the scrollable area appears using the scrollable area attribute character
  - Bound the area with the attribute character
  - Specify the area name immediately after the first attribute character
  - The top line of the scrollable area is reserved for the **`More: -+`** indicator that ISPF automatically displays
  - Minimum width is 20 characters, minimum depth is 2 lines

```
)BODY
%---------- Panel Tile ----------
+===>_ZCMD                       +
       +Static Data +

$SCRL1                $
$                     $  +Static
$                     $  +Data
$                     $
```

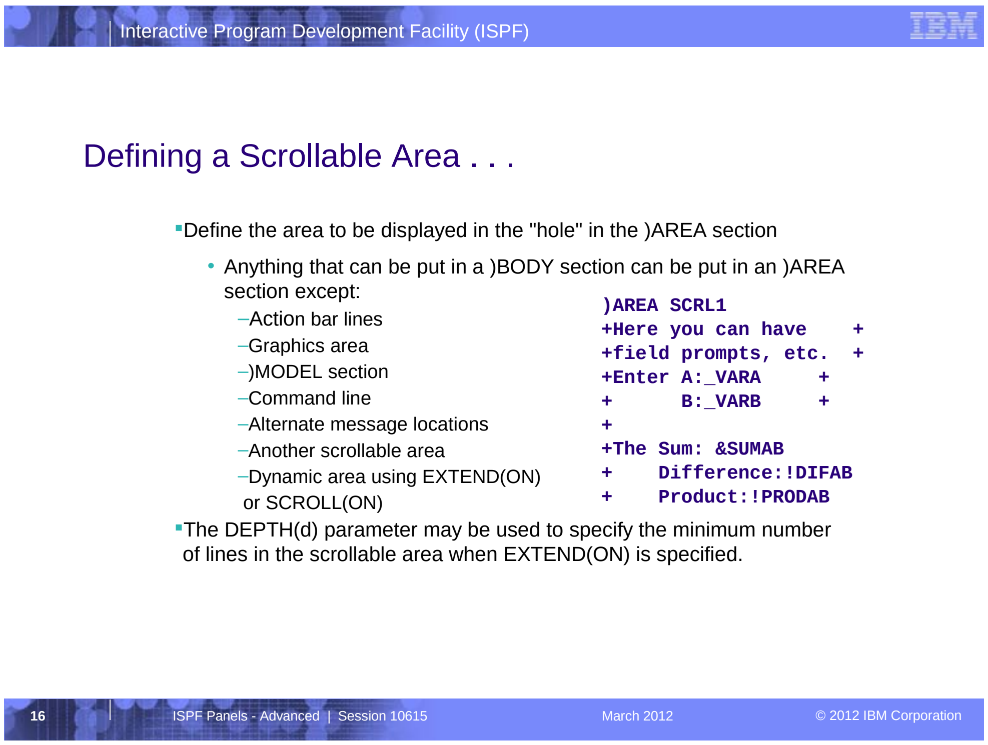
The area that we want to define as a scrollable area is **bound** with the attribute character (**$** was defined as the attribute character for the scrollable area on the previous page).

The **name** of the scrollable area (in this example it is **SCRL1**), is specified immediately after the first attribute character for the scrollable area.

Notes:

•The first line of the scrollable area is reserved for the indicator: **`MORE: -+`**

•The minimum size of a scrollable area is 20 characters by 2 lines.



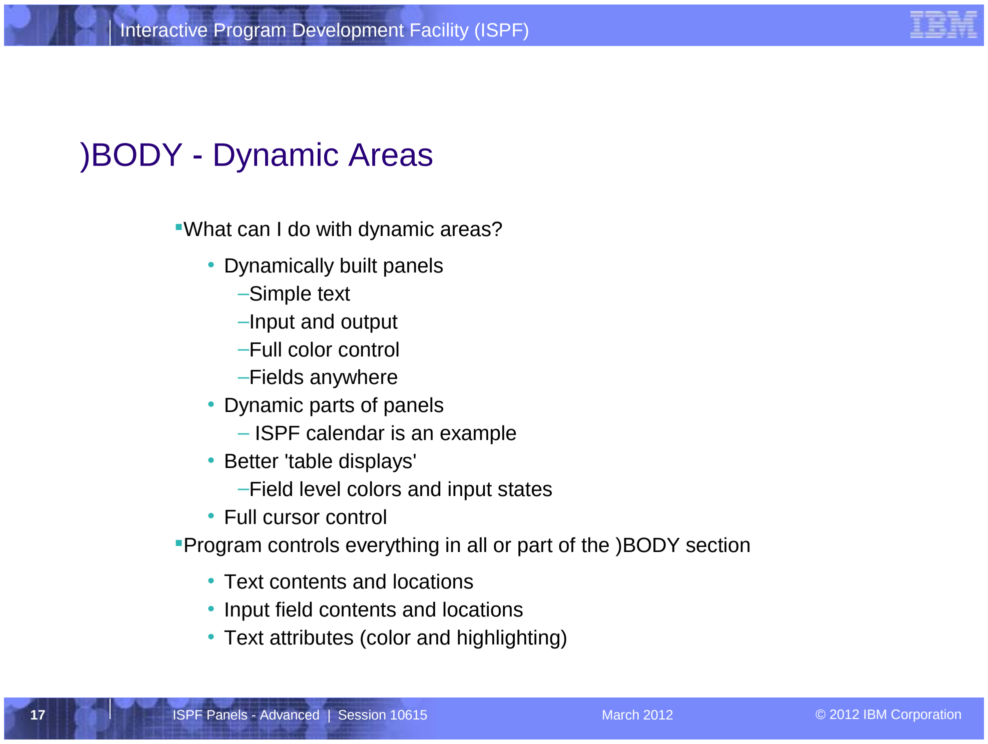
Defining a Scrollable Area . . .

- Define the area to be displayed in the "hole" in the)AREA section
  - Anything that can be put in a)BODY section can be put in an)AREA section except:
    - Action bar lines
    - Graphics area
    -)MODEL section
    - Command line
    - Alternate message locations
    - Another scrollable area
    - Dynamic area using EXTEND(ON) or SCROLL(ON)

```
)AREA SCRL1
+Here you can have    +
+field prompts, etc.  +
+Enter A:_VARA     +
+      B:_VARB     +
+
+The Sum: &SUMAB
+    Difference:!DIFAB
+    Product:!PRODAB
```

- The DEPTH(d) parameter may be used to specify the minimum number of lines in the scrollable area when EXTEND(ON) is specified.

The)AREA section defines the format of the data displayed in the associated scrollable area.



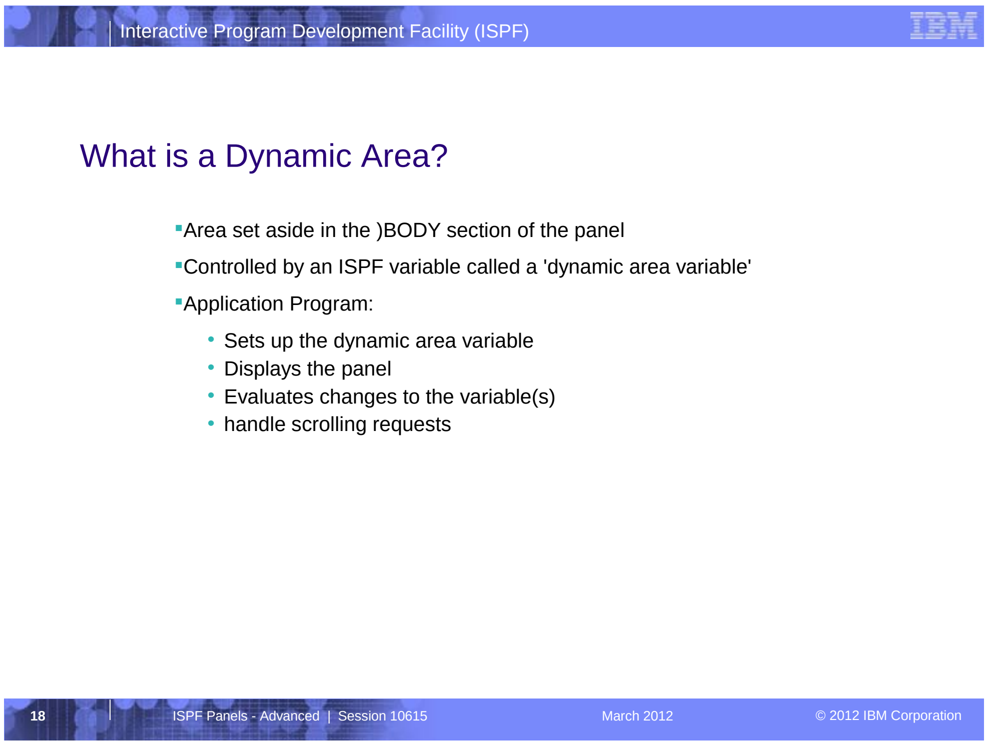
)BODY - Dynamic Areas

- What can I do with dynamic areas?
  - Dynamically built panels
    - Simple text
    - Input and output
    - Full color control
    - Fields anywhere
  - Dynamic parts of panels
    - ISPF calendar is an example
  - Better 'table displays'
    - Field level colors and input states
  - Full cursor control
- Program controls everything in all or part of the)BODY section
  - Text contents and locations
  - Input field contents and locations
  - Text attributes (color and highlighting)

