ISPF Panels
Beyond the Basics

Doug Nadel

Research Triangle Park
North Carolina
USIB3Z3Z at IBMMAIL

March, 2000

IBM

ISPF Panels
Beyond the Basics

Doug Nadel

Research Triangle Park
North Carolina
USIB3Z3Z at IBMMAIL

March, 2000

IBM

Contents ISPF

The following presentation is intended to be given on at the SHARE Winter 1999 Conference in San Francisco, California. This presentation discusses advanced techniques for using ISPF panels in ISPF version 4 and as such contains no IBM* restricted information.

The following terms, which are marked with an asterisk at their first occurrence, are trademarks of the International Business Machines Corporation.

IBM
CUA

The "notes" pages included with the foils indicate the topics to be discussed with each foil. They are not intended to be a complete script but they are representative of the presentation content.

IBM's plans are subject to change. Nothing in this document is intended to create any representations or warranties. IBM warranties are contained in the applicable IBM license agreements.

This document contains sample code. IBM PROVIDES THIS CODE ON AN "AS IS" BASIS WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Session Number: 2631

Subject: ISPF Panels - Beyond the Basics

Speaker: Doug Nadel

1. Basic panel concepts for DISPLAY panels
   (SELECT panels not covered)
2. CUA* Attributes
3. Windowing Concepts
4. Pulldowns
5. Scrollable areas
   - Dialog panels
   - Tutorial panels
6. Dynamic Areas
   - Scrolling
   - Shadow Variables
7. New Version 4 Constructs
8. Panel Exits
9. DTL comparisons

→ Features of ISPF Version 4 and higher

- Version 3.5 Service discontinued Nov, 1997
- Version 4.1 Service discontinued April, 1997

This presentation covers advanced panel topics using ISPF versions 3.3 through 4.5. I will be using REXX as the primary programming language because of its flexibility. However, for simple examples, I will also use CLIST and PL/I.

The topics which will be covered will include some basic panel concepts for review, Dynamic areas, Scrollable Areas, Tutorial Panels, Panel Exits, and some basic comparisons between 'old style' panels and Dialog Tag Language (DTL) panels.

Some of the new constructs for Version 4 will be discussed.

Types of panels

1. DISPLAY panels

   Invoked via 'ISPEXEC DISPLAY' service

   Primarily for data entry or output

2. SELECT panels

   Invoked via 'ISPEXEC SELECT' service

   Primarily for providing selection lists such as option menus.

3. Tutorial Panels

   Invoked by ISPF when HELP is requested

ISPF has three main types of panels: DISPLAY panels, SELECT panels, and tutorial panels.

Display panels are the panels that are shown with the 'ISPEXEC DISPLAY PANEL(...)' service. They are the most common ISPF panel type, and are used for data entry, progress messages, and output-only panels.

Select panels are used mainly for presenting options menus such as the ISPF/PDF Primary Option Panel. Select panels are invoked with the 'ISPEXEC SELECT PANEL(...)' service. The setting of the variable &ZSEL to an ISPF command string in the)PROC section of a select panel causes the invocation of a command, program or another select panel.

Although all of the concepts discussed in this presentation are applicable to select panels, they will not be covered as a separate topic here.

Tutorial panels are panels which are displayed by the ISPF tutorial processor (ISPTUTOR). They have a few idiosyncrasies, mostly in the area of special variables used by the tutorial processor. Again, although all of the concepts discussed in this presentation are applicable to tutorial panels, they will not be covered as a separate topic here, except where necessary.

Displaying a panel

1. Set up variables to be shown or updated

2. Call ISPEXEC DISPLAY service (or ISPLINK for compiled languages)

3. Check return code to see if user pressed END or ENTER

The general method of displaying a panel is to set up variables, call ISPEXEC or ISPLINK with the DISPLAY service and the panel name, and then, after the panel is displayed, check the return code from the display.

The variable setup is done with the VDEFINE or VREPLACE service in compiled languages and assembler. In REXX and CLIST, ISPF has access to the language variables, so no special variable creation is necessary. You can just assign values to variables, and as long as those variable names match ISPF conventions (up to 8 characters, starting with a letter, and containing national characters or numbers; REXX stem variables are not supported), the variables in the CLIST or EXEC will be available to ISPF and can be used in panels.

Panel Sections

)ATTR	Defines attribute characters, dynamic and graphic areas, etc.
)BODY	Defines where panel elements will be, among other things.
)INIT	Contains logic for before the panel is displayed the 1st time by the program.
)REINIT	Contains logic for before the panel is redisplayed by ISPF (for errors) or without a panel name.
)PROC	Allows logic for after the panel is displayed, such as variable verification, message setting, etc.
)END	Tells ISPF that the panel definition is complete.

Panels are divided into processing sections. Each section contains information that ISPF needs to create a screen image, and create, verify, or change data that is to be displayed on the screen. The main, or most common sections are listed here.

)ATTR This section defines how ISPF should use certain characters. The characters, which the panel developer chooses, can be used to define field characteristics, color, highlighting, etc., to tell ISPF where special types of fields (such as scrollable areas, dynamic areas, and graphic areas) begin and end, and how to display some of the data (Caps on or off for example).

)BODY Defines where panel elements will be, what the static text will look like, what the panel size is, where the command line is, and other things. When examining a panel, This is usually the first place to look.

)INIT Contains logic for before the panel is displayed the first time by the program. It is usually used to set variables which will be displayed, or to set variables which have special meaning to ISPF, to indicate such things as the associated tutorial panel name, whether the panel is a primary selection panel, etc.

)REINIT Contains logic for before the panel is redisplayed by ISPF (for errors) or without a panel name.

)PROC Allows logic for after the panel is displayed, such as variable verification, message setting, etc. Most of the time, the majority of panel logic will be placed in this area.

)END Tells ISPF that the panel definition is complete. Any characters after the)END are ignored by ISPF.

Some other panel sections

)PANEL	Defines panel as a CUA* panel. Specifies keylist used.
)CCSID	Defines Coded Character Set Identifier (CCSID)
)ABC	Defines action bar choices
)ABCINIT	Defines action bar initial processing
)ABCPROC	Defines action bar choice selection processing
)AREA	Defines a scrollable area
)LIST	Defines a selectable list (listbox in GUI)
)PNTS	Point and shoot field definition
)HELP	Associates fields with tutorial panel names (field level help).
)MODEL	Defines table display area for table display service TBDISPL.

```
)HELP
  FIELD(somefld) PANEL(somepan)
```

Other less commonly used panel sections are listed here.

)PANEL Defines the panel as a CUA panel. It also Specifies the name of the keylist to be in effect while this panel is displayed.

)CCSID Defines Coded Character Set Identifier (CCSID).

)ABC Defines action bar choices on CUA panels.

)ABCINIT Defines action bar initial processing. It is required if the)ABC section is present. This defines the processing to take place before that action bar choice is displayed.

)ABCPROC Defines action to be taken when the action bar choice is selected by the user.

)AREA Defines a scrollable area. The contents of this section often look like the contents of the)BODY section. In effect, it is a small, scrollable)BODY section that is imbedded into the real)BODY section.

)HELP Associates fields in the)BODY or)AREA sections with tutorial panel names. This is how field level help is implemented. If the cursor is on a field which has a tutorial panel associated with it, and the user presses HELP, the associated panel is displayed under the control of the ISPF tutorial processor.

)MODEL Defines the table display area for table display service TBDISPL. The)MODEL section provides ISPF with a list of table variables to display, and the attributes (color, case, highlighting, etc.) of each column of variables. More than one line of variables can be supplied in the)MODEL section.

- Defines attribute characters used to:
  - Define input and output field characteristics
  - Start and end of dynamic areas
  - Highlighting within dynamic areas
  - Start and end of scrollable areas
  - Start and end of graphic areas
  - Start of action bar pull down names
  - Characteristics of pull down choices

In the)ATTR section, you will find the definitions of attribute characters. Often you will see sections that define how a particular character in the)BODY section will affect highlighting or color on the panel. The most common of these is to use the percent sign (%) for output highlighting.

In addition to that type of definition, characters can be assigned to have effects in dynamic areas, and shadow variables, as well as to define the starting and ending locations of special areas.

- Defines where fields are physically placed on the screen and some of the characteristics (from)ATTR section)
  - Static text
  - Input and output fields
  - Dynamic areas
  - Scrollable areas
  - Graphic areas
  - Action bar pull downs
- Also defines window size, command line, and other things.

In addition to the placement of fields, (static, input, dynamic, scrollable, graphic, action bar choices, etc.), the)BODY statement defines what the windows size will be if this panel is displayed in a window (using the ADDPOP/REMPOP services), what field will act as the command line, how to expand the panel for large screens (3278 Mod 5, etc.), where the message fields should be located, and some DBCS (double byte character support) characteristics.

Panels and messages can be displayed in windows.

- **Panels should have WINDOW(columns,rows) coded in the)BODY section.**

- **Programs must place ADDPOP and REMPOP around DISPLAY or SELECT.**

```
ISPEXEC ADDPOP  /* add pop-up window */
ISPEXEC DISPLAY PANEL(panelname)
ISPEXEC REMPOP  /* remove pop-up window */
```

Both panels and messages can be displayed in windows. You may specify the size of the window in the)BODY section of the panel. For example:

```
)BODY WINDOW(46,9)
```

would tell ISPF to display a window which is 46 columns wide and 9 columns deep **not including the borders and associated attribute bytes**.

If a program is to display a panel in a window, the program must issue the ADDPOP service before the window is displayed, and the REMPOP service after the window is displayed. Otherwise the panel is shown in a full screen (not in a window).

When deciding on the size of the panel, remember that some users will run with the Program Function keys showing (PFSHOW ON) and you will want to leave room for the PF key display. Also remember that ISPF requires 2 bytes on either side and 2 lines on the top and bottom for the window border.

Messages can be placed in windows also. This is accomplished in one of two ways:
1. Set the .TYPE and or .WINDOW values in the message definition.
2. Create long messages that are greater than the width of the panel.

Windows can be moved by the user.

1. Place cursor on a panel border.

2. Press enter.

3. Place cursor where you want the upper left corner.

Only top-most window can be moved.

Windows can be moved by the user of your application. They may want to do this to see information that is behind the window, or in the case of field level error messages, to move the window out of the way so that they can enter data.

Window movement is accomplished by telling ISPF to move the window (by placing the cursor on the window border and pressing enter), and moving the cursor to the location where the upper left corner of the window should be moved to. ISPF will make any necessary adjustments to insure that the whole window still fits on the screen.

CUA Attributes allow you provide color consistency for fields which have the same function.

- **Input fields**
- **Panel Titles**
- **Warning messages**
- **many others**

Attributes are specified in the)ATTR section

```
)ATTR
   % TYPE(ET)                          /* emphasized text    */
   _ TYPE(NEF) PAD(USER) CAPS(ON) /* normal entry field */
   + TYPE(NT)                          /* Normal text        */
)BODY
```

Each user can customize how each field looks using the CUAATTR command.

ISPF allows you to use new attributes in your panel definitions to allow for consistent color presentation across the ISPF session. These attributes, referred to as CUA Attributes are set in the)ATTR section, and are types. Each corresponds to a CUA panel element.

Individual users can set the colors for individual CUA attributes by using the CUAATTR command from any command line or from the settings panels (option 0).

Note that you can override the default attribute characters in the)ATTR section, as the example shows.

Non-CUA attributes

Value	Usage
TEXT	Text (protected) field
INPUT	Input (unprotected) field
OUTPUT	Output (protected) field
DATAIN	Input (unprotected) field in a dynamic area
DATAOUT	Output (protected) field in a dynamic area
CHAR	Character attributes in a dynamic area
GRPBOX	Group box
PS	Point-and-shoot

CUA attributes

Value	Usage
AB	AB unselected choices
ABSL	AB separator line
CEF	Choice entry field
CH	Column heading
CT	Caution text
DT	Descriptive text
EE	Error emphasis
ET	Emphasized text
FP	Field prompt
LEF	List entry field
LI	List items
LID	List item description
NEF	Normal entry field
NT	Normal text
PIN	Panel instruction
PT	Panel title
RP	Reference phrase
SAC	Select available choices
SC	Selected choice
SI	Scroll information
SUC	Select unavailable choices
VOI	Variable output information
WASL	Work area separator line
WT	Warning text

This foil lists the attributes, both CUA and Non-CUA, which can be specified by TYPE() in the)ATTR section of the panel definition.

There are many options for each of the attribute types. See the manuals for details on each attribute.

Action bar on top of the panel

Each action bar may have multiple choices when selected.

