Contents

This PilotAPI help file was created by Matt Peterson (mattp@usa.net) for use with Darrin Massena's Alternate Software Development Kit for the USR Pilot. For more info on the ASDK, see http://www.massena.com/darrin/pilot

I've modeled this closely after the Win32 API reference, as that is the help file I am most used to working with. If you have any suggestions for improvements please let me know.

This is a work in progress with many enhancements planned:

Finish formatting for all functions (at over 7000 lines of data and over 600 functions) this may take some time

Group functions ("String Manager Functions", "Time Functions", etc.)

Add Events, Constants, and Types

"From the field" information, not documented in USRobotics' SDK

Sample code

CtlDrawControl

Draws a control object (and the text in it) on screen. The control is drawn only if its usable attribute is TRUE.

```
void CtlDrawControl (
    ControlPtr ControlP         // address of control object
);
```

Parameters

ControlP
Pointer to the control object to draw.

Return Value

Returns nothing.

Remarks

Sets the visible attribute to TRUE.

See Also

CtlSetUsable, CtlShowControl

CtlEraseControl

Erase a usable and visible control object and its frame from the screen.

```
void CtlEraseControl (
    ControlPtr ControlP         // address of control object
);
```

Parameters

ControlP
Pointer to control object to erase.

Return Value

Retirns nothing.

Remarks

Sets the visible attribute to FALSE.

CtlGetLabel

Returns a character pointer to a control's text label.

```
CharPtr CtlGetLabel (
    ControlPtr ControlP         // address of control object
);
```

Parameters

ControlP
Pointer to control object.

Return Value

Returns a pointer to a null-terminated string.

See Also

CtlSetLabel

CtlGetValue

Return the current value (on or off) of the specified control. This function is valid only for push buttons and check boxes. The return value is undefined for other control types.

```
short CtlGetValue (
    ControlPtr ControlP          // address of control object
);
```

Parameters

ControlP
Pointer to a control object.

Return Value

Returns the current value of the control; 0 = off, 1 = on.

See Also

CtlSetValue

CtlHandleEvent

Handle event in the specified control object.

```
Boolean CtlHandleEvent (
    ControlPtr ControlP,        // address of control object
    EventPtr EventP             // address of   event structure
);
```

Parameters

ControlP
Pointer to control object.
EventP
Pointer to an EventType structure.

Return Value

Returns TRUE if an event is handled by this function. Events that are handled are:
penDownEvent if the pen is within the bounds of the control
ctlEnterEvent and ctlExitEvent, if the control ID in the event data matches the control's ID.

Remarks

The control object's usable, enabled, and visible attributes must be TRUE. This routine handles three type of events: penDownEvent, ctlEnterEvent, and ctlRepeatEvent.

When this routine receives a penDownEvent, it checks if the pen position is within the bounds of the control object. If it is, a ctlEnterEvent is added to the event queue and the routine exits.

When this routine receives a ctlEnterEvent, the control object is inverted.

When this routine receives a ctlEnterEvent or ctlRepeatEvent, it checks that the control ID in the passed event record matches the ID of the specified control. If they match, this routine tracks the pen until it comes up or until it leaves the ob-ject's bounds. When that happens, ctlSelectEvent is sent to the event queue if the pen came up in the bounds of the control. If the pen exits the bounds, a ctlExitEvent is sent to the event queue.

CtlHideControl

Set a control's usable attribute to FALSE and erase the control from the screen. This function calls CtlEraseControl.

```
void CtlHideControl (
    ControlPtr ControlP         // address of control object
);
```

Parameters

ControlP
Pointer to the control object to hide.

Return Value

Returns nothing.

Remarks

A control that is not usable doesn't draw and doesn't respond to the pen.
Sets the visible and the usable attributes to FALSE.

See Also

CtlShowControl

CtlHitControl

Simulate tapping a control. This function adds a ctlSelectEvent to the event queue.

```
void CtlHitControl (
    ControlPtr ControlP         // address of control object
);
```

Parameters

ControlP
Pointer to a control object.

Return Value

Returns nothing.

Remarks

Useful for testing.

CtlEnabled

Return TRUE if the control is enabled. Disabled controls do not re-spond to the pen.

```
Boolean CtlEnabled (
    ControlPtr ControlP         // address of control object
);
```

Parameters

ControlP
Pointer to control object.

Return Value

Returns TRUE if enabled, FALSE if not.

See Also

CtlSetEnabled

CtlSetEnabled

Set a control as enabled or disabled. Disabled controls do not re-spond to the pen.

```
void CtlSetEnabled (
    ControlPtr ControlP,        // address of control object
    Boolean enable              //
);
```

Parameters

ControlP
Pointer to a control object.

enable
TRUE to set enabled, FALSE to set not enabled.

Return Value

Returns nothing.

See Also

CtlEnabled

CtlSetLabel

Set the current label for the specified control object. If the control object currently has its usable and visible attributes set to TRUE, redraw it with the new label.

```
void CtlSetLabel (
    ControlPtr ControlP,        // address of control object
    CharPtr newLabel            //
);
```

Parameters

ControlP
Pointer to a control object.

newLabel
Pointer to the new text label. Must be a null-terminated string.

Return Value

Returns nothing.

Remarks

This function resizes the width of the control to the size of the new label.
The pointer passed to this function is stored in the control's data structure; the control does not make a copy of the string passed.

See Also

CtlGetLabel

CtlSetUsable

Set a control usable or not usable.

```
void CtlSetUsable (
    ControlPtr ControlP,        // address of control object
    Boolean usable
);
```

Parameters

ControlP
Pointer to a control object.

usable
TRUE to set usable, FALSE to set not usable.

Return Value

Returns nothing.

Remarks

Does not usually update the control.

See Also

CtlEraseControl, CtlDrawControl

CtlSetValue

Set the current value (on or off) of the specified control. If the control is visible, it is visually updated.

```
void CtlSetValue (
    ControlPtr ControlP,        // address of control object
    short newValue              //
);
```

Parameters

ControlP
Pointer to a control object.

newValue
0 = off, non-zero = on.

Return Value

Returns nothing.

Remarks

Does not usually update the control.
This function works only with push buttons and check boxes.
Other controls ignore calls to this function.

See Also

CtlGetValue

CtlShowControl

Set a control's usable attribute to TRUE and draw the control on the screen. This function calls CtlDrawControl.

```
void CtlShowControl (
    ControlPtr ControlP         // address of control object
);
```

Parameters

ControlP
Pointer to a control object.

Return Value

Returns nothing.

Remarks

If the control is already usable, this function is the functional equivalent of CtlDrawControl. Sets the visible and the usable attributes to TRUE.

See Also

CtlHideControl

FldCalcFieldHeight

Determine the height of a field for a string.

```
Word FldCalcFieldHeight (
    CharPtr chars,
    Word maxWidth
);
```

Parameters

chars
Pointer to a null-terminated string.

maxWidth
Maximum line width in pixels.

Return Value

Returns total number of lines needed to draw the string passed.

FldCompactText

Compact the memory block that contains the text of the field to release any unused space.

void **FldCompactText (**
    FieldPtr fld
);

Parameters

fld
    Pointer to a field object (FieldType data structure).

Return Value

Returns nothing.

Remarks

As characters are added to the text of a field, the block that contains the text is grown. The block is expanded in chunks so that it doesn't have to expand each time a character is added. This results in some unused space in the text block.
Needs to be called by applications on field objects which edit data records in place before the field is unlocked, or at any other time when a compact field is desirable; for example, when the form is being closed.

FldCopy

Copy the current selection to the text clipboard.

```
void FldCopy (
    FieldPtr fld
);
```

Parameters

fld
Pointer to a field object (FieldType data structure).

Return Value

Returns nothing.

Remarks

This function leaves the current selection highlighted.
This functions replaces anything previously in the text clipboard.
If there is no selection, this function does nothing.

See Also

FldCut, FldPaste

FldCut

Copy the current selection to the text clipboard, delete the selection from the field, and redraw the field.

```
void FldCut (
    FieldPtr fld
);
```

Parameters

fld
Pointer to a field object (FieldType data structure).

Return Value

Returns nothing.

Remarks

Anything previously in the text clipboard is replaced by this function.
If there is no selection, this function does nothing.

See Also

FldCopy, **FldPaste**

FldDelete

Delete the specified range of characters from the field and redraw the field.

void **FldDelete (**
    FieldPtr fld,
    Word start,
    Word end
);

Parameters

fld
    Pointer to the field object to delete from.
start
    Starting character position.
end
    Ending character position.

Return Value

Returns nothing.

See Also

FldInsert

FldDirty

Return true if the field has been modified by the user since the text value was set (FldSetText).

```
Boolean FldDirty (
    FieldPtr fld
);
```

Parameters

fld
Pointer to a field object (FieldType data structure)

Return Value

Returns TRUE if the field has been modified by the user, FALSE if the field has not been modified.

See Also

FldSetDirty

FldDrawField

Draw the text of the field. The field's usable attribute must be TRUE or the field won't be drawn.

```
void FldDrawField (
    FieldPtr fld
);
```

Parameters

fld
Pointer to a field object (FieldType data structure).

Return Value

Returns nothing.

Remarks

This function does not erase the area behind the field before drawing.
If the field has the focus, the blinking insertion point is displayed in the field.

See Also

FldEraseField

FldEraseField

Erase the text of a field and turn off the insertion point if it's in the field.

```
void FldEraseField (
    FieldPtr fld
);
```

Parameters

fld
Pointer to a field object (FieldType data structure).

Return Value

Returns nothing.

Remarks

The function does not modify the contents of the field.
If the field has the focus, the blinking insertion point is turned off.

See Also

FldDrawField

FldFreeMemory

Release the memory allocated to the text of a field and the word-wrapping information.

```
void FldFreeFieldMemory (
    FieldPtr fld
);
```

Parameters

fld
Pointer to a field object (FieldType data structure).

Return Value

Returns nothing.

Remarks

This function releases the memory allocated to hold the text of a field (the memory block pointed to by the text member of the FieldType data structure) and the memory allocated to hold the display lines information (the memory block pointed to by the lines member in the FieldType data structure).
This function does not affect the display of the field.

FldGetAttributes

Return the attributes of a field.

void **FldGetAttributes (**
    FieldPtr fld,
    FieldAttrPtr attrP
);

Parameters

fld
    Pointer to a FieldType structure.

attrP
    Pointer to FieldAttrType, see Field.h.

Return Value

Returns nothing.

See Also

FldSetAttributes

FldGetBounds

Return the current bounds of a field.

void **FldGetBounds (**
    FieldPtr fld,
    RectanglePtr rect
);

Parameters

fld
    Pointer to a field object (FieldType data structure).

rect
    Pointer to a RectangleType structure.

Return Value

Returns the field's bounds in the RectangleType structure reference by bounds.

Remarks

Returns the rect field of the FieldType structure.

See Also

FldSetBounds

FldGetFont

Return the ID of the font used to draw the text of a field.

```
FontID FldGetFont (
    FieldPtr fld
);
```

Parameters

fld
Pointer to a field object (FieldType data structure).

Return Value

Returns the ID of the font.

See Also

FldSetFont

FldGetInsPtPosition

Return the string position of the insertion point.

```
Word FldGetInsPtPosition (
    FieldPtr fld
);
```

Parameters

fld
Pointer to a field object (FieldType data structure).

Return Value

Returns the character position of insertion point.

Remarks

The insertion point position number is to the left of the string position number. In multiline fields, line feeds are counted as part of the string and the position number after the line feed is the beginning of the next line.

See Also

FldSetInsPtPosition

FldGetMaxChars

Return the maximum number of characters the field accepts.

```
Word FldGetMaxChars (
    FieldPtr fld
);
```

Parameters

fld
Pointer to a field object (FieldType data structure).

Return Value

Returns the maximum number of characters the user is allowed to enter.

See Also

FldSetMaxChars

FldGetScrollPosition

Return the string position of the first character in the first line of a field.

```
Word FldGetScrollPosition (
    FieldPtr fld
);
```

Parameters

fld
Pointer to a field object (FieldType data structure).

Return Value

Returns the character position of first visible character.

See Also

FldSetScrollPosition

FldGetSelection

Return the current selection of a field.

```
void FldGetSelection (
    FieldPtr fld,
    WordPtr startPosition,
    WordPtr endPosition
);
```

Parameters

fld
Pointer to a field object (FieldType data structure).

startPosition
Pointer to start-character position of selected range of characters.

endPosition
Pointer to end-character position of selected range of characters.

Return Value

Returns the start and end position in startPosition and endPosition.

Remarks

The first character in a field is at position zero.
If the user has selected the first five characters of a field, startPosition will contain the value 0 and endPosition the value 5.

See Also

FldSetSelection

FldGetTextAllocatedSize

Return the number of characters allocated to hold the field's text string. Don't confuse this number with the length of the text string.

```
Word FldGetTextAllocatedSize (
    FieldPtr fld
);
```

Parameters

fld
Pointer to a field object.

Return Value

Returns the number of characters allocated for the field's text.

