New Features in Panther 4.50

Panther 4.50 supports a few new features. Two of these are specific to deployment under Microsoft Windows. A third also has an impact on web deployments. And one of the features is also supported in character mode and Motif. These features are:

- Framesets and Splitters (Windows and Web)
- Tooltip support (Windows only)
- New window options (Windows only)
- New status bar functions (Windows, Motif and character mode)

Panther 4.50 also includes a sample application to demonstrate the use of the Frameset and Splitters feature. This sample application also serves to illustrate how to use the TreeView, ListView and ImageList ActiveX controls. See the section below entitled “The Frameset Sample Application” for further information.

The following document describes the new features.

Framesets and Splitters

A Splitter is a Microsoft Windows control that divides a window into a number of subareas, each of which is populated separately. Each subarea is called a "pane". The movable dividing line between two panes is called a "mullion".

In Panther, support for splitters is implemented by means of a special kind of screen called a frameset. The number and configuration of the panes into which each frameset is divided is determined in the Panther editor by placing special widgets called splitters on the frameset. Each pane is itself a Panther widget that has properties associated with it. The most important property of a pane is the name of a screen that will be rendered inside the pane at runtime. The screens associated with a frameset's panes are not displayed along with the splitters in the Panther editor. They must be created and edited separately.

Creating And Editing Framesets

In the Panther editor a new frameset is created by choosing `File→New→Frameset` on the editor's main menu. This will bring up a blank frameset. Such a frameset is much like an ordinary Panther screen, in that it supports a similar set of properties, is stored in libraries, and is generally worked with in much the same way. But a frameset can hold only two kinds of widgets: splitters and panes.

Splitter Widgets

A frameset is divided into multiple panes by placing splitter widgets on it. There are three kinds of splitter widgets – vertical splitters, horizontal splitters and two-way splitters.

- A vertical splitter will divide the frameset into two or columns, and will be represented in the frameset screen in the editor by one or more vertical mullions.
- A horizontal splitter will divide the frameset into two or more rows, and will be represented in the editor by one or more horizontal mullions.
- A two-way splitter will divide the frameset into four panes, and will be represented in the editor by one vertical and one horizontal mullion.

A splitter can be configured to divide the area it was placed in into further rows or columns by changing the splitter's Number of Rows or Number of Columns property.

Placing a splitter in an empty frameset divides the whole frameset into panes. Subsequent splitter placements in that frameset will necessarily have to be placed within previously defined panes. Such splitter placements will divide the panes the splitter was placed in into smaller panes. That is to say, splitters can be nested.

Splitters are selected in the editor by clicking on one of the mullions that represents the splitter. A splitter can also be selected from the widget list.

When a splitter widget is selected in the editor, the properties window is updated to display the properties settable for splitter widgets.

The following are the properties for splitter widgets:

```
IDENTITY
       Name
       Comments
       MEMO TEXT
              Memo1
              etc.

GEOMETRY
       Rows
       → Row 1 Height
       → Row 2 Height
          etc.
       Columns
       → Col 1 Width
       → Col 2 Width
          etc.

WEB OPTIONS
       No Border
       Spacing
       Col 1 Width
       Col 2 Width
       Etc.
       HTML OPTIONS
              Attributes
```

The `Rows` and `Columns` properties determine how many panes each splitter defines. These can have integer values from 2 to 16. Changing this property will create or destroy panes associated with the splitter. Newly created panes will be added to the right or below the existing panes. Likewise, panes destroyed by changes to this property will be removed from the right hand side and from the bottom of the screen.

The height and width properties, for rows and columns respectively, will be used at runtime to render the frameset. These properties can be set in the editor by dragging one of the mullions associated with a splitter, or it can be entered in the properties window. The size properties are array-valued, with one occurrence for each row or column. Note, however, that the height of the bottommost row, and the width of the rightmost column, cannot be changed. These are computed by the sizes of the other rows or columns, and the height of the container.

Like the other size-related properties in Panther, the height and width properties for splitter rows and columns can be specified using a variety of units. The default is grid units. In many cases pixels will be the best unit to use.

The Web Options and HTML Options are relevant to web deployment of framesets. See “Web Deployment of Framesets” below.

Pane Widgets

Unlike other widgets, pane widgets are not explicitly created by Panther users. Pane widgets are implicitly created when splitter widgets are placed on a Screen.