A panel with pulldowns:

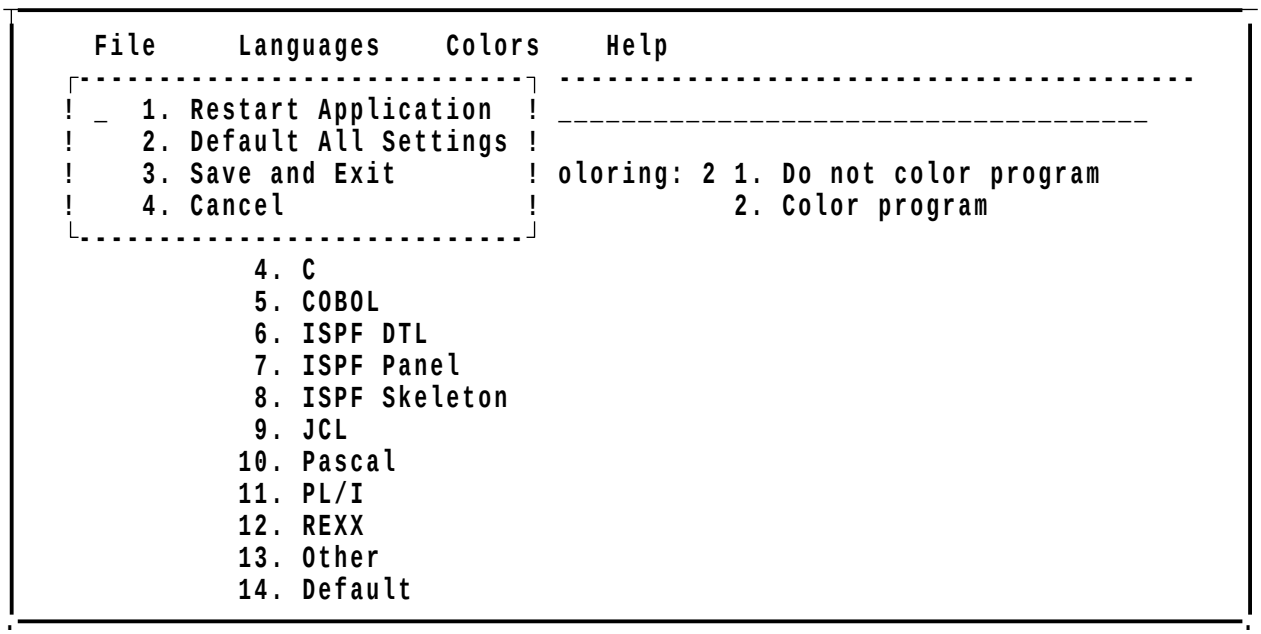
```
   File     Languages    Colors    Help
------------------------------------------------------------------------
Command ===> ____________________________________________________

Language:  1 1. Automatic      Coloring: 2 1. Do not color program
             2. Assembler                  2. Color program
             3. BookMaster
             4. C
             5. COBOL
             6. ISPF DTL
             7. ISPF Panel
             8. ISPF Skeleton
             9. JCL
            10. Pascal
            11. PL/I
            12. REXX
            13. Other
            14. Default
```

Pulldowns allow you to have an action bar at the top of the screen and each action bar choice may have multiple actions associated with it. You may code pulldowns in either panel language or DTL. What I will be showing here is how to code them in panel language.

Selection of an action bar choice is done by placing the cursor on the action bar choice and pressing enter. When the enter key is pressed, a pulldown is shown which gives additional choices which can be selected by number or by cursor positioning.

The example panel shows how a panel with an action bar might look.

The same panel with 'File' selected.



This shows how the panel would look once the 'File' choice is selected. The pulldown, in this case, contains 4 items which can be selected by number or by choice. If the user presses the END, Exit, or CANCEL key, the pulldown is removed with no action taken.

While the pulldown is displayed, the rest of the panel becomes read-only.

- Panel defines each pulldown and its actions

- There are 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

- The action bar must be added to the)BODY section

A panel can define pulldowns using the)ABC,)ABCINIT, and)ABCPROC panel sections.

)ABC DESC(...) Defines action names and response. This is done with keywords:

- PDC DESC(...) defines the name shown on the panel.
- ACTION RUN(...) defines a string to be placed on the command line when this action is selected.

)ABCINIT Allows setup before pulldown display. This allows you to alter 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 be needed in special cases.

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 foil shows an example of an)ABC section of a panel. It has 4 actions, defined by the PDC (pull down choice) keyword. Each PDC section has an ACTION section which tells ISPF what action to take when the pulldown choice is selected.

When a pulldown 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 and then simulates an Enter key. This means that you can have the processing be handled by the command table, program logic, or panel logic.

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).

- Add attributes for types AB, ABSL to)ATTR

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

- Add)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 >
```

- Add action bar and separator line to top of)BODY

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

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

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

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 DESC in)ABC to menu name
   - Remember to assign .ZVERB in the)ABCINIT section
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 pulldown from the)ABC section
4. Add an action bar separator line (optional)

   To process the action bar selections, alter your program or panel logic, or add selections to your command table.

```
)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
```

A complete pulldown example /Notes

This is a working example of a panel which will appear in a window (if an ADDPOP was issued). It has 2 pulldowns, Menu and Help. It shows the use of some CUA Attributes, overriding the defaults (&, + , and _) and it shows an example of adding processing to the)ABCINIT section for the Help pulldown.

If the Help pulldown is selected, ISPF will automatically simulate an extra enter key because of the .RESP=ENTER in the)ABCINIT section. Thus the pulldown 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 pulldown is selected for the 1st time, ISPF will run the HELP command and will never actually show the pulldown. This is a method you can use to have action bar choices automatically take some action.

The panel looks like this when displayed:

```
  Menu   Help
 ---------------------------------------------
              Panel Title
 Command ===> _______________________________

 This is normal text.  Right? ___  (Yes, No)
```

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

```
  Menu   Help
 ┌──────────────┐ ------------------------------
 ! _  1. Save   ! el Title
 !    2. End    ! _______________________________
 !    3. Cancel !
 └──────────────┘ ext.  Right? ___  (Yes, No)
```

What is a Scrollable Area?

- Like a)BODY section of indeterminate length
- Lets you use one panel instead of many
- Uses same basic syntax as an 'old')BODY section
- ISPF handles UP/DOWN scrolling if the image does not fit on the screen or in the window.
- ISPF gives an indicator of whether there is more data above or below the current data.

```
 More:   +     You can only scroll forwards
 More: –       You can only scroll backwards
 More: – +     You can scroll backwards or forwards
(Blank)        All data is currently shown
```

  - → ISPF updates the indicator with any interaction (split screen, PF keys, etc)

A scrollable area can be compared to the)BODY section of the panel. However, it can be much longer then the)BODY section, and you can have multiple scrollable areas defined within the)BODY section.

This allows you to define one panel, and allow your end-user to scroll the information, rather than defining multiple panels.

They are very simple to define. The)BODY section definition resembles that of a dynamic area, and the)AREA section, where you define the information to be displayed within the area, uses the same syntax that you would use if you defined it within the)BODY section.

The two major differences between Dynamic and Scrollable areas area that 1) the scrollable area definition must be done at panel creation time, rather than at run-time, like the dynamic areas. However for scrollable areas, ISPF handles all of the scrolling of the data, along with updating the scroll indicator whenever any interaction with the panel occurs.

Scrollable Areas ...

Why use scrollable areas?

- Applications with large panels can run on small screens
- Less application logic (one panel rather than multiple)
- Simpler implementation than dynamic areas
  - Familiar syntax
  - Scrolling handled by ISPF
- Easier tutorial implementation
  - One panel rather than multiple, no need to use ZCONT in tutorials

Scrollable areas are extremely useful for applications which must run on multiple screen sizes and/or have large amounts of data that is grouped. For example, if you have a data entry application, where the information to enter does not fit on a normal terminal screen, you can either create multiple panels, and have your application display them, or create a scrollable area with all the information and have the end-user scroll through it.

Advantages to the scrollable area are that it allows your application to have less logic to display the information. In this example, your application would need to determine what the end-user wants to do and put up the appropriate panel each time. Also, the syntax is much the same as that of the)BODY section, and the implementation is easier than that of dynamic areas.

Scrollable areas are also available in tutorials, and allow the information on one panel rather than many.

-)ATTR section must contain AREA definition

```
)ATTR
 @ AREA(SCRL)   EXTEND(ON)
```

EXTEND	ON|OFF - Causes the scrollable area to extend to the last line on the screen if necessary.

-)BODY section must use the area character, and an area name:

In the example, the scrollable area is bounded by '@' characters and uses the area name 'surv'.

```
+ @scarea                                    @
```

- The)AREA section follows the)BODY section:

```
)AREA scarea
+First Name%===>_FNAME    + (nickname)
+Last  Name%===>_LNAME    + (surname)
```

To define a scrollable area, the)ATTR section much contain an attribute byte definition for AREA(SCRL), in the same manner as dynamic areas. You may specify EXTEND either OFF or ON, with OFF being the default.

The)BODY section contains the definition for the placement and size of the scrollable area, in the same manner as the dynamic area definition. You specify an area name in the)BODY section definition, which maps to an)AREA section, defined later, which contains the data to be displayed within the area.

The)AREA section is defined like a miniature)BODY section, which fits into the scrollable area dimensions in the action)BODY section of the panel.

You may have multiple scrollable areas on the panel. If more than one scrollable area exists, cursor placement determines which area to scroll.

A scrollable area cannot contain another area with scroll capabilities such as another scrollable area, a scrollable dynamic area, or a scrollable graphic area.

- a Panel may have more than 1 scrollable area

  → Cursor position is used to determine which area to scroll

- Area definition in)BODY may be more than one line.

- Scrollable areas may not contain other scrollable areas but they MAY contain dynamic areas with EXTEND(OFF).

```
)ATTR
 $ AREA(SCRL) EXTEND(ON)
 # TYPE(INPUT) PAD(_) CAPS(OFF)
 @ TYPE(INPUT) PAD(_) CAPS(ON)
)BODY
%--------------------  SUPERSEEDY SEED COMPANY  ----------------------%
%COMMAND ===>_ZCMD                                                    +
+

+  Press%PF8+to go forward,%PF7+to go back, %PF3+to exit
 $surv                                                                $
)AREA surv
+   Thank you for participating in this survey.  Please rest
    assured that the information you provide will remain%strictly
    confidential+unless we feel like divulging it.

%   Information about you
+
+      Last  Name ............#lname                          +
+      First Name ............#fname                          +
+      Middle Initial.........@z+

+      Street Address.........#saddr                              +
+      Apartment number.......#aaddr                              +
+      City...................#caddr                              +
+      State or Province......@z +       (Use 2 letter abbreviation) +
+      Zip or Postal Code.....@pcode   +

+      Occupation.............#occ                                  +
+      Number of years at
+        your current job.....#yr+
       Annual income          #inc    +  (optional)
+      Size of your garden....@gsz     + (Porch, Backyard, Farm, None)
+
%   Information about our services

+      Where did you hear about our services?#howhear                    +
+      What similar services have you used?  #simserv                    +
```

Scrollable panel example (1 of 2) /Notes

This part of the panel shows the)ATTR section, the)BODY section, and the beginning of the)AREA section.

The area name 'surv' is used in both the)BODY section and the)AREA tag. This is how the connection is made between area definitions in the)BODY and)AREA sections.

Note that there is an attribute character in column 1 for each of the lines. This is not required, but it is generally a good idea because the beginning of the scrollable area uses the same attributes that precede the scrollable area unless the area contains attributes of its own.

The area can take up all or part of the line in the)BODY section. This example illustrates that, although the)AREA section contents start in column 1, the display starts where the attribute byte is in the)BODY section.

```
%      Please rate the following categories on a scale of 1 to 10.
%          1 <--------------- 6 -----------------> 10
%        Poor               Average               Excellent
%      Please use a 0 for 'Not Applicable'

 +         Catalog .....................#Z +
 +         Sales representatives.......#Z +
 +         Time to process your order..#Z +
 +         General merchandise.........#Z + (General seeds and plants)
 +         Specialty merchandise.......#Z + (Rare plants, tools, etc.)
 +         Return Policy...............#Z +
 +         Complaint Department........#Z +
 +         Our Lawyers.................#Z +

%   Information about your buying habits

+      Do you subscribe to our%gardening fashions catalog? @GF +
+      Do you buy garden%supplies for other people? @GO +
+      Would you be interested in buying a%really big bridge? @BB +
)INIT
  .ZVARS = '(MI,PADDR,CAT,REP,TIME,GEN,SPEC,RETP,COMP,SUIT)'
)PROC
  VER (&LNAME,NB)
  VER (&FNAME,NB)
  VER (&SADDR,NB)
  VER (&CADDR,NB)
  VER (&PADDR,NB,LIST,AK,AS,HI,GA,NC,SC,NY,KY,QC,BC,WA,HI,MI,TX,AL,FL)
  VER (&YR,NUM)
  VER (&INC,NUM)
  &GSZ = TRANS( TRUNC(&GSZ,1) P,PORCH B,BACKYARD, F,FARM N,NONE *,'?')
  &GF  = TRANS( TRUNC(&GF,1) Y,YES N,NO *,'?')
  &GO  = TRANS( TRUNC(&GO,1) Y,YES N,NO *,'?')
  &BB  = TRANS( TRUNC(&BB,1) Y,YES N,NO *,'?')
  VER (&GSZ,NB,LIST,PORCH,BACKYARD,FARM,NONE)
  VER (&CAT  ,NB,RANGE,0,10)
  VER (&REP  ,NB,RANGE,0,10)
  VER (&TIME ,NB,RANGE,0,10)
  VER (&GEN  ,NB,RANGE,0,10)
  VER (&SPEC ,NB,RANGE,0,10)
  VER (&RETP ,NB,RANGE,0,10)
  VER (&COMP ,NB,RANGE,0,10)
  VER (&SUIT ,NB,RANGE,0,10)
  VER (&GF   ,NB,LIST,YES,NO)
  VER (&GO   ,NB,LIST,YES,NO)
  VER (&BB   ,NB,LIST,YES,NO)
)END
```

Scrollable panel example (2 of 2) /Notes

This section shows the remainder of the)AREA section, the)INIT section and the)PROC section.

The)INIT section in this example only contains the assignment of names to 'Z' variable place holders. The assignment follows the normal conventions; Each variable named 'Z' in the BODY, and AREA is associated with a real variable name in the order it is found in the panel definition. (This allows you to have fields with names that are larger than the field size).

Verification is done as if the area was just part of the regular)BODY section.