See Also

FldSetTextAllocatedSize

FldGetTextHandle

Return a handle to the block that contains the text string of a field.

```
Handle FldGetTextHandle (
    FieldPtr fld
);
```

Parameters

fld
Pointer to a field object (FieldType data structure).

Return Value

Returns the handle of the text string of a field; 0 is a possible value.

Remarks

If 0 is returned, no handle has been allocated for the field pointer.

See Also

FldSetTextHandle, FldGetTextPtr

FldGetTextHeight

Return the number of lines of text that the specified field has.

```
Word FldGetTextHeight (
    FieldPtr fld
);
```

Parameters

fld
Pointer to a field object (FieldType data structure).

Return Value

Returns the number of lines with text.

Remarks

Empty lines are not counted.

See Also

FldCalcFieldHeight

FldGetTextLength

Return the length of the text string of a field object.

```
Word FldGetTextLength (
    FieldPtr fld
);
```

Parameters

fld
Pointer to a field object (FieldType data structure).

Return Value

Returns the length of a field's text string.

FldGetTextPtr

Return a pointer to the text string of a field or null.

```
CharPtr FldGetTextPtr (
    FieldPtr fld
);
```

Parameters

fld
Pointer to a field object (FieldType data structure).

Return Value

Returns a pointer to the text string of a field; NULL is a possible value.

See Also

FldSetTextPtr, FldGetTextHandle

FldGetVisibleLines

Return the number of lines that can be displayed within the visible bounds of the field.

```
Word FldGetVisibleLines (
    FieldPtr fld
);
```

Parameters

fld
Pointer to a field object (FieldType data structure).

Return Value

Returns the number of lines.

FldGrabFocus

Turn the insertion point on (if the specified field is visible) and position the blinking insertion point in the field.

```
void FldGrabFocus (
    FieldPtr fld
);
```

Parameters

fld
    Pointer to a field object (FieldType data structure).

Return Value

Returns nothing.

Remarks

This function sets the field attribute hasFocus to TRUE.

See Also

FldReleaseFocus

FldHandleEvent

Handles the following events: keyDownEvent, penDownEvent, and fldEnterEvent. The field's editable and usable attributes must be set to TRUE.

```
Boolean FldHandleEvent (
    FieldPtr fld,
    EventPtr EventP
);
```

Parameters

fld
Pointer to a field object (FieldType data structure).

EventP
Pointer to an event (EventType data structure).

Return Value

Returns TRUE if the event was handled.

Remarks

When a keyDownEvent occurs, the keystroke appears in the field if it is a printable character or manipulates the insertion point if it is a "movement" character. The field is automatically updated.
When a penDownEvent occurs, an "editable" field sends a fldEnterEvent to the event queue.
When a fldEnterEvent occurs, the field grabs the focus and the insertion point is placed in the specified position.
If the event alters the contents of the field, this function visually updates the field.
This function does not handle any events if the field is not editable.

FldInsert

Replace the current selection with the string passed.

```
Boolean FldInsert (
    FieldPtr fld,
    CharPtr insertChars,
    Word insertLen
);
```

Parameters

fld
Pointer to the field object to insert to.

insertChars
Text string to be inserted.

insertLen
Length of the text string to be inserted.

Return Value

Returns TRUE if the string was successfully inserted, otherwise FALSE.

Remarks

If there is no current selection, the string passed is inserted at the position of the insertion point.

See Also

FldPaste, FldDelete, FldCut, FldCopy

FldMakeFullyVisible

Cause a dynamically resizable field to expand its height to make its text fully visible.

```
Boolean FldMakeFullyVisible (
    FieldPtr fld
);
```

Parameters

fld
Pointer to a field object.

Return Value

Returns TRUE if the field was not fully visible, otherwise FALSE.

Remarks

If the field's height changes, this function sends a fldHeightChangedEvent via the event queue.

Caveats

If the field is in a table, the table resizes it; otherwise, it is not re-sized.

FldPaste

Replace the current selection in the field with the contents of the text clipboard.

```
void FldPaste (
    FieldPtr fld
);
```

Parameters

fld
Pointer to a field object (FieldType data structure).

Return Value

Returns nothing

Remarks

This insertion point is positioned after the last character inserted and the field is scrolled, if necessary, so the insertion point is visible.
If there is no current selection, the clipboard text is inserted at the position of the insertion point. If there is no text in the clipboard, do not delete the current selection.

See Also

FldInsert, **FldDelete**, **FldCut**, **FldCopy**

FldRecalculateField

Update the structure that contains the word-wrapping information for each visible line.

void **FldRecalculateField (**
    FieldPtr fld,
    Boolean redraw
);

Parameters

fld
    Pointer to a field object (FieldType data structure).

redraw
    If TRUE, redraws the field.

Return Value

Returns nothing.

Remarks

If necessary this function reallocates the memory block that contains the displayed lines information, the block pointed to by the lines member in the FieldType data structure.
Call this function if the field data structure is modified in a way that invalidates the visual appearance of the field.

FldReleaseFocus

Turn the blinking insertion point off if the field is visible and has the current focus; reset the Graffiti state; and reset the undo state.

```
void FldReleaseFocus (
    FieldPtr fld
);
```

Parameters

fld
Pointer to a field object (FieldType data structure).

Return Value

Returns nothing.

Remarks

This function sets the field attribute hasFocus to FALSE.

See Also

FldGrabFocus

FldScrollable

Return TRUE if the field is scrollable in the direction specified.

```
Boolean FldScrollable (
    FieldPtr fld,
    DirectionType direction
);
```

Parameters

fld
Pointer to a field object (FieldType data structure).

direction
“up” or “down.”

Return Value

Returns TRUE if the field is scrollable, FALSE otherwise.

See Also

FldScrollField

FldScrollField

Scroll a field up or down by the number of lines specified.

void **FldScrollField (**
    FieldPtr fld,
    Word linesToScroll,
    DirectionType direction
);

Parameters

fld
    Pointer to a field object (FieldType data structure).

linesToScroll
    Number of lines to scroll.

direction
    “up” or “down.”

Return Value

Returns nothing.

Remarks

Can’t scroll right or left.
The field object is redrawn if it’s scrolled.

See Also

FldScrollable

FldSendChangeNotification

Send a fldChangedEvent via the event queue.

void **FldSendChangeNotification (**
    FieldPtr fld
);

Parameters

fld
    Pointer to a field object.

Return Value

Returns nothing.

FldSendHeightChangeNotification

Send a fldHeightChangedEvent via the event queue.

```
void FldSendHeightChangeNotification (
    FieldPtr fld,
    Word pos,
    Short numLines
);
```

Parameters

fld
Pointer to a field object.

pos
Character position of the insertion point.

numLines
New number of lines in the field.

Return Value

Returns nothing.

FldSetAttributes

Set the attributes of a field.

void **FldSetAttributes (**
    FieldPtr fld,
    FieldAttrPtr attrP
);

Parameters

fld
    Pointer to a FieldType structure.

attrP
    Pointer to the attributes.

Return Value

Returns nothing.

See Also

FldGetAttributes

FldSetBounds

Change the position and or size of a field.

```
void FldSetBounds (
    FieldPtr fld,
    RectanglePtr rect
);
```

Parameters

fld
Pointer to a field object (FieldType data structure).

rect
Pointer to a RectangleType structure that contains the new bounds of the display.

Return Value

Returns nothing.

Remarks

If the field is visible, the field is redrawn within its new bounds.
The memory block that contains the word-wrapping information will be resized if the number of visible lines is changed.
The insertion point is assumed to be off when this routine is called.

Caveats

Don't change the width of the object while it is visible.

See Also

FldGetBounds

FldSetDirty

Set whether the field has been modified.

void **FldSetDirty (**
    FieldPtr fld,
    Boolean dirty
);

Parameters

fld
    Pointer to a field object.

dirty
    TRUE if the text is modified.

Return Value

Returns nothing.

See Also

FldDirty

FldSetFont

Set the font of the field, update the word-wrapping information and draw the field if the field is visible.

void **FldSetFont (**
    FieldPtr fld,
    FontID fontID
);

Parameters

fld
    Pointer to a field object (FieldType data structure).

fontID
    ID of new font.

Return Value

Returns nothing.

See Also

FldGetFont

FldSetInsPtPosition

Set the location of the insertion point for a given string position.

void **FldSetInsPtPosition (**
    FieldPtr fld,
    Word pos
);

Parameters

fld
    Pointer to a field object (FieldType data structure).

pos
    Character position of insertion point.

Return Value

Returns nothing.

Remarks

If the position is beyond the visible text, the insertion point is disabled.

See Also

FldGetInsPtPosition

FldSetMaxChars

Set the maximum number of characters the field accepts.

void **FldSetMaxChars (**
    FieldPtr fld,
    Word maxChars
);

Parameters

fld
    Pointer to a field object (FieldType data structure).

maxChars
    Maximum number of characters the user may enter.

Return Value

Returns nothing.

Remarks

Line feed characters are included when the number of characters is determined.

See Also

FldGetMaxChars

FldSetScrollPosition

Set the string position of the first character in the first line of a field. Redraw the field if necessary.

void **FldSetScrollPosition (**
    FieldPtr fld,
    Word pos
);

Parameters

fld
    Pointer to a field object (FieldType data structure).

pos
    Character position of first visible character.

Return Value

Returns nothing.

See Also

FldGetScrollPosition

FldSetSelection

Set the current selection in a field and highlight the selection if the field is visible.

```
void FldSetSelection (
    FieldPtr fld,
    Word startPosition,
    Word endPosition
);
```

Parameters

fld
Pointer to a field object (FieldType data structure)

startPosition
Starting character position of the character range to highlight.

endPosition
End character position of the character range to highlight.

Return Value

Returns nothing.

Remarks

This function does not affect the display; the highlight is not redrawn until the field is redrawn.
To cancel a selection, set both startPosition and endPosition to the same value.
If startPosition equals endPosition, the current selection is unhighlighted.

FldSetText

Set the text value of the field, update the word-wrapping information, and place the insertion point after the last visible character.

```
void FldSetText (
    FieldPtr fld,
    VoidHand textHandle,
    Word offset,
    Word size
);
```

Parameters

fld
Pointer to a field object (FieldType data structure).

textHandle
Handle of a block containing a null-terminated text string.

offset
Offset from start of block to start of the text string.

size
Allocated size of text string, not the string length.

Return Value

Returns nothing.

Remarks

The pointer passed is stored in the field's structure; in other words this function does not make a copy of the string passed.
If a size of zero is passed, the size is computed as the block size, less the offset passed. If more text is set than there is room for in memory, an error occurs.

WARNING: This routine does not free the memory block that holds the current text value.

See Also

FldSetTextPtr, FldSetTextHandle

FldSetTextAllocatedSize

Set the number of characters allocated to hold the field's text string. Don't confuse this with the length of the text string.

void **FldSetTextAllocatedSize (**
    FieldPtr fld,
    Word allocatedSize
);

Parameters

fld
    Pointer to a field object.

allocatedSize
    Number of characters to allocate for the text.

Return Value

Returns nothing.

See Also

FldGetTextAllocatedSize

FldSetTextHandle

Set the handle of the block that contains the text string of a field.

```
void FldSetTextHandle (
    FieldPtr fld,
    Handle textHandle
);
```

Parameters

fld
Pointer to a field object (FieldType data structure).

textHandle
Handle of a field's text string; 0 is a possible value.

Return Value

Returns nothing.

See Also

FldSetTextPtr, FldSetText

FldSetTextPtr

Set the field's text to point to a text string.

```
void FldSetTextPtr (
    FieldPtr fld,
    CharPtr textP
);
```

Parameters

fld
Pointer to a field object (FieldType data structure).

textP
Pointer to a null-terminated string.

Return Value

Returns nothing.

Remarks

Since the field cannot resize a pointer (only handles can be re-sized), the field must be not editable; if the field is editable, an error occurs.
This function does not visually update the field.

See Also

FldSetTextPtr, FldSetTextHandle

FldSetUsable

Set a field usable or nonusable.

```
void FldSetUsable (
    FieldPtr fld,
    Boolean usable
);
```

Parameters

fld
Pointer to a FieldType structure.

usable
TRUE to set usable, FALSE to set nonusable.

Return Value

Returns nothing.

Remarks

A nonusable field does not display or accept input.

See Also

FldEraseField, FldDrawField

FldUndo

Undo the last change made to the field object. Changes include typing, backspaces, delete, paste, and cut.

void **FldUndo (**
    FieldPtr fld
);

Parameters

fld
    Pointer to the field that has the focus.

Return Value

Returns nothing.

See Also

FldPaste, **FldCut**, **FldCopy**

FldWordWrap

Given a string and a width, return the number of characters that can be displayed using the current font.

```
Word FldWordWrap (
    CharPtr chars,
    Word maxWidth
);
```

Parameters

chars
Pointer to a null-terminated string.

maxWidth
Maximum line width in pixels.

Return Value

Returns the number of characters.