ISPF allows a section of an ISPF panel to be defined as a dynamic area.
Dynamic areas can contain:
•Simple Text

•Input fields, and

•Output fields.
The Calendar on the ISPF primary menu is and example of a dynamic area.
Dynamic area could be used for a "better" table display, as is the case with the ISPF data set list (option 3.4).
It is the application program that controls EVERYTHING within the Dynamic area, including:
•Text and where it is located
•Input fields and where they are located (and how long)
•Text attributes (including color and highlighting)



What is a Dynamic Area?

- Area set aside in the)BODY section of the panel
- Controlled by an ISPF variable called a 'dynamic area variable'
- Application Program:
  - Sets up the dynamic area variable
  - Displays the panel
  - Evaluates changes to the variable(s)
  - handle scrolling requests

A Dynamic Area is an area set aside in the)BODY section of a panel and is controlled by an ISPF variable (called a dynamic area variable). The value of the dynamic area variable defines the data displayed in the dynamic area.

It is the responsibility of the application program displaying the panel with the dynamic area to:
•Set up the dynamic area variable

•Display the panel

•Evaluate changes to the data within the dynamic area and then update any ISPF variables

•Handle any requests to scroll the dynamic area (UP, DOWN, TOP, BOTTOM)

The Dynamic Area Variable

- Long string of characters
- Contains all lines to be shown in the dynamic area
  - Data wraps on screen to create multiple lines
- Contains all attributes, text, and space for input fields
  - Attributes are defined in the)ATTR section by using
    - TYPE (DATAOUT)
    - TYPE (DATAIN)

A dynamic area can be thought of as a long string of characters that wraps to create the multiple lines within the area. The string of characters contains the attribute characters, text, and input/output fields. Attributes are defined in the)ATTR section using TYPE(DATAOUT) and TYPE(DATAIN) options.

Dynamic Area - Attributes

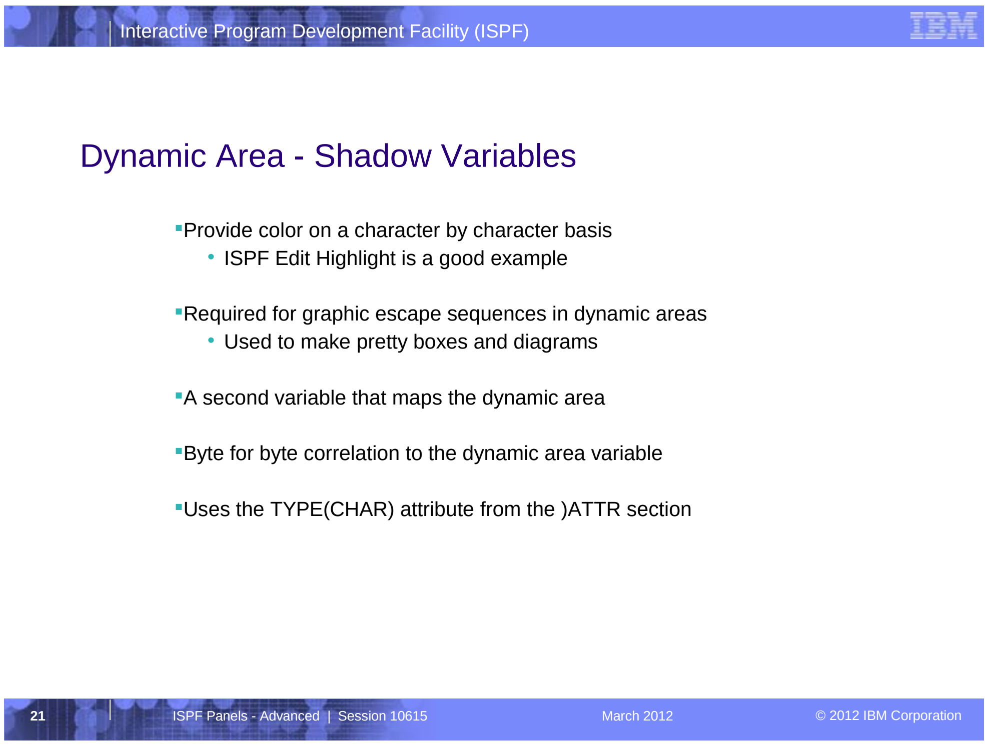
- The attribute section)ATTR defines all attributes used in the dynamic area
  - Attributes are reserved characters given special meaning
    - Examples:

```
% TYPE(DATAOUT) COLOR(YELLOW) INTENS(HIGH)
_ TYPE(DATAIN)  COLOR(PINK)
```

-)ATTR section also defines a character to delimit the dynamic area:

```
@ AREA(DYNAMIC) EXTEND(OFF)
@ AREA(DYNAMIC) EXTEND(ON)
```

From the example above, the **%** attribute character identifies in the dynamic area an output/text area that will be yellow and high intensity. The _ (underscore) attribute character is used for an input field that is pink.
An attribute character is also required to define the dynamic area within the)BODY section. The examples above use the @ attribute character to define where in the)BODY section the dynamic area is located. Like scrollable areas, dynamic areas can be defined with either EXTEND(ON) or EXTEND(OFF).



Dynamic Area - Shadow Variables

- Provide color on a character by character basis
  - ISPF Edit Highlight is a good example
- Required for graphic escape sequences in dynamic areas
  - Used to make pretty boxes and diagrams
- A second variable that maps the dynamic area
- Byte for byte correlation to the dynamic area variable
- Uses the TYPE(CHAR) attribute from the)ATTR section

Defining a shadow variable for a Dynamic Area allows the dynamic area to be colored on a character by character basis (eg: ISPF Edit highlighting). A shadow variable is also required to generate graphic escape sequences in dynamic area for generating boxes and diagrams.

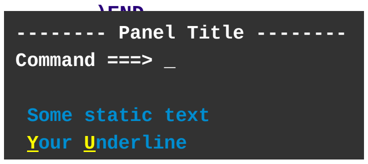
The shadow variable is another dialog variable that maps the dynamic area byte for byte. The TYPE(CHAR) attribute in the)ATTR section is used to define attributes used in the shadow variable.

Shadow Variable - Example

TYPE(CHAR) will not conflict with characters in the dynamic variable

Shadow variable maps dynamic area variable byte for byte

```
)ATTR DEFAULT(%+_)
 @ AREA(DYNAMIC) SCROLL(OFF)
 Y TYPE(CHAR) COLOR(YELLOW) HILITE(USCORE)
)BODY
%-------- Panel Title --------
%Command ===>_ZCMD           +
+
+ Some static text
+ @DYNVAR,SHADVAR          @
)INIT
  &DYNVAR = 'Your Underline'
  &SHADVAR= 'Y    Y        '
)END
```



```
-------- Panel Title --------
Command ===> _

 Some static text
 Your Underline
```

Here is a simple example showing how a dynamic area can be defined.

The @ attribute byte is used to identify the location of the dynamic area. It is non-extendable and non-scrollable.

The **Y** attribute character defines the character that will be used in the shadow variable value to display data in the dynamic area as yellow and underlined.