When a vertical or horizontal splitter is placed on an empty frameset, two panes are created. When another splitter is placed in a previously defined pane, new panes are added. Previously defined panes are not destroyed. Panes are added to the top and left, shifting existing panes to the right and down.

A pane widget is selected in the editor by clicking on a frameset screen that contains at least one splitter widget. You can also use the widget list to select a pane.

Note that if a frameset contains at least one splitter, it might not be clear how to select the frameset itself. Clicking on one of the mullions between two panes will select a splitter. Clicking on the background of the frameset will select one of the panes. The frameset will be selected when it has focus, and none of the widgets on the frameset screen is selected. A quick way to do that is to click on a pane, and then shift-click on the same pane again. The first click selects the pane, and deselects everything else. The shift-click deselects the pane, leaving the frameset itself selected. You can also select the frameset using the widget list.

Pane widgets display the following properties in the properties window when selected:

```
IDENTITY
       Name
       Comments
       MEMO TEXT
              Memo1
              etc.

FORMAT/DISPLAY
       Form Name

WEB OPTIONS
       No Border
       Scrolling
       No Resize
       Margin Height
       Margin Width
       HTML OPTIONS
              Attributes
```

The `Form Name` property is used to designate which screen to render inside the pane at runtime. This can be the name of any Panther screen, or the name of another frameset. It is possible to nest framesets within one another.

The Web Options and HTML Options are relevant to web deployment of framesets. See “Web Deployment of Framesets” below.

Frameset Properties

A frameset supports a set of properties much like that of an ordinary screen. When a frameset is selected in the editor, the Properties window displays the following properties:

```
IDENTITY
       Title
       Inherit From
       Comments
       Dialog
       → Keep In Frame
```

```
        → etc.
        3D
        Mnemonic Position
        Java tag
        MEMO TEXT
               Memo1
               etc.

GEOMETRY
        Height
        Width
        Resizable
        Resize Function
        Startup
        Max/Min
        Grid Height
        Grid Width

COLOR
        Color Type
        etc.

FONT
        Font Name
        etc.

FOCUS
        Entry Function
        Exit Function
        JPL Procedures
        Control Strings
        Menu Name
        Menu Script File

DISPLAY
        System menu
        Title bar
        Pointer
        Icon
        Wallpaper Pixmap

WEB OPTIONS
        Display Window
        HTML OPTIONS
               Head Markup
               NOFRAMES Markup
        BROWSER OPTIONS
               JavaScript
               VBScript
               On Load
               On Unload
```

These properties are used in the same way as the corresponding properties of screens. The properties under the Web Options heading and the HTML Options and Browser Options headings are relevant to web deployment of framesets.

Many of the properties of screens aren't relevant to framesets. The corresponding properties of the screens displayed in the frameset's panes would be the ones relevant at runtime.

The wallpaper and color properties deserves special mention. The specified wallpaper or color will only be seen in an empty pane, that is, a pane that has no screen associated with it. A pane that has a screen associated with it will display the background color or wallpaper of that screen.

Programming With Framesets

At runtime, the frameset and the screens displayed in its panes all act as sibling windows. Focus can move from window to window within a frameset, and whatever other windows might be open, as among sibling windows in a Panther application that doesn't use framesets. The frameset itself cannot normally get focus after it has been opened.

Opening a frameset

A frameset is opened like any other screen. When a frameset is opened, it goes through the usual set of screen entry operations (unnamed JPL, screen entry function, etc.) Note, however, that all of these operations take place before the screens designated to be opened in the frameset's panes have been opened. This means that processing associated with the frameset's entry procedures cannot reference fields on the screens that will be opened in the frameset's panes.

After the frameset's entry procedures, the screens designated to be opened in the frameset's panes will be opened. They will be opened from bottom to top, and from right to left. So the pane in the lowest and rightmost position will be populated first, and the pane in the topmost and leftmost position will be populated last. Each screen opened in a pane will itself go through the normal screen entry events. The screens are opened in the reverse or Panther's usual left-to-right, top-to-bottom order so that the screen in the upper left corner will be the last screen opened and will be the one to retain focus. In this manner none of the screens placed in the panes undergo two entry events while opening the frameset.

Once all of the frameset's panes have been populated by their designated screens, the screen in the topmost and leftmost pane will gain focus. Focus will go to the first available field in that screen. Only at this point will the keyboard stack be opened. Any logical keys ungotten by any of the screen entry procedures, including that of the frameset itself will be read by the first screen to get focus.