InsPtEnable

Enable or disable the insertion point. When the insertion point is disabled it is invisible, when it is enabled it blinks.

void **InsPtEnable (**
    Boolean enableIt
);

Parameters

enable
    TRUE = enable, FALSE = disable

Return Value

Returns nothing.

Remarks

This function is called by the Form functions when a text field loses or gains the focus, and by the Windows function when a region of the display is copied (WinCopyRectangle).

See Also

InsPtEnabled

InsPtEnabled

Return TRUE if the insertion point is enabled or FALSE if it is disabled.

Boolean **InsPtEnabled (**
    void
);

Parameters

None.

Return Value

Returns TRUE if the insertion point is enabled (blinking), returns FALSE if the insertion point is disabled (invisible).

See Also

InsPtEnable

InsPtGetHeight

Return the height of the insertion point.

short **InsPtGetHeight (**
    void
);

Parameters

None.

Return Value

Returns the height of the insertion point, in pixels.

InsPtGetLocation

Return the screen-relative position of the insertion point.

```
void InsPtGetLocation (
    short *x,
    short *y
);
```

Parameters

x
Pointer to top-left position of insertion point's x coordinate.

y
Pointer to top-left position of insertion point's y coordinate.

Return Value

Returns nothing. Stores the location in x and y.

Remarks

This function is called by the Field functions. An application would not normally call this function.

InsPtSetHeight

Set the height of the insertion point.

void **InsPtSetHeight (**
    short height
);

Parameters

height
    Height of the insertion point in pixels.

Return Value

Returns nothing.

Remarks

Set the height of the insertion point to match the character height of the font used in the field that the insertion point is in. When the current font is changed, the insertion point height should be set to the line height of the new font.
If the insertion point is visible when its height is changed, it is erased and redrawn with its new height.

See Also

InsPtGetHeight

InsPtSetLocation

Set the screen-relative position of the insertion point.

```
void InsPtSetLocation (
    short x,
    short y
);
```

Parameters

x
Number of pixels from the left side of the display.

y
Number of pixels from the top of the display.

Return Value

Returns nothing.

Remarks

The position passed to this function is the location of the top-left corner of the insertion point. This function should be called only by the Field functions.

See Also

InsPtGetLocation

InsPtCheckBlink

void **InsPtCheckBlink (**
    void
);

WARNING: For System Use Only.

InsPtInitialize

void **InsPtInitialize (**
    void
);

WARNING: For System Use Only.

FrmAlert

Create a modal dialog from an alert resource and display it until the user selects a button in the dialog.

```
Word FrmAlert (
    Word alertId
);
```

Parameters

alertId
    ID of the alert resource.

Return Value

Returns the item number of the button the user selected. A button's item number is determined by its order in the alert dialog; the first button has the item number 0 (zero).

See Also

FrmDoDialog, FrmCustomAlert

FrmCloseAllForms

Sends a frmCloseEvent to all open forms.

void **FrmCloseAllForms (**
    void
);

Parameters

None.

Remarks

Can be called by applications to ensure that all forms are closed cleanly before exiting PilotMain(); that is, before termination.

See Also

FrmSaveAllForms

FrmCopyLabel

Copy the passed string into the data structure of the specified label object in the active form.

```
void FrmCopyLabel (
    FormPtr frm,
    Word labelID,
    CharPtr newLabel
);
```

Parameters

frm
Pointer to memory block that contains the form.

labelID
ID of form label object.

newLabel
Pointer to a null-terminated string.

Return Value

Returns nothing.

Remarks

The size of the new label must not exceed the size of the label defined in the resource. When defining the label in the resource, specify an initial size at least as big as any of the strings that will be assigned dynamically. Redraw the label if the form's usable attribute and the label's visible attribute are set.

See Also

FrmGetLabel

FrmCopyTitle

Copy the title passed over the form's current title. If the form is visible, the new title is drawn.

```
void FrmCopyTitle (
    FormPtr frm,
    CharPtr newTitle
);
```

Parameters

frm
Memory block that contains the form.

newTitle
Pointer to the new title string.

Return Value

Returns nothing.

Remarks

The size of the new title must not exceed the title size defined in the resource. When defining the title in the resource, specify an initial size at least as big as any of string to be assigned dynamically.

See Also

FrmGetTitle

FrmCustomAlert

Create a modal dialog from an alert resource and display the dialog until the user taps a button in the alert dialog.

```
Word FrmCustomAlert (
    Word alertId,
    CharPtr s1,
    CharPtr s2,
    CharPtr s3
);
```

Parameters

alertId
Resource ID of the alert.

s1
String to replace '^1'

s2
String to replace '^2'

s3
String to replace '^3'

Return Value

Returns the button number the user tapped (first button is zero).

Remarks

A button's item number is determined by its order in the alert template; the first button has the item number zero.
Up to three strings can be passed to this routine. They are used to replace the "text replacement variables" ^1, ^2 and ^3 that are contained in the message string of the alert resource.

See Also

FrmAlert, FrmDoDialog

FrmDeleteForm

Release the memory occupied by a form.
Any memory allocated to objects in the form is also released.

```
void FrmDeleteForm (
    FormPtr frm
);
```

Parameters

frm
    Pointer to memory block that contains the form.

Return Value

Returns nothing.

Remarks

This function does not modify the display.

See Also

FrmInitForm, FrmReturnToForm

FrmDispatchEvent

Dispatch an event to the application's handler for the form.

```
Boolean FrmDispatchEvent (
    EventPtr eventP
);
```

Parameters

eventP
Pointer to an event.

Return Value

Returns nothing.

Remarks

The event is dispatched to the current form unless the form ID is specified in the event data, as, for example, with frmOpenEvent.

See Also

FrmSetEventHandler, FrmHandleEvent

FrmDoDialog

Display a modal dialog until the user taps a button in the dialog.

```
Word FrmDoDialog (
    FormPtr frm
);
```

Parameters

frm
Pointer to memory block that contains the form.

Return Value

Returns the number of the button the user tapped (first button is zero).

Remarks

A button's item number is determined by its order in the alert template; the first button has an item number of 0 (zero).

See Also

FrmInitForm, FrmCustomAlert

FrmDrawForm

Draw all objects in a form and the frame around the form.

```
void FrmDrawForm (
    FormPtr frm
);
```

Parameters

frm
Pointer to the memory block that contains the form.

Return Value

Returns nothing.

Remarks

Saves the bits behind the form using the bitsBehindForm field.

See Also

FrmEraseForm, FrmInitForm

FrmEraseForm

Erase a form from the display.

```
void FrmEraseForm (
    FormPtr frm
);
```

Parameters

frm
Pointer to the memory block that contains the form.

Return Value

Returns nothing.

Remarks

If the region obscured by the form was saved by FrmDrawForm, this function restores that region.

See Also

FrmDrawForm

FrmGetActiveForm

Return the currently active form.

```
FormPtr FrmGetActiveForm (
    void
);
```

Parameters

None.

Return Value

Returns the pointer to the memory block that contains the form.

See Also

FrmGetActiveFormID, FrmSetActiveForm

FrmGetActiveFormID

Return the ID of the currently active form.

```
Word FrmGetActiveFormID (
    void
);
```

Parameters

None.

Return Value

Returns the currently active form's ID number.

See Also

FrmGetActiveForm

FrmGetControlGroupSelection

Return the item number of the control selected in a group of controls.

```
Byte FrmGetControlGroupSelection (
    FormPtr frm,
    Byte groupNum
);
```

Parameters

frm
Pointer to memory block that contains the form.

groupNum
Control group number.

Return Value

Returns the item number of the selected control, -1 if none is selected.

Remarks

The item number is the index into the form's objects data structure.

See Also

FrmGetObjectId, FrmGetObjectPtr, FrmSetControlGroupSelection

FrmGetControlValue

Return the on/off state of a control.

```
short FrmGetControlValue (
    FormPtr frm,
    Word objIndex
);
```

Parameters

frm
Pointer to memory block that contains the form.

objIndex
Item number of the object.

Return Value

Returns the state of the control: 1 = on; 0 = off.

Remarks

The caller must specify a valid index. This function is used only for push button and check box control objects.

See Also

FrmGetObjectIndex, FrmSetControlValue

FrmGetFirstForm

Return the first form in the window list.

```
FormPtr FrmGetFirstForm (
    void
);
```

Parameters

None.

Return Value

Returns a pointer to a form, or NULL if there are no forms.

Remarks

The window list is a LIFO stack. The last window created is the first window in the window list.

FrmGetFocus

Return the item (index) number of the object (UI element) that has the focus.

```
Word FrmGetFocus (
    FormPtr frm
);
```

Parameters

frm
Pointer to memory block that contains the form.

Return Value

Returns the index of the object (UI element) that has the focus, or -1 if none does.

See Also

FrmGetObjectId, FrmGetObjectPtr, FrmSetFocus

FrmGetFormBounds

Return the visual bounds of the form; the region returned includes the form's frame.

```
void FrmGetFormBounds (
    FormPtr frm,
    RectanglePtr r
);
```

Parameters

frm
Pointer to memory block that contains the form.

r
Pointer to a RectangleType structure that will contain the bounds.

Return Value

Returns bounds of the form in r.

FrmGetFormId

Return the resource ID of a form.

```
Word FrmGetFormId (
    FormPtr frm
);
```

Parameters

frm
Pointer to memory block that contains the form.

Return Value

Returns form resource ID.

See Also

FrmGetFormPtr

FrmGetFormPtr

Return a pointer to the form that has the specified ID.

```
FormPtr FrmGetFormPtr (
    Word formId
);
```

Parameters

formId
Form ID number.

Return Value

Returns a pointer to the memory block that contains the form, or NULL if the form is not in memory.

See Also

FrmGetFormId

FrmGetGadgetData

Return the value stored in the data field of the gadget object.

```
VoidPtr FrmGetGadgetData (
    FormPtr frm,
    Word objIndex
);
```

Parameters

frm
Pointer to memory block that contains the form.

objIndex
Item number of the gadget object.

Return Value

Returns a pointer to the custom gadget's data.

Remarks

Gadget objects provide a way for an application to attach custom gadgetry to a form. In general, the data field of a gadget object contains a pointer to the custom object's data structure.

See Also

FrmSetGadgetData

FrmGetLabel

Return pointer to the text of the specified label object in the specified form.

```
CharPtr FrmGetLabel (
    FormPtr frm,
    Word labelID
);
```

Parameters

frm
Pointer to memory block that contains the form.

labelID
ID of the label object.

Return Value

Returns pointer to the label string.

Remarks

Does not make a copy of the string; returns a pointer to the string.
The object must be a label.

See Also

FrmCopyLabel

FrmGetNumberOfObjects

Return the number of objects in a form.

```
Word FrmGetNumberOfObjects (
    FormPtr frm
);
```

Parameters

frmPtr
Pointer to memory block that contains the form.

Return Value

Returns the number of objects in the specified form.

See Also

FrmGetObjectPtr, FrmGetObjectId

FrmGetObjectBounds

Retrieve the bounds of an object given its form and index.

```
void FrmGetObjectBounds (
    FormPtr frm,
    Word ObjIndex,
    RectanglePtr r
);
```

Parameters

frm
Pointer to memory block that contains the form.

ObjIndex
Index of an object in the form.

r
Pointer to the rectangle containing the object bounds.

Return Value

Returns nothing. The object's bounds are returned in r.

See Also

FrmGetObjectPositon, FrmGetObjectIndex, FrmSetObjectPositon

FrmGetObjectId

Return the ID of the specified object.

```
Word FrmGetObjectId (
    FormPtr frm,
    Word objIndex
);
```

Parameters

frm
Pointer to memory block that contains the form.

objIndex
Index of an object in the form.

Return Value

Returns the ID number of a object.

Remarks

The application developer specifies a unique object ID.

See Also

FrmGetObjectPtr, FrmGetObjectIndex

FrmGetObjectIndex

Return the item number of an object. The item number is the position of the object in the form's objects list.

```
Word FrmGetObjectIndex (
    FormPtr frm,
    Word objID
);
```

Parameters

frmPtr
Pointer to memory block that contains the form.

objID
ID of an object in the form.

Return Value

Returns the item number of an object (the first item number is 0).

See Also

FrmGetObjectPtr, FrmGetObjectId

FrmGetObjectPositon

Return the coordinate of the specified object relative to the form.

void **FrmGetObjectPositon (**
    FormPtr frm,
    Word objIndex,
    SWordPtr x,
    SWordPtr y
);

Parameters

frm
    Pointer to memory block that contains the form.

objIndex
    Item number of the object.

x
    Pointer to window-relative x position.

y
    Pointer to window-relative y position.

Return Value

Returns nothing.

Remarks

The function name is misspelled (the second “i” is missing).

See Also

FrmGetObjectBounds, **FrmSetObjectPositon**

FrmGetObjectPtr

Return a pointer to the data structure of an object in a form.

```
void * FrmGetObjectPtr (
    FormPtr frm,
    Word objIndex
);
```

Parameters

frm
Pointer to memory block that contains the form.

objIndex
Item number of the object.

Return Value

Returns pointer to an object in the form.

