Paper 160-25

Not on the Menu...,
Visualizing Application Path Choices

Tom Miron, *Miron InfoTec, Inc., Madison, WI*

Pick-List Menus (PMENU)

The most common application interface is the pick-list menu. Within the SAS system, these menus are called PMENU's and are usually created with PROC PMENU. PMENU's allow for menu hierarchies, that is, selecting a menu item can open another menu. PMENU's are used throughout the SAS Display Manager (DM) and are similar to the menu interface used with other graphical interface systems, including most Windows family applications. See Fig. 1.

Within the SAS system, there is extensive support for PMENU's:

- menus can be assigned with the frame general attribute screen. No coding and recompile are required to changed menus
- PMENU related commands, functions, and methods are provided with the display manager, SCL, and SAS/AF classes
- support for autoloading of TOOLBOX's related by name to PMENU entries (Windows platforms)
- support for item gray/ungray, checkbox and radio button items
- support for accelerator key combinations

What's the Issue?

Hierarchical, pick-list menus have stood the test of time and are familiar to most application users, as such, they should get first consideration when designing an application interface. PMENU's are a good choice when your application paths are organized as a fixed hierarchy.

There are situations were alternatives to PMENU's should be considered. First, PMENU's are unavoidably hierarchical. If this hierarchical view does not fit well with your application or data organization, PMENU's can be cumbersome and confusing. PMENU's also limit the user's view of the application structure. Most of the time users are offered only a top-level view of the items on the menu bar. Sub-menu functions are only displayed when a menu bar item is selected. Essential, or at least useful, functions can be buried under layers of sub-menus. Users may not be aware of application features because they have not explored all menu paths.

PMENU's are text based and afford limited opportunity to create a visual representation of your application layout.

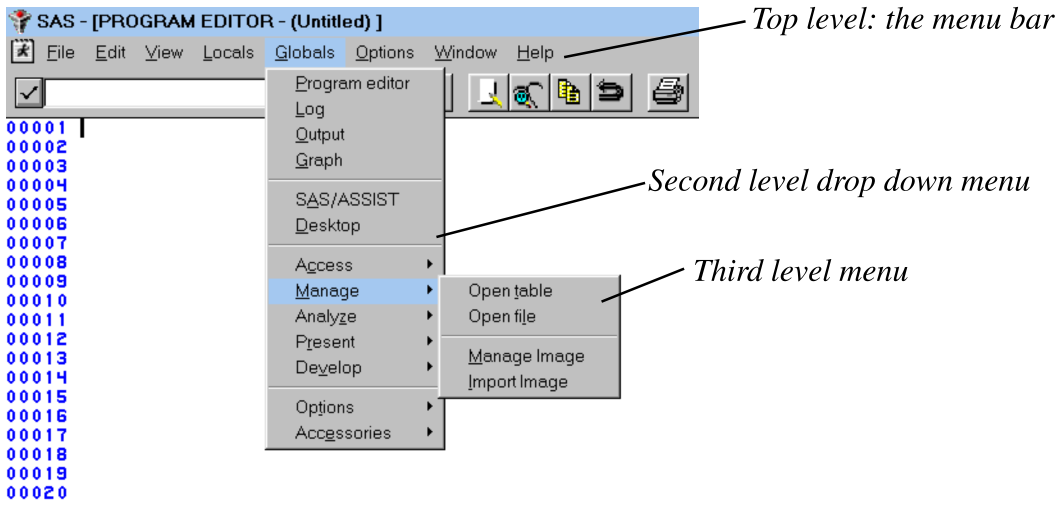


Fig. 1 - Three levels of PMENU in the SAS Display Manager

Help Users Visualize Your Application

There are several alternatives to PMENU's to consider when designing your SAS software application interface. You can use these alternatives to visually present application paths and data structures. Keep in mind that all of these alternatives can be used in addition to PMENU's. You can offer users the familiarity of PMENU's along with the efficiency and esthetics of a graphical interface.

The rest of this paper presents examples of various widgets used as selection objects in SAS/AF Frame applications. The examples are from SAS release 6.12 running on Windows, but should be valid for later releases and any GUI platform.