```
--------------------  SUPERSEEDY SEED COMPANY  ---------------------
COMMAND ===>


  Press PF8 to go forward, PF7 to go back,  PF3 to exit
                                                          More:     +
    Thank you for participating in this survey.  Please rest
    assured that the information you provide will remain strictly
    confidential unless we feel like divulging it.

    Information about you

       Last  Name ..... John_____________________________
       First Name ..... Public___________________________
       Middle Initial..... Q

       Street Address..... 100 Main Street_____________________
       Apartment number....... 1B_______________________________
       City................... Anytown__________________________
       State or Province...... QC        (Use 2 letter abbreviation)
       Zip or Postal Code..... H3C3A8__

       Occupation............. Village Idiot_____________________
       Number of years at
```

```
--------------------  SUPERSEEDY SEED COMPANY  ---------------------
COMMAND ===>


  Press PF8 to go forward, PF7 to go back,  PF3 to exit
                                                          More:   - +
       Number of years at
         your current job..... 30
       Annual income          25_____   (optional)
       Size of your garden.... FARM____  (Porch, Backyard, Farm, None)

    Information about our services

       Where did you hear about our services? Radio Free Apex
       What similar services have you used?   _____________________________

       Please rate the following categories on a scale of 1 to 10.
           1 <--------------- 6 -----------------> 10
          Poor              Average              Excellent
       Please use a 0 for 'Not Applicable'

           Catalog ..................... 10
           Sales representatives....... 9_
           Time to process your order.. 1_
```

These screens show the display of the scrollable area panel as the end-user enters data into the input fields as scrolls. Notice that the scroll indicator (the More: - +) is updated according to whether or not there is additional information in each direction (forward and backward).

Scrolled to the bottom

```
--------------------  SUPERSEEDY SEED COMPANY  ---------------------
COMMAND ===>
THERE IS NO ADDITIONAL INFORMATION BELOW THIS LINE.

  Press PF8 to go forward, PF7 to go back,  PF3 to exit
                                                          More:   -
          1 <--------------- 6 -----------------> 10
         Poor              Average               Excellent
       Please use a 0 for 'Not Applicable'

           Catalog ..................... 9
           Sales representatives....... 1_
           Time to process your order.. 1_
           General merchandise......... 1_  (General seeds and plants)
           Specialty merchandise....... 1_  (Rare plants, tools, etc.)
           Return Policy............... 1_
           Complaint Department........ 1_
           Our Lawyers................. 10

     Information about your buying habits

        Do you subscribe to our gardening fashions catalog?  NO_
        Do you buy garden supplies for other people?  NO_
        Would you be interested in buying a really big bridge?  YES
```

When there is no more information to display, and the user requests a scroll function in that direction, a message is displayed stating that there is no more information to scroll to in that direction.

- Work the same as in other panels, except

  → UP/DOWN scrolling is accomplished with RIGHT/LEFT keys.

  This is in keeping with the functions of LEFT and RIGHT in tutorials.

Scrollable Areas in Tutorials /Notes

You can define scrollable areas within your tutorial panels, as well. If you have tutorial panels which relate to the same field that you currently have continued with ZCONT, you can put all the information on one panel using scrollable areas, and not have to continue them in that manner. This allows for cursor-dependent scrolling, also.

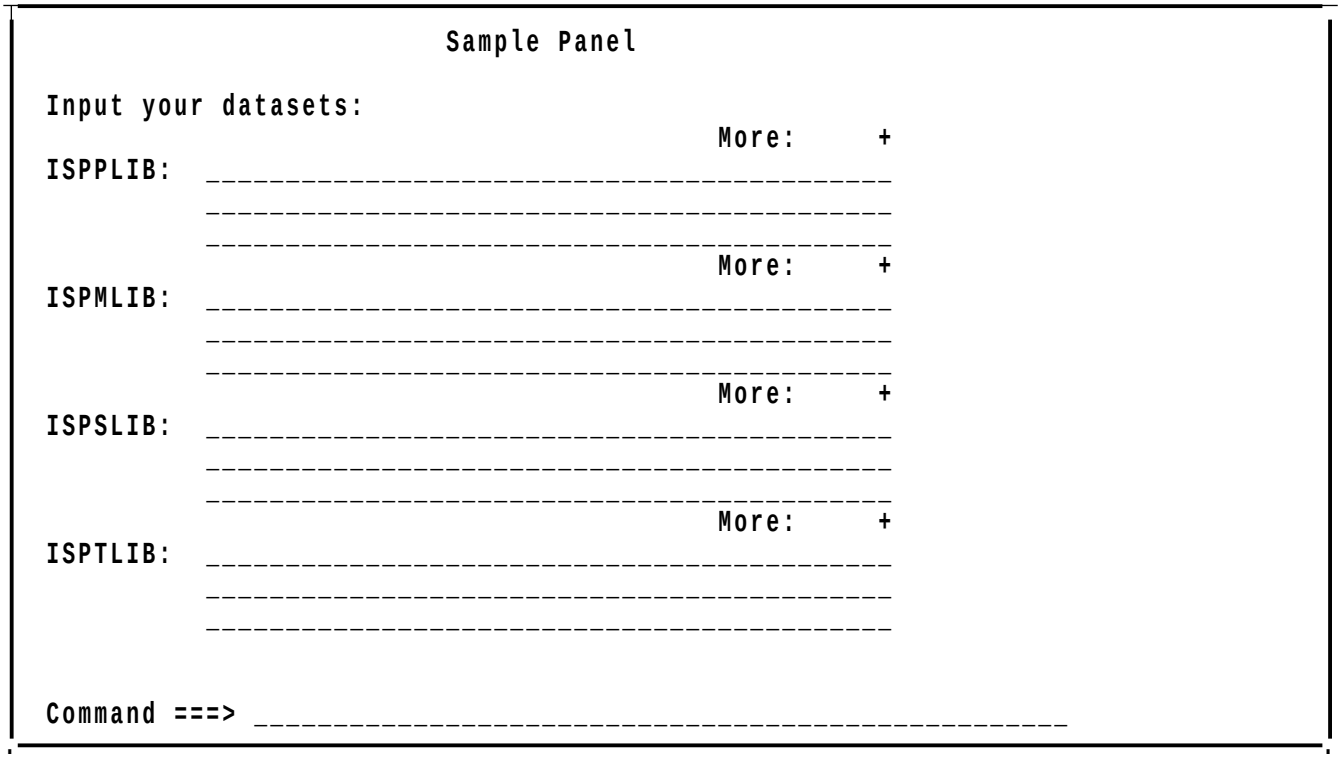
Up and down scrolling is done with the RIGHT and LEFT commands, to keep it consistent with previous function of tutorial panels. You can also use the ENTER key to scroll down.

```
)PANEL
)ATTR
 | TYPE(PT)
 . TYPE(FP)
 ? TYPE(NT)
 _ TYPE(NEF) PADC(_)
 $ AREA(scrl)
)BODY WINDOW(65,22) CMD(ZCMD)
|                        Sample Panel
?Command ===>_ZCMD
?
?Input your datasets:
          $scrl1 - - - - - - - - - - - - - - - - - - - - $
.ISPPLIB: $                                             $$
          $                                             $$
          $                                             $$
          $scrl2 - - - - - - - - - - - - - - - - - - - - $$
.ISPMLIB: $                                             $$
          $                                             $$
          $                                             $$
          $scrl3 - - - - - - - - - - - - - - - - - - - - $$
.ISPSLIB: $                                             $$
          $                                             $$
          $                                             $$
          $scrl4 - - - - - - - - - - - - - - - - - - - - $$
.ISPTLIB: $                                             $$
          $                                             $$
          $                                             $$
)AREA scrl1
._plibds1                                       ?
._plibds2                                       ?
._plibds3                                       ?
._plibds4                                       ?
._plibds5                                       ?
._plibds6                                       ?
._plibds7                                       ?
._plibds8                                       ?
._plibds9                                       ?
._plibds10                                      ?
)AREA scrl2
._mlibds1                                       ?
._mlibds2                                       ?
      ...          etc...for each of the 4 scrollable areas
)INIT
&ZCMD = ' '
)PROC
)END
```

Multiple scrollable areas /Notes

This is a simple example showing how to use multiple scrollable areas on a single panel. It is used to input dataset names for the normal ISPF DDnames. Up to 10 datasets are available for each of the four DDnames given. All this information would not fit on one panel, but the information is all related, and is an excellent use of a scrollable area.

We could have made this one long scrollable area, but didn't want to force the user to scroll if they only had one or two datasets for each DDname.



- First line is ALWAYS reserved for scroll indicator
  - If all data is displayed, the line is left blank
  - Notice: The field prompts outside of the scrollable area are coded on the 2nd line for this reason

Notice that the DDnames (ISPPLIB, etc) are coded outside of the scrollable area on the second line. This is due to the fact that ISPF reserves the first line of the scrollable area for the scroll indicator, and we wanted the DDname to be displayed on the line with the first input field. They are outside of the scrollable are so that they never get scrolled out of sight of the end-user.

What is a Dynamic Area?

- Area in)BODY section to be filled in at run-time.

- Lets your program decide how your panel will look.

- Offers a great deal of flexibility!

  Program retains control of:

  - Colors (both field color and at the character level)
  - Field size
  - Input fields
  - Verification
  - Scrolling (Up, Down, Right, Left)

Dynamic areas allow you to define what a part of your panel looks like at run time. That means you can create input or output fields under program control. It also means that you can have complete control of the characteristics of every character position in the dynamic area. For example, you could display a three character word and have every word be a different color.

Dynamic areas work by allowing you to tell ISPF the name of a variable which contains the data to be displayed. Within that variable, you can imbed attribute characters defined in the)ATTR section which will have special meaning to ISPF, such as 'start display in green', or 'start an input field'. The attributes types are DATAIN, CHAR, and DATAOUT, and can be used to define input fields, text, and output fields, just like in the)BODY section, as well as the colors to be associated with those fields/text.

You can also define USERMOD and DATAMOD fields within your dynamic area. These allow your program to determine a) If a user has entered anything in the field, even if it is the same characters as are currently there, b) if the user changed anything in the field.

Dynamic areas offer a great deal of flexibility. For example, if you need a table with right/left scrolling capability, you can use dynamic areas to implement it. If you just want to highlight titles with the 1st character in a different color, you can use dynamic areas without even having your program know about it (This is done by setting up all the necessary variables in panel logic). A simple example of this will be shown later on.

Why use dynamic areas?

- Special display requirements
  - Field level color control
  - Character level attribute support
  - Best use of different screen sizes.
  - Nonstandard "table" display
    - Right-Left scrolling
    - Field (or character) level highlighting and colors
    - Better control of what information gets displayed

Dataset List (option 3.4) uses a dynamic area.

There are many times you may want to use dynamic areas.

If your screen sizes are varied (24 lines, 32, 43, etc), you may want to use dynamic areas to display your data. If you want character level highlighting you will need to use dynamic areas. And since your program decides what fields get displayed, you may only generate certain fields for certain users. This can be done without complicated panel logic, although it does increase the complexity of the program which drives the panel.

```
)ATTR DEFAULT($+_)
  _ TYPE(INPUT)   INTENS(HIGH) CAPS(ON)
  ! AREA(DYNAMIC) SCROLL(ON) EXTEND(ON)
 01 TYPE(DATAOUT) INTENS(LOW)                          /* LEFT HEADER     */
 02 TYPE(DATAOUT) INTENS(LOW)          COLOR(PINK)     /* MESSAGE AREA    */
 03 TYPE(DATAOUT) INTENS(LOW)          COLOR(TURQ)     /* VOLUME NAME     */
 04 TYPE(DATAOUT) INTENS(LOW)                          /* SEPARATOR LINES */
 05 TYPE(DATAIN) INTENS(LOW) CAPS(ON) COLOR(YELLOW)   /* DATASET NAME    */
 06 TYPE(DATAOUT) INTENS(LOW)          COLOR(GREEN)    /* TRK, % XT, DEV  */
 07 TYPE(DATAOUT) INTENS(LOW)          COLOR(WHITE)    /* DSORG, RECFM ETC*/
 08 TYPE(DATAOUT) INTENS(LOW)          COLOR(RED)      /* CREATE, EXP, ETC*/
 09 TYPE(DATAOUT) INTENS(LOW)          COLOR(BLUE)     /* END OF DATA LINE*/
 10 TYPE(DATAOUT) INTENS(LOW)          COLOR(RED)      /* BELOW END       */
)BODY WIDTH(80)
$&ZDLTITLE                                                                +
$COMMAND ===>_ZCMD                                        $SCROLL ===>_ZUSC+
+
!ZDATA ------------------------------------------------------------------------!
!                                                                              !
! ----------------------------------------------------------------------------!
)INIT
  .HELP = ISR34014
)PROC
  VPUT (ZUSC) PROFILE
)END
```

This panel is an old modified copy of the product panel ISRUDSL0. This panel is what the data set list utility (OPTION 3.4) of ISPF/PDF uses to display the list of datasets. When you scroll left or right, option 3.4 does not change panels, it only changes the value of the variable ZDATA which is then displayed in a dynamic area on this panel.

Option 3.4 was written to use different attribute characters for different items in the list. Using this panel, you can show the header in the default low intensity color, the message areas in pink, the volume names in turquoise, the data set names in yellow, etc.

To see a good example of how dynamic areas work scroll right and left in option 3.4 Data Set List.

-)ATTR section must contain AREA definition

```
)ATTR
 @ AREA(DYNAMIC)   SCROLL(ON) EXTEND(ON)
```

-)ATTR section may contain DATAIN and DATAOUT definitions

```
01 TYPE(DATAOUT)  COLOR(RED)
02 TYPE(DATAIN)   COLOR(GREEN) PAD(_)
```

-)BODY section must use the attribute character defined for the dynamic area, and a program variable, which will contain the data which is to be put into this area at display-time:

  In this example, the dynamic area is bounded by '@' characters and displays the variable 'DYNAREA'.