See Also

FrmGetObjectIndex, FrmGetObjectId

FrmGetObjectType

Return the type of an object.

```
FormObjectKind FrmGetObjectType (
    FormPtr frm,
    Word objIndex
);
```

Parameters

frm
Pointer to memory block that contains the form.

objIndex
Item number of the object.

Return Value

Returns FormObjectKind of the item specified.

FrmGetTitle

Return a pointer to the title string of a form.

```
CharPtr FrmGetTitle (
    FormPtr frm
);
```

Parameters

frm
    Pointer to memory block that contains the form.

Return Value

Returns a pointer to title string.

Remarks

This is a pointer to the internal structure, not a copy.

See Also

FrmCopyTitle, FrmSetTitle

FrmGetUserModifiedState

Return TRUE if an object in the form has been modified by the user since it was initialized or since last call to FrmSetNotUserModified.

```
Boolean FrmGetUserModifiedState (
    FormPtr frm
);
```

Parameters

frm
Pointer to the memory block that contains the form.

Return Value

Returns TRUE if an object was modified, FALSE otherwise.

Remarks

Returns TRUE if the dirty attribute of the form has been set.

See Also

FrmSetNotUserModified

FrmGetWindowHandle

Return the window handle of a form.

```
WinHandle FrmGetWindowHandle (
    FormPtr frm
);
```

Parameters

frm
Pointer to memory block that contains the form.

Return Value

Returns the handle of the memory block that the form is in. Since the form structure begins with the WindowType structure, this is also a WinHandle.

FrmGotoForm

Send a frmCloseEvent to the current form; send a frmLoadEvent and a frmOpenEvent to the specified form.

```
void FrmGotoForm (
    Word formId
);
```

Parameters

formId
ID of the form to display.

Return Value

Returns nothing.

Remarks

The form event handler (FrmHandleEvent) erases and disposes of a form when it receives a frmCloseEvent.

See Also

FrmPopupForm

FrmHandleEvent

Handle the event that has occurred in the form.

```
Boolean FrmHandleEvent (
    FormPtr frm,
    EventPtr event
);
```

Parameters

frm
Pointer to the memory block that contains the form.

event
Pointer to the event data structure.

Return Value

Returns TRUE if the event was handled.

See Also

FrmDispatchEvent

FrmHelp

Display the specified help message until the user taps the done button in the help dialog.

```
void FrmHelp (
    Word helpMsgId
);
```

Parameters

helpMsgId
    Resource ID of help message string.

Return Value

Returns nothing.

Remarks

The ID passed is the resource ID of a string resource that contains the help message. The help message is displayed in a modal dialog with vertical scrolls if necessary.

FrmHideObject

Erase the specified object and set its attribute data so that it does not redraw or respond to the pen.

```
void FrmHideObject (
    FormPtr frm,
    Word objIndex
);
```

Parameters

frm
Pointer to memory block that contains the form.

objIndex
Item number of the object.

Return Value

Returns nothing.

See Also

FrmGetObjectIndex, FrmShowObject

FrmInitForm

Load and initialize a form resource.

```
FormPtr FrmInitForm (
    Word rscID
);
```

Parameters

rscID
Resource ID of the form.

Return Value

Returns a pointer to the form memory block.
Displays an error message if the form has already been initialized.

Remarks

This function does not affect the display nor make the form active.

See Also

FrmDoDialog, FrmDeleteForm

FrmPopupForm

Send a frmOpenEvent to the specified form. This routine differs from FrmGotoForm in that the current form is not closed.

```
void FrmPopupForm (
    Word formId
);
```

Parameters

formID
Resource ID of form to open.

Return Value

Returns nothing.

See Also

FrmGotoForm

FrmReturnToForm

Erase and delete the currently active form and make the specified form the active form.

```
void FrmReturnToForm (
    Word formId
);
```

Parameters

formID
Resource ID of the form to return to.

Return Value

Returns nothing.

Remarks

It is assumed that the form being returned to is already loaded into memory and initialized. Passing a form ID of 0 returns to the first form in the window list, which is the last form to be loaded.

See Also

FrmGotoForm, FrmPopupForm

FrmSaveAllForms

Send a frmSaveEvent to all open forms.

void **FrmSaveAllForms (**
    void
);

Parameters

None.

Return Value

Returns nothing.

See Also

FrmCloseAllForms

FrmSetActiveForm

Set the active form. All input (key and pen) is directed to the active form.

```
void FrmSetActiveForm (
    FormPtr frm
);
```

Parameters

frm
Pointer to memory block that contains the form.

Return Value

Returns nothing.

Remarks

A penDownEvent outside the form but within the display area is ignored.

See Also

FrmGetActiveForm

FrmSetCategoryLabel

Set the category label displayed on the title line of a form. If the form's visible attribute is set, redraw the label.

```
void FrmSetCategoryLabel (
    FormPtr frm,
    Word objIndex,
    CharPtr newLabel
);
```

Parameters

frm
Pointer to memory block that contains the form.

objIndex
Item number of the object.

newLabel
Pointer to the name of the new category.

Return Value

Returns nothing.

Remarks

The pointer to the new label is saved in the object.

FrmSetControlGroupSelection

Set the selected control in a group of controls.

```
void FrmSetControlGroupSelection (
    FormPtr frm,
    Byte groupNum,
    Word controlID
);
```

Parameters

frm
Pointer to memory block that contains the form.

groupNum
Control group number.

controlID
ID of control to set.

Return Value

Returns nothing.

Remarks

This function unsets all the other controls in the group. The display is updated.

See Also

FrmGetControlGroupSelection

FrmSetControlValue

Turn a control on or off.

```
void FrmSetControlValue (
    FormPtr frm,
    Word objIndex,
    short newValue
);
```

Parameters

frm
Pointer to memory block that contains the form.

objIndex
Item number of the object.

newValue
New control value (non-zero equals on).

Return Value

Returns nothing.

Remarks

The display is not changed.

See Also

FrmGetControlValue

FrmSetEventHandler

Set the event handler callback routine for the specified form.

void **FrmSetEventHandler (**
    FormPtr frm,
    FormEventHandlerPtr handler
);

Parameters

frm
    Pointer to memory block that contains the form.

handler
    Address of a function.

Return Value

Returns nothing.

Remarks

FrmHandleEvent calls this handler whenever it receives an event.
This routine should be called right after a form resource is loaded.
The callback routine is the mechanism for dispatching events to an application. The tutorial explains how to use callback routines.

See Also

FrmDispatchEvent

FrmSetFocus

Set the focus of a form to the specified object.

```
void FrmSetFocus (
    FormPtr frm,
    Word objIndex
);
```

Parameters

frm
Pointer to memory block that contains the form.

objIndex
Item number of the object (UI element) that gets the focus.

Return Value

Returns nothing.

See Also

FrmGetFocus, FrmGetObjectIndex

FrmSetGadgetData

Store the value passed in the data field of the gadget object.

```
void FrmSetGadgetData (
    FormPtr frm,
    Word objIndex,
    VoidPtr data
);
```

Parameters

frmPtr
Pointer to memory block that contains the form.

objIndex
Item number of the object.

data
Application-defined value.

Return Value

Returns nothing.

Remarks

Gadget objects provide a way for an application to attach custom gadgetry to a form. In general, the data field of a gadget object contains a pointer to the custom object's data structure.

See Also

FrmGetGadgetData, FrmGetObjectIndex

FrmSetNotUserModified

Clear the flag that keeps track of whether or not the form has been modified by the user.

void **FrmSetNotUserModified (**
    FormPtr frm
);

Parameters

frm
    Pointer to memory block that contains the form.

Return Value

Returns nothing.

See Also

FrmGetUserModifiedState

FrmSetObjectPositon

Set the window-relative coordinate of the specified object.

```
void FrmSetObjectPositon (
    FormPtr frm,
    Word objIndex,
    SWord x,
    SWord y
);
```

Parameters

frm
Pointer to memory block that contains the form.

objIndex
Item number of the object.

x
Window-relative coordinate.

y
Window-relative coordinate.

Return Value

Returns nothing.

Remarks

Does not update the display. Presently only label objects are affected.

See Also

FrmGetObjectPositon, FrmGetObjectIndex, FrmGetObjectBounds

FrmSetTitle

Set the title of a form. If the form is visible, draw the new title.

```
void FrmSetTitle (
    FormPtr frm,
    CharPtr newTitle
);
```

Parameters

frm
Pointer to memory block that contains the form.

newTitle
Pointer to the new title string.

Return Value

Returns nothing.

Remarks

Draws the title if the form is visible.
Saves the pointer to the passed title string. Does not make a copy.

See Also

FrmGetTitle, FrmCopyTitle, FrmCopyLabel

FrmShowObject

Set an object (UI element) as usable. If the form is visible, draw the object.

void **FrmShowObject (**
    FormPtr frm,
    Word objIndex
);

Parameters

frm
    Pointer to memory block that contains the form.

objIndex
    Item number of the object.

Return Value

Returns nothing.

See Also

FrmHideObject, **FrmGetObjectIndex**

FrmUpdateScrollers

Visually update the field scroll arrow buttons.

```
void FrmUpdateScrollers (
    FormPtr frm,
    Word upIndex,
    Word downIndex,
    Boolean scrollableUp,
    Boolean scrollableDown
);
```

Parameters

frm
Pointer to a form.

upIndex
Index of the up-scroller button.

downIndex
Index of the down-scroller button.

scrollableUp
TRUE if the up-scroll should be active.

scrollableDown
TRUE if the down-scroll should be active.

Return Value

Returns nothing.

See Also

FrmGetObjectIndex

FrmUpdateForm

Send a frmUpdateEvent to the specified form.

```
void FrmUpdateForm (
    Word formId,
    Word updateCode
);
```

Parameters

formID
    Resource ID of form to open.

updateCode
    If the update code is frmRedrawUpdateCode, the form reinitializes its global variables and redraws itself. Otherwise, the form reinitializes its global variables but does not redraw itself.

Return Value

Returns nothing.

FrmVisible

Return TRUE if the form is visible (is drawn).

```
Boolean FrmVisible (
    FormPtr frm
);
```

Parameters

frm
Pointer to memory block that contains the form.

Return Value

Returns TRUE if visible, FALSE if not visible.

See Also

FrmDrawForm, FrmEraseForm

LstDrawList

Draw the list object if it is usable.

```
void LstDrawList (
    ListPtr list
);
```

Parameters

list
Pointer to list object (ListType data structure).

Return Value

Returns nothing.

Remarks

If there are more choices than can be displayed, this function ensures that the current selection is visible. If possible, the current selection is displayed at the top. The current selection is highlighted. If the list is disabled, it's drawn grayed-out (strongly discouraged).
If it's empty, nothing is drawn. If it's not usable, nothing is drawn.
This function sets the visible attribute to TRUE.

See Also

FrmGetObjectPtr, **LstPopupList**, **LstEraseList**

LstEraseList

Erase a list object.

```
void LstEraseList (
    ListPtr ListP
);
```

Parameters

ListP
Pointer to a list object (ListType data structure).

Return Value

Returns nothing.

Remarks

The visible attribute is set to FALSE by this function.

See Also

FrmGetObjectPtr, LstDrawList

LstGetNumberOfItems

Return the number of items in a list.

```
Word LstGetNumberOfItems (
    ListPtr ListP
);
```

Parameters

ListP
Pointer to a list object (ListType data structure).

Return Value

Returns the number of items in a list.

See Also

FrmGetObjectPtr, LstSetListChoices

LstGetSelection

Return the currently selected choice in the list. If there is no selection, return NoListSelection (-1).

```
Word LstGetSelection (
    ListPtr ListP
);
```

Parameters

ListP
Pointer to list object.

Return Value

Returns the item number of the current list choice. The list choices are numbered sequentially, starting with 0; -1 = none.

See Also

FrmGetObjectPtr, LstSetListChoices, LstSetSelection, LstGetSelectionText

LstGetSelectionText

Return a pointer to the text of the specified item in the list or NULL if no such item exists.

```
CharPtr LstGetSelectionText (
    ListPtr ListP,
    Word itemNum
);
```

Parameters

ListP
Pointer to list object.

itemNum
Item to select (0 = first item in list).

Return Value

Returns pointer to the text of the current selection, or NULL if out of bounds.

Remarks

This is a pointer within ListType structure, not a copy.

See Also

FrmGetObjectPtr, LstSetListChoices

LstHandleEvent

Handle event in the specified list; the list object must have its usable and visible attribute set to TRUE. (This routine handles two types of events, penDownEvent and lstEnterEvent; see Remarks).

```
Boolean pascal LstHandleEvent (
    ListPtr listP,
    EventPtr pEvent
);
```

Parameters

listP
Pointer to a list object (ListType data structure).

pEvent
Pointer to an EventType structure.

Return Value

Return TRUE if the event was handled. The following cases will result in a return value of TRUE:
A penDownEvent within the bounds of the list.
A lstEnterEvent with a list ID value that matches the list ID in the list data structure.