Using Frame Widgets as "Menus"

You can use any Frame widget (aka control or object) as a selectable screen element. "Selection" means the user clicks (or double clicks) on the widget or places the cursor on the widget and hits enter. You can associate a widget with a command. This corresponds to the association of a PMENU item with a command.

You assign a command to a widget via the build time object attribute window. You don't need to code SCL. You don't need to recode and recompile when a widget or its command changes.

Presenting a Two Level Structure with Containers

Figure 2 shows two commonly used selection widgets: the block and push button. Note the use of containers to group the blocks. Grouping blocks within titled containers is equivalent to presenting a two-level hierarchy. In this example, the top-level items are Tables, Enter Billable Items, Invoices, etc. Within each container sub-items are presented as selectable blocks. Block color can also be used to visually group related items.

A command is assigned to the selection of each block via its build time attributes window. See Figure 3.

The basic frame SCL to handle any application specific command, including those assigned as build time widget attributes, is shown in Figure 4. You could also handle a user selection by coding an object label section in your frame SCL or overriding the widget's _OBJECT_LABEL_ or _SELECT_ methods. Object label sections and method overrides involve more work and complexity, and, in most situations, are just not necessary.

Not only can you assign a command to the each block, you can assign a command to a container. This allows you present a full two level structure (containers and blocks within containers) and assign an application function (command) to both levels, something that is not possible with PMENU's.

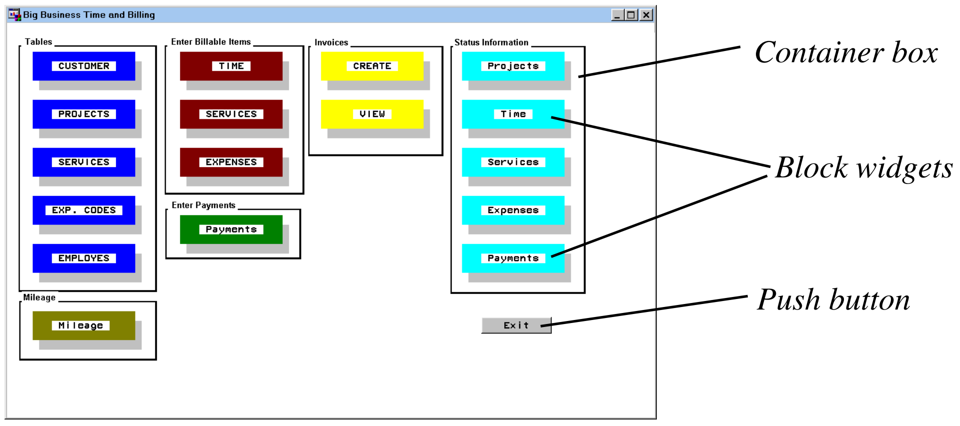
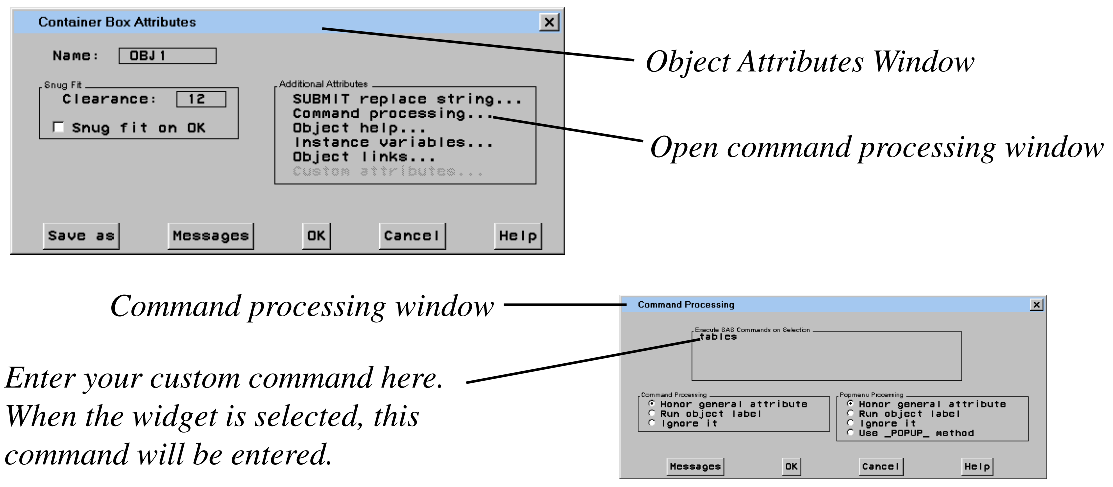