```
+This is extendable  @DYNAREA                    @
```

  → The default color is whatever preceded the dynamic area in the)BODY section.

Panels with dynamic areas on them need two basic things that 'regular' panels do not have. They need changes to the)ATTR section and changes to the)BODY section.

The)ATTR section must have a character defined which will tell ISPF that when this character is encountered, a dynamic area starts, and when it is encountered again (must be on the same line), that is where the dynamic area ends. Multiple lines can be specified (in the shape of a "box", and ISPF will wrap the text specified by the variable within the box.

You can also define characters that, when found in the dynamic area will affect the display. The attributes are similar to regular)BODY character attributes except that instead of using TYPE(TEXT) and TYPE(INPUT), you use TYPE(DATAOUT) and TYPE(DATAIN). This means that you can use the same character with different meanings in dynamic areas and the)BODY itself. You can, for example, have the letter R just be displayed in the)BODY, but if it is in the dynamic area variable, it will start showing output in the color RED.

```
)ATTR
 ...
  R  TYPE(DATAOUT) COLOR(RED)
 ...
)BODY
```

The)BODY section must define where the dynamic area is, and what variable will contain the data that is displayed. The definition starts and ends with the character defined as AREA(DYNAMIC) in the)ATTR section. Immediately following the first instance of that character is the name of the variable.

```
)ATTR
 @ AREA(DYNAMIC) SCROLL(ON) EXTEND(ON) DATAMOD(15)
```

SCROLL ON|OFF - Tells ISPF to let the application know when the user requests a scroll of the dynamic area.

EXTEND ON|OFF - Causes the dynamic area to extend to the last line on the screen.

DATAMOD Character or hex code - Defines a character to be set in your variable if the data following it is modified by the user.

USERMOD Character or hex code - Defines a character to be set in your variable if the area is typed in, even if the value is not changed.

USERMOD and DATAMOD cause the TYPE(DATAIN) character to be altered in the displayed variable.

SCROLL All scrolling for the dynamic area MUST be done by your program. However, ISPF allows you to specify that your program plans to do scrolling, by specifying SCROLL(ON) or SCROLL(OFF). If SCROLL(ON) is specified, when the user requests a scrolling function, ISPF will pass control to your program (by completing the DISPLAY service) and setting the SCROLL value in ZVERB.

EXTEND EXTEND allows the depth of the dynamic area to be automatically increased, if required, to match the panel)BODY depth with the depth of the physical screen. This is useful when your application will run on different terminal sizes, so that your application can take advantage of the extra lines.

USERMOD and DATAMOD If you specify USERMOD and/or DATAMOD for attribute characters, when the dynamic area variable is returned to the dialog, the usermod-code and the datamod-code are used to replace the attribute character of each field that has bee modified. One or both can be specified for an attribute character within the dynamic area. USERMOD specifies that the user had typed into the field, but the data itself has not changed. DATAMOD specifies that the data has changed, either by user input or a capitalization/justification, etc.

With EXTEND(OFF) you must outline the entire dynamic area.

```
)ATTR
 @ AREA(DYNAMIC)   SCROLL(ON)   EXTEND(OFF)
 R  TYPE(DATAOUT)  COLOR(RED)
 W  TYPE(DATAOUT)  COLOR(White)
)BODY
%———————  EXAMPLE USING A DYNAMIC AREA WITH EXTEND(OFF) ——————————%
%COMMAND ===>_ZCMD                                    %SCROLL ===>_AMT +
%
+
+This is not          @DYNAREA                              @
+ extendable.         @@@@@@@                               @@@@@@@
+                     @@@@@@@                               @@@@@@@
+Dynamic area is      @@@@@@@                               @@@@@@@
+  seven lines long   @@@@@@@                               @@@@@@@
+                     @@@@@@@                               @@@@@@@
+                     @@@@@@@                               @@@@@@@
%
% This line will be shown below the dynamic area.
)INIT
 &dyn1 = 'RThis is the text which will  Rbe d'
 &dyn2 = 'isplayed in the area.  AsRyou can see,WspecialRcar'
 &dyn3 = 'e    Rmust be taken to space the   Rtext correctly, and p'
 &dyn4 = 'ut the  Rdataout attribute bytes in   Rthe correct places.'
 &dynarea = '&dyn1.&dyn2.&dyn3.&dyn4'
)END
```

```
---------  EXAMPLE USING A DYNAMIC AREA WITH EXTEND(OFF) ----------
COMMAND ===>                                        %SCROLL ===>


This is not          This is the text which will
 extendable.         be displayed in the area.  As
                     you can see, special care
Dynamic area is      must be taken to space the
  seven lines long   text correctly, and put the
                     dataout attribute bytes in
                     the correct places.

 This line will be shown below the dynamic area.
```

EXTEND(OFF) example /Notes

In this example, the variable to contain the dynamic area data is called 'DYNAREA'. The total size of the dynamic area is 7 lines long because of the EXTEND(OFF). The data outside of the dynamic area is not altered by the presence of the dynamic area. If the user presses a scroll key (UP, DOWN, RIGHT, or LEFT), control will be returned to the program because of the SCROLL(ON).

With EXTEND(ON) you must outline the start of the dynamic area, and may extend it further down the screen. ISPF will extend it further if necessary.

```
)ATTR
 @ AREA(DYNAMIC)   SCROLL(ON) EXTEND(ON)
 R  TYPE(DATAOUT)  COLOR(RED)
 W  TYPE(DATAOUT)  COLOR(WHITE)
)BODY
%———————  EXAMPLE USING A DYNAMIC AREA WITH EXTEND(ON) ———————————%
%COMMAND ===>_ZCMD                                    %SCROLL ===>_AMT +
%
+
+This is not extendable, @DYNAREA                          @
+                        @                                 @
+but the last line is.   @                                 @

%And this line is at the bottom.
)INIT
 &dyn1 = 'RThis is the text which will  R'
 &dyn2 =  'be displayed in the area.  AsR'
 &dyn3 =  'you can see,WspecialRcare    R'
 &dyn4 =  'must be taken to space the   R'
 &dyn5 =  'text correctly, and put the  R'
 &dyn6 =  'dataout attribute bytes in   R'
 &dyn7 =  'the correct places.          '
 &dynarea = '&dyn1.&dyn2.&dyn3.&dyn4.&dyn5.&dyn6.&dyn7.'
)END
```

```
---------  EXAMPLE USING A DYNAMIC AREA WITH EXTEND(ON) -----------
COMMAND ===>                                          SCROLL ===>


This is not extendable,  This is the text which will
                         be displayed in the area.  As
but the last line is.    you can see, special care
but the last line is.    must be taken to space the
but the last line is.    text correctly, and put the
but the last line is.    dataout attribute bytes in
but the last line is.    the correct places.
but the last line is.
but the last line is.
but the last line is.
but the last line is.
but the last line is.
but the last line is.
but the last line is.

And this line is at the bottom.
```

In this example, the variable to contain the dynamic area data is called 'DYNAREA'. The total size of the dynamic area will be determined at run time (because of the EXTEND(ON) in the)ATTR section. The data outside of the dynamic area on the last line will be repeated on every line from where it is in the definition, down to the last line on the screen, because the dynamic area will be extended. If the user presses a scroll key (UP, DOWN, RIGHT, or LEFT), control will be returned to the program.

- Allow for character level highlighting and graphic escape characters.

  - Alleviates problems with blanks due to field attributes

    1. Add a second variable to the dynamic area definition:

```
+This is extendable  @DYNAREA,SHADOW                @
```

    2. Add TYPE(CHAR) definitions to ATTR section

```
B TYPE(CHAR) COLOR(BLUE)   INTENS(LOW) HILITE(REVERSE)
W TYPE(CHAR) COLOR(WHITE)  INTENS(HIGH) GE(&MYGE)
```

    3. Characters in shadow variable override field highlighting for that individual character that it associates to in the dynamic area string.

  - Programs can check ZGE to determine if terminal supports graphic escape characters and set dynamic area accordingly

TYPE(CHAR) indicates to ISPF that you are defining a character attribute as opposed to a field attribute. Value keywords are COLOR, HILITE, and GE.

The color may be specified by a dialog variable, which is set by the application. This allows the color to be changed within the application. If this is not defined, the color of the underlying field attribute is used.

The valid values for HILITE are USCORE for underscore, BLINK for blinking, and REVERSE for reverse video. The 'value' can be a dialog variable which is set by the application. This allows the extended highlighting to be changed within the application. If this is not defined, the highlighting of the underlying field attribute is used.

GE may be specified as ON or OFF (OFF is the default). The 'value' can be specified by a dialog variable that is set by the application. If the terminal supports Graphic Escape order, a GE order is placed before the character code in the order stream. If the terminal does not support graphic escape, or the value is not '40'X - 'FE'X, then no GE order is placed in the order stream prior to the character, and the character is displayed as a blank. The application can check the new Z-variable 'ZGE' to determine if the terminal being used supports graphic escape.

A dynamic area must be defined in your panel. The dynamic area name must be followed by a comma, followed by the shadow variable name, followed immediately by a blank.

The shadow variable name follows the same naming rules as the dynamic area name.

When defining the shadow variable 'text', be sure that the spacing is correct, so that the character attribute is associated with the appropriate character in the dynamic area text. Any Character attribute which maps to the location of a field attribute byte will be ignored.

The following is a panel and REXX exec example to show:

1. Dynamic area panel

2. Mixing dynamic and static areas

3. Extending a dynamic area to fill the screen.

4. Scrolling up and down.

5. Use of a shadow variable for character level attributes.

Dynamic area example /Notes

This is a simple example of dynamic areas. It includes defining the dynamic and shadow variable within the)BODY section, the setting of the dynamic and shadow variable text strings, and the REXX code to handle simple scrolling requests.

A panel named DYNAREA

- Shows how static input/output areas coexist with a dynamic area
- Shows how an extendable dynamic area can be specified
- Shows how to use a shadow variable for character level attributes.
- Driven by the REXX exec on the next page

```
)ATTR
 @ AREA(DYNAMIC)   SCROLL(ON) EXTEND(ON)
 01 TYPE(DATAOUT)  COLOR(RED)
 02 TYPE(DATAOUT)  COLOR(BLUE)
 03 TYPE(DATAOUT)  COLOR(GREEN)
 04 TYPE(DATAOUT)  COLOR(WHITE)
 r  TYPE(CHAR) COLOR(RED) HILITE(REVERSE)
 g  TYPE(CHAR) COLOR(GREEN) HILITE(REVERSE)
 b  TYPE(CHAR) COLOR(BLUE) HILITE(REVERSE)
 $  TYPE(TEXT)     COLOR(YELLOW)
)BODY
%———————————————  EXAMPLE FOR USING A DYNAMIC AREA  ———————————————%
%COMMAND ===>_ZCMD                                      %SCROLL ===>_AMT +
%
+ This area is fixed.   size: &size
+
+ This is an input field%===>_somevar +
+
+This is extendable  @DYNAREA,DYNSHAD             @

       $This should be at the bottom of the screen when in full screen.
)END
```

- Variable DYNSHAD is the 'shadow variable'.

This panel shows several things. First, the panel defines a dynamic area, but it also contains static text (field prompts), an output field (size), and an input field (somevar). All of these elements, and many more if needed, can coexist together.

The dynamic area defined in the panel is defined with SCROLL(ON) and EXTEND(ON). This allows the application to scroll the data within the dynamic area when the user requests it, and ISPF will, at display time, extend the dynamic area depth so that the depth of the)BODY section equals that of the terminal that the panel is being displayed on.

It also shows how to use a shadow variable and character-level attributes to highlight individual characters within the string, overriding the field attribute for just one character, with no intervening space from a field attribute. **The shadow variable is optional.**. Shadow variables are only needed when you want character level attributes.

Character level attributes tend to generate more complex 3270 data streams, so users on remote lines will see slower performance.