Cursor movement and window management using framesets

If a frameset is the only window open, cursor movement will behave much as though the screens in each of the panes were in separate sibling windows. Tabbing order in each pane will be governed by the usual rules. If you want a `TAB` from a field in one pane to lead to a field in another pane, you will have to write a routine that calls `sm_wselect`, and then perhaps `sm_gofield` to put focus where you want it.

When focus leaves one pane, the screen in that pane undergoes a screen exit event, with the `K_HIDE` flag set. Likewise when focus enters a pane, the screen in that pane undergoes a screen entry event with the `K_EXPOSE` flag set.

If a frameset is not the only screen open, things get more complicated. To understand how things work, it's best to think in terms of the window stack, and to remember that the frameset itself is a window on the stack, as are the screens in the frameset's panes.

When a frameset is first opened it is the active window while it opens the screens that are to populate its panes. Imagining a simple scenario of opening a frameset with three panes, the sequence of entry and exit events would be as follows:

- frameset enter (`K_ENTRY`)
- frameset exit (`K_EXIT | K_EXPOSE`)
- pane3 enter (`K_ENTRY`)
- pane3 exit (`K_EXIT | K_EXPOSE`)
- pane2 enter (`K-ENTRY`)
- pane2 exit (`K_EXIT | K_EXPOSE`)

- pane1 (`K_ENTRY`)

at this point the window stack would look like:

- pane1 (active)
- pane2
- pane3
- frameset

After its initial entry the frameset itself will not, under normal circumstances, get focus, or undergo entry or exit events. Thus the frameset will tend to linger at the bottom of the window stack.

If the user clicks a button on the first pane that opens another window, a sibling to the frameset itself, the following entry and exit events would occur:

- pane1 exit (`K_EXIT | K_EXPOSE`)
- otherwindow enter (`K_ENTRY`)

at which point the window stack would then look like this:

- otherwindow (active)
- pane1
- pane2
- pane3
- frameset

Let's suppose a user now clicks on pane3 in the frameset. The following entry and exit events would occur:

- otherwindow exit (`K_EXIT | K_EXPOSE`)
- pane3 entry (`K_ENTRY`)

and the window stack would end up looking like:

- pane3 (active)
- otherwindow
- pane1
- pane2
- frameset

Other much more complicated scenarios could be envisioned. Things would get very complicated in the case where framesets are nested. In general, most people will probably not want to do anything on screen hide events (`K_EXIT | K_EXPOSE`) for screens that are in framesets.

Closing framesets

A screen in Panther is typically closed in one of three ways:

- by the user clicking on the 'x' control on the screen's title bar
- programmatically, with the function `sm_jclose`
- by means of the logical key `EXIT`

A frameset is closed in the same manner.

Clicking on the 'x' control on the title bar will cause the logical key `EXIT` to be issued, and so the first and third of these methods will be same.

When a screen gets the logical key `EXIT` from the key stack, it calls `sm_jclose`. So really all three of these methods are fundamentally the same.

The function `sm_jclose` is one of those that implicitly operates on the screen that currently has focus. If the screen that currently has focus is in a frameset, then a call to `sm_jclose` will close the whole frameset.

Closing a screen that is in a frameset will cause the whole frameset to be closed. The screens in the frameset will all be closed, in the order in which they are found in the window stack. This list of screens includes the screens in the panes of any other framesets that are nested in the frameset being closed. After all the screens in all of the panes of the framesets have been closed, the framesets themselves will be closed. The outermost frameset will always be the last to close.

As an example, consider the following scenario. The window stack looks like this:

- pane1 (active)
- pane2
- otherwindow
- pane3.1
- pane3.2
- pane3.3
- pane3 (nested frameset)
- frameset

(This means to indicate that pane3 in the frameset is itself a frameset, containing three panes. The panes in that nested frameset are the ones designated 3.1, 3.2, and 3.3.)

If a call to `sm_jclose` is now issued, the window stack is reordered to place all the panes contained in the frameset (including those in the panes of the nested frameset) on the top, followed by all the framesets.

- pane1 (active)
- pane2
- pane3.1
- pane3.2
- pane3.3
- pane3 (nested frameset)
- frameset
- otherwindow

And the screens will be closed in order, from the top down. Until the outermost frameset that the screen that had been active when `sm_jclose` was called is closed.