Fig. 2 - Blocks, containers, and a push button



Container Box Attributes

Name: OBJ1

Snug Fit
Clearance: 12
☐ Snug fit on OK

Additional Attributes
SUBMIT replace string...
Command processing...
Object help...
Instance variables...
Object links...
Custom attributes...

Object Attributes Window

Open command processing window

Command processing window

Enter your custom command here. When the widget is selected, this command will be entered.

Command Processing

Execute SAS Commands on Selection
tables

Command Processing
[✓ Honor general attribute]
Run object label
Ignore it

Popmenu Processing
[✓ Honor general attribute]
Run object label
Ignore it
Use _POPUP_ method

Fig. 3 - Widget attribute windows used to assign a command

```
INIT:
    control allcmds;
 return;

MAIN:
    word = word( 1, 'l' );
    select( word );
       when( 'tables' ) do;
         link tables;
         call nextword();
       end;
       when( 'invview' ) do;
         link invview;
         call nextword();
       end;
       when( 'invcrt' ) do;
         link invcrt;
         call nextword();
       end;
     otherwise;
   end;
return;
```

Force MAIN section to run for all commands

Get first word in command queue and convert to lowercase

Check for application (custom) commands

Link routine to handle a command.

Purge command queue of custom command so it's not passed to SAS/AF, where it would be flagged as an unknown, invalid command.

If not a custom command, do nothing. Let AF handle it.

Fig. 4 - Frame SCL to handle custom commands

You Can Use Any Widget

You can use any widget as a selectable object and process its selection via a custom command assigned at build time. Figure 5 shows some examples.

When you want to present a series of buttons, it's usually more convenient to use a single toolbar widget than several push buttons. A command can be assigned to each button at build time through the toolbar attributes window.

User selections are handled by intercepting the command associated with a widget or individual toolbar button as shown in Fig. 4.

Custom Graphic Images

The IMAGE widget is great for giving your application a customized look. Most common graphic file formats are supported for images. Fig. 6 shows an example with 3 scanned company logos in TIFF format. As described above, you can assign a command to the image object at build time through its attribute windows.

Set the region outline attributes to push button and the image will depress as it's selected. The push button attribute also allows you to set tooltip help that displays when the user holds the mouse cursor over the image. See Fig. 7.