```
/* REXX – Dynamic area example */
Address ispexec                          /* Calls go to ISPF       */
red   = '01'x                            /* Assign colors to       */
blue  = '02'x                            /*   Attribute bytes      */
green = '03'x                            /*   found in the data    */
white = '04'x
maxlines = 600                           /* set max number of lines*/
dyndata  = ''                            /* initialize data        */
shadata  = ''                            /* initialize shadow var  */

Do a = 1 to maxlines by 3                /* Create some dummy data */
  dyndata=dyndata||white||left('This is'red  ||'red  'white||a  , 29)
  dyndata=dyndata||white||left('This is'blue ||'blue 'white||a+1, 29)
  dyndata=dyndata||white||left('This is'green||'green'white||a+2, 29)
  shadata=shadata||' r                                      '
  shadata=shadata||' b                                      '
  shadata=shadata||' g                                      '
End

/*    Add a bottom of data maker to the end of the data      */
dyndata = dyndata||blue||centre(green||'BOTTOM'||blue,29,'*')
shadata = shadata||'                                      '

curline = 1;                             /* set current line #     */

/*--------------------------------------------------------------------*/
/* Display loop until end or error                                    */
/*--------------------------------------------------------------------*/

Do Until disprc > 0
  dynarea = substr(dyndata,1+(curline-1)*30) /* set dynamic variable */
  dynshad = substr(shadata,1+(curline-1)*30) /* set shadow  variable */
  size = length(dynarea)                 /* Set a scalar variable  */
  'ISPEXEC DISPLAY PANEL(DYNAREA)'       /* Display the data       */
  disprc = rc                            /* save return code       */
  'ISPEXEC VGET (ZVERB,ZSCROLLA,ZSCROLLN)' /* get scroll values     */
  SELECT                                 /* Process scrolling      */
    When(zverb  = 'UP')   Then           /* Scroll up              */
     If zscrolla = 'MAX' Then            /*  if scroll was max     */
      curline = 1                        /*    scroll to top       */
     Else                                /*  else a number is known*/
      curline = max(1,curline-zscrolln);  /* (maximum is top)       */
    When(zverb  = 'DOWN') Then           /* Scroll down            */
     If zscrolla = 'MAX' Then            /*  if scroll was max     */
      curline = maxlines                 /*    scroll to bottom    */
     Else                                /*  else a number is known*/
      curline = min(maxlines,curline+zscrolln); /* (max is bottom)   */
    Otherwise;                       /* could use left & right too  */
  End
End                                      /* End of display loop    */
```

This REXX routine shows how to set up a variable for use as a dynamic area variable, how to set up the shadow variable associated with that area, how to display the panel, and how to process UP and DOWN scrolling. RIGHT/LEFT scrolling is ignored in this example.

The data to be shown is set up to imbed the hex characters '01'x, '02'x, '03'x, and'04'x to represent colors red, blue, green, and white respectively. It will highlight each of the color names in their respective color, and show a list with colored color names, and line numbers.

The exec also sets up a shadow variable, which highlights the first character of each line with a color in reverse video. This 'shadowing' effect is only available in dynamic areas, and allows a change of the attribute at the character level, with no intervening 'blank' space caused by a field attribute.

Note that the shadow variable (variables shadata and dynshad in this example) is optional. You could leave it out of the panel and the exec and you would not have character level coloring.

The display is handled by setting a variable which contains the program data (dyndata) and then setting a second variable which will actually display the data (DYNAREA). The second variable is needed in this case to handle scrolling. The same manipulation with the length of the dyndata variable is done to the associated shadow variable, dynshad.

The variable dyndata starts with the first line to be displayed. When a scroll request is received, the program calculates the offset into DYNDATA of the first displayed line, and creates DYNAREA from there.

In a compiled language, you do not necessarily need to VDEFINE a second variable (the one actually used in the panel), since VREPLACE makes setting the second variable very easy.

Scrolled to top

```
--------------------  EXAMPLE FOR USING A DYNAMIC AREA  --------------------
COMMAND ===>                                                SCROLL ===> CSR

 This area is fixed.   size: 18030

 This is an input field ===>

This is extendable   This is red   1
This is extendable   This is blue  2
This is extendable   This is green 3
This is extendable   This is red   4
This is extendable   This is blue  5
                       .
                       .
                       .
This is extendable   This is blue  14
This is extendable   This is green 15

       This should be at the bottom of the screen when in full screen.
```

Scrolled to bottom

```
--------------------  EXAMPLE FOR USING A DYNAMIC AREA  --------------------
COMMAND ===>                                                SCROLL ===> CSR

 This area is fixed.   size: 60

 This is an input field ===>

This is extendable   This is green 600
This is extendable   ********** BOTTOM ***********
This is extendable
This is extendable
This is extendable
                       .
                       .
                       .
This is extendable
This is extendable
This is extendable

       This should be at the bottom of the screen when in full screen.
```

This shows how the example looks when scrolled to the top (UP MAX) and to the bottom (DOWN MAX). Notice the 'size:' field. That is the size of the DYNAREA variable. It does not have to exactly match the size of the dynamic area that is displayed. ISPF handles that for you.

If you want to have DOWN MAX fill the screen with data, you can, in the)PROC section, set a variable using the LVLINES(...) function and base your calculations on that.

- ISPF returns ZSCROLLN, ZSCROLLA, ZVERB

  ZVERB Direction of scroll (UP, DOWN, LEFT, RIGHT)

  ZSCROLLA Amount to scroll (MAX, CSR, HALF, DATA, number, etc.)

  ZSCROLLN Number of lines to scroll

→ UP/DOWN scrolling

- If all of your data is in one variable, use (ZSCROLLN*area width) to locate the start of the displayed variable.
- In a compiled language, VREPLACE is good for creating the displayed variable.

→ LEFT/RIGHT scrolling

- Left/right scrolling usually involves creating a new variable to display.
  This is because the dynamic area uses a contiguous string.

Scrolling a dynamic area /Notes

When the user requests a scrolling action for the dynamic area, ISPF returns several variables which contain information about the requested scroll. This information is checked by the application and is then used by the application to reset the dynamic area variable.

ZVERB Direction of scroll (UP, DOWN, LEFT, RIGHT)

ZSCROLLA Amount to scroll (MAX, CSR, HALF, DATA, number, etc)

ZSCROLLN Number of lines to scroll

- Character level coloring is available through shadow variables.

- You can use ISPEXEC PQUERY to get area sizes.

  LVLINE() function in)PROC section may be used to determine last visible line (for DOWN MAX)

- you can allow for automatic expansion for 3278 Mod-5 by coding / / in the dynamic area definition in the)BODY section and coding a variable width on the)BODY statement.

- For extendable areas, other characters on the line are replicated on each 'extended' line.

From panel ISREFR01 (edit user panel):

```
)BODY WIDTH(&ZWIDTH) EXPAND(//)
%EDIT -----.ZTITLE ------------------/-/----------------%COLUMNS.ZCL  .ZCR  %%
%COMMAND ===>_ZCMD                  / /                   %SCROLL ===>_Z   %
!ZDATA ------------------------------/-/-------------------------------------!
!                                   / /                                      !
! ----------------------------------/-/--------------------------------------!
)PROC
   &LASTLINE = &LVLINE(ZDATA)
)END
```

ISPF has several functions which can be used to help get additional information to assist in the definition of the dynamic area variable or scrolling requests.

PQUERY can help get information about areas contained within the panel. You specify the area name, and ISPF can return such things as the area type, and its' width and depth.

The LVLINE built-in function can be called from the)INIT,)REINIT, or)PROC sections. It returns the line number of the last line within a dynamic, graphic, or scrollable area which was visible to the end-user on the currently displayed panel. This is extremely useful when the user can be in split-screen mode.

Your application can be coded to take advantage of larger-size terminal screens by using extendable areas, and by coding variable widths (on the)BODY section) and using the automatic expansion feature of ISPF. On the)BODY section, you code the EXPAND keyword, and then specify the characters you will place in the text of the panel when you wish to expand. The character in the)BODY that you place between these two characters will be used when the expansion is done.

What is DTL

- Dialog Tag Language
- SGML for panels
  - → Frees developer from many panel design decisions
  - → Easy way to create CUA compliant dialogs

Why use DTL

- CUA compliance
- Standards within a group
- Application keylists outside of profiles
- Global changes
  - → DTL files imbed sections
    1. Often used phrases
    2. Variable class definitions
    3. Action Bar definitions

Dialog Tag Language is a tag-based markup language, such as BookMaster. DTL was created for various reasons, including

1. Ease of use of markup tags. The tag names directly relate to the 'element' that they create, such as a < P > tag creates a paragraph.

2. Flexibility in application development. GML's can be quickly changed and regenerated. Use of imbed "entities" allow global changes across your application panels, and aids in reuse.

3. It provides consistency in dialog elements when many programmers are working on the same application, or for new programmers.

4. DTL enforces some formatting rules defined by CUA, so your programmers do not have to be experts in CUA to define CUA conforming application elements.

5. It enables NLS support and provides translations of various keywords, such as the "More:" used on Scrollable Areas.

   Application keylists can be created outside of using actual profiles, by using <KEYL> and <KEYI> tags. The keylists are created as xxxKEYS where xxx is your application ID used when converting the GML.

   Using DTL Imbed files, or Entities, allows for reuse within your application development. This is useful for imbedding panel sections, such as an action bar definition which is used globally, keylists, variable classes within an application, and often used words or phrases, such as your company name.

```
<.DOCTYPE DM SYSTEM (
   <.ENTITY keys SYSTEM "guikey">
   <.ENTITY cmpn SYSTEM "gbl">
   <.ENTITY states SYSTEM "guiste"> )>

&keys;

<varclass name=nme  type='char 30'>
<varclass name=ini  type='char 1'>
<varclass name=addr type='char 35'>
<varclass name=zip  type='char 7'>
<varclass name=ch25 type='char 25'>

<varclass name=yrs  type='char 2'>
  <checkl>
     <checki TYPE=NUM >
  </checkl>

<varclass name=sal  type='char 7'>
  <checkl>
     <checki TYPE=NUM >
  </checkl>

<varclass name=ste  type='char 2'>
  <xlatl format=upper>
  </xlatl>
  <checkl>
     <checki &states; >
  </checkl>

<varclass name=gard type='char  8'>
  <xlatl format=upper>
  </xlatl>
  <checkl msg=gui201>
     <checki type=values parm1=eq parm2="PORCH BACKYARD FARM NONE"
  </checkl>

<varclass name=yesno type='char 3'>
  <xlatl format=upper>
  </xlatl>
  <checkl>
     <checki type=values parm1=eq parm2="YES NO"
  </checkl>

<varclass name=num10 type='char 2'>
  <checkl>
     <checki type=range parm1=0 parm2=10
  </checkl>
```

The next foils contain the GML source code for the panel shown in the Scrollable Area section of this presentation. There are some minor differences between the two, such as indentations.

The <!DOCTYPE DM SYSTEM ()> statement is the first in the GML file, and it defines to ISPF that it is a GML file for conversion. It also defines the ENTITY's that will be used for the file during the conversion process.

The &keys is an imbed file (shown later in the notes pages) which contains the entire keylist mapping definition, which will be included into the source file when the conversion is done.

The <VARCLASS> tag contains the name of "classes" of data, which will then be used to associate a variable to a class. This defines data to be character, numeric, DBCS, or mixed (DBCS and SBCS). It specifies the length, and optionally specifies a default message to be displayed for type failures. You may also specify translates <XLATL> for things such as FORMAT=UPPER, which will use CAPS(ON) for the field. Checks for certain values or ranges can also be done in this area with the <CHECKL> and <CHECKI> tags.

```
<varlist>

<vardcl name=lname varclass=nme>
<vardcl name=fname varclass=nme>
<vardcl name=mi    varclass=ini>
<vardcl name=saddr varclass=addr>
<vardcl name=aaddr varclass=addr>
<vardcl name=caddr varclass=addr>
<vardcl name=paddr varclass=ste>
<vardcl name=pcode varclass=zip>
<vardcl name=occ   varclass=addr>
<vardcl name=yr    varclass=yrs>
<vardcl name=inc   varclass=sal>
<vardcl name=gsz   varclass=gard>
<vardcl name=howhear varclass=ch25>
<vardcl name=simserv varclass=ch25>
<vardcl name=cat     varclass=num10>
<vardcl name=rep     varclass=num10>
<vardcl name=time    varclass=num10>
<vardcl name=gen     varclass=num10>
<vardcl name=spec    varclass=num10>
<vardcl name=retp    varclass=num10>
<vardcl name=comp    varclass=num10>
<vardcl name=suit    varclass=num10>
<vardcl name=GF      varclass=yesno>
<vardcl name=GO      varclass=yesno>
<vardcl name=BB      varclass=yesno>

</varlist>
```

The <VARLIST> and </VARLIST> define the start and end of the variable list. Inside you code your variable declarations.

The <VARDCL> tag associates a variable to be used in the GML with the classes that were just defined.

The <VARCLASS> and <VARDCL> tags ARE NOT REQUIRED for your conversion but it is an easy way to get certain verification for variables done in the)PROC sections, and is also required for portability to the OS/2 Dialog Manager.

```
<PANEL NAME = SURVEY
       KEYLIST = gui1
       HELP = survyh
       DEPTH = 24
       WIDTH = 76  >
       &cmpn;
<info>
<P COMPACT>Press<HP>PF8</HP>to go forward,<HP>PF7</HP>to go back,
<HP>PF3</HP>to exit
</info>
<area depth=2 marginw=2 extend=on>
<info>
<p compact>
Thank you for participating in this survey.  Please rest assured
that the information you provide will remain<HP>strictly
confidential</hp>unless we feel like divulging it.
</info>
<info>
<p><hp>Information about you</hp>
</info>
<divider>
<DTACOL PMTWIDTH = 23 entwidth = 31>
<DTAFLD DATAVAR = lname
        USAGE = in
        required=yes
        PMTLOC = before>
        Last  Name
</DTAFLD>
<DTAFLD DATAVAR = fname
        USAGE = in
        required=yes
        PMTLOC = before>
        First Name
<DTAFLD DATAVAR = Mi
        USAGE = in
        entwidth = 1>
        Middle Initial
<DIVIDER>
<DTAFLD DATAVAR = saddr
        USAGE = in
        required=yes
        entwidth = 35>
        Street Address
```

The <PANEL> tag defines certain characteristics of the panel, such as the panel name (member name), a keylist if one is to be used when the panel is displayed, panel help if one is needed, width and depth. Other characteristics can also be specified. For complete documentation of ALL DTL tags, see the ISPF Dialog Tag Language Guide and Reference manual.

Other tags are then used, and most are very easy to understand what they will create. I will give a brief summary of those tags used on this foil:

- INFO - defines an information region, this is required for some tags to be nested under, such as the < P > tag.
- P - defines a paragraph. The default is to format with a blank line prior to the paragraph text. However, using the COMPACT keyword will suppress that blank line.
- HP - defines a highlighted phrase. This text will be displayed in turquoise (the default, it can be changed using CUAATTR utility by the end-user), which stands out from the rest of the text.
- AREA - defines a portion of the panel body, one or more of which can be scrollable. If the DEPTH attribute is used, it defines a scrollable area within your)BODY section, and all information defined within its' bounds are added to an)AREA section.
- DIVIDER - creates one or more divider lines, which can be blank or solid/dashed.
- DTACOL - defines default parameters for DTAFLD's within it.
- DTAFLD - defines an input and/or output field on the panel, along with the characteristics of that field.