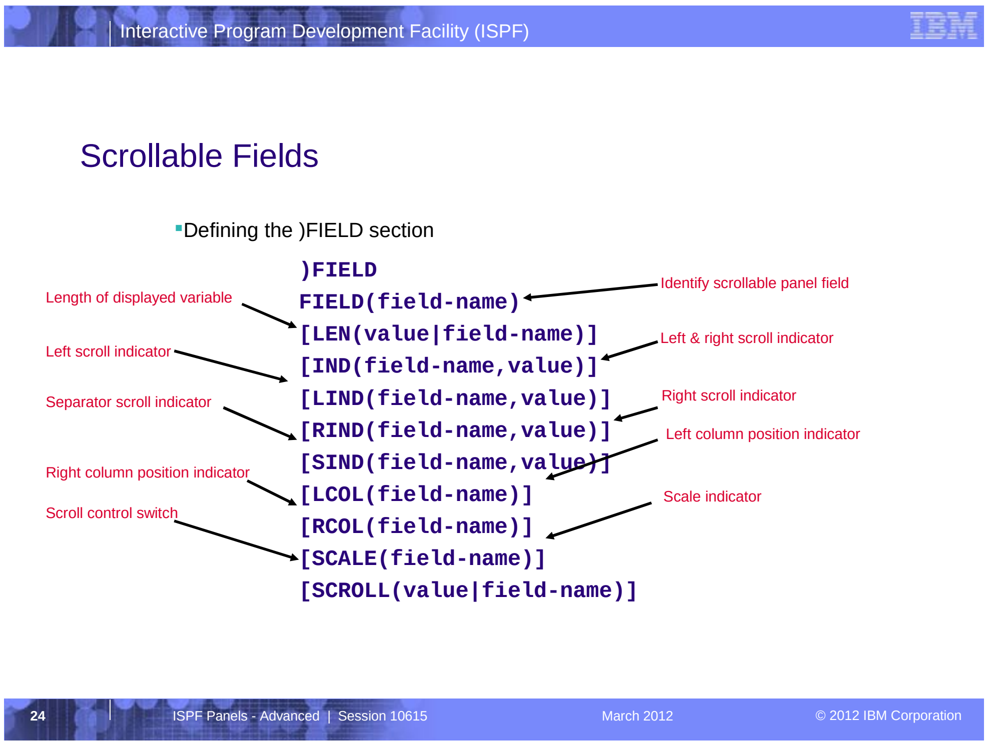
The dynamic area variable name is **DYNVAR**. The shadow variable name is **SHADVAR**.

The)INIT section of the panel contains logic to set values for the variables DYNVAR and SHADVAR. Note that the position of the Y characters in the value for SHADVAR cause the uppercase Y and U in the dynamic area to be displayed as yellow and underlined.

)BODY - Scrollable Fields

- Support for Scrollable field new in z/OS 1.5 (ISPF 5.5)
- Allow display of data larger than the panel field size
  - maximum field length is 32K
- New)FIELD section used to define fields that are scrollable
- LEFT, RIGHT and (new) ZEXPAND primary commands change display of scrollable fields

Scrollable Fields is a feature of ISPF added with z/OS 1.5. It was also made available to z/OS 1.2 and later via APAR OW57368 (PTF UA02839).
A scrollable field allows the display of dialog variables that are larger than the field on an ISPF panel. In fact the maximum data length supported for a scrollable field is 32K.
A new **)FIELD** section is used to define a field that is scrollable. The **LEFT** and **RIGHT** commands are used to scroll the data displayed in a scrollable field. The new **ZEXPAND** command displays the data from a scrollable field in a popup window.



Scrollable Fields

- Defining the)FIELD section

```
)FIELD
FIELD(field-name)
[LEN(value|field-name)]
[IND(field-name,value)]
[LIND(field-name,value)]
[RIND(field-name,value)]
[SIND(field-name,value)]
[LCOL(field-name)]
[RCOL(field-name)]
[SCALE(field-name)]
[SCROLL(value|field-name)]
```

Identify scrollable panel field
Length of displayed variable
Left & right scroll indicator
Left scroll indicator
Right scroll indicator
Separator scroll indicator
Left column position indicator
Right column position indicator
Scale indicator
Scroll control switch

The Syntax of the **)FIELD** section is shown here. Options are provided to allow the association of other variables with the scrollable field for:

•field length

•LEFT/RIGHT Scroll indicators (defaults: - +)

•Separator lines (default: <->)

•Left and Right displayed column values

•Scale lines - `----+----1----+----2----+----3...`

•Scroll control - ON or OFF

Scrollable Fields - Example

```
)ATTR
 | TYPE(OUTPUT) CAPS(OFF) JUST(ASIS )
 _ TYPE(INPUT)  CAPS(OFF) JUST(ASIS ) FORMAT(MIX)
)BODY
%------ Left/Right Scroll Panel ------
%OPTION  ===>_ZCMD
+ Field              Value
+ -------------------------------
+ Value            :_LR1         +
+ Scroll Indicator :|lr1in
+ Left & Right     :|lril      |lrir
+ Left/Right cols  :_Lr1lf     _lr1ri
+ Length           :_lr1ln
+ Scale            :|lr1sc       +
+ Separator        :|lrisp       +
)FIELD
 Field(LR1)
 Ind(lr1in,'<>')
 Lind(lril,'<') Rind(lrir,'>')
 Lcol(lr1lf) Rcol(lr1ri)
 Len(lr1ln)
 Scale(lr1sc)
 Sind(lrisp)
)END
```

Shown here is the source of a panel demonstrating the use of scrollable fields. The panel has a single **scrollable field** for variable **LR1**. The panel also displays fields for the following variables associated with current state of the scrollable field. These associations are established through the) FIELD section definition.

•**LR1IN** – scroll indicator with value ‘<>’

•**LRIL** & **LRIR** – Left and Right scroll indicators with values ‘<‘ and ‘>’

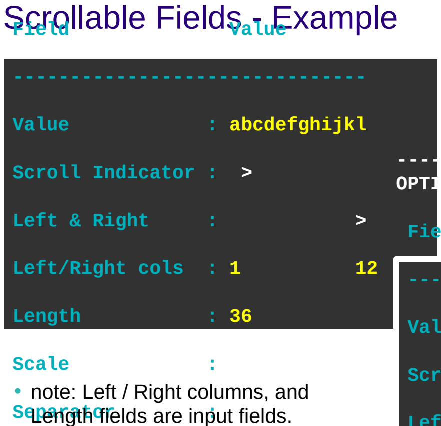
•**LR1LF** & **LR1RI** – Left and Right column values

•**LR1LN** – Field data length

•**LR1SC** – Scale line

•**LR1SP** – separator line

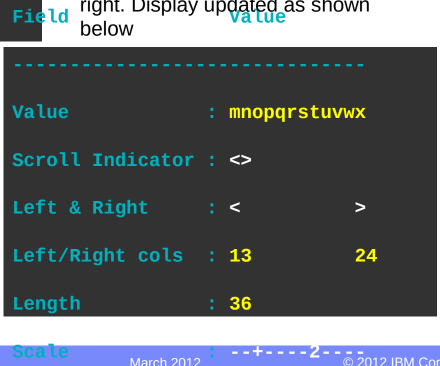
Scrollable Fields - Example



```
Field                Value
------------------------------
Value             : abcdefghijkl
Scroll Indicator  :  >
Left & Right      :             >
Left/Right cols   : 1           12
Length            : 36
Scale             :
Separator         :
```

```
----
OPTI
```