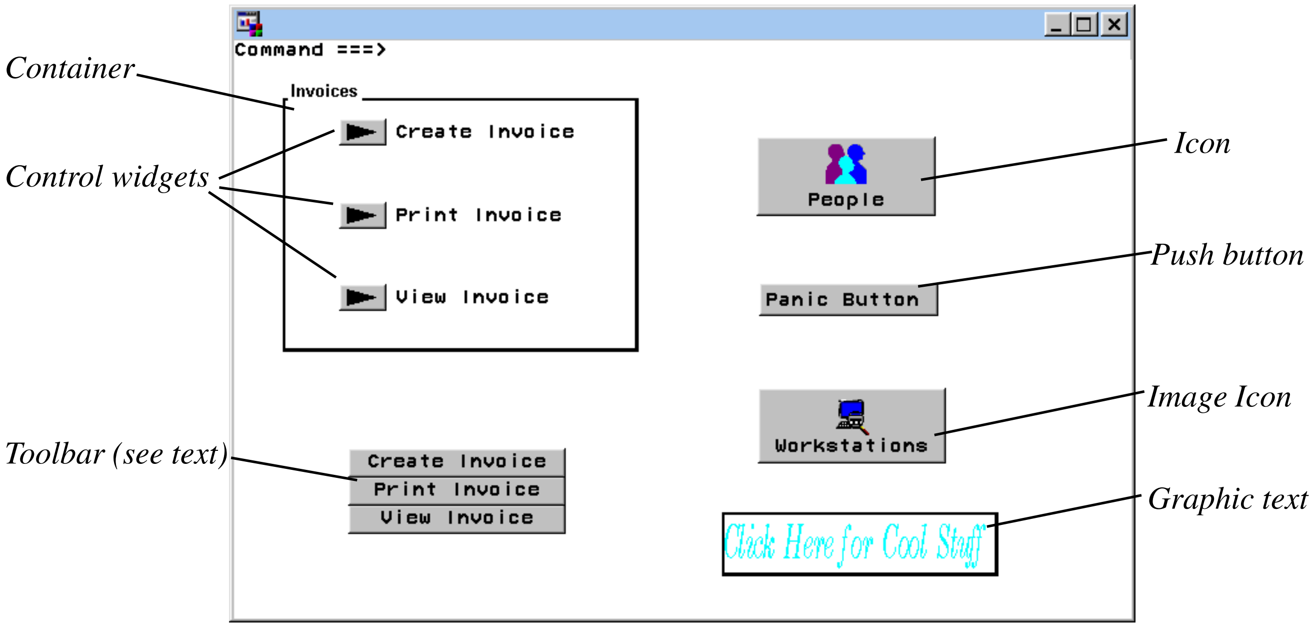


Fig. 5 - Presenting selections with widgets

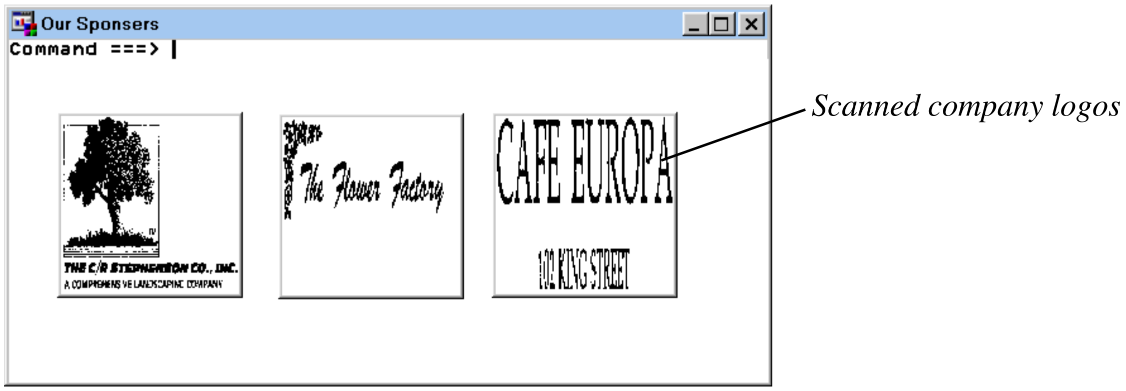
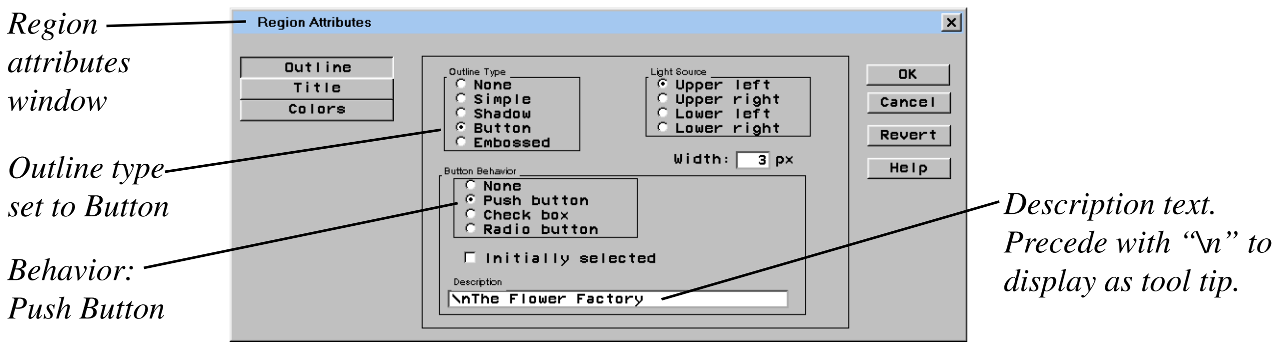