Remarks

When this routine receives a penDownEvent, it checks if the pen position is within the bounds of the list object. If it is, this routine tracks the pen until the pen comes up. If the pen comes up within the bounds of the list, a lstEnterEvent is added to the event queue, and the routine is exited.
When this routine receives a lstEnterEvent, it checks that the list ID in the event record matches the ID of the specified list. If
there is a match, this routine creates and displays a popup window containing the list's choices, and the routine is exited.
If a penDownEvent is received while the list's popup window is displayed, and the pen position is outside the bounds of the popup window, the window is dismissed. If the pen position is within the bounds of the window, this routine tracks the pen until it comes up. If the pen comes up outside the list object, a lstEnterEvent is added to the event queue.

LstMakeItemVisible

Make an item visible, preferably at the top. If the item is already visible, no changes are made.

```
LstMakeItemVisible (
    ListPtr ListP,
    Word itemNum
);
```

Parameters

ListP
Pointer to a list object (ListType data structure).

itemNum
Item to select (0 = first item in list).

Return Value

Returns nothing.

Remarks

Does not visually update the list. You must call LstDrawList to update it.

See Also

FrmGetObjectPtr, LstSetSelection, LstSetTopItem, LstDrawList

LstPopupList

Display a modal window that contains the items in the list.

```
short LstPopupList (
    ListPtr ListP
);
```

Parameters

ListP
    Pointer to list object.

Return Value

Returns the list item selected, or -1 if no item was selected.

Remarks

Saves the previously active window. Creates and deletes the new popup window.

See Also

FrmGetObjectPtr

LstSetDrawFunction

Set a callback function to draw each item instead of drawing the item's text string.

```
void LstSetDrawFunction (
    ListPtr list,
    ListDrawDataFuncPtr func
);
```

Parameters

list
Pointer to list object.

func
Pointer to function which draws items.

Return Value

Returns nothing.

Remarks

This function also adjusts topItem to prevent a shrunken list from being scrolled down too far. Use this function for custom draw functionality.

See Also

FrmGetObjectPtr, LstSetListChoices

LstSetHeight

Set the number of items visible in a list.

```
void LstSetHeight (
    ListPtr ListP,
    Word visibleItems
);
```

Parameters

ListP
Pointer to list object.

visibleItems
Number of choices visible at once.

Return Value

Returns nothing.

Remarks

This function does not redraw the list if it is already visible.

See Also

FrmGetObjectPtr

LstSetListChoices

Set the items of a list to the array of text strings passed to this function.
This function does not affect the display of the list.

```
void LstSetListChoices (
    ListPtr ListP,
    char ** itemsText,
    UInt numItems
);
```

Parameters

ListP
Pointer to a list object.

itemsText
Pointer to an array of text strings.

numItems
Number of choices in the list.

Return Value

Returns nothing.

Remarks

If the list is visible, erases the old list items.

See Also

FrmGetObjectPtr, LstSetSelection, LstSetTopItem, LstDrawList, LstSetHeight, LstSetDrawFunction

LstSetPosition

Set the position of a list.

```
void LstSetPosition (
    ListPtr ListP,
    short x,
    short y
);
```

Parameters

ListP
    Pointer to a list object

x
    Left bound.

y
    Top bound.

Return Value

Returns nothing.

Remarks

The list is not redrawn. Don't call this function when the list is vis-ible.

See Also

FrmGetObjectPtr

LstSetSelection

Set the selection for a list.

```
void LstSetSelection (
    ListPtr ListP,
    Word itemNum
);
```

Parameters

ListP
    Pointer to a list object.

itemNum
    Item to select (0 = first item in list, -1 = none).

Return Value

Returns nothing.

Remarks

The old selection, if any, is unselected. If the list is visible, the selected item is visually updated. The list is scrolled to the selection, if necessary.

See Also

FrmGetObjectPtr, LstSetSelection

LstSetTopItem

Set the item visible. The item cannot become the top item if it's on the last page.

```
void LstSetTopItem (
    ListPtr ListP,
    UInt itemNum
);
```

Parameters

ListP
Pointer to list object.

itemNum
Item to select (0 = first item in list).

Return Value

Returns nothing.

Remarks

Does not update the display.

See Also

FrmGetObjectPtr, LstSetSelection, LstMakeItemVisible, LstDrawList, LstEraseList

MenuDispose

Release any memory allocated to support the menu management.

```
void MenuDispose (
    MenuBarPtr MenuP
);
```

Parameters

MenuP
Pointer returned by MenuInit; this is a pointer to a MenuBarType data structure.

Return Value

Returns nothing.

Remarks

This function is useful for applications that have multiple menu bars. It frees all memory allocated by a menu, resets the command status, and restores the saved bits to the screen.

See Also

MenuInit, MenuDrawMenu

MenuDrawMenu

Draw the current menu bar and the last pull-down that was visible.

```
void MenuDrawMenu (
    MenuBarPtr MenuP
);
```

Parameters

MenuP
Pointer to a MenuBarType data structure.

Return Value

Returns nothing.

Remarks

If a pull-down menu was visible the last time the menu bar was visible, the pull-down menu is also drawn.
The first time a menu bar is drawn no pull-down menu is displayed.
The menu bar and the pull-down menu are drawn in front of all the applications windows.
Screen regions obscured by the menus are saved by this function and restored by MenuEraseStatus.

See Also

MenuInit, MenuEraseStatus, MenuDispose

MenuEraseStatus

Erase the menu command status.

```
void MenuEraseStatus (
    MenuBarPtr MenuP
);
```

Parameters

MenuP
Pointer to a MenuBarType data structure, or NULL for the current menu.

Return Value

Returns nothing.

See Also

MenuInit

MenuGetActiveMenu

Returns a pointer to the current menu.

```
MenuBarPtr MenuGetActiveMenu (
    void
);
```

Parameters

None.

Return Value

Returns a pointer to the current menu, NULL if there is none.

See Also

MenuSetActiveMenu

MenuHandleEvent

Handle events in the current menu. This routine handles two types of events, penDownEvent and winEnterEvent.

```
Boolean MenuHandleEvent (
    MenuBarPtr MenuP,
    EventPtr event,
    WordPtr error
);
```

Parameters

MenuP
Pointer to a MenuBarType data structure.

event
Pointer to an EventType structure.

error
Error (or 0 if no error).

Return Value

Returns TRUE if the event is handled. (If the event is a penDownEvent within the menu bar or the menu, or the event is a keyDownEvent that the menu supports.)

Remarks

When MenuHandleEvent receives a penDownEvent, it checks if the pen position is within the bounds of the menu object. If it is, MenuHandleEvent tracks the pen until it comes up. If the pen comes up within the bounds of the menu, a winEnterEvent is added to the event queue, and the routine is exited.
When MenuHandleEvent receives a winEnterEvent, it checks that the menu ID in the event record matches the ID of the specified menu. If there is a match, MenuHandleEvent creates and displays a popup window containing the menu's choices, and the routine is exited.
If a penDownEvent is received while the menu's popup window is displayed, and the pen position is outside the bounds of the popup
window, the menu is dismissed. If the pen position is within the bounds of the window MenuHandleEvent tracks the pen until it comes up. If the pen comes up in the menu, a winExitEvent is added to the event queue.

MenuInit

Load a menu resource from a resource file.

```
MenuBarPtr MenuInit (
    Word resourceId
);
```

Parameters

resourceId
ID that identifies the menu resource.

Return Value

Returns the pointer to a memory block allocated to hold the menu resource (a pointer to a MenuBarType data structure).

Remarks

The menu is not usable until MenuSetActiveMenu is called.

See Also

MenuSetActiveMenu, MenuDispose

MenuSetActiveMenu

Set the current menu.

```
MenuBarPtr MenuSetActiveMenu (
    MenuBarPtr MenuP
);
```

Parameters

MenuP
    Pointer to the memory block that contains the new menu, or NULL for none.

Return Value

Returns a pointer to the menu that was active before the new menu was set, or NULL if no menu was active.

See Also

MenuGetActiveMenu

TblDrawTable

Draw a table.

```
void TblDrawTable (
    TablePtr table
);
```

Parameters

table
    Pointer to a table object.

Return Value

Returns nothing.

See Also

TblEraseTable, TblRedrawTable, TblSetCustomDrawProcedure

TblEditing

Check whether a table is in edit mode.

```
Boolean TblEditing (
    TablePtr table
);
```

Parameters

table
Pointer to a table object.

Return Value

Returns TRUE if the table is in edit mode, FALSE otherwise.

Remarks

The table is in edit mode while the user edits a text item.

TblEraseTable

Erase a table object.

```
void TblEraseTable (
    TablePtr table
);
```

Parameters

table
    Pointer to a table object.

Return Value

Returns nothing.

See Also

TblDrawTable, TblSetCustomDrawProcedure, TblRedrawTable

TblFindRowData

Return the row number that contains the specified data value.

```
Boolean TblFindRowData (
    TablePtr table,
    ULong data,
    WordPtr rowP
);
```

Parameters

table
    Pointer to a table object.

data
    Row data to find.

rowP
    Pointer to the row number (return value).

Return Value

Returns TRUE if a match was found, FALSE otherwise.

See Also

TblGetRowData, TblFindRowID

TblFindRowID

Return the number of the row that matches the specified ID.

```
Boolean TblFindRowID (
    TablePtr table,
    Word id,
    WordPtr rowP
);
```

Parameters

table
    Pointer to a table object.

id
    Row ID to find.

rowP
    Pointer to the row number (return value)

Return Value

Returns TRUE if a match was found, FALSE otherwise.

See Also

TblFindRowData

TblGetBounds

Return the bounds of a table.

```
void TblGetBounds (
    TablePtr table,
    RectanglePtr r
);
```

Parameters

table
    Pointer to a table object.

r
    Pointer to a RectangleType structure.

Return Value

Returns nothing. Stores the bounds in r.

See Also

TblGetItemBounds

TblGetColumnSpacing

Return the spacing after the specified column.

```
Word TblGetColumnSpacing (
    TablePtr table,
    Word column
);
```

Parameters

table
Pointer to a table object.

column
Column number (zero-based).

Return Value

Returns the spacing after column (in pixels).

See Also

TblGetColumnWidth, TblSetColumnSpacing, TblSetColumnUsable

TblGetColumnWidth

Return the width of the specified column.

```
Word TblGetColumnWidth (
    TablePtr table,
    Word column
);
```

Parameters

table
Pointer to a table object.

column
Column number (zero-based).

Return Value

Returns the width of a column (in pixels).

See Also

TblGetColumnSpacing, TblSetColumnWidth, TblSetColumnUsable

TblGetCurrentField

Return a pointer to the field structure in which the user is currently editing a text item.

```
FieldPtr TblGetCurrentField (
    TablePtr table
);
```

Parameters

table
Pointer to a table object.

Return Value

Returns FieldPtr, or NULL if the table is not in edit mode.

See Also

TblGetSelection

TblGetItemBounds

Return the bounds of an item in a table.

```
void TblGetItemBounds (
    TablePtr table,
    Word row,
    Word column,
    RectanglePtr r
);
```

Parameters

table
Pointer to a table object.

row
Row of the item (zero-based).

column
Column of the item (zero-based).

r
Pointer to a structure that holds the bounds of the item.

Return Value

Returns nothing. Stores the bounds in r.

TblGetItemInt

Return the integer value stored in a table item.

```
Word TblGetItemInt (
    TablePtr table,
    Word row,
    Word column
);
```

Parameters

table
Pointer to a table object.

row
Row of the item to select (zero-based).

column
Column of the item to select (zero-based).

Return Value

Returns the integer value.

See Also

TblSetItemInt

TblGetLastUsableRow

Return the last row in a table that is usable (visible).

```
Word TblGetLastUsableRow (
    TablePtr table
);
```

Parameters

table
Pointer to a table object.

Return Value

Returns the row index (zero-based) or -1 if there are no usable rows.

See Also

TblGetRowData, TblGetRowID

TblGetNumberOfRows

Return the number of rows in a table.

```
Word TblGetNumberOfRows (
    TablePtr table
);
```

Parameters

table
Pointer to a table object.

Return Value

Returns the number of rows in the specified table.

TblGetRowData

Return the data value of the specified row.

```
ULong TblGetRowData (
    TablePtr table,
    Word row
);
```

Parameters

table
Pointer to a table object.

row
Row of the item to select (zero-based).

Remarks

The data value is a placeholder for application-specific values.

See Also

TblGetRowID, TblSetRowData

TblGetRowHeight

Return the height of the specified row.

```
Word TblGetRowHeight (
    TablePtr table,
    Word row
);
```

Parameters

table
Pointer to a table object.

row
Row to get (zero-based).

Return Value

Returns the height in pixels.

See Also

TblGetItemBounds, TblSetRowHeight

TblGetRowID

Return the ID value of the specified row.

```
Word TblGetRowID (
    TablePtr table,
    Word row
);
```

Parameters

table
Pointer to a table object.

row
Row for which the ID will be returned (zero-based).

Return Value

Returns the ID value of the row in the table.