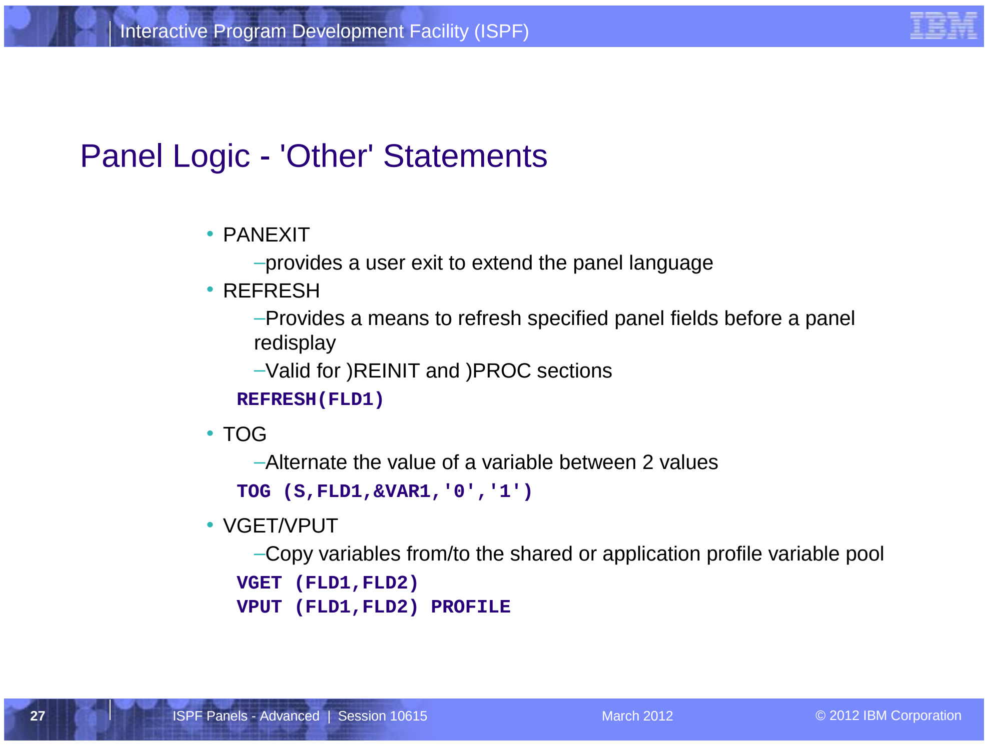
- Scrollable field showing first 12 characters of a 36 byte field, plus associated indicator, column, length, scale and separator fields
- Position cursor on value and scroll right. Display updated as shown below



```
Field                Value
------------------------------
Value             : mnopqrstuvwx
Scroll Indicator  : <>
Left & Right      : <           >
Left/Right cols   : 13          24
Length            : 36
Scale             : --+----2----
```

- note: Left / Right columns, and Length fields are input fields.
  - modifying these fields will also alter the display

Shown here are examples of the panel display with some suitable data. The variable **LR1** is **36 characters** longs initialized with the 26 letters of the alphabet followed by 0-9. Initially the display shows the **first 12** characters (abcdefghijkl) and only the RIGHT scroll indicators (as we are at the maximum LEFT position). If we position the cursor over the value and scroll RIGHT, the display is updated to show the **next 12** characters, both the LEFT and RIGHT scroll indicators are shown and the separator line indicates we can scroll both directions, and the left/right column values are updated. Also the scale line is updated to reflect the current position.



Panel Logic - 'Other' Statements

- PANEXIT
  - provides a user exit to extend the panel language
- REFRESH
  - Provides a means to refresh specified panel fields before a panel redisplay
  - Valid for)REINIT and)PROC sections

```
REFRESH(FLD1)
```

- TOG
  - Alternate the value of a variable between 2 values

```
TOG (S,FLD1,&VAR1,'0','1')
```

- VGET/VPUT
  - Copy variables from/to the shared or application profile variable pool

```
VGET (FLD1,FLD2)
VPUT (FLD1,FLD2) PROFILE
```

Here are some 'other' panel logic statements, not covered during the Dialog Developers Boot Camp.

•PANEXIT - used to define a panel exit - we will look at this shortly

•REFRESH - refreshes 1 or more panel fields before a redisplay of a panel

•TOG - toggle between 2 values if the data in FLD1 is modified

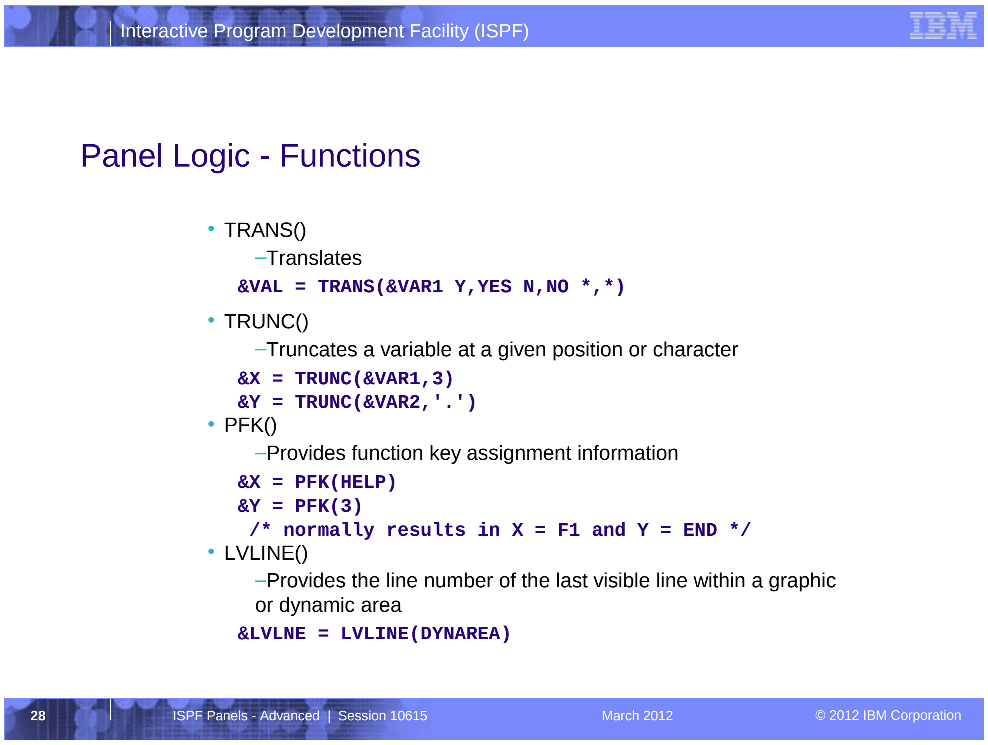
•VGET/VPUT - Copy a dialog variable from/to the shared or application profile pool

There are other statements, not shown here, and not often used

•VEDIT

•EXIT

•GOTO



Panel Logic - Functions

- TRANS()
  - Translates

```
&VAL = TRANS(&VAR1 Y,YES N,NO *,*)
```

- TRUNC()
  - Truncates a variable at a given position or character

```
&X = TRUNC(&VAR1,3)
&Y = TRUNC(&VAR2,'.')
```

- PFK()
  - Provides function key assignment information

```
&X = PFK(HELP)
&Y = PFK(3)
 /* normally results in X = F1 and Y = END */
```

- LVLINE()
  - Provides the line number of the last visible line within a graphic or dynamic area

```
&LVLNE = LVLINE(DYNAREA)
```

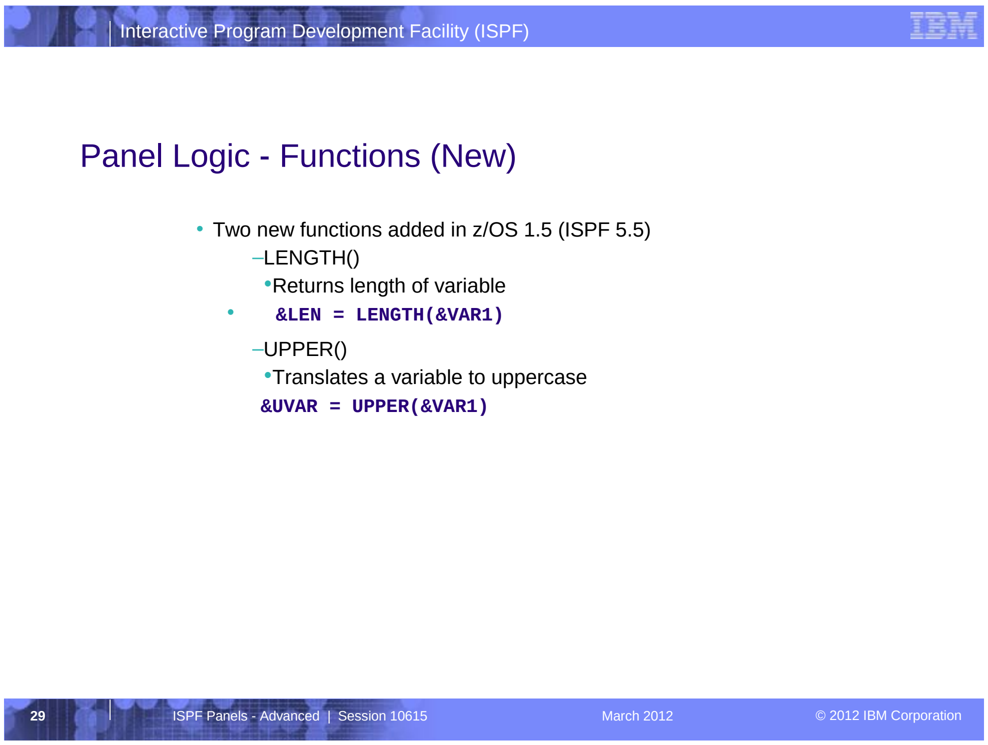
There are many built-in functions available for use in panel procedures. Some of the main ones are:

•TRANS - to translate values from 1 value to another. Most often used in menu selection panels (as we saw earlier). The '*,*' as the last pair is the anything-else condition. In the example, if &VAR1 is neither 'Y' or 'N' the value of VAR1 is put into VAL.