Fig. 6 - Image widgets as selection objects


Region attributes window
Outline type set to Button
Behavior: Push Button
Description text. Precede with “\n” to display as tool tip.
Outline
Title
Colors
OK
Cancel
Revert
Help

Region Attributes

Outline Type
( ) None
( ) Simple
( ) Shadow
(•) Button
( ) Embossed

Light Source
(•) Upper left
( ) Upper right
( ) Lower left
( ) Lower right

Width: 3 px

Button Behavior
( ) None
(•) Push button
( ) Check box
( ) Radio button

☐ Initially selected

Description
\nThe Flower Factory

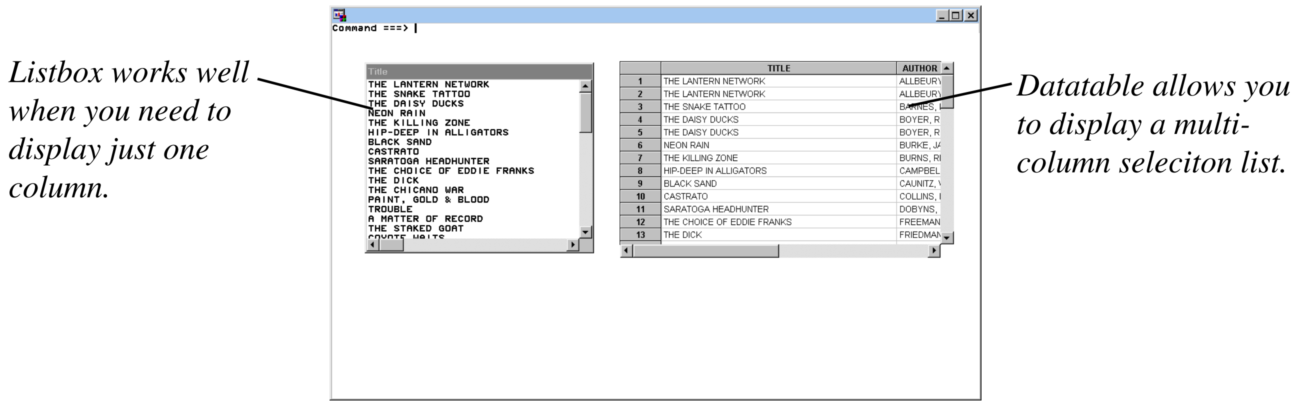
Fig. 7 - Make your image act as a push button with tooltip help

Presenting Long Flat Lists

If you need to offer users a long, one-dimensional list of items, PMENU’s are not appropriate and the placement of too many widgets on a frame is impractical and confusing. The DATATABLE and LISTBOX widgets work well for this sort presentation. These widgets allow users to scroll through a long list.

With LISTBOX and DATATABLE, assigning a command attribute to the widget is not useful, since you want to capture the selection of an item within the widget, not selection of the widget itself. You will need SCL to process user selections from tables or lists. Typically, this SCL would be in the object label section of your frame SCL. Figure 8 shows a frame with a listbox and datatable and Figure 9 shows example SCL to handle selections.

In the example, both the listbox and datatable display the same SAS table. Without special coding, the listbox can display only one column from a table. Datatables allow you to display multiple columns. Datatables also offer more control over appearance and behavior, at the expense of more complex coding. On the other hand, there are several options for populating lists in addition to SAS datasets including: build time manual text entry, an SCL list that exists at run time, and LIST type catalog entries.