See Also

TblGetRowData, TblSetRowHeight

TblGetSelection

Return the row and column of the currently selected table item.

```
Boolean TblGetSelection (
    TablePtr table,
    WordPtr rowP,
    WordPtr columnP
);
```

Parameters

table
Pointer to a table object.

rowP
Pointer to a Word variable in which to store the row (zero-based).

columnP
Pointer to a Word variable in which to store the column (zero-based).

Return Value

Returns TRUE if the item is highlighted, FALSE if not.

See Also

TblSetRowSelectable

TblGrabFocus

Put a table into edit mode.

```
void TblGrabFocus (
    TablePtr table,
    Word row,
    Word column
);
```

Parameters

table
Pointer to a table object.

row
Current row to be edited (zero-based).

column
Current column to be edited (zero-based).

Return Value

Returns nothing.

Remarks

Displays an error if the row or column passed is out of bounds. An editable field must exist in the coordinates passed to this function.

See Also

TblReleaseFocus

TblHandleEvent

Handle an event for the table.

```
Boolean TblHandleEvent (
    TablePtr table,
    EventPtr event
);
```

Parameters

table
    Pointer to a table object.

event
    The event to be handled.

Return Value

Returns TRUE if the event was handled, FALSE if it was not.

TblInsertRow

Insert a row into the table before the specified row.
The number of rows in the table is not increased; the last row in the table is removed.

```
void TblInsertRow (
    TablePtr table,
    Word row
);
```

Parameters

table
Pointer to a table object.

row
Row to insert (zero-based).

Return Value

Returns nothing.

Remarks

If the row parameter is greater than or equal to the number of rows in the table, an error is displayed.

See Also

TblRemoveRow, TblSetRowUsable, TblSetRowSelectable, TblMarkRowInvalid

TblMarkRowInvalid

Mark the image of the specified row invalid.

```
void TblMarkRowInvalid (
    TablePtr table,
    Word row
);
```

Parameters

table
Pointer to a table object.

row
Row of the item to select (zero-based).

Remarks

After calling this function, call TblRedrawTable to redraw all rows marked invalid. Rows not marked invalid are not redrawn.

Return Value

Returns nothing.

See Also

TblRemoveRow, TblSetRowUsable, TblSetRowSelectable, TblMarkTableInvalid, TblRowInvalid

TblMarkTableInvalid

Mark the image of all the rows in a table invalid.

```
void TblMarkTableInvalid (
    TablePtr table
);
```

Parameters

table
Pointer to a table object.

Return Value

Returns nothing.

Remarks

After calling this function, you must call TblRedrawTable to redraw all rows. Rows not marked invalid do not draw.

See Also

TblEraseTable, TblRedrawTable, TblMarkTableInvalid

TblRedrawTable

Redraw the rows of the table that are marked invalid.

```
void TblRedrawTable (
    TablePtr table
);
```

Parameters

table
    Pointer to a table object.

Return Value

Returns nothing.

See Also

TblMarkTableInvalid

TblReleaseFocus

Release the focus.

```
void TblReleaseFocus (
    TablePtr table
);
```

Parameters

table
Pointer to a table object.

Return Value

Returns nothing.

Remarks

If the current item is a text item, the memory allocated for editing is released and the insertion point is turned off.

See Also

TblGrabFocus

TblRemoveRow

Remove the specified row from the table.

```
void TblRemoveRow (
    TablePtr table,
    Word row
);
```

Parameters

table
Pointer to a table object.

row
Row to remove (zero-based).

Return Value

Returns nothing.

Remarks

The number of rows in the table is not decreased; an unusable row is added to the end of the table. If an invalid row is specified, an error is displayed.
This function does not visually update the display.

See Also

TblInsertRow, TblSetRowUsable, TblSetRowSelectable, TblMarkRowInvalid

TblRowInvalid

Determine whether a row is invalid. Invalid rows need to be re-drawn.

```
Boolean TblRowInvalid (
    TablePtr table,
    Word row
);
```

Parameters

table
Pointer to a table object.

row
Row number (zero-based).

Return Value

Returns TRUE if the row is invalid, FALSE if it's valid.

See Also

TblMarkRowInvalid

TblRowSelectable

Determine whether the specified row is selectable. Rows that are not selectable don't highlight when touched.

```
Boolean TblRowSelectable (
    TablePtr table,
    Word row
);
```

Parameters

table
    Pointer to a table object.

row
    Row of the item to select (zero-based).

Return Value

Returns TRUE if the row is selectable, FALSE if it's not.

TblRowUsable

Determine whether the specified row is usable.

```
Boolean TblRowUsable (
    TablePtr table,
    Word row
);
```

Parameters

table
Pointer to a table object.

row
Row number (zero-based).

Return Value

Returns TRUE if the row is usable, FALSE if it's not.

Remarks

Rows that are not usable do not display.

See Also

TblRowSelectable, TblGetLastUsableRow

TblSelectItem

Select (highlight) the specified item. If there is already a selected item, it is unhighlighted.

```
void TblSelectItem (
    TablePtr table,
    Word row,
    Word column
);
```

Parameters

table
Pointer to a table object.

row
Row of the item to select (zero-based).

column
Column of the item to select (zero-based).

Return Value

Returns nothing.

See Also

TblRowSelectable, TblGetItemBounds, TblGetItemInt

TblSetColumnSpacing

Set the spacing after the specified column.

```
void TblSetColumnSpacing (
    TablePtr table,
    Word column,
    Word spacing
);
```

Parameters

table
Pointer to a table object.

column
Column number (zero-based).

spacing
Spacing after the column.

Return Value

Returns nothing.

See Also

TblSetColumnUsable

TblSetColumnUsable

Set a column in a table usable or unusable.

```
void TblSetColumnUsable (
    TablePtr table,
    Word row,
    Boolean usable
);
```

Parameters

table
    Pointer to a table object.

column
    Column of the item to select (zero-based).

usable
    True for usable or false for not usable.

Return Value

Returns nothing.

Remarks

Columns that are not usable do not display.

See Also

TblMarkRowInvalid

TblSetColumnWidth

Set the width of the specified column.

```
void TblSetColumnWidth (
    TablePtr table,
    Word column,
    Word width
);
```

Parameters

table
Pointer to a table object.

column
Column number (zero-based).

width
Width of the column (in pixels).

Return Value

Returns nothing.

See Also

TblGetColumnWidth

TblSetCustomDrawProcedure

Set the custom draw callback procedure for the column specified.

```
void TblSetCustomDrawProcedure(
    TablePtr table,
    Word column,
    VoidPtr drawCallback
);
```

Parameters

table
Pointer to a table object.

column
Column of table.

drawCallback
Callback function.

Note: The callback procedure should have this prototype:

```
void drawCallback (
    VoidPtr table,
    Word row,
    Word column,
    RectanglePtr bounds
);
```

Return Value

Returns nothing.

Remarks

The custom draw callback function is used to draw table items with a TableItemStyleType of customTableItem (see table.h).

See Also

TblDrawTable

TblSetItemInt

Set the integer value of the specified item.

```
void TblSetItemInt (
    TablePtr table,
    Word row,
    Word column,
    Word value
);
```

Parameters

table
Pointer to a table object.

row
Row of the item (zero-based).

column
Column of the item (zero-based).

value
Any byte value (an integer).

Return Value

Returns nothing.

Remarks

An application can store what it wants in an item's integer value.

See Also

TblGetItemInt, TblSetItemPtr

TblSetItemPtr

Set the item to the specified pointer value.

```
void TblSetItemPtr (
    TablePtr table,
    Word row,
    Word column,
    VoidPtr value
);
```

Parameters

table
Pointer to a table object.

row
Row of the item (zero-based).

column
Column of the item (zero-based).

value
Pointer to data to display in the table item.

Return Value

Returns nothing.

Remarks

An application can store whatever it wants in the table item.

See Also

TblSetItemInt

TblSetItemStyle

Set the item to display its data in a style; for example, text, numbers, dates, and so on.

```
void TblSetItemStyle (
    TablePtr table,
    Word row,
    Word column,
    TableItemStyleType type
);
```

Parameters

table
Pointer to a table object.

row
Row of the item (zero-based).

column
Column of the item (zero-based).

type
See Table.h.

Return Value

Returns nothing.

See Also

TblSetCustomDrawProcedure

TblSetLoadDataProcedure

Set the load-data callback procedure for the specified column.

```
void TblSetLoadDataProcedure(
    TablePtr table,
    Word column,
    TableLoadDataFuncPtr loadDataCallback
);
```

Parameters

table
Pointer to a table object.

column
Column of table.

loadDataCallback
Callback procedure.

Note: The callback procedure should have this prototype:

```
VoidHand LoadDataCallback (
    VoidPtr table,
    Word row,
    Word column,
    Boolean editable,
    WordPtr dataOffset,
    WordPtr dataSize
);
```

For a text style item, the callback procedure should return the handle of a block that contains a null-terminated text string, the offset from the start of the block to the start of the string, and the amount of space allocated for the string.

Return Value

Returns nothing.

Remarks

The callback function is used to obtain the data values of a table item.

See Also

TblSetCustomDrawProcedure

TblSetRowData

Set the data value of the specified row.
The data value is a placeholder for application-specific values.

```
void TblSetRowData (
    TablePtr table,
    Word row,
    ULong data
);
```

Parameters

table
Pointer to a table object.

row
Row of the item to select (zero-based).

data
Application-specific data.

Return Value

Returns nothing.

See Also

TblGetRowData

TblSetRowHeight

Set the height of the specified row.

```
void TblSetRowHeight (
    TablePtr table,
    Word row,
    Word height
);
```

Parameters

table
    Pointer to a table object.

row
    Row to set (zero-based).

height
    New height in pixels.

Return Value

Returns nothing.

See Also

TblGetRowHeight

TblSetRowID

Set the ID value of the specified row.

```
void TblSetRowID (
    TablePtr table,
    Word row,
    Word id
);
```

Parameters

table
    Pointer to a table object.

row
    Row of the item to select (zero-based).

id
    ID to identify a row.

Return Value

Returns nothing.

See Also

TblGetRowID

TblSetRowSelectable

Set a row in a table to selectable or nonselectable.

```
void TblSetRowSelectable (
    TablePtr table,
    Word row,
    Boolean selectable
);
```

Parameters

table
Pointer to a table object.

row
Row of the item to select (zero-based).

selectable
TRUE or FALSE.

Return Value

Returns nothing.

Remarks

Rows that are not selectable don't highlight when touched.

See Also

TblRowSelectable, TblSetRowUsable

TblSetRowUsable

Set a row in a table to usable or unusable. (Rows that are not usable do not display.)

```
void TblSetRowUsable (
    TablePtr table,
    Word row,
    Boolean usable
);
```

Parameters

table
Pointer to a table object.

row
Row of the item to select (zero-based).

usable
TRUE or FALSE.

Return Value

Returns nothing.

See Also

TblRowUsable, TblSetRowSelectable

TblSetSaveDataProcedure

Set the save-data callback procedure for the specified column.

```
void TblSetSaveDataProcedure
    TablePtr table,
    Word column,
    VoidPtr saveDataCallback
);
```

Parameters

table
Pointer to a table object.

column
Column of table.

saveDataCallback
Callback function.

Note: The callback procedure should have this prototype:

```
VoidPtr SaveDataCallback(
    VoidPtr table,
    Word row,
    Word column
);
```

Remarks

The callback procedure is called when the table object determines the data of a text object needs to be saved.

Return Value

Returns nothing.

See Also

TblSetCustomDrawProcedure

TblUnhighlightSelection

Unhighlight the currently selected item in a table.

```
void TblUnhighlightSelection (
    TablePtr table
);
```

Parameters

table
    Pointer to a table object.

Return Value

Returns nothing.

WinAddWindow

Add the specified window to the active windows list.

```
void WinAddWindow (
    WinHandle winHandle
);
```

Parameters

winHandle
Handle of a window.

Return Value

Returns nothing.

Remarks

The active windows list contains all windows in the current application's user interface.

See Also

WinCreateWindow, WinRemoveWindow

WinClipRectangle

Clip a rectangle to the clipping rectangle of the draw window.

```
void WinClipRectangle (
    RectanglePtr r
);
```

Parameters

r
Pointer to a structure holding the rectangle to clip.

Remarks

The draw window is the window to which all drawing functions send their output.
The rectangle returned in r is the intersection of the rectangle passed and the draw window's clipping bounds.

Return Value

Returns nothing.

See Also

WinCopyRectangle, WinDrawRectangle, WinEraseRectangle, WinGetClip

WinCopyRectangle

Copy a rectangular region from one place to another (either between windows or within a single window).

```
void WinCopyRectangle (
    WinHandle srcWin,
    WinHandle dstWin,
    RectanglePtr srcRect,
    SWord destX,
    SWord destY,
    ScrOperation mode
);
```

Parameters

srcWin
Window from which the rectangle is copied.

dstWin
Window to which the rectangle is copied.

srcRect
Bounds of the region to copy.

destX
Top bound of the rectangle in destination window.

destY
Left bound of the rectangle in destination window.

mode
The method of transfer from the source to the destination window (see window.h).

Return Value

Returns nothing.