When closing the screens contained in the frameset, normal screen events would occur, just as though a series of sibling windows were being closed one after another. In this case, the event sequence would be:

- pane1 exit (K_EXIT)
- pane2 enter (K_ENTRY | K_EXPOSE)
- pane2 exit (K_EXIT)
- pane3.1 enter (K_ENTRY | K_EXPOSE)
- pane3.1 exit (K_EXIT)
- pane3.2 enter (K_ENTRY | K_EXPOSE)
- pane3.2 exit (K_EXIT)
- pane3.3 enter (K_ENTRY | K_EXPOSE)
- pane3 exit (K_EXIT)
- pane3 enter (K_ENTER | K_EXPOSE)
- pane3 exit (K_EXIT)
- frameset enter (K_ENTRY | K_EXPOSE)
- frameset exit (K_EXIT)
- otherwindow enter (K_ENTRY | K_EXPOSE)

Which would leave the window stack as:

otherwindow (active)

The reordering of the window stack prior to closing the screens is not accompanied by screen events, except in the unusual circumstance that sm_jclose is issued when a frameset has focus. Let's consider such a scenario. If the window stack looks like:

- frameset
- pane1
- otherwindow
- pane2
- pane3

And sm_jclose is called, the window stack will be reordered to place all the screens in the frameset at the top, followed by all the framesets contained in the outermost frameset. In this case, the following events would occur:

- frameset exit (K_EXIT | K_EXPOSE)
- pane1 enter (K_ENTRY | K_EXPOSE)

The rest of the reordering, to place the frameset after all the screens in its panes, occurs without any screen events taking place, and will leave the window stack as:

- pane1 (active)
- pane2
- pane3
- frameset
- otherwindow

And the screens will close, with CLOSE and EXPOSE events occurring as described in the previous scenario.

Note that this latter scenario is very unusual. Under normal circumstances a frameset will only gain focus when it is first opened, and will not be in focus when user-written code is being executed. The only way to force a frameset to gain and retain focus is by explicitly giving it focus with a call to sm_wselect or sm_n_wselect.

Runtime properties

Screen and Frameset properties

Screens and framesets have a property called PR_FRAMESET that returns a handle to the frameset the screen or frameset is inside, just as the PR_GRID property returns a handle to the grid a particular field is in. If a screen or frameset is not in a frameset PR_FRAMESET returns "".

Additionally, screens and forms have a PR_SPLITTER property that returns a handle to the splitter that defines the pane in which the form is contained. If the screen is not in a frameset, this property will return "".

Screens and forms also have a PR_PANE property that returns a handle to the pane that contains the form. If the screen is not in a frameset, this property will return "".

Splitter Properties

A splitter will return PV_SPLITTER when PR_WIDGET_TYPE is queried.

The PR_SPLITTER_ROWS and PR_SPLITTER_COLS properties of splitter widgets are read-only at runtime. Since these properties are read-only, you cannot change a frameset's pane configuration on the fly. To change the number of subwindows in a frameset, you will have to nest framesets.

PR_SPLITTER_ROWS returns the number of rows in the splitter. This will be an integer from 1 to 16, but it cannot be 1 if the number of columns is 1.

PR_SPLITTER_COLS returns the number of columns in the splitter. This will be an integer from 1 to 16, but it cannot be 1 if the number of rows is 1.

The PR_SPLITTER_ROW_HEIGHT property gets or sets the height of each row. This is a multi-occurrence property, indexed from 1 to PR_SPLITTER_ROWS. The height of the bottommost row cannot be set. It is computed based on the sum of the other row heights and the size of the container.

The PR_SPLITTER_COL_WIDTH property gets or sets the width of each column. This is a multi-occurrence property, indexed from 1 to PR_SPLITTER_COLS. The width of the rightmost column cannot be set. It is computed based on the sum of the other column widths and the size of the container.

The PR_MEMBER property is an indexed property that will return handles for the panes in the splitter. The pane in the first row, first column will have the index 1. The pane in the first row, second column will have index 2. The pane in the second row, first column will have index PR_SPLITTER_COLS +1. In general the formula to determine a pane's index is:

```
index = (row - 1) + PR_SPLITTER_COLS + column
```

PR_FRAMESET will return a handle to the frameset that the splitter is contained in.

If a splitter is contained in a pane defined by another splitter, the property PR_SPLITTER will return a handle to that other splitter and the property PR_PANE will return a handle to the pane in which the splitter is contained.

Pane Properties

The PR_PANE_FORM property of pane widgets can be changed at runtime. Changing the screen associated with a pane at runtime should close the screen currently in that pane, and then open the newly specified screen in its place. Clearing the PR_PANE_FORM property of a pane will leave the pane empty. Since issuing sm_jclose

will close the whole frameset, clearing the PR_PANE_FORM property of a pane is the only way to close a single screen in a frameset.