Listbox works well when you need to display just one column.
Datatable allows you to display a multi-column selecitonlist.

```
Command ===>
```

Title
THE LANTERN NETWORK
THE SNAKE TATTOO
THE DAISY DUCKS
NEON RAIN
THE KILLING ZONE
HIP-DEEP IN ALLIGATORS
BLACK SAND
CASTRATO
SARATOGA HEADHUNTER
THE CHOICE OF EDDIE FRANKS
THE DICK
THE CHICANO WAR
PAINT, GOLD & BLOOD
TROUBLE
A MATTER OF RECORD
THE STAKED GOAT

	TITLE	AUTHOR
1	THE LANTERN NETWORK	ALLBEUR
2	THE LANTERN NETWORK	ALLBEUR
3	THE SNAKE TATTOO	BARNES,
4	THE DAISY DUCKS	BOYER, R
5	THE DAISY DUCKS	BOYER, R
6	NEON RAIN	BURKE, J
7	THE KILLING ZONE	BURNS, R
8	HIP-DEEP IN ALLIGATORS	CAMPBEL
9	BLACK SAND	CAUNITZ,
10	CASTRATO	COLLINS,
11	SARATOGA HEADHUNTER	DOBYNS,
12	THE CHOICE OF EDDIE FRANKS	FREEMAN
13	THE DICK	FRIEDMAN

Fig. 8 - Long selection lists: listbox and datatable

```
 INIT:
   attrs = makelist();
   attrs = setnitemc( attrs, 'y', 'select_rows' );
   call notify( 'table', '_set_attributes_', attrs );
   attrs = dellist( attrs );
return;

TABLE:
   list = makelist();
   call notify( 'table', '_get_selections_', list);
   link tabsel;
   list = dellist( list, 'y' );
return;

LISTBOX:
   call notify( 'listbox', '_get_last_sel_', row, selected, text );
   if not selected then return;
   link listsel;
return;
```

Force highlight of full row when user clicks anywhere in that row.

Object label for datatable. This code runs when table is selected.

Get row/column coordinates of the selected row.

Link to selection handling routine. The routine will have to relate the selected row coordinates to data values using other DATATABLE methods.

Object label for listbox. This code runs when the listbox is selected.

Get last selected row

If not still selected just return

Link to listbox selection handler. The text of the selected item is returned in the variable TEXT passed in the _GET_LAST_SEL_ method above.

Fig. 9 - Frame SCL to handle listbox and datatable selection lists.

Maps

You can effectively present geographic or spacial choices with the map widget. This class is new with SAS release 6.12 and is documented in the on-line help. The addition of this widget to the standard SAS/AF class library greatly simplifies the use of maps as a selection interface.

When using a map, you normally want to intercept which map area was selected, not just the fact that the map itself was selected. This means you can't just assign a command to the map and code a response to the command. You need to code an object label section to determine which map area has been selected.

The details of SAS maps are beyond the scope of this paper, but for any SAS map you need to assign a map dataset and a response dataset. The MAP widget allows you to do this at build time. Once your map is set up, capturing the selected area is fairly simple. Figure 10 shows a frame with the county map of Indiana. Figure 11 show the object label section for the map in the frame SCL. The object label code allows you to determine which area has been selected. In the example, ID value is the county code number.