•TRUNC - truncates a variable. Truncation can either occur at a specified position, or at a particular character. The remaining data after the truncation point is stored in the .TRAIL control variable.

•PFK - provides function key information

•LVLINE - provides the line number of the last visible line within a graphic or dynamic area. This is useful to determine how many lines to scroll (up or down). (And it takes into account the presence or not of the function key display)



Panel Logic - Functions (New)

- Two new functions added in z/OS 1.5 (ISPF 5.5)
  - LENGTH()
    - Returns length of variable
    - `&LEN = LENGTH(&VAR1)`
  - UPPER()
    - Translates a variable to uppercase

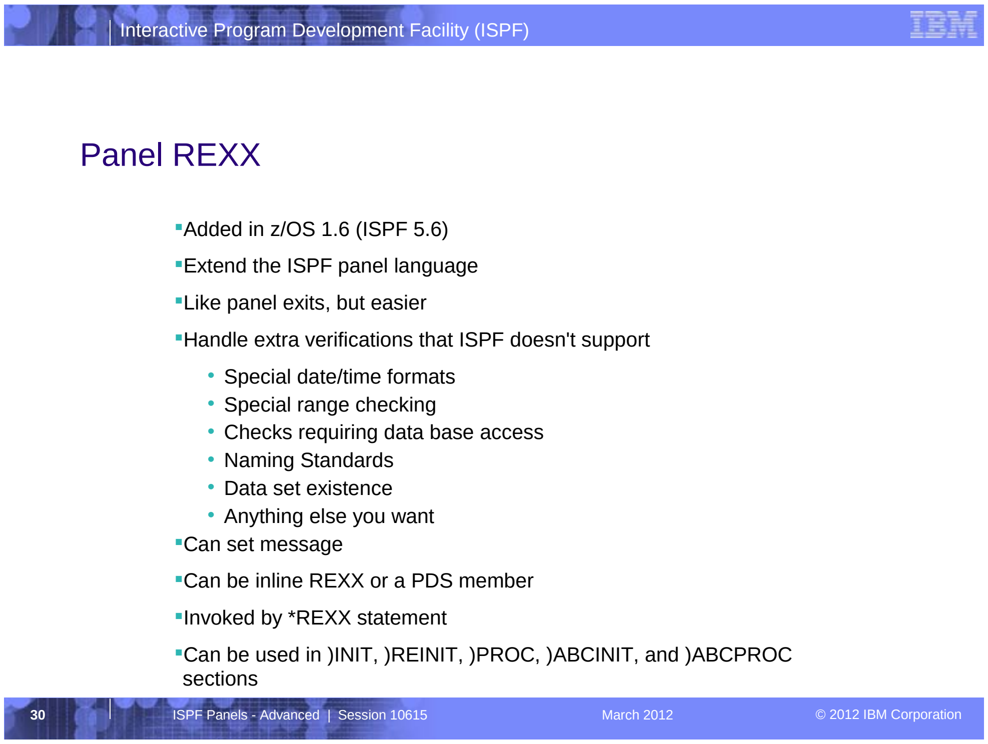
```
&UVAR = UPPER(&VAR1)
```

Two new functions were added in z/OS 1.5 (ISPF 5.5):

•**LENGTH** - can occur on the right side of an assignment statement to evaluate the length of a dialog variable. The variable length returned will be the maximum of the actual length of the variable if it exists and the length specified in the)FIELD section.

•**UPPER** - can occur on the right side of an assignment statement and will return the uppercase value of a variable.

Support added to z/OS 1.2 and above via APAR OW57368 (PTF UA02839).

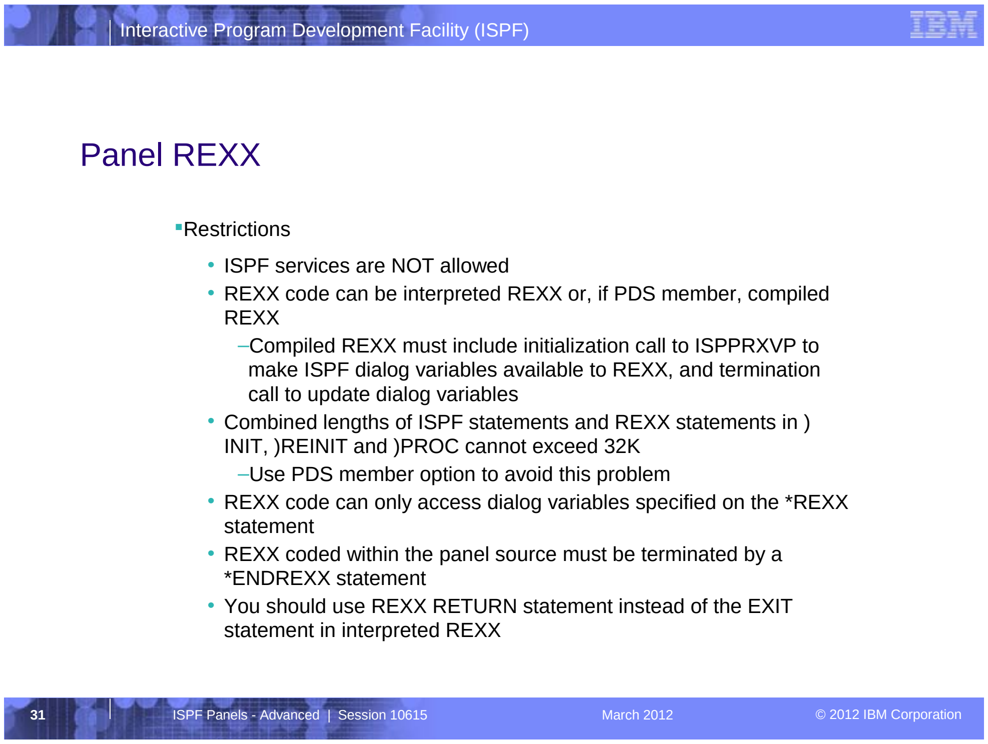


Panel REXX

- Added in z/OS 1.6 (ISPF 5.6)
- Extend the ISPF panel language
- Like panel exits, but easier
- Handle extra verifications that ISPF doesn't support
  - Special date/time formats
  - Special range checking
  - Checks requiring data base access
  - Naming Standards
  - Data set existence
  - Anything else you want
- Can set message
- Can be inline REXX or a PDS member
- Invoked by *REXX statement
- Can be used in)INIT,)REINIT,)PROC,)ABCINIT, and)ABCPROC sections

With z/OS 1.6 support was added to allow REXX to be invoked from ISPF panel procedures. This extended the processing capability in a panel procedure by making available the many features and functions of REXX. Tasks previously difficult or impossible to do from a panel procedure are now simple, such as reading data from a data set or performing mathematical calculations.

While it was possible to use a panel exit to invoke REXX from a panel procedure, using panel REXX is much simpler. The *REXX statement is used to invoke panel REXX and identify the ISPF variables to be processed by the REXX. The REXX can be coded directly into the panel procedure after the *REXX statement or the name of a member containing the REXX can be specified with the *REXX statement.