The PR_PANE_ROW property gets the row number of the pane.

The PR_PANE_COL property gets the column number of the pane.

The PR_PANE_INDEX property gets the index of this pane. The indices are the same as those used by the PR_MEMBER property of framesets. The JPL code

```
index = mypane->pane_index
```

is equivalent to:

```
index = (mypane->PR_PANE_ROW - 1) + \
       (@widget(mypane->PR_SPLITTER)->PR_SPLITTER_COLS + \
       mypane->PR_PANE_COL
```

The PR_MEMBER property of a pane will return an object id for the pane's contents. This will either be the id of a form (a screen or another frameset), or a splitter, or nothing.

The PR_FRAMESET property returns an object id for the frameset in which the pane is contained.

The PR_SPLITTER property returns an object id for the splitter that contains the pane.

Web Deployment of Framesets

If a frameset is brought up in a jserver, HTML to represent the frameset will be generated and sent to the browser.

The properties in the Web Properties sections of the Properties window control various options relevant to HTML generation of framesets.

Frameset Web Properties

Framesets support the NOFRAMES Markup property. The text entered here will be retured to the browser in the event that the frameset is accessed by a browser that does not support frames. The value of this property is indexed (like the JavaScript and VBScript properties). If no value is provided for the NOFRAMES Markup property, the following text will be returned to browsers that do not support frames:

> This Page Requires Frames.

The NOFRAMES Markup property is accessible at runtime, its C constant name is PR_FRAME_NOFRAMES and its JPL mnemonic is frame_noframes.

Splitter Web Properties

Splitter Widgets display the following properties under the WEB OPTIONS heading in the properties window:

```
No Border
Spacing
Row 1 Height
Row2 Height
…
Col 1 Width
```

```
Col 2 Width
…
```

The No Border property defaults to No. Hence by defaults borders will be shown. If No Border is set to Yes the HTML output is `FRAMEBORDER="0"`. The No Border property is referred to at runtime as `PR_FRAME_NOBORDER` in C and as `frame_noborder` in JPL.

Spacing is the width of the mullions between panes. In MSIE this is the `FRAMESPACING` attribute. In Netscape this is the `BORDER` attribute. Setting `Spacing = 0` has the same effect as setting `No Border = Yes`. The Spacing property is referred to at runtime as `PR_FRAME_SPACING` in C and as `frame_spacing` in JPL.

The height and width properties in the WEB OPTIONS section can be used to override the similar properties in the GEOMETRY section. The latter are used for GUI display, and will be used for HTML generation as well, unless height and width properties are explicitly specified in the WEB OPTIONS section.

There will be one height shown for each row, and one width for each column. If there is only one row, then no heights will be shown, and if there is only one column, then no widths will be shown.

The WEB heights and widths can be either given as percentages or in pixels. They also accept the special value *. Omitted entries default to *, which tells the browser to choose whatever size it wants. Typically it will divide the available space equally among the regions specified with stars.

For example:

Row 1 Height = 25%
Row 2 Height =
Row 3 Height = 25%
Row 4 Height = *

will be output as ROWS="25%,*,25%,*". The browser will allocate half the available space to the two specified rows, and divide the remaining space equally among the rows specified with stars. In this case, that would result in four rows of equal size.

Heights and widths should either be given in percentages or in pixels. Do not mix units.

Pane Web Propeerties

Like Splitters, panes each have the No Border property. As noted above, it defaults to No. If set to Yes the 3-D border around the pane may be removed (or it may not be, this is browser-dependent behavior). Setting No Border to Yes for some panes and No for other panes can yield strange and unsightly results. It is recommended that you use the Splitter-level `No Border` property if possible.

The Scrolling property supports three values: Auto, No and Yes. It defaults to Auto. If set to Auto, scroll bars will appear only when needed. If set to `No`, scroll bars will never appear and if set to `Yes` scroll bars will always appear. The `Scrolling` property is referred to at runtime as `PR_FRAME_SCROLLING` in C and as `frame_scrolling` in JPL.

The Margin Height and Margin Width properties control the space between the splitter lines and the HTML inside the pane. It defaults to 13 or 14 pixels. However, if you set one and not the other, the unspecified one will default to 0. These are referred to at runtime as `PR_FRAME_MARGIN_HEIGHT` and `PR_FRAME_MARGIN_WIDTH` in C and as `frame_margin_height` and `frame_margin_width` in JPL.