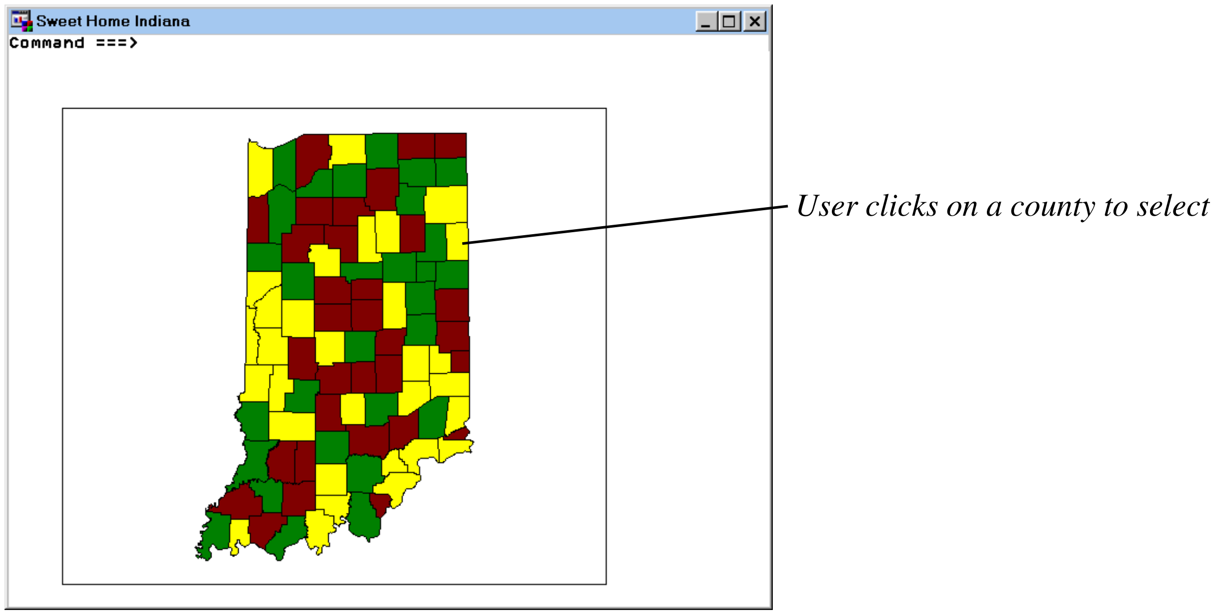


Fig. 10 - Map widget

Map object label (name)

Attribute request list. Insert a numeric item named ID.

_GET_VALUE_ method returns attribute named in LIST.

The selected area (ID) is returned. In this case, ID is a county code.

```
MAP:
   list = makelist();
   list = setnitemn( list, ., ‘id’ );
   call notify( ‘map’, ‘_get_value_’, list );
   county_code = getnitemn( list, ‘id’ );
   list = dellist( list );
return;
```

Fig. 11 - Object label SCL to handle area selection in a map

Flow Charts

Some application paths can best be visualized as nodes in a process or tasks in a project. For these, consider the PROCESS FLOW DIAGRAM (PFD) widget. The PFD graphic can be constructed at build time using built-in drawing tools. As with maps, a full discussion of the PFD is off topic here, but you can use a one level PFD as a selection object without bothering with the more complex PFD features.

Figure 12 shows a one-level, PFD representing phases in the software development process. Users can select a phase node to view information about projects at that phase. For example, which projects are the Development Team working on, or which tasks are in the Test Phase. Figure 13 shows the PFD object label SCL that allows you to determine which node has been selected.

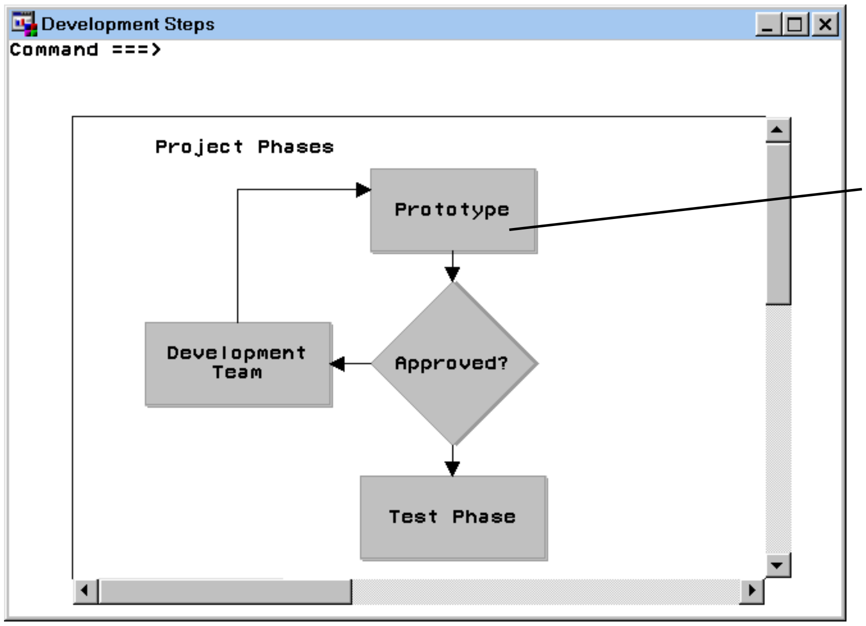


Fig. 12 - Process flow diagram

PFD object label (name)

```
PFD:
   list = makelist();
   call notify( 'pfd', '_get_info_', list );
   node_name = getniteml( list, 'text' );
   list = dellist( list );
return;
```

Return info about the selected node in LIST.