Panel REXX

- Restrictions
  - ISPF services are NOT allowed
  - REXX code can be interpreted REXX or, if PDS member, compiled REXX
    - Compiled REXX must include initialization call to ISPPRXVP to make ISPF dialog variables available to REXX, and termination call to update dialog variables
  - Combined lengths of ISPF statements and REXX statements in)INIT,)REINIT and)PROC cannot exceed 32K
    - Use PDS member option to avoid this problem
  - REXX code can only access dialog variables specified on the *REXX statement
  - REXX coded within the panel source must be terminated by a *ENDREXX statement
  - You should use REXX RETURN statement instead of the EXIT statement in interpreted REXX

Some restrictions apply to the processing that can be done in panel REXX:

•ISPF services cannot be called

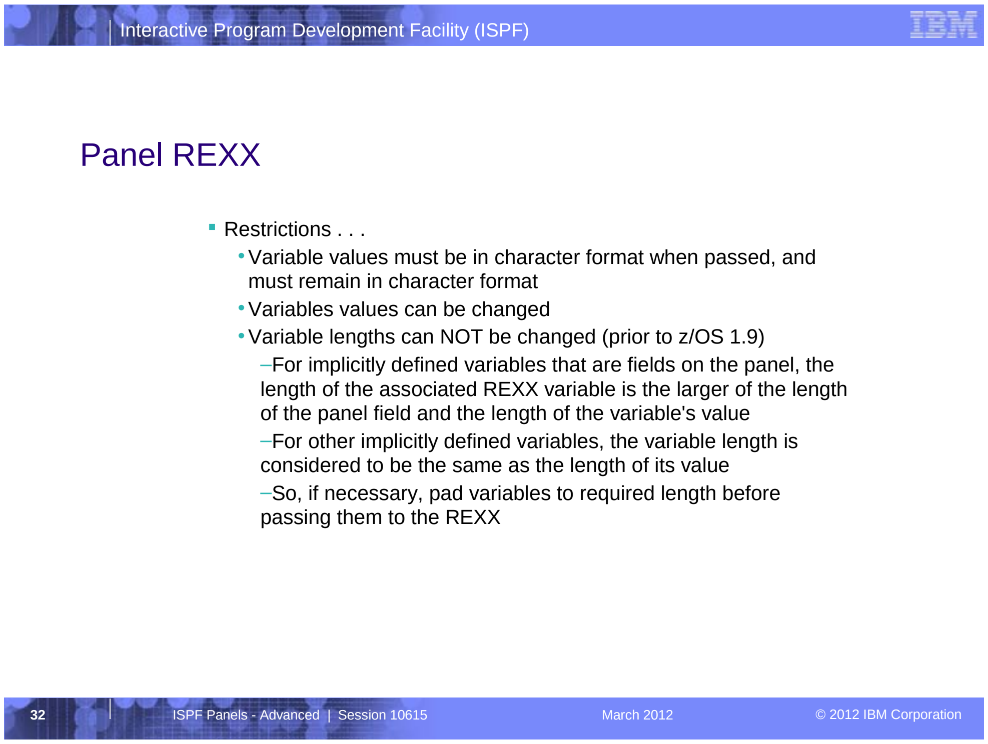
•Compiled REXX requires calls to module ISPPRXVP to make ISPF dialog variables available to REXX and to update the variable back in the ISPF variable pool.

•Coding REXX in the panel procedure could result in error message ISPP321 if the combined length of panel statements exceeds 32K.

•The REXX can only process the ISPF dialog variables identified on the *REXX statement.

•The *ENDREXX statement is required to terminate REXX coded directly into a panel procedure.

•If EXIT rather than RETURN is used for interpreted REXX, the call to ISPPRXVP generated by ISPF to update the dialog variables will not be executed. Therefore updates to these variables from within the REXX will not be ‘seen’ by ISPF.



Panel REXX

- Restrictions . . .
  - Variable values must be in character format when passed, and must remain in character format
  - Variables values can be changed
  - Variable lengths can NOT be changed (prior to z/OS 1.9)
    - For implicitly defined variables that are fields on the panel, the length of the associated REXX variable is the larger of the length of the panel field and the length of the variable's value
    - For other implicitly defined variables, the variable length is considered to be the same as the length of its value
    - So, if necessary, pad variables to required length before passing them to the REXX

•Dialog variables passed to panel REXX must be in character format and must remain in character format.

•On releases prior to z/OS 1.9, panel REXX cannot change the length of the value of a dialog variable. This generally required the value of the dialog variable to be padded to the required length before being passed to panel REXX. After changes provided with z/OS 1.9, this is no longer necessary.

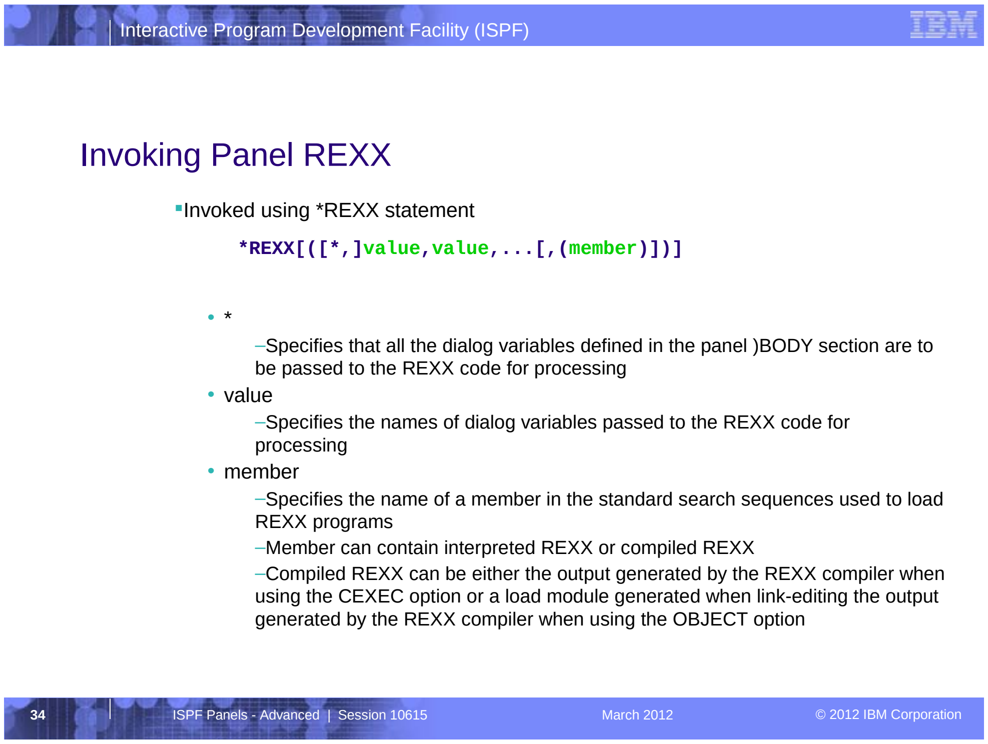
Panel REXX

- Return Codes
  - The exit should return one of the following values:

0	- Operation successful
8	- Panel REXX defined failure ISPF sets the .MSG control variable and redisplays the panel
20 or other	- Severe error in the exit routine

Dialog variable ZRXRC is provided by ISPF for panel REXX to set a return code. Dialog variable ZRXMSG is provided by ISPF for panel REXX to set the ID for a message to be displayed when the panel REXX sets a return code of 8.

Note: It is not necessary to specify ZRXRC or ZRXMSG on the *REXX statement.



Invoking Panel REXX

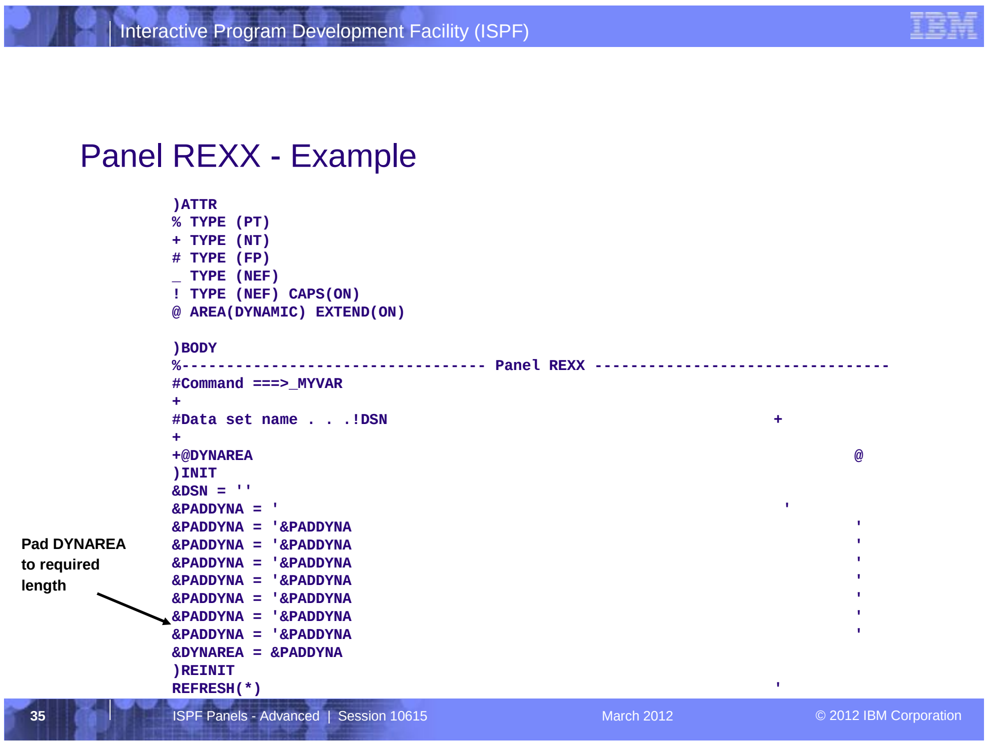
- Invoked using *REXX statement

```
*REXX[([*,]value,value,...[,(member)])]
```