Tooltip Support

Tooltip support has been implemented for screens displayed on Microsoft Windows. A tooltip is a pop-up containing a text message that appears next to a widget in response to a mouse hover event. A mouse hover event occurs when the mouse pointer remains over a widget for approximately one-and-a-half seconds. Tooltips can be specified for most widgets. The exceptions are lines, boxes, grid frames and tab decks.

Adding Tooltips in the Editor

The property Tooltip Text is found in the HELP category in the Properties Window in the editor. This is a string valued property that simply holds the text to be shown in the tooltip for a widget. If no tooltip text is specified for a widget, it will not display a tooltip at runtime. It is not possible to have a widget display an empty tooltip.

Tooltip Text is not a multi-valued property. All the occurrences of an array will show the same tooltip at runtime.

The Tooltip Text property can be changed at runtime. Its value in C is `PR_TOOLTIP_TEXT` and its JPL mnemonic is `tooltip_text`.

Controlling Tooltip Appearance

Tooltip appearance is, in general, determined by Windows desktop settings. There is, however, a property called `tooltip_style` (`PR_TOOLTIP_STYLE` in C) that can be used to get and set the Windows tooltip style bits.

The `tooltip_style` property is of application scope. It applies to the tooltips associated with all widgets and to the tooltips associated with the toolbar.

The following table lists the bits that can be set with the `tooltip_style` property, the Windows bits they correspond to, and the values so defined.

Panther bit	Windows bit	Value
PV_TOOLTIP_ALWAYS	TTS_ALWAYSTIP	0x01
PV_TOOLTIP_NOPREFIX	TTS_NOPREFIX	0x02
PV_TOOLTIP_NOANIMATE	TTS_NOANIMATE	0x10
PV_TOOLTIP_NOFADE	TTS_NOFADE	0x20
PV_TOOLTIP_BALLOON	TTS_BALLOON	0x40

Some of these bits are relevant only to the latest versions of the Windows operating system. Other bits are undefined, but may be defined in future releases of Windows.

`PV_TOOLTIP_ALWAYS` is set by default. This means that the tooltip will be displayed even when the Panther application is not active. Clearing this bit will prevent tooltips from being displayed when the Panther application is inactive. However, tooltips will always display on an inactive form within an active Panther application.

`PV_TOOLTIP_NOPREFIX` is off by default and is irrelevant to Panther applications.

`PV_TOOLTIP_NOANIMATE` and `PV_TOOLTIP_NOFADE` are off by default. They are used to override Windows 2000 desktop settings. If a user's desktop is set up with the tooltip animation and fading features enabled tooltips will display those behaviors. Setting these bits will override the desktop preferences and turn off animation or fading, respectively, for tooltips in the Panther application.

PV_TOOLTIP_BALLOON is off by default. It is defined only in Windows 2000 or if Microsoft Internet Explorer 5 is installed on an earlier version of Windows. Setting this bit will cause tooltips to be displayed in a "balloon," like the voice balloon used in comic strips, rather than the usual plain rectangle.

Examples of use:

This JPL code will turn off the TOOLTIP_ALWAYS bit and prevent tooltips from being displayed for Panther widgets when the Panther application is not active:

```
@app()->tooltip_style = @app()->tooltip_style & ~ PV_TOOLTIP_ALWAYS
```

This will set both the BALLOON bit and the NOANIMATE bit:

```
@app()->tooltip_style = @app()->tooltip_style | \
       PV_TOOLTIP_BALLOON | PV_TOOLTIP_NOANIMATE
```

New Window Styles

Panther 4.5 supports some new options for opening windows under the Windows operating system. Previously windows could either be opened as MDI windows, or as dialogs. An MDI window cannot be moved outside the MDI frame. A dialog can be moved outside the MDI frame, but blocks access to the MDI frame – when a dialog is opened, focus cannot be given to any MDI windows, nor can the MDI menu be accessed until the dialog is closed.

The new options are implemented by the properties Keep in Frame and Topmost. These are screen-level properties and are found under the IDENTITY section in the Properties Window.

The Keep In Frame Property

The Keep in Frame property is a subproperty of the Dialog property. If a window is specified as a dialog, the new option doesn't apply.