Remarks

Copies the bits of the window inside the rectangle region.

WinCreateWindow

Create a new window and add it to the window list.

```
WinHandle WinCreateWindow (
    RectanglePtr bounds,
    FrameType frame,
    Boolean modal,
    Boolean focusable,
    WordPtr error
);
```

Parameters

bounds
Display relative bounds of the window.

frame
Type of frame around the window (see window.h).

modal
TRUE if the window is modal.

focusable
TRUE if the window can be the active window.

error
Pointer to any error encountered by this function.

Return Value

Returns handle for the new window.

Remarks

Windows created by this routine draw to the display, see WinCreateOffscreenWindow.
New windows are created disabled, and must be enabled before they accept input.

See Also

WinCreateOffscreenWindow, WinDeleteWindow, WinInitializeWindow

WinCreateOffscreenWindow

Create a new off-screen window and add it to the window list.

```
WinHandle WinCreateOffscreenWindow (
    SWord width,
    SWord height,
    WindowFormatType format,
    WordPtr error
);
```

Parameters

width
Width of the window in pixels.

height
Height of the window in pixels.

format
Either screenFormat or genericFormat.

error
Pointer to any error encountered by this function.

Return Value

Returns the handle of the new window.

Remarks

Windows created with this routine draw to a memory buffer instead of the display.
The memory buffer has two formats: screen format and generic format. Screen format is the native format of the video system, windows in this format can be copied to the display faster. The generic format is device-independent.

See Also

WinCreateWindow, WinAddWindow

WinDeleteWindow

Remove a window from the window list and free the memory used by the window.

```
void WinDeleteWindow (
    WinHandle winHandle,
    Boolean eraseIt
);
```

Parameters

winHandle
    Handle of window to delete.

eraseIt
    If TRUE, the window is erased before it is deleted.

Return Value

Returns nothing.

See Also

WinCreateWindow

WinDisableWindow

Disable a window but leave it on the active windows list (list of all windows in the system).

```
void WinDisableWindow (
    WinHandle winHandle
);
```

Parameters

winHandle
Handle of window to disable.

Return Value

Returns nothing.

Remarks

Disabled windows ignore all pen input and cannot be made the current window or the draw window.
Windows are usually disabled when they are removed from the screen.
This function does not affect the visual appearance of the window.

See Also

WinEnableWindow, WinDeleteWindow

WinDisplayToWindowPt

Convert a display-relative coordinate to a window-relative coordinate.
The coordinate returned is relative to the display window.

```
void WinDisplayToWindowPt (
    SWordPtr extentX,
    SWordPtr extentY
);
```

Parameters

extentX
    Pointer to x coordinate to convert.

extentY
    Pointer to y coordinate to convert.

Return Value

Returns nothing.

See Also

WinWindowToDisplayPt

WinDrawBitmap

Draw a bitmap at the given x and y coordinates.

```
void WinDrawBitmap (
    BitmapPtr bitmapP,
    SWord x,
    SWord y
);
```

Parameters

bitmapP
    Pointer to a bitmap.

x
    The x coordinate of the top-left corner.

y
    The y coordinate of the top-left corner.

Return Value

Returns nothing.

See Also

WinEraseRectangle

WinDrawChars

Draw the specified characters in the draw window.

```
void WinDrawChars (
    CharPtr chars,
    Word len,
    SWord x,
    SWord y
);
```

Parameters

chars
Pointer to the characters to draw.

len
Number of characters to draw.

x
Left bound of first character to draw.

y
Top bound of first character to draw.

Return Value

Returns nothing.

Remarks

Before calling this function, you may call WinSetUnderlineMode and FntSetFont.

See Also

WinDrawInvertedChars, WinEraseChars, WinSetUnderlineMode

WinDrawGrayLine

Draw a line in the draw window.

```
void WinDrawGrayLine (
    SWord x1,
    SWord y1,
    SWord x2,
    SWord y2
);
```

Parameters

x1
x coordinate of the start of the line.

y1
y coordinate of the start of the line.

x2
x coordinate of the end of the line.

y2
y coordinate of the end of the line.

Return Value

Returns nothing.

See Also

WinDrawLine

WinDrawGrayRectangleFrame

Draw a gray rectangular frame in the draw window.

```
void WinDrawGrayRectangleFrame (
    FrameType frame,
    RectanglePtr r
);
```

Parameters

frame
Type of frame to draw.

r
Pointer to the rectangle to frame.

Return Value

Returns nothing.

Remarks

The standard gray pattern is not used by this routine; rather, the frame is drawn so that the top-left pixel of the frame is always on.

See Also

WinDrawRectangleFrame

WinDrawInvertedChars

Draw the specified characters inverted (background color) in the draw window.

```
void WinDrawInvertedChars(
    CharPtr chars,
    Word len,
    SWord x,
    SWord y
);
```

Parameters

chars
Pointer to the characters to draw.

len
Number of characters to draw.

x
Left bound of first character to draw.

y
Top bound of first character to draw.

Return Value

Returns nothing.

Remarks

The characters are drawn in the background color and the off pixels are drawn in the foreground color. Before calling this function, you may call WinSetUnderlineMode and FntSetFont.

See Also

WinDrawChars

WinDrawLine

Draw a line in the draw window.

```
void WinDrawLine (
    short x1,
    short y1,
    short x2,
    short y2
);
```

Parameters

x1
    x coordinate of the start of the line.

y1
    y coordinate of the start of the line.

x2
    x coordinate of the end of the line.

y2
    y coordinate of the end of the line.

Return Value

Returns nothing.

See Also

WinDrawGrayLine, WinEraseLine, WinFillLine

WinDrawRectangle

Draw a black rectangle in the draw window; the rectangle can have square or round corners.

```
void WinDrawRectangle (
    RectanglePtr r,
    Word cornerDiam
);
```

Parameters

r
    Pointer to the rectangle to draw.

cornerDiam
    Diameter of rounded corners. Zero for square corners.

Return Value

Returns nothing.

Remarks

The cornerDiam parameter specifies the diameter of four imaginary circles used to form the rounded corners. An imaginary circle is placed within each corner tangent to the rectangle on two sides.

See Also

WinFillRectangle, WinEraseRectangle

WinDrawRectangleFrame

Draw a rectangular frame around the specified region in the draw window.

```
void WinDrawRectangleFrame (
    FrameType frame,
    RectanglePtr
);
```

Parameters

frame
Type of frame to draw.

r
Pointer to the rectangle to frame.

Return Value

Returns nothing.

Remarks

The frame is drawn outside the specified region.

See Also

WinEraseRectangleFrame, WinGetFramesRectangle, WinDrawGrayRectangleFrame, WinDrawWindowFrame

WinDrawWindowFrame

Draw the frame of the current drawing window.

```
void WinDrawWindowFrame (
    void
);
```

Parameters

None.

Return Value

Returns nothing.

See Also

WinDrawRectangleFrame, WinGetDrawWindow

WinEnableWindow

Enable a window.

```
void WinEnableWindow (
    WinHandle winHandle
);
```

Parameters

winHandle
    Handle of the window to enable.

Return Value

Returns nothing.

Remarks

Enabled windows accept pen input and can be made the active window.
This routine does not affect the visual appearance of the window.

See Also

WinDisableWindow, WinSetActiveWindow

WinEraseChars

Erase specified characters in the draw window.

```
void WinEraseChars (
    CharPtr chars,
    Word len,
    SWord x,
    SWord y
);
```

Parameters

chars
Pointer to the characters to erase.

len
Number of characters to erase.

x
Left bound of first character to erase.

y
Top bound of first character to erase.

Return Value

Returns nothing.

See Also

WinDrawChars

WinEraseLine

Erase a line in the draw window.

```
void WinEraseLine (
    SWord x1,
    SWord y1,
    SWord x2,
    SWord y2
);
```

Parameters

x1
x coordinate of the start of the line.

y1
y coordinate of the start of the line.

x2
x coordinate of the end of the line.

y2
y coordinate of the end of the line.

Return Value

Returns nothing.

See Also

WinDrawLine

WinEraseRectangle

Erase a rectangle in the draw window. (The rectangle can have round or square corners; see WinDrawRectangle.)

```
void WinEraseRectangle (
    RectanglePtr r,
    Word cornerDiam
);
```

Parameters

r
    Pointer to the rectangle to erase.

cornerDiam
    Diameter of rounded corners; zero for square corners.

Return Value

Returns nothing.

See Also

WinDrawRectangle

WinEraseRectangleFrame

Erase a rectangular frame in the draw window.

```
void WinEraseRectangleFrame (
    FrameType frame,
    RectanglePtr r
);
```

Parameters

frame
Type of frame to erase.

r
Pointer to the rectangular frame.

Return Value

Returns nothing.

See Also

WinDrawRectangleFrame

WinEraseWindow

Erase the contents of the draw window.
The frame around the draw window is not erased by this routine.

```
void WinEraseWindow (
    void
);
```

Parameters

None.

Return Value

Returns nothing.

See Also

WinEnableWindow

WinFillLine

Fill a line in the draw window with the current pattern. You can set the current pattern with WinSetPattern.

```
void WinFillLine (
    SWord x1,
    SWord y1,
    SWord x2,
    SWord y2
);
```

Parameters

x1
x coordinate of the start of the line.

y1
y coordinate of the start of the line.

x2
x coordinate of the end of the line.

y2
y coordinate of the end of the line.

Return Value

Returns nothing.

See Also

WinSetPattern, WinDrawLine

WinFillRectangle

Draw a rectangle with current pattern. (The rectangle can have square or round corners.)

```
void WinFillRectangle (
    RectanglePtr r,
    Word cornerDiam
);
```

Parameters

r
Pointer to the rectangle to draw.

cornerDiam
Diameter of rounded corners. Zero for square corners.

Return Value

Returns nothing.

Remarks

You can set the current pattern with WinSetPattern.

See Also

WinSetPattern. WinDrawRectangle

WinGetActiveWindow

Return the window handle of the active window.

```
WinHandle WinGetActiveWindow (
    void
);
```

Parameters

None.

Return Value

Returns the handle of the active window.

See Also

WinSetActiveWindow, WinGetDisplayWindow, WinGetFirstWindow, WinGetDrawWindow, WinRemoveWindow

WinGetClip

Return the clipping rectangle of the draw window.

```
void WinGetClip (
    RectanglePtr r
);
```

Parameters

r
Pointer to a structure to hold the clipping bounds.

Return Value

Returns nothing.

See Also

WinSetClip

WinGetDisplayExtent

Return the width and height of the display (the screen).

```
void WinGetDisplayExtent (
    SWordPtr extentX,
    SWordPtr extentY
);
```

Parameters

extentX
    Pointer to the width of the display.

extentY
    Pointer to the height of the display.

Return Value

Returns nothing.

WinGetDisplayWindow

Return the window handle of the display window.

```
WinHandle WinGetDisplayWindow (
    void
);
```

Parameters

None.

Return Value

Returns handle of display window.

Remarks

The display window is created by the system at start-up; its size is the same as the physical display (screen).

See Also

WinGetDisplayExtent, WinGetActiveWindow, WinGetDrawWindow

WinGetDrawWindow

Return the window handle of the current draw window.

```
WinHandle WinGetDrawWindow (
    void
);
```

Parameters

None.

Return Value

Returns handle of draw window

See Also

WinGetDisplayWindow, WinGetActiveWindow, WinSetDrawWindow

WinGetFirstWindow

Return a pointer to the first window in the linked list of windows.

```
WinHandle WinGetFirstWindow (
    void
);
```

Parameters

None.

Return Value

Returns handle of first window.

Remarks

This function is usually used by the system only.

See Also

WinAddWindow, WinGetActiveWindow

WinGetFramesRectangle

Return the region needed to draw a rectangle with the specified frame around it.

```
void WinGetFramesRectangle (
    FrameType frame,
    RectanglePtr r,
    RectanglePtr obscuredRect
);
```

Parameters

frame
    Type of frame drawn around the rectangle.

r
    Pointer to the rectangle to frame.

obscuredRect
    Pointer to the rectangle obscured by the frame.

Return Value

Returns nothing.

Remarks

Frames are always drawn around (outside) a rectangle.

See Also

WinGetWindowBounds

WinGetPattern

Return the current fill pattern.

```
void WinGetPattern (
    CustomPatternType pattern
);
```

Parameters

pattern
Pattern buffer to hold pattern.

Return Value

Returns nothing.

Remarks

The fill pattern is used by WinFillLine and WinFillRectangle.
A pattern defines an 8-x-8 bit pattern. The pattern is tiled to fill the specified region. The pattern structure is eight bytes long, the first byte is the first row of the pattern.

See Also

WinSetPattern

WinGetWindowBounds

Return the bounds of the current draw window in display-relative coordinates.

```
void WinGetWindowBounds (
    RectanglePtr r
);
```

Parameters

r
    Pointer to a rectangle.

Return Value

Returns nothing.