- *
  - Specifies that all the dialog variables defined in the panel)BODY section are to be passed to the REXX code for processing
- value
  - Specifies the names of dialog variables passed to the REXX code for processing
- member
  - Specifies the name of a member in the standard search sequences used to load REXX programs
  - Member can contain interpreted REXX or compiled REXX
  - Compiled REXX can be either the output generated by the REXX compiler when using the CEXEC option or a load module generated when link-editing the output generated by the REXX compiler when using the OBJECT option

The *REXX statement is used to invoke panel REXX. It identifies the names of the dialog variables to be passed to the panel REXX for processing. A list of variable names can be specified or if all dialog variables referenced in the)BODY section may be processed by the panel REXX then just an * (asterisk) can be specified.

The REXX can be coded in the panel procedure immediately after the *REXX statement. Alternatively if there is a large amount of REXX code or the code is common amongst a number of panels the name of a member containing the REXX code can be specified on the *REXX statement.



Panel REXX - Example

```
)ATTR
% TYPE (PT)
+ TYPE (NT)
# TYPE (FP)
_ TYPE (NEF)
! TYPE (NEF) CAPS(ON)
@ AREA(DYNAMIC) EXTEND(ON)

)BODY
%---------------------------------- Panel REXX ---------------------------------
#Command ===>_MYVAR
+
#Data set name . . .!DSN                                           +
+
+@DYNAREA                                                                   @
)INIT
&DSN = ''
&PADDYNA = '                                                        '
&PADDYNA = '&PADDYNA                                                        '
&PADDYNA = '&PADDYNA                                                        '
&PADDYNA = '&PADDYNA                                                        '
&PADDYNA = '&PADDYNA                                                        '
&PADDYNA = '&PADDYNA                                                        '
&PADDYNA = '&PADDYNA                                                        '
&PADDYNA = '&PADDYNA                                                        '
&DYNAREA = &PADDYNA
)REINIT
REFRESH(*)                                                         '
```

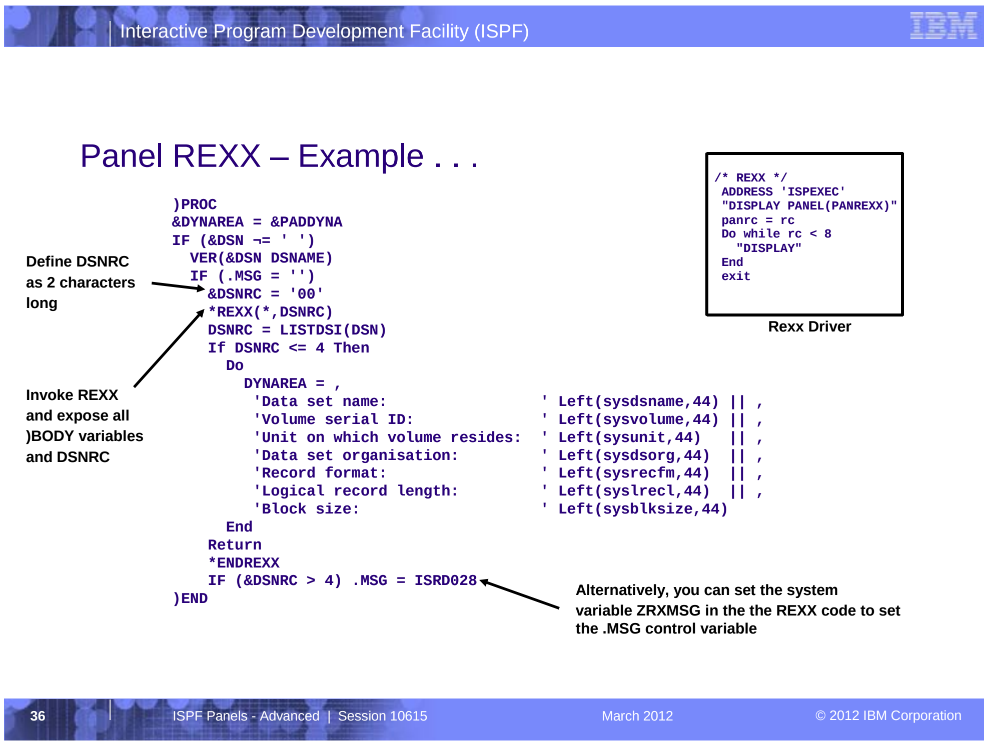
Pad DYNAREA
to required
length

This and the following page show an example of a panel using panel REXX.

The)BODY section contains some static text and 3 variables (MYVAR, DSN, DYNAREA), 1 of which (DYNAREA) is a dynamic area.

The)INIT section initializes the DSN and the DYNAREA variables to blanks, and pads the DYNAREA variable to the length required by the REXX code.

Note: The padding of variable DYNAREA is not required on z/OS 1.9 or later.



Panel REXX – Example . . .

```
)PROC
&DYNAREA = &PADDYNA
IF (&DSN ¬= ' ')
  VER(&DSN DSNAME)
  IF (.MSG = '')
    &DSNRC = '00'
    *REXX(*,DSNRC)
    DSNRC = LISTDSI(DSN)
    If DSNRC <= 4 Then
      Do
        DYNAREA = ,
         'Data set name:                 ' Left(sysdsname,44) || ,
         'Volume serial ID:              ' Left(sysvolume,44) || ,
         'Unit on which volume resides:  ' Left(sysunit,44)   || ,
         'Data set organisation:         ' Left(sysdsorg,44)  || ,
         'Record format:                 ' Left(sysrecfm,44)  || ,
         'Logical record length:         ' Left(syslrecl,44)  || ,
         'Block size:                    ' Left(sysblksize,44)
      End
    Return
    *ENDREXX
    IF (&DSNRC > 4) .MSG = ISRD028
)END
```

Define DSNRC as 2 characters long

Invoke REXX and expose all)BODY variables and DSNRC

Alternatively, you can set the system variable ZRXMSG in the the REXX code to set the .MSG control variable

```
/* REXX */
 ADDRESS 'ISPEXEC'
 "DISPLAY PANEL(PANREXX)"
 panrc = rc
 Do while rc < 8
   "DISPLAY"
 End
 exit
```

Rexx Driver

In the)PROC section the VER statement is used to validate the user entered a correctly formed data set name.
The dialog variable DSNRC is set to a value of "00" to cause the variable to be 2 characters in length.

The Panel REXX is invoked with the *REXX statement, exposing all variables, including the DSN variable, defined in the)BODY section and the DSNRC variable which was implicitly defined in the)PROC section.