The default value for Keep in Frame is Yes. A non-dialog window with Keep In Frame set to Yes will behave exactly like the MDI windows Panther has always supported. Setting Keep In Frame to No will make the screen open as a non-MDI window, that is not a modal dialog. Such a screen can be moved outside the MDI frame, but does not block access to the MDI windows, or to the MDI's menu bar.

When Keep in Frame is set to No, two further properties appear in the Properties window. These are Keep On Top and Parent Window. The Parent Window property is relevant only to windows that have Keep in Frame set to No, and Keep On Top also set to No. The Parent Window property will not appear in the Properties window if the Keep On Top property is set to Yes.

Unlike dialogs, screens that have Keep in Frame set to No can be minimized or maximized. The behavior when they are minimized is determined by the Parent Window property.

Windows outside the MDI frame do not appear in the Windows menu list in the standard MDI application menu. And they cannot be accessed by function keys like `Ctrl+F6`. They can, however be accessed by Windows shortcuts such as `Alt+F6`.

The Parent Window Property

Windows that have the Keep in Frame property set to No and are not designated as Topmost windows can either have no parent or they can be parented by the MDI frame. If the parent is designated as None then the screen can be hidden by the MDI frame when the latter is given focus. If the screen is designated as parented by the MDI frame, then the screen will always appear on top of the MDI frame. In this case, even if focus is given to the MDI frame, or one of the windows inside the MDI frame, the non-Keep in Frame window will always be displayed, and will not be obscured behind the MDI frame. Note that a non-Keep in Frame window whose parent is the MDI frame can still be hidden behind other windows that are not part of the Panther application. To designate that a window should never be obscured behind another window, whether that other window be part of the Panther application or not, use the Keep on Top property.

When a non-Keep In Frame window is minimized, the location of the icon representing ot depends on the window's parent. A non-Keep In Frame window with no parent will be represented by an item on task bar. A non-Keep in frame window whose parent is the MDI frame will be represented by an icon in the desktop's lower left, just above the taskbar.

The Keep On Top Property

A non-Keep in Frame window can be designated as a Topmost window by setting the Keep On Top property to Yes. A Topmost window will generally be rendered on top of any other window on the desktop, no matter

which application is active. The only windows that can be placed on top of a Topmost window are other Topmost windows.

Runtime Access to the Window Options Properties

At runtime the properties `PR_KEEP_IN_FRAME`, `PR_KEEP_ON_TOP`, and `PR_WINDOW_PARENT` are read only. Once a window has been opened, its status with regard to these settings cannot be changed.

The Frameset Sample Application

In the directory `$SMBASE\samples\frameset` there are three files: a .lib file, a database file and a README file. These constitute the sample application provided to illustrate the use of framesets. This application uses ODBC to access its database. To run the frameset sample application you must have the ODBC database driver installed.

This application also provides examples of code to manipulate a few common ActiveX controls: the TreeView control, the ListView control and the ImageList control. The code for all the application's functionality is in the screens that are found in the library Biblio.lib. The JPL that manipulates the ActiveX controls has been commented and written in such a way that it can easily be re-used.

See the `samples\frameset\README` file for more information, regarding installing and running the frameset sample.

New Status Bar Functions

This section describes several new functions that can be used to create and manipulate subsections of the status bar.

sm_sb_insert
Inserts a status bar section

```
int sm_sb_insert( int sectno, int type, int length );
```

`sectno` The index, in the array of sections, of the section to be added.

`type` The type of the new section, one of the following constants:

```
SBS_TEXT
SBS_SEPARATOR
SBS_SYSTEM_TIME
SBS_ELAPSED_TIME
SBS_OVERLAY
SBS_CAPS
SBS_NUM
SBS_SCROLL
```

`length` The length of the section to be added.

Returns

• The section number given to the new section
-1 Failure

Description

This function inserts a new section on the status bar. When the status bar is initially created it contains a single section of type `SBS_MSGLINE`. This initial section is the one written to by the various Panther functions that send messages to the status line. The initial `SBS_MSGLINE` section occupies the `0` position in the array of status bar sections. Newly added sections must be placed to the right of the initial section, hence the value of the first argument to `sm_sb_insert` cannot be `0`. If you supply a negative value to the first parameter, the newly added section will be the rightmost, no matter how many sections already exist.

The newly added section must be one of several pre-defined types, as specified by the second argument to `sm_sb_insert`. You cannot add a second section of type `SBS_MSGLINE`. Hence, the valid values for the second argument are as follows:

`SBS_TEXT` This type of section is used to display text. Text is written to such a section using the funtion `sm_sb_settext`.