```
<DTAFLD DATAVAR = aaddr
        USAGE = in
        required=yes
        entwidth = 35>
        Apartment Number
<DTAFLD DATAVAR = caddr
        USAGE = in
        required=yes
        entwidth = 35>
        City
<DTAFLD DATAVAR = paddr
        USAGE = in
        required=yes
        deswidth= 35
        entwidth = 2>
        State or Province
        <DTAFLDD>    (Use 2 letter abbreviation)</dtafldd>
<DTAFLD DATAVAR = pcode
        USAGE = in
        entwidth = 7>
        Zip or Postal Code
<DIVIDER>
<DTAFLD DATAVAR = occ
        USAGE = in
        entwidth = 35>
        Occupation
<DTAFLD DATAVAR = yr
        USAGE = in
        entwidth = 2>
        Number of years at your current job
<DTAFLD DATAVAR = inc
        USAGE = in
        required=no
        deswidth= 35
        entwidth = 7>
        Annual income
        <SOURCE TYPE=PROC>
&GSZ = TRANS(TRUNC(&GSZ,1) P,PORCH B,BACKYARD F,FARM N,NONE *,'?'
        </SOURCE>
        <DTAFLDD>(optional)</dtafldd>
<DTAFLD DATAVAR = gsz
        USAGE = in
        deswidth= 29
        entwidth = 8>
        Size of your garden
        <DTAFLDD>(Porch, Backyard, Farm, None)</dtafldd>
</DTACOL>
```

Other tags on this foil:

- DTAFLDD - Data field descriptor text
- SOURCE - Defines panel processing statements

The <SOURCE> tag allows you to place panel logic statements directly into the processing sections of the panel, if you need additional logic that ISPF does not provide. A good example of this is if you wish to invoke the PANEXIT statement from within the)PROC section of the panel for a panel exit.

```
<divider>
<info>
<P><hp>Information about our services</hp>
</info>
<divider>
<DTAFLD DATAVAR = howhear
        USAGE = in
        pmtwidth = 38
        entwidth = 26>
        Where did you hear about our services?
<DTAFLD DATAVAR = simserv
        USAGE = in
        pmtwidth = 38
        entwidth = 26>
        What similar services have you used?
<info>
<LINES>
Please rate the following categories on a scale of 1 to 10.
    1 &ltsym;———————————— 6 ————————————————————&gtsym;10
   Poor              Average                 Excellent
Please use a 0 for 'Not Applicable'

</lines>
</info>
<DTACOL pmtwidth = 27 entwidth = 2 >
<DTAFLD DATAVAR = cat
        USAGE = in
        required = yes>
        Catalog
<DTAFLD DATAVAR = rep
        USAGE = in
        required = yes>
        Sales representatives
<DTAFLD DATAVAR = time
        USAGE = in
        required = yes>
        Time to process your order
<DTAFLD DATAVAR = gen
        USAGE = in
        deswidth = 26
        required = yes>
        General merchandise
        <dtafldd> (General seeds and plants) </dtafldd>
```

Other tags on this foil:

- LINES - unformatted text within an information region. The text is put is EXACTLY as you have it typed.

```
<DTAFLD DATAVAR = spec
        USAGE = in
        deswidth = 26
        required = yes>
        Specialty merchandise
        <dtafldd> (Rare plants, tools, etc) </dtafldd>
<DTAFLD DATAVAR = retp
        USAGE = in
        required = yes>
        Return Policy
<DTAFLD DATAVAR = comp
        USAGE = in
        required = yes>
        Complaint Department
<DTAFLD DATAVAR = suit
        USAGE = in
        required = yes>
        Our Lawyers
        <SOURCE TYPE=PROC>
&GF = TRANS(TRUNC(&GF,1) Y,YES N,NO *,'?'
&GO = TRANS(TRUNC(&GO,1) Y,YES N,NO *,'?'
&BB = TRANS(TRUNC(&BB,1) Y,YES N,NO *,'?'
        </SOURCE>
</dtacol>
<info>
<P><hp>Information about your buying habits</hp>
</info>
<divider>
<DTAFLD DATAVAR = gf
        USAGE = in
        entwidth=3
        required = yes>
        Do you subscribe to our<hp>gardening fashions catalog?</hp>
<DTAFLD DATAVAR = go
        USAGE = in
        entwidth=3
        required = yes>
        Do you buy garden<hp>supplies for other people?</hp>
<DTAFLD DATAVAR = bb
        USAGE = in
        entwidth=3
        required = yes>
        Would you be interested in buying a<hp>really big bridge?</hp>
</area>
<CMDAREA>
</PANEL>
```

Other tags on this foil:

- CMDAREA - specifies your command line. You can specify a help panel name and your prompt text (the default is 'Command').

Following are the GML imbed files:

File GUIKEY:

```
<KEYL NAME=gui1 >
  <KEYI KEY=F1 CMD=Help FKA=Yes>Help
  <KEYI KEY=F3 CMD=Exit FKA=Yes>Exit
  <KEYI KEY=F7 CMD=Up fKA=yes>Bkwd
  <KEYI KEY=F8 CMD=Down FKA=yes>Fwd
  <KEYI KEY=F12 CMD=Cancel FKA=Yes>Cancel
</KEYL>
```

File GUISTE:

```
TYPE=VALUES PARM1=EQ
PARM2="AK AS HI GA NC SC NY KY QC BC WA HI MI TX AL FL"
```

File GBL:

```
SUPERSEEDY SEED COMPANY
```

```
)PANEL KEYLIST(GUI1,GUI)
)ATTR DEFAULT(_+%) FORMAT(MIX)
  | TYPE(PT)
  . TYPE(FP)
  @ TYPE(NT)
  # TYPE(ET)
  $ TYPE(NEF) PADC(_)
  . TYPE(NEF) CAPS(ON) PADC(_)
  \ TYPE(DT)
  . TYPE(WASL) SKIP(ON)
  / AREA(SCRL) EXTEND(ON)
)BODY WINDOW(76,24) CMD(ZCMD)
|                          SUPERSEEDY SEED COMPANY                          @
@Command ===>$Z
@
@Press#PF8@to go forward,#PF7@to go back,#PF3@to exit@                        @
/SAREA37                                                                      /
/                                                                             /
```

The next foils are the edited source that was generated.

When the GML's are converted, all of the attribute bytes in the generated panel are hex codes. I have edited the source and changed these hex codes into characters for the use in this presentation.

As you can see, we generated this panel using CUA Attribute types in the)ATTR section. This is an option on the conversion process. There are two TYPE(NEF) fields generated, since we declared a VARCLASS which also said FORMAT=UPPER on an XLATL tag.

The scrollable area, seen here below the panel information line, matches to an)AREA section, shown on the next foil.

```
)AREA SAREA37
@ Thank you for participating in this survey.  Please rest assured that@  @
@@the information you provide will remain#strictly confidential@unless we@@@
@@feel like divulging it.@                                                @
@
@#Information about you@                                                  @
  .
 .Last  Name  . . . . . .$Z                          @
 .First Name  . . . . . .$Z                          @
 .Middle Initial  . . . .$Z@
  .
 .Street Address  . . . .$Z                              @
 .Apartment Number  . . .$Z                              @
 .City  . . . . . . . . .$Z                              @
 .State or Province . . ..Z @\    (Use 2 letter abbreviation)    @
 .Zip or Postal Code  . .$Z      @
  .
 .Occupation  . . . . . .$Z                              @
 .Number of years at your
 .current job . . . . . .$Z @
 .Annual income . . . . .$Z      @\(optional)                        @
 .Size of your garden . ..Z       @\(Porch, Backyard, Farm, None)@
@
@#Information about our services@                                         @
  .
 .Where did you hear about our services?$Z                          @
 .What similar services have you used?  $Z                          @
@
@@ Please rate the following categories on a scale of 1 to 10.@           @@
@@     1 <———————————— 6 ————————————————————————>10@                   @@@
@@    Poor              Average                    Excellent@            @@@@
@@ Please use a 0 for 'Not Applicable'@                                   @@
@                                                                         @
 .Catalog . . . . . . . . . .$Z @
 .Sales representatives . . .$Z @
 .Time to process your order $Z @
 .General merchandise . . . .$Z @\(General seeds and plants)@
 .Specialty merchandise . . .$Z @\(Rare plants, tools, etc) @
 .Return Policy . . . . . . .$Z @
 .Complaint Department  . . .$Z @
 .Our Lawyers . . . . . . . .$Z @
@
@#Information about your buying habits@                                   @
  .
 .Do you subscribe to our#gardening fashions catalog?..Z  @
 .Do you buy garden#supplies for other people?..Z  @
 .Would you be interested in buying a#really big bridge?..Z  @
```

Here is the entire text of the scrollable area. You will notice it is very similar to the example we had earlier. There are some minor differences, but the text is very much the same.

```
)INIT
  .ZVARS = '( ZCMD LNAME FNAME MI SADDR AADDR CADDR PADDR PCODE +
              OCC YR INC GSZ HOWHEAR    +
              SIMSERV CAT REP TIME GEN SPEC RETP COMP SUIT GF GO BB)'
  .HELP = SURVYH
  &ZCMD = ' '
)PROC
  VER(&LNAME,NONBLANK)
  VER(&FNAME,NONBLANK)
  VER(&SADDR,NONBLANK)
  VER(&AADDR,NONBLANK)
  VER(&CADDR,NONBLANK)
  VER (&PADDR
  LIST,AK,AS,HI,GA,NC,SC,NY,KY,QC,BC,WA,HI,MI,TX,AL,FL)
  VER(&PADDR,NONBLANK)
  VER (&YR NUM)
  VER (&INC NUM)
  &GSZ = TRANS(TRUNC(&GSZ,1) P,PORCH B,BACKYARD F,FARM N,NONE *,'?'
  VER (&GSZ LIST,PORCH,BACKYARD,FARM,NONE MSG=GUI201)
  VER (&CAT RANGE,0,10)
  VER(&CAT,NONBLANK)
  VER (&REP RANGE,0,10)
  VER(&REP,NONBLANK)
  VER (&TIME RANGE,0,10)
  VER(&TIME,NONBLANK)
  VER (&GEN RANGE,0,10)
  VER(&GEN,NONBLANK)
  VER (&SPEC RANGE,0,10)
  VER(&SPEC,NONBLANK)
  VER (&RETP RANGE,0,10)
  VER(&RETP,NONBLANK)
  VER (&COMP RANGE,0,10)
  VER(&COMP,NONBLANK)
  VER (&SUIT RANGE,0,10)
  VER(&SUIT,NONBLANK)
  &GF = TRANS(TRUNC(&GF,1) Y,YES N,NO *,'?'
  &GO = TRANS(TRUNC(&GO,1) Y,YES N,NO *,'?'
  &BB = TRANS(TRUNC(&BB,1) Y,YES N,NO *,'?'
  VER (&GF LIST,YES,NO)
  VER(&GF,NONBLANK)
  VER (&GO LIST,YES,NO)
  VER(&GO,NONBLANK)
  VER (&BB LIST,YES,NO)
  VER(&BB,NONBLANK)
)END
```

Generated 'Panel' source /Notes

The processing sections are shown in this foil. Again, you will notice some slight differences, including the way that the conversion utility formats the VER statements onto different lines in some cases. It also generates multiple VER statements for a single variable, such as a VER for NONBLANK, and a VER for the RANGE.

```
                          SUPERSEEDY SEED COMPANY

Press PF8 to go forward, PF7 to go back, PF3 to exit
                                                              More:     +
 Thank you for participating in this survey.  Please rest assured that
 the information you provide will remain strictly confidential unless we
 feel like divulging it.