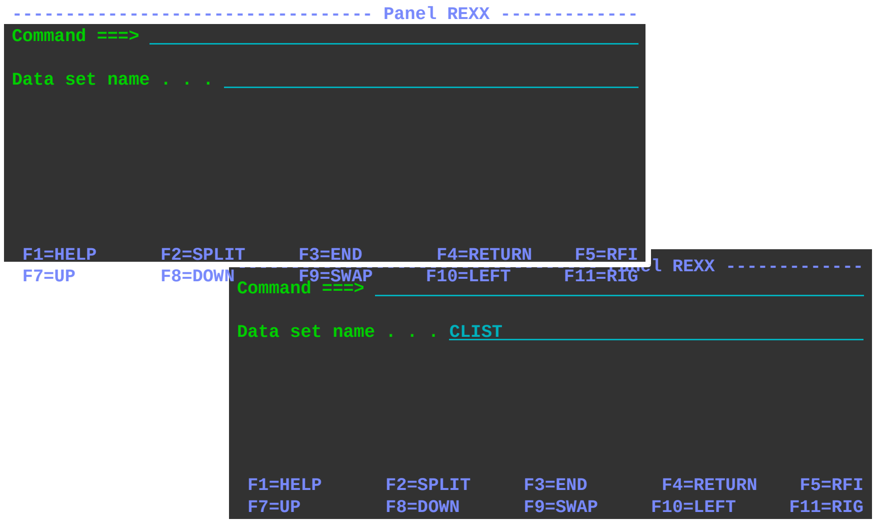
The REXX code calls the TSO function LISTDSI to determine if the data set exists, and to get the attributes of the data set returned in the REXX variables SYSDSNAME, SYSVOLUME, and so on.

The REXX code formats the information returned by LISTDSI into the DYNAREA variable and uses the REXX RETURN statement to return to ISPF.

The REXX code is terminated by the *ENDREXX statement.

If the REXX code returns a return code > 4, the .MSG control variable is set to ISRD028 to cause that message to be displayed.

Panel REXX – Example . . .

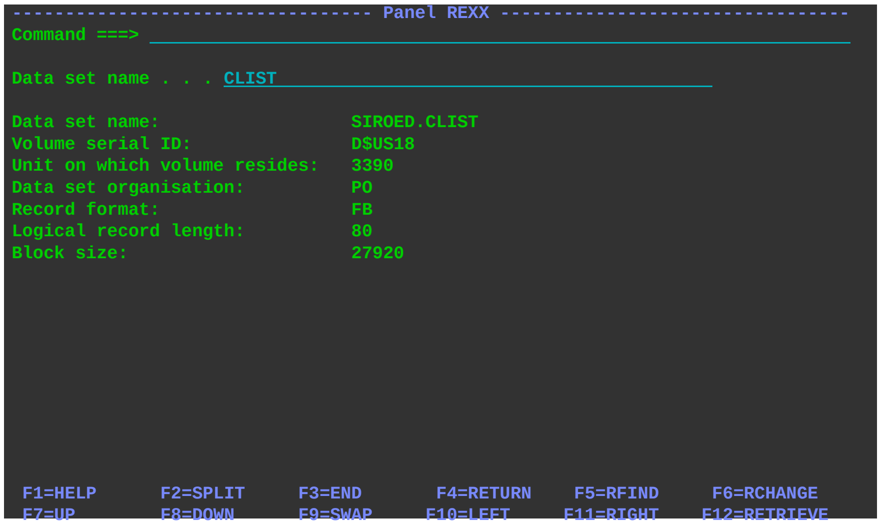


```
---------------------------------- Panel REXX ------------
Command ===> ______________________________________________

Data set name . . . _______________________________________





 F1=HELP       F2=SPLIT      F3=END        F4=RETURN     F5=RFI
 F7=UP         F8=DOWN       F9=SWAP       F10=LEFT      F11=RIG
```

```
---------------------------------- Panel REXX ------------
Command ===> ______________________________________________

Data set name . . . CLIST__________________________________



 F1=HELP       F2=SPLIT      F3=END        F4=RETURN     F5=RFI
 F7=UP         F8=DOWN       F9=SWAP       F10=LEFT      F11=RIG
```

This and the following page show an example of invoking our panel. When the panel is displayed the user enters the name of the data set that will have it’s attributes displayed.

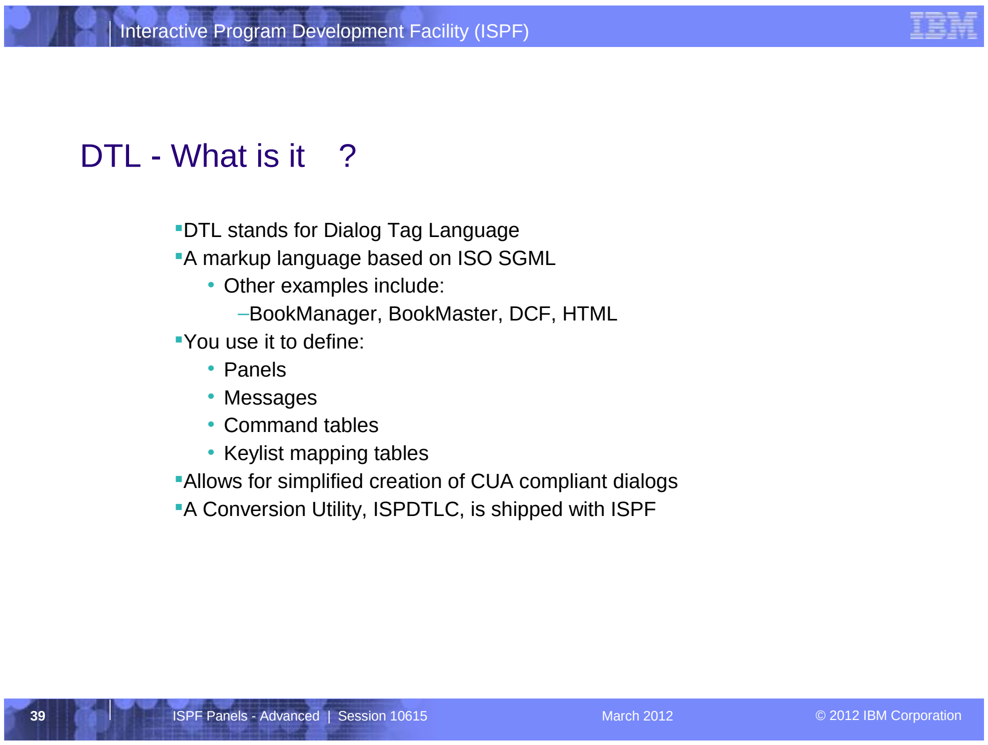
Panel REXX – Example . . .



```
--------------------------------- Panel REXX --------------------------------
Command ===> ________________________________________________________________

Data set name . . . CLIST

Data set name:                SIROED.CLIST
Volume serial ID:             D$US18
Unit on which volume resides: 3390
Data set organisation:        PO
Record format:                FB
Logical record length:        80
Block size:                   27920









 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

Shown here is the display of the data set attributes formatted into the dynamic area by the panel REXX code.

Note: ISPF ships a sample of panel REXX code in the source module ISRVCHIL in the ISPF REXX exec library (ISP.SISPEXEC). This panel REXX code is invoked in the)INIT section of the panel ISRVCALP, which is shipped in the ISPF panel library. This panel REXX code is used to enable color highlighting of the entries in the trace data set generated by the ISPVCALL utility. ISPVCALL is used by the ISPF product support team to assist in debugging customer reported problems.



DTL - What is it ?

- DTL stands for Dialog Tag Language
- A markup language based on ISO SGML
  - Other examples include:
    - BookManager, BookMaster, DCF, HTML
- You use it to define:
  - Panels
  - Messages
  - Command tables
  - Keylist mapping tables
- Allows for simplified creation of CUA compliant dialogs
- A Conversion Utility, ISPDTLC, is shipped with ISPF

DTL is the acronym for “Dialog Tag Language”. It is markup language based on ISO SGML.

ISO - International Standards Organization

SGML - Standard Generalized Markup Language

Other examples of SGML include Bookmanager, Bookmaster, DCF, HTML, ...

In ISPF you can use DTL to define:

•Panels

•Messages

•Command Tables

•Keylists

ISPF ships the conversion utility ISPDTLC to convert the DTL into standard ISPF panels, messages, command and keylist tables.

References

- ISPF Publications
  - SC34-4821 Dialog Developer's Guide and Reference
  - SC34-4816 Reference Summary
- Dialog Tag Language
  - SC34-4824 Dialog Tag Language Guide and Reference
  - S2627 SHARE presentation from Dallas (and Austin)

The main reference material to help you develop ISPF panels is the **ISPF Dialog Developer's Guide and Reference**

•Chapter 6 is the main reference material,   while

•Chapter 4 provides Common User Access (CUA) Guidelines, and

•Chapter 5 provides many guidelines and examples of ISPF panels

•Chapter 7 provides additional information for Help and Tutorial panels

For those wishing to venture into DTL, the Dialog Tag Language Guide will help you.