Node label text is returned in the TEXT list. Each list item is a line of the label.

Fig. 13 - Object SCL to capture selected node label

Orgcharts (Directories)

PMENU's can represent application path hierarchies, but in some cases the number of nodes and levels in a hierarchy may be too large to effectively display via menu lists. Also, it's not easy to make PMENU's dynamic, that is, make the menu items change based on some underlying data. Org chart widgets can overcome both these limitations.

The most difficult part about using org charts is getting your data into the proper format to populate the chart. For more on this I refer you to the "SAS/AF Software Frame Class Dictionary" and a paper I presented at SUGI 22, "How to Use the SAS/AF Frame Orgchart Object".

Figure 14 shows the employee organization for the "New Technologies Staff". The underlying data in the example is the ORGSAMP1 table, a sample provided with the ORGCHART class. Figure 15 shows the object label SCL used to determine the name of the person selected.

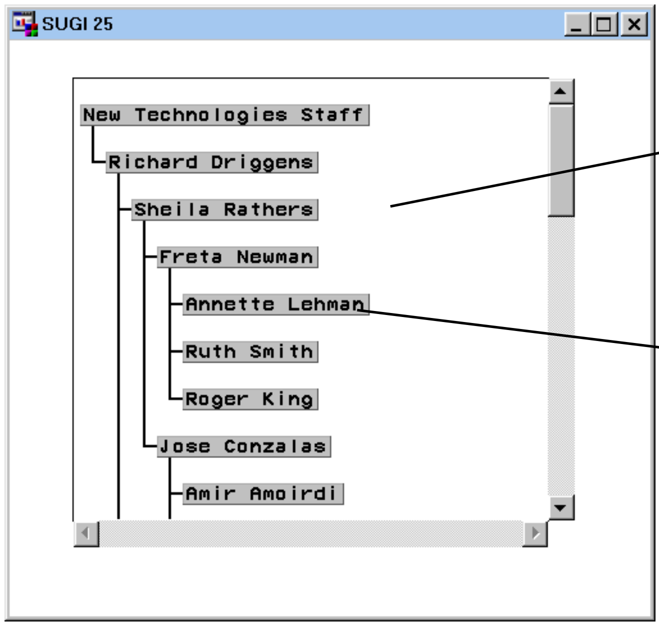


Fig. 14 - Personnel orgchart

Orgchart object label (name)

Get selected node number in variable NODEID. If nothing selected just return.

```
ORGCHART:
   call notify( ‘orgchart’, ‘_get_selected_’, widgetid, nodeid );
   if not nodeid then return;
   call notify( ‘orgchart’, ‘_get_current_’, nodeid, ‘text’, list );
   name = getnitemc( list, ‘text’ );
return;
```

Get information about selected node. Request is for node TEXT returned as a named item in LIST.

Employee name is the text displayed on the node.

Fig. 15 - Object SCL for orgchart

Conclusion

You can use any Frame widget as a selectable object. This means you can use widgets arrayed on a frame as an alternative to pick-list menus (PMENU’s) to present application features. Widgets can be grouped by container boxes to present a hierarchy of options. Wise use of selection widgets presents the user with an intuitive view of your application features and paths.

The easiest way to detect that a widget has been selected is to assign a custom command to the object at build time then test for that command in the MAIN section of the frame’s SCL. No object label SCL or method overrides are necessary.

The map, orgchart, PFD, listbox, and datatable widgets required object label SCL to capture user selection. Though these widgets can be complex, you need to understand only one or two methods in order to use them effectively as selection objects.