See Also

WinGetWindowExtent

WinGetWindowExtent

Return the width and height of the current draw window.

```
void WinGetWindowExtent (
    SWordPtr extentX,
    SWordPtr extentY
);
```

Parameters

extentX
    Pointer to the width of the draw window.

extentY
    Pointer to the height of the draw window.

Return Value

Returns nothing.

See Also

WinGetWindowBounds, WinGetWindowFrameRect

WinGetWindowFrameRect

Return a rectangle, in display-relative coordinates, that defines the size and location of a window and its frame.

```
void WinGetWindowFrameRect (
    WinHandle winHandle,
    RectanglePtr r
);
```

Parameters

winHandle
    Handle of window whose coordinates are desired.

r
    Pointer to the coordinates of the window.

Return Value

Returns nothing.

See Also

WinGetWindowBounds

WinGetWindowPointer

Return a pointer to the specified window's WindowType structure.

```
WinPtr WinGetWindowPointer (
    WinHandle winHandle
);
```

Parameters

winHandle
    Handle of a window.

Return Value

Returns nothing.

See Also

WinGetActiveWindow

WinInitializeWindow

Initialize the screen-dependent members of a WindowType structure and set the window's clipping bounds to the window's bounds.

```
void WinInitializeWindow (
    WinHandle winHandle
);
```

Parameters

winHandle
    Handle of a window.

Return Value

Returns nothing.

See Also

WinCreateWindow

WinInvertChars

Invert the specified characters in the draw window.

```
void WinInvertChars (
    CharPtr chars,
    Word len,
    SWord x,
    SWord y
);
```

Parameters

chars
    Pointer to the characters to invert.

len
    Number of characters to invert.

x
    Left bound of first character to invert.

y
    Top bound of first character to invert.

Return Value

Returns nothing.

See Also

WinDrawInvertedChars, WinDrawChars

WinInvertLine

Invert a line in the draw window.

```
void WinInvertLine (
    SWord x1,
    SWord y1,
    SWord x2,
    SWord y2
);
```

Parameters

x1
x coordinate of the start of the line.

y1
y coordinate of the start of the line.

x2
x coordinate of the end of the line.

y2
y coordinate of the end of the line.

Return Value

Returns nothing.

See Also

WinInvertRectangle, WinInvertRectangleFrame, WinDrawLine, WinEraseLine

WinInvertRectangle

Invert a rectangle in the draw window. (The rectangle can have square or round corners.)

```
void WinInvertRectangle (
    RectanglePtr r,
    Word cornerDiam
);
```

Parameters

r
Pointer to the rectangle to invert.

cornerDiam
Diameter of rounded corners; zero for square corners.

Return Value

Returns nothing.

See Also

WinInvertLine, WinInvertRectangleFrame, WinDrawRectangle

WinInvertRectangleFrame

Invert a rectangular frame in the draw window.

```
void WinInvertRectangleFrame (
    FrameType frame,
    RectanglePtr r
);
```

Parameters

frame
Type of frame to invert.

r
Pointer to the rectangular frame to invert.

Return Value

Returns nothing.

See Also

WinInvertRectangle, WinInvertLine, WinDrawRectangleFrame, WinEraseRectangleFrame

WinModal

Return TRUE if the specified window is modal.

```
Boolean WinModal (
    WinHandle winHandle
);
```

Parameters

winHandle
    Handle of a window.

Return Value

Returns TRUE if modal, otherwise FALSE.

Remarks

A window is modal if it cannot lose the focus.

WinRemoveWindow

Remove the specified window from the window list.

```
void WinRemoveWindow (
    WinHandle winHandle
);
```

Parameters

winHandle
Handle of a window.

Return Value

Returns nothing.

Remarks

Does not free the memory used by the window.

See Also

WinAddWindow, WinDeleteWindow, WinGetFirstWindow

WinResetClip

Reset the clipping rectangle of the draw window to the portion of the draw window that is within the bounds of the display.

```
void WinResetClip (
    void
);
```

Parameters

None.

Return Value

Returns nothing.

See Also

WinSetClip

WinRestoreBits

Copy the contents of the specified window to the draw window and delete the passed window.

```
void WinRestoreBits (
    WinHandle winHandle,
    SWord destX,
    SWord destY
);
```

Parameters

winHandle
Handle of window to copy and delete.

destX
x coordinate in the draw window to copy to.

destY
y coordinate in the draw window to copy to.

Return Value

Returns nothing.

Remarks

This routine is generally used to restore a region of the display that was saved with WinSaveBits.

See Also

WinSaveBits

WinSaveBits

Create an offscreen window and copy the specified region from the draw window to the offscreen window.

```
WinHandle WinSaveBits (
    RectanglePtr sourceP,
    WordPtr error
);
```

Parameters

sourceP
Pointer to the bounds of the region to save, relative to the display.

error
Pointer to any error encountered by this function.

Return Value

Returns the handle of the window containing the saved image, or zero if an error occurred.

Remarks

The offscreen window is the same size as the region to copy.

See Also

WinRestoreBits

WinScrollRectangle

Scroll a rectangle in the draw window.

```
void WinScrollRectangle (
    RectanglePtr r,
    DirectionType direction,
    SWord distance,
    RectanglePtr vacated
);
```

Parameters

r
Pointer to the rectangle to scroll.

direction
Direction to scroll (up, down, left, or right).

distance
Distance to scroll in pixels.

vacated
Pointer to the rectangle that needs to be redrawn because it has been vacated as a result of the scroll.

Return Value

Returns nothing.

Remarks

The rectangle scrolls within its own bounds. Any portion of the rectangle that is scrolled outside its bounds is clipped.

WinSetActiveWindow

Make a window the active window.

```
void WinSetActiveWindow (
    WinHandle winHandle
);
```

Parameters

winHandle
    Handle of a window

Return Value

Returns nothing.

Remarks

The active window is not actually set in this routine; flags are set to indicate that a window is being exited and another window is being entered. The routine EvtGetEvent sends a winExitEvent and a winEnterEvent when it detects these flags. The active window is set by EvtGetEvent when it sends the winEnterEvent. The draw window is also set to the new active window, when the active window is changed.
All user input is directed to the active window.

See Also

WinAddWindow, WinGetActiveWindow

WinSetClip

Set the clipping rectangle of the draw window.

```
void WinSetClip (
    RectanglePtr r
);
```

Parameters

r
    Pointer to a structure holding the clipping bounds.

Return Value

Returns nothing.

See Also

WinClipRectangle, WinSetClip, WinGetClip

WinSetDrawWindow

Set the draw window. (All drawing operations are relative to the draw window.)

```
WinHandle WinSetDrawWindow (
    WinHandle winHandle
);
```

Parameters

winHandle
    Handle of a window.

Return Value

Returns the draw window.

See Also

WinGetDrawWindow, WinSetActiveWindow

WinSetPattern

Set the current fill pattern.

```
void WinSetPattern (
    CustomPatternType pattern
);
```

Parameters

pattern
Pattern to use.

Return Value

Returns nothing.

Remarks

The fill pattern is used by WinFillLine and WinFillRectangle.

See Also

WinGetPattern

WinSetUnderlineMode

Set the graphic state to enable or disable the underlining of characters.

```
UnderlineModeType WinSetUnderlineMode (
    UnderlineModeType mode
);
```

Parameters

mode
New underline mode type, one of noUnderline, grayUnderline, solidUnderline.

Return Value

Returns the previous underline mode type.

See Also

WinDrawChars

WinWindowToDisplayPt

Convert a window-relative coordinate to a display-relative coordinate.

```
void WinWindowToDisplayPt (
    SWordPtr extentX,
    SWordPtr extentY
);
```

Parameters

extentX
Pointer to x coordinate to convert.

extentY
Pointer to y coordinate to convert.

Return Value

Returns nothing.

Remarks

The coordinate passed is assumed to be relative to the draw window.

See Also

WinDisplayToWindowPt

CategoryCreateList

Read a database's categories and set categories.

```
void CategoryCreateList (
    DmOpenRef db,
    ListPtr lst,
    Word currentCategory,
    Boolean showAll
);
```

Parameters

db
    Database containing categories to extract.

lst
    List object to load categories into.

currentCategory
    Will be set as the current selection in the resulting list.

showAll
    TRUE if an "All" category should be included in the list.

Return Value

Returns nothing.

CategoryEdit

Event handler for the Edit Categories dialog.

```
Boolean CategoryEdit (
    DmOpenRef db,
    WordPtr category
);
```

Parameters

db
Database containing the categories to be edited.

category
Current category.

Return Value

Returns TRUE if any of the following conditions are true:
the current category is renamed
the current category is deleted
the current category is merged with another category

CategoryFind

Return the index of the category that matches the name passed.

```
Word CategoryFind (
    DmOpenRef db,
    CharPtr name
);
```

Parameters

db
Database to search for the passed category.

name
Category name.

Return Value

Returns the category index.

CategoryFreeList

Unlock or free memory locked or allocated by CategoryCreateList which was attached to the passed List object.

```
void CategoryFreeList
    DmOpenRef db,
    ListPtr lst
);
```

Parameters

db
Database containing the categories.

1st
Pointer to the category list containing the memory to be freed.

Return Value

Returns nothing.
Comment Calling this function does not remove the categories from the passed database.

CategoryGetName

Return the name of the specified category.

```
void CategoryGetName (
    DmOpenRef db,
    Word index,
    CharPtr name
);
```

Parameters

db
Database that contains the categories.

index
Category index.

name
Buffer to hold category name. Buffer should be dmCategoryLength in size.

Return Value

Stores the category name in the name buffer passed.

CategoryGetNext

Given a category index this routine return the index of the next category.
Categories are not stored sequentially.

```
Word CategoryGetNext (
    DmOpenRef db,
    Word index
);
```

Parameters

db
Database that contains the categories.

index
Category index.

Return Value

Category index of next category.

CategoryTruncateName

Truncate a category name so that it's short enough to display.

```
void CategoryTruncateName (
    CharPtr name,
    Word maxWidth
);
```

Parameters

name
Category name to truncate.

maxWidth
Maximum size, in pixels, of truncated category (including ellipsis).

Return Value

Returns nothing

CategorySetTriggerLabel

Set the label displayed by the category trigger. The category name is truncated if it's to long.

```
void CategorySetTriggerLabel (
    ControlPtr ctl,
    CharPtr name
);
```

Parameters

ctl
Pointer to control object to relabel.

label
Pointer to the name of the new category.

Return Value

Returns nothing.

CategorySelect

Process the selection and editing of categories.

```
Boolean CategorySelect (
    DmOpenRef db,
    FormPtr frm,
    Word ctlID,
    Word lstID,
    Boolean title,
    WordPtr categoryP,
    CharPtr categoryName
);
```

Parameters

db
Database that contains the categories.

frm
Form that contains the category popup list.

ctlID
ID of the popup trigger.

lstID
ID of the popup list.

title
True if the popup trigger is on the title line.

categoryP
Current category (index into db structure).

categoryName
Name of the current category.

Return Value

Returns TRUE if any of the following conditions are true:
the current category is renamed
the current category is deleted
the current category is merged with another category

GetCharAttr

Return a pointer to the characters attributes array which is used by the character classification and character conversion macros (such as isalpha and toascii).

```
WordPtr GetCharAttr (
    void
);
```

Parameters

None

Return Value

A pointer to the attributes array. See CharAttr.h for an explanation of the attributes.

GetCharCaselessValue

Return a pointer to an array that maps all characters to an assigned caseless and accentless value. This should be used for finding text.

```
BytePtr GetCharCaselessValue (
    void
);
```

Parameters

None.

Return Value

A pointer to the sort array.
The compiler pads each byte out to a word so each index position contains two characters.
Note: array[x].high = sort value for character 2x+1.

GetCharSortValue

Return a pointer to an array that maps all characters to an assigned sorting value. This should be used for ordering (sorting) text.

```
BytePtr GetCharSortValue (
    void
);
```

Parameters

None.

Return Value

Returns a pointer to the attributes array.
The compiler pads each byte out to a word so each index position contains two characters.
Note: array[x].low = sort value for character 2x.

ClipboardAddItem

Add the item passed to the specified clipboard. The format parameter determines which clipboard (text, ink, etc.) the item is added to.

```
void ClipboardAddItem (
    ClipboardFormatType format,
    VoidPtr ptr,
    Word length
);
```

Parameters

format
Text, ink, bitmap, etc.

ptr
Pointer to the item to place on the clipboard.

length
Size of the item to place on the clipboard.

Return Value

Returns nothing.

See Also

FldCut, FldCopy

ClipboardGetItem

Return the handle of the contents of the clipboard of a specified type and the length of a clipboard item.

```
VoidHand ClipboardGetItem (
    ClipboardFormatType format,
    WordPtr length
);
```

Parameters

format
Text, ink, bitmap, etc.

length
Pointer to the length of the clipboard item.

Return Value

Handle of the clipboard item.

FntAccentHeight

Return the height of an accent of the characters in the current font.
The height of an accent is the distance between the top of the character cell and the top a non-accent capital letter.

```
short FntAccentHeight (
    void
);
```

Parameters

None.

Return Value

Height of an accent (in pixels).

FntAscent

Return the ascent of the