 Information about you

 Last  Name  . . . . . _______________________________
 First Name  . . . . . _______________________________
 Middle Initial  . . . . _

 Street Address  . . . . _____________________________________
 Apartment Number  . . . _____________________________________
 City  . . . . . _____________________________________
 State or Province . . . __      (Use 2 letter abbreviation)
 Zip or Postal Code  . . _______

 Occupation  . . . . . _____________________________________
 Number of years at your
 current job . . . . . __
Command ===> ____________________________________________________________
```

DTL/'Old style' sample /Notes

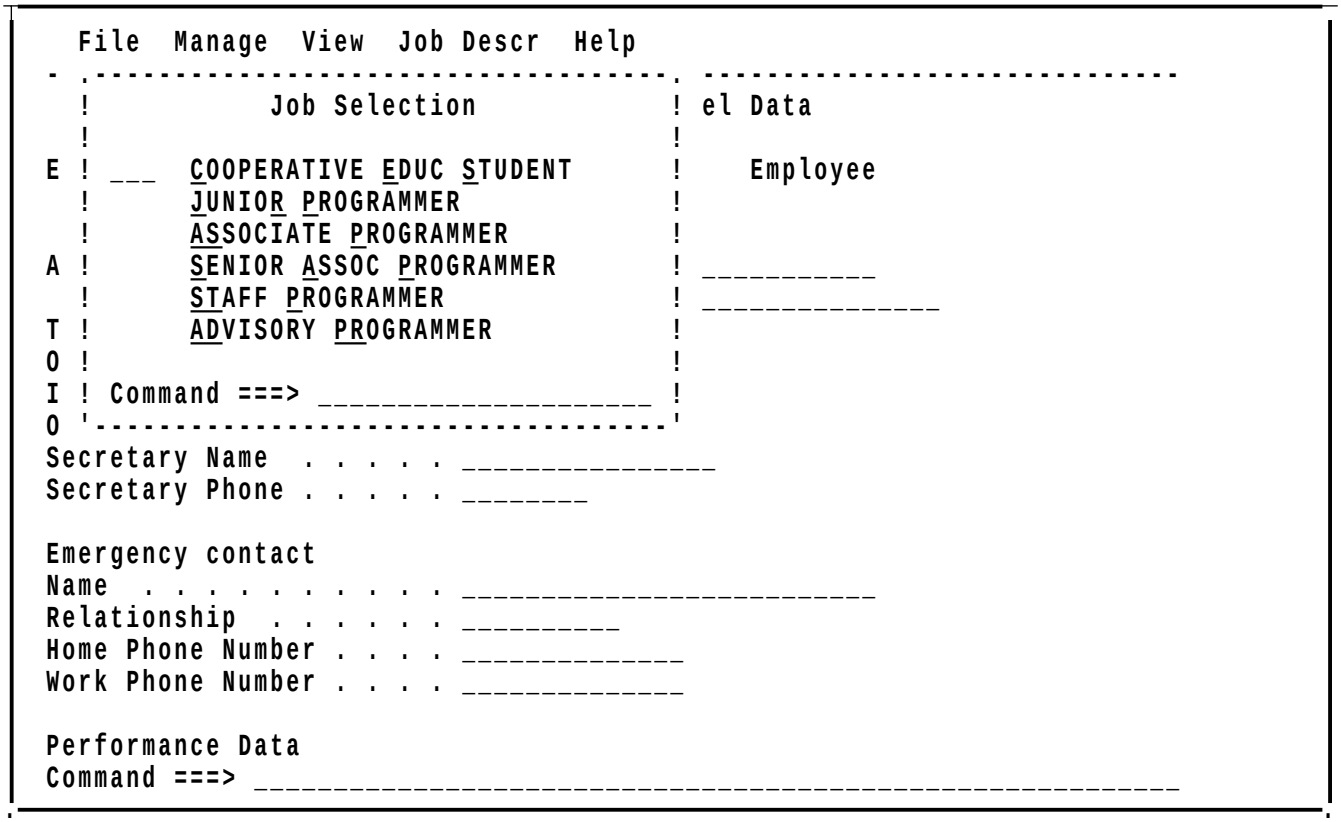
This is the first display....I will not show all of them.

Again, the indentations are a little different, but the general use of the panel is the same.

```
<.DOCTYPE DM SYSTEM ()>
<varclass name=vjb type='char 3'>
<varlist>
<vardcl name=jobd varclass=vjb>
</varlist>
<PANEL NAME = cua2job
       KEYLIST = cu2s
       DEPTH = 10
       WIDTH = 35  >
       Job Selection

<REGION DIR=HORIZ>
   <DTAFLD DATAVAR=jobd pmtwidth=0 entwidth=3>
   </dtafld>
   <DA name=job extend=off scroll=on depth=6 width=25 shadow=jobshad>
   <attr attrchar=@ TYPE=CHAR COLOR=YELLOW HILITE=USCORE >
   <attr attrchar=# TYPE=DATAOUT COLOR=TURQ>
   </DA>
</region
<CMDAREA>
</PANEL>
<.── Comment Line >
```

This is a brief example of DTL to generate a dynamic area, including a shadow variable and attributes for DATAOUT and TYPE(CHAR) in the)ATTR section of the panel.

The dialog would have to set the variables 'job' and 'jobshad', which would contain the dynamic area string and its' associated shadow variable.


File Manage View Job Descr Help
Job Selection
el Data
COOPERATIVE EDUC STUDENT
JUNIOR PROGRAMMER
ASSOCIATE PROGRAMMER
SENIOR ASSOC PROGRAMMER
STAFF PROGRAMMER
ADVISORY PROGRAMMER
Command ===>
Employee
Secretary Name . . . . .
Secretary Phone . . . . .
Emergency contact
Name . . . . . .
Relationship . . . . . .
Home Phone Number . . . .
Work Phone Number . . . .
Performance Data
Command ===>

This is the display of the dynamic area panel, as shown within an existing application. As you can see, the datafield is shown on the right side of the dynamic area. This is done by using a <REGION> tag, and defining it with a DIR=HORIZ for a horizontal region. This states that those elements defined should be placed horizontally across the panel. They can be used in conjunction with vertical regions (<REGION DIR=VERT>) to set columns of information horizontally across the panel.

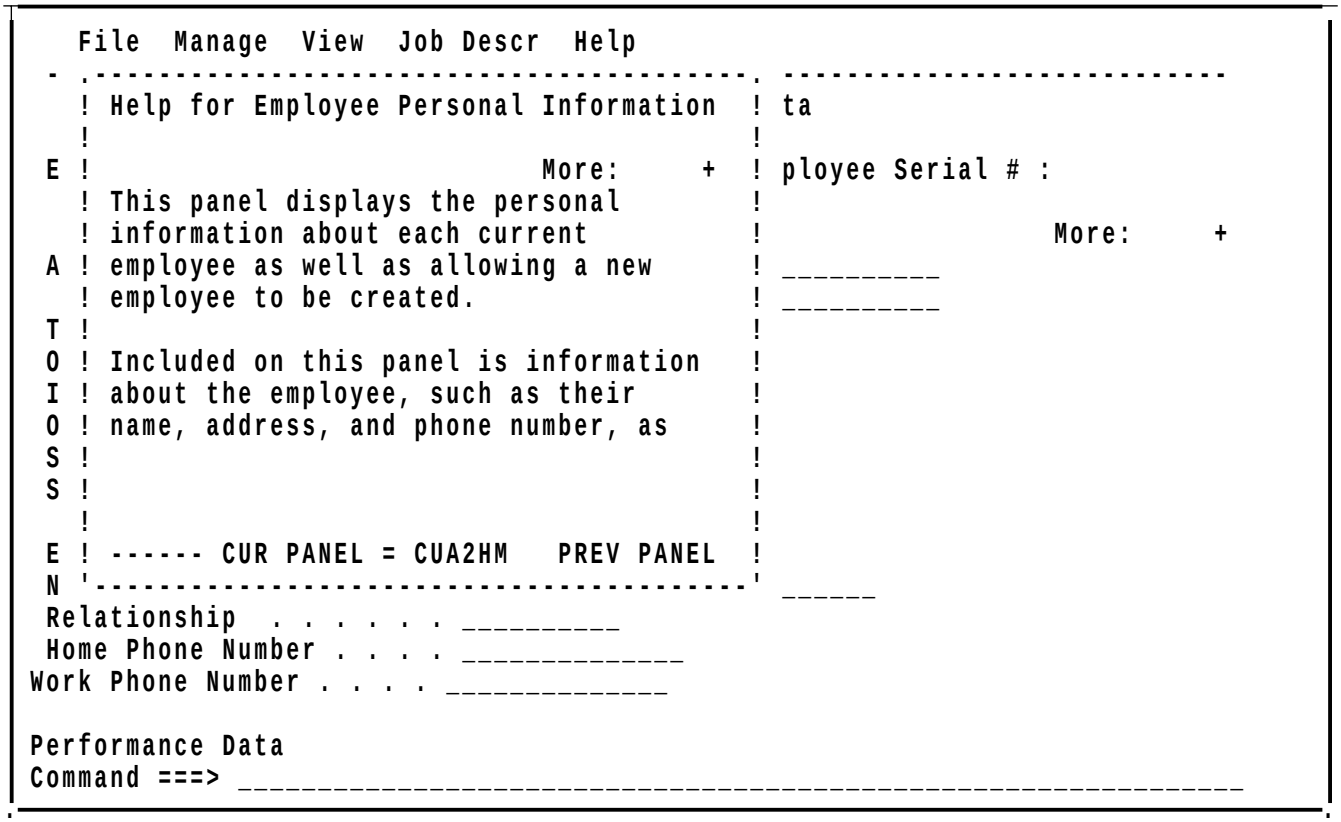
The underlined characters are also shown in yellow, whereas the rest of the text is in turquoise.

```
<.DOCTYPE DM SYSTEM ()>

<help  NAME = cua2hm
       KEYLIST = CUA2H
       DEPTH = 15
       WIDTH = 40  >
       Help for Employee Personal Information

<area depth=9>
<info>
<p>This panel displays the personal information about each current employee
as well as allowing a new employee to be created.
<p>Included on this panel is information about the employee, such as their
name, address, and phone number, as well as information about the employees
current position.
<p>When entering a new employee the name and <RP help=cua2hsn>serial number</RP>
are changed to input fields to allow these to be entered.
</info>
</area>

</help>
```

This is an example of a tutorial panel, written using the Dialog Tag Language. You will notice that instead of a <PANEL> tag, it uses a <HELP> tag. This defines it as a tutorial, rather than an application panel.

Most elements are similar between what can be placed in an application panel and that which can be in a tutorial panel. The tutorial panels can even have input and output fields and action bars. There are a few constructs which are not available in tutorial panels, such as <LSTCOL> and <LSTFLD> tags, which are used to create)MODEL lines for table displays.

One construct which IS allowed ONLY on tutorial panels are Reference phrases (<RP> tags). Reference phrases look like Highlighted Phrases (<HP> tags), but associate a help panel with the word or phrase. The end-user can request help with their cursor on that word or phrase, and ISPF will display the associated help panel.


File Manage View Job Descr Help
Help for Employee Personal Information
More: +
This panel displays the personal
information about each current
employee as well as allowing a new
employee to be created.
Included on this panel is information
about the employee, such as their
name, address, and phone number, as
------ CUR PANEL = CUA2HM PREV PANEL
ta
ployee Serial # :
More: +
Relationship . . . . . .
Home Phone Number . . . .
Work Phone Number . . . .
Performance Data
Command ===>

- **You can pop up a help panel when the cursor is in a field and HELP is requested.**

```
)BODY
   This is a field===>_fldname      +
)HELP
   FIELD(fldname) HELP(panelnme)
)END
```

Use)HELP section (last section).

Field Level Help /Notes

ISPF provides facilities for field level help. Field level help is presented when the user places the cursor on a field and presses the HELP key. The field can by a type INPUT, or OUTPUT, or a dynamic area.

The field and its field level help panel are specified in the)HELP section, which is normally the last section in the panel definition. You may specify the same panel for more than one field. Field level help panels are tutorial panels and may have reference phrases within them.

Field level help comes up in a window, so you should use the WINDOW() parameter on the)BODY statement of the help panel.

- Point & Shoot
  - → Enables cursor sensitive selections
    - Sets a named variable to a value
  - → Creates buttons in GUI
  - → Displays in the 'Point and Shoot' color
- TYPE(PS) is similar to type text
- Attribute PAS(ON|OFF) on TYPE(INPUT) and TYPE(OUTPUT)
-)PNTS section
  - → Note that Point and Shoot in dynamic areas must be handled by the program.

)PNTS section

```
)PNTS
  FIELD(field|ZPSxxyyy) VAR(var) VAL(value)
```

field Any named field

xx 00 for body section or 01 to 99 for number of the area in which field appears

yyy 001 to 999 for relative position of field within the)BODY or)AREA

var Name of variable to set when field is selected

value Value to assign to variable. May not use substitution.

Mnemonics

You can specify Mnemonics on pulldowns for the GUI interface. They have no effect on 3270 displays.

Unavailable Choices

You can cause a pull-down choice to be unavailable ('greyed out') if a given variable is one (1).

```
)ABC DESC('Utilities') MNEM(1)
     PDC DESC('Library')   UNAVAIL(ZUT1) MNEM(1)
                             ACTION RUN(ISRROUTE) PARM('U1')
     PDC DESC('Data set')  UNAVAIL(ZUT2) MNEM(1)
                             ACTION RUN(ISRROUTE) PARM('U2')
     PDC DESC('Move/Copy') UNAVAIL(ZUT3) MNEM(1)
                             ACTION RUN(ISRROUTE) PARM('U3')
```

UNAVAIL and/or MNEM must come before the ACTION keyword.

```
)ATTR DEFAULT(%+_)
 @ TYPE(PS)
 # TYPE(OUTPUT) PAS(ON) caps(off)
 < TYPE(INPUT)  PAS(ON) caps(off)
 . type(AB)
 • type(ABsl)
)ABC desc('ABchoice')
  PDC DESC('Reset In') unavail(ur1) ACTION RUN(R1)
  PDC DESC('Reset Out') unavail(ur2) ACTION RUN(R2)
)ABCINIT
  .zvars = somethin
  if (&inval = &Z) &ur1 = 1
  Else &ur1 = 0
  if (&outval = &Z) &ur2 = 1
  Else &ur2 = 0
)BODY
+. ABchoice +
•————————————————————————————————————————————
+              %Point & Shoot Example+
%Option ===>_zcmd                              +
+
           %1  @Selection 1+   % 3@Selection 3+
           %2  @Selection 2+   % 4@Selection 4+

Input field button  (<infld +): &inval       +
Output field button (#outfld+): &outval      +
)INIT
  &infld  = 'Press'
  &outfld = 'Press'
)REINIT
  REFRESH(*)
)PROC
if (&zcmd = 'R1')
  &zcmd = &Z
  &inval = &Z
if (&zcmd = 'R2')
  &zcmd = &Z
  &outval = &Z
&ZSEL = TRANS ( TRUNC (&zcmd,'.')
  1,'CMD(%SAY Selection 1 was pressed... &zcmd)'
  2,'CMD(%SAY Selection 2 was pressed... &zcmd)'
  3,'CMD(%SAY Selection 3 was pressed... &zcmd)'
  4,'CMD(%SAY Selection 4 was pressed... &zcmd)'
 ' ',' '
   *,'?' )
)PNTS
FIELD(zps00001) VAR(zcmd) VAL(1)
FIELD(zps00002) VAR(zcmd) VAL(3)
FIELD(zps00003) VAR(zcmd) VAL(2)
FIELD(zps00004) VAR(zcmd) VAL(4)
FIELD(infld)    VAR(inval)  VAL('In')
FIELD(outfld)   VAR(outval) VAL('out')
)END
```

What are they?

Panel exits allow you to perform extra data manipulation and verification in the)INIT,)REINIT, and/or)PROC section of a panel.

Input

Exit data	Number passed to data
Panel Name	Name of panel invoking the exit
Panel Section	Panel area invoking the exit I)INIT R)REINIT P)PROC
Message id	Message id to display if an error occurs (output)
Array Dim	Number of variables passed
Varname	Array of variable names
Varlen	Array of variable lengths
Varval	Contiguous storage containing variable values

Panel exits allow you to add additional logic to your panel which can be used to update variables, verify variables, and perform functions which regular ISPF panel language can not accomplish. For example, setting up variables before a display, such as shadow variables, or variables based on constantly changing system values can be done in panel exits. If you want to verify a field based on different criteria, this can be done in a panel exit.

Panel exits receive as input individual variable names, lengths, and values, as well as a variable which can be used to store data between invocations. The panel exit also receives an indicator of what panel is calling the exit, and what section of the panel the exit is being called from. It can be called from the)INIT,)REINIT, or)PROC sections of the panel.

Another example of using panel logic is command line checking or alteration. Aside from using the panel exit to do variable verification, it can also be used to examine and preprocess the command line. For commands that may be seen by your application, (not ones in the ISPF or application command table), you can examine the command line during)PROC section processing, and alter it if you want to before control returns to the program that displayed the panel.

Output

Returns a return code in Register 15:

0	Successful operation
8	'Verification' failed. Display message passed back and redisplay panel.
Other	Severe error. ISPF fails the display.

Restrictions

- Must be in a compiled language (not REXX)
- Exit can not issue ISPF services
- Can change passed variable VALUES, but NOT the lengths of the variables.

Tip

- Make sure variables passed in are padded to allow for any updates to be made to them. ISPF may backscan off blanks before passing value to the exit.

A panel exit has several methods of output.

A panel exit can return a return code in register 15. A zero means successful completion. An 8 means display a returned message id. Any other return code will be assumed by ISPF to indicate a severe error.

As mentioned, the exit can pass back a message id to display.

The exit can change the 'exdata' exit data word. This can be used as an anchor pointer to exit data.

Lastly, the exit can change the values of the variables that are passed into it. However: The exit must not change the lengths of the variables passed to it.

Panel exits do not have access to ISPF services (ISPLINK, ISPEXEC).

Panel exits must be in a compiled language. Compiled REXX can not be used.

One thing to look out for: ISPF panel logic may strip trailing blanks off of variables before passing them to the panel exit. If you plan to update a variable, you may want to pad it with non-blank characters in the panel logic before passing it to the exit so that you can pass back a longer (non-blank) string than would have been passed in.

Your dialog can preload the exit before calling DISPLAY or SELECT, or it may have ISPF load the exit dynamically.

- **Preload**

```
PANEXIT((ZCMD, VAR1, VAR2),PGM,address,exit-data,MSG=msgid)
```

  - **address is a variable name containing the address of the exit**

- **Dynamic load**

```
PANEXIT((ZCMD, VAR1, VAR2),LOAD,modname,exit-data,MSG=msgid)
```

  - **modname is the actual name of the exit entry point.**
  - **The exit stays loaded for the entire select level.**

Dynamic loading or Preloading /Notes

ISPF will load your exit dynamically if you request it to. The exit is loaded when first used, and remains loaded until the current SELECT level completes. If an intervening SELECT level occurs between panel displays, the exit may be reloaded.

This sample, written in PL/I is a panel exit for the Editor. It causes the phrase containing the cursor to be highlighted, the same way browse highlights words. A phrase is any non-blank string, delimited by blanks).

I copied panel ISREFR01 and made a few changes.

1. **Added a shadow variable to the dynamic area.**

2. **Added W as a TYPE(CHAR) in the)ATTR section**

3. **Setup some variables to pass to the exit the)REINIT section.**

4. **Added a call to my exit (HILITE) in the)REINIT section.**

The sample panel exit I am providing causes the editor to highlight the phrase containing the cursor in a manner similar to the way Browse highlights the cursor. This is done by using a modified copy of panel ISREFR01, the supplied alternate (or user) panel for edit. I made the following changes to that panel:

1. I added a shadow variable (called SHADOW) to the)BODY of the panel. It is this shadow variable which will cause the highlighting to occur, since the character level attributes of the shadow variable supersede the default highlighting of edit's dynamic area.

2. The)ATTR section was modified to add the character 'W' as an attribute character for the shadow variable. For the characters in edit's dynamic area which I want to highlight, I will add a 'W' to the corresponding position in the shadow variable. The W is defined in this example as TYPE(CHAR) COLOR(WHITE) INTENS(HIGH).

3. In order for the exit to know where the cursor is, and to have a shadow variable to update, I set up some variables before the exit is called. I used the fact that the editor sets up the ZWIDTH variable automatically to tell ISPF the screen width to use. I used some built-in functions (.CURSOR, .CSRPOS) to get cursor position and, most importantly, I allocated some space for the shadow variable. I know that the shadow variable has to be the same length as the ZDATA variable, and that I can not change the length of any variables. Therefore, I just assigned ZDATA to the shadow variable, to allocate the space, then the 1st thing the exit will do is to set all of the characters in the shadow variable to blanks.

4. I also added the actual call to the exit (which I call HILITE) to the)REINIT section. I used)REINIT because the 1st time the screen is shown, the cursor is usually on the command line, so the exit is not necessary in the)INIT section. It is not needed in the)PROC section because no post-display processing is needed in this example.

(Notes continued on next page)

Alternate edit panel

→ **May be copied to ISREDDE (primary edit panel)**

```
)ATTR
  _ TYPE(INPUT) CAPS(OFF) INTENS(HIGH) FORMAT(&MIXED)
  ! AREA(DYNAMIC) EXTEND(ON) SCROLL(ON) USERMOD(20)
  . TYPE(OUTPUT) INTENS(HIGH) PAD(-)
 01 TYPE(DATAOUT) INTENS(LOW)
 02 TYPE(DATAOUT) INTENS(HIGH)
 03 TYPE(DATAOUT) SKIP(ON) /* FOR TEXT ENTER CMD. FIELD */
 04 TYPE(DATAIN)  INTENS(LOW)  CAPS(OFF) FORMAT(&MIXED)
 05 TYPE(DATAIN)  INTENS(HIGH) CAPS(OFF) FORMAT(&MIXED)
 06 TYPE(DATAIN)  INTENS(LOW)  CAPS(IN)  FORMAT(&MIXED)
 07 TYPE(DATAIN)  INTENS(HIGH) CAPS(IN)  FORMAT(&MIXED)
 08 TYPE(DATAIN)  INTENS(LOW)  FORMAT(DBCS)
 09 TYPE(DATAIN)  INTENS(LOW)  FORMAT(EBCDIC)
 0A TYPE(DATAIN)  INTENS(LOW)  FORMAT(&MIXED)
 20 TYPE(DATAIN)  INTENS(LOW) CAPS(IN) FORMAT(&MIXED)
  W TYPE(CHAR) COLOR(WHITE)  INTENS(HIGH)
)BODY WIDTH(&ZWIDTH) EXPAND(//)
%EDIT -----.ZTITLE ------------------/-/--------------------%COLUMNS.zcl.zcr%%
%COMMAND ===>_ZCMD                  / /                     %SCROLL ===>_z   %
!ZDATA,SHADOW ----------------------/-/-------------------------------------!
!                                   / /                                     !
! ---------------------------------/-/-------------------------------------!
)INIT
  .HELP = ISR20000          /* Default tutorial name                 */
  .ZVARS = 'ZSCED'          /* Scroll amount variable name           */
  &MIXED = MIX              /* Set format mix                        */
  IF (&ZPDMIX = N)          /* If ebcdic mode requested              */
    &MIXED = EBCDIC         /*  Set format ebcdic                    */

)REINIT
  &SHADOW = &ZDATA          /* Allocate space for shadow variable    */
  &FIELD = .CURSOR          /* Save cursor field name                */
  &POSIT = .CSRPOS          /* Save offset into field                */
                            /* NOTE: 1St shadow reference is         */
                            /* really a reference to ZDATA's data.   */
                            /* Problems may arise using ZDATA itself */
  PANEXIT ((ZWIDTH,FIELD,POSIT,ZDATA,SHADOW),LOAD,HILITE)
  REFRESH(ZCMD,ZSCED,ZDATA,ZTITLE,ZCL,ZCR)
  .HELP = ISR20000          /* Default tutorial name     */

)PROC
  REFRESH(ZCMD,ZSCED,ZDATA,ZTITLE,ZCL,ZCR)
  &ZCURSOR = .CURSOR
  &ZCSROFF = .CSRPOS
  &ZLVLINE = LVLINE(ZDATA)

)END
```

The sample code is PL/I. It will compile and work. I compiled it with the IBM OS PL/I Optimizing compiler Version 2 Release 2 Mod level 1. In case you are not familiar with the terminology, a 'BASED' variable is a variable whose location in storage is determined (based on) another variable (a pointer).

I suggest that you try this exit to see how it works. If you don't include all of my comments, it will not take long at all to type in... Or I can send you a copy on diskette, or via IBMMAIL, INTERNET, or Bitnet.

Declarations

```
*process system(mvs) optimize(time);
 hilite:
    proc(exdata,panname,pansect,msgid,araydim,varname,varlen,varval)
        options(main noexecops) reorder;
    declare
      exdata   fixed bin(31),       /* Not used here                */
      panname  char(8),             /* Panel name                   */
      pansect  char(1),             /* Panel section                */
      msgid    char(8),             /* Last message id              */
      araydim  fixed bin(31),       /* Array dimension              */
      varname  char(8),             /* Variable names passed in     */
      varlen   fixed bin(31),       /* Variable value lengths kludge */
      varval   fixed bin(31);       /* String of values             */
    declare
      varlen_array(5) fixed bin(31) based(addr(varlen)), /* Array of
                                       lengths                      */
      zwidth_p ptr,                 /* Pointer to width of screen   */
      field_p  ptr,                 /* Pointer to field name        */
      posit_p  ptr,                 /* Pointer to cursor position   */
      zdata_p  ptr,                 /* Pointer to zdata dynamic area */
      shadow_p ptr,                 /* Pointer to shadow variable   */
      zwidth   char(3)     based(zwidth_p),/* Width of screen       */
      field    char(5)     based(field_p), /* Cursor field name     */
      posit    char(8)     based(posit_p), /*   And position in field*/
      zdata    char(10000) based(zdata_p), /* Dynamic area(edit data)*/
      shadow   char(10000) based(shadow_p); /* Shadow variable area  */
    declare
      a        fixed bin(31),       /* Index into shadow or zdata   */
      chars(10000) char(1) based,   /* Based array of single chars  */
      realpos  fixed bin(31),       /* Binary version of var 'posit' */
      width    fixed bin(31),       /* Binary version if var 'zwidth'*/
      st_line  fixed bin(31);       /* Start(or end) of line in zdata*/
```

Logic

```
/*****************************************************************/
/* Get pointers to the values of each passed in variable         */
/*****************************************************************/

zwidth_p = addr(varval);
field_p  = addr(zwidth_p -> chars(varlen_array(1)+1));
posit_p  = addr(field_p  -> chars(varlen_array(2)+1));
zdata_p  = addr(posit_p  -> chars(varlen_array(3)+1));
shadow_p = addr(zdata_p  -> chars(varlen_array(4)+1));

substr(shadow,1,varlen_array(5)) = ' '; /* Blank shadow variable */
if field = 'ZDATA' then                 /* If cursor is in ZDATA */
  do;
    /*************************************************************/
    /* Get position of cursor within zdata field                 */
    /*************************************************************/

    realpos = substr(posit,1,varlen_array(3)); /* Convert to bin */
    width = substr(zwidth,1,varlen_array(1));  /* Convert to bin */
    st_line = ((realpos - 1) / width) * width + 8;

    if realpos >= st_line then      /* If cursor in data area    */
      do;
        /*********************************************************/
        /* Scan backwards and forwards looking for a blank.      */
        /* For each non-blank character, put a 'W' in shadow var.*/
        /*********************************************************/

        do a = realpos to st_line by -1 /* Scan backwards        */
                          while(substr(zdata,a,1) .= ' ');
          substr(shadow,a,1) = 'W';     /* put char in shadow    */
        end;
        st_line = st_line + width - 9;  /* calculate end of line*/
        st_line = min(st_line,varlen_array(5)); /* don't overflow*/
        do a = realpos to st_line while(substr(zdata,a,1) .= ' ');
          substr(shadow,a,1) = 'W'; /* update shadow for fwd scan*/
        end;
      end;
  end;
end hilite;
```