`SBS_SEPARATOR` This type of section is used to mark a boundary between two other sections. In character mode, it is equivalent to `SBS_TEXT`, and you can write whatever character you wish to it, to mark the section boundary. In a GUI `SBS_SEPARATOR` sections aren't displayed in a recessed style.

`SBS_SYSTEM_TIME` This type of section displays the system time. The format for the time displayed is set by the function `sm_sb_format`. The default format shows the time in a 12-hour clock, with an AM/PM indicator.

`SBS_ELAPSED_TIME` This type of section displays the time elapsed since the section was created. The format for the time displayed is set by the function `sm_sb_format`. The default format shows the time in the form '00:00:00'.

SBS_OVERLAY	This type of section displays the state of Panther's insert/overstrike mode. The length parameter is ignored if this is the type specified. In character mode the length defaults to 3, and will either display 'OVR' or be blank. In a GUI the 'OVR' indicator may be grayed out rather than blanked.
SBS_CAPS	This type of section displays the CAPS LOCK state of the keyboard. The length parameter is ignored if this is the type specified. This type is not supported in character mode.
SBS_NUM	This type of section displays the NUM LOCK state of the keyboard. The length parameter is ignored if this is the type specified. This type is not supported in character mode.
SBS_SCROLL	This type of section displays the SCROLL LOCK state of the keyboard. The length parameter is ignored if this is the type specified. This type is not suported in characted mode.

Other than for SBS_TEXT and SBS_SEPARATOR, you can have only one section of each type on the status bar. Calls to sm_sb_insert that specify a type that already exists on the status bar will have no effect. You can insert any number of SBS_TEXT or SBS_SEPARATOR sections.

The length parameter is the length, in characters, of the section to be added. The length specified should be greater than or equal to the lenth of any text that might be placed in that section. This parameter is ignored for some section types, as noted above.

Note that the length of the status bar as a whole remains constant, and that the message line section initially occupies all of it. In Motif, the message line section is always 255 characters long, so any sections placed after it will appear displaced by 255 character positions. As a result, in Motif sections added to the status bar will probably not be visible unless the window containing the status bar is very wide. To compensate for this you can, in Motif, add a trailing SBS_SEPARATOR section that's wide enough to force the section to the right of the message line section to become visible.

Since in the GUIs the screen space allocated to status line sections is font-dependent, you may need to experiment with different lengths to get the status line sections to appear the way you want them.

See Also sm_sb_delete, sm_sb_format, sm_sb_gettext, sm_sb_settext

sm_sb_delete

Deletes a status bar section

```
int sm_sb_delete( int sectno );
```

`sectno` The number of the section to be deleted.

Returns

0 Success
-1 Failure

Description

This function deletes a section of the status bar as specified by its index in the array of staus bar sections. The initial section, the message line, always has section number `0` and cannot be deleted. Hence, the argument to this function must always be `>=1`.

sm_sb_format

Sets a format string for a status bar section

```
int sm_sb_format( int sectno, char* format )
```

`sectno`	The number of the section for which to specify a format string.
`format`	A format string for a section.

Returns

0 Success
-1 Failure

Description

This function sets the format string for a status bar section. This is relevant only for sections of type `SBS_SYSTEM_TIME` and `SBS_ELAPSED_TIME`, as descibed above.

The second argument, `format`, represents a format string. Valid date/time format strings are described in the documentation for the function `sm_sdtime`. Note that an `SBS_ELAPSED_TIME` section displays a clock that starts at midnight when the section is created. So a format string for a section of that type should be chosen so that it is meaningful in that context.

sm_sb_gettext

Get contents of a status bar section

```
char* sm_sb_gettext( int sectno );
```

`sectno`	The number of the section being queried.

Returns

• The section's contents
-1 Error

Description

This function gets the contents of a status bar section. The text returned is as shown on the status bar, and may differ from the text set with `sm_sb_settext` if that text contained formatting tokens.

sm_sb_settext

Set contents of a status bar section

```
int sm_sb_settext( int sectno, char* text );
```

`sectno`	The number of the section to update
`text`	The text to place in that section

Returns

0 Success
-1 Failure

Description

This function assigns contents to a section of the status bar. The text specified as the second argument to this function may contain formatting tokens such as `%A` and `%K`. See the section entitled “Providing Status Line Help Text” in the Editors Guide and the description of the JPL command `msg` for descriptions of the valid formatting and key value display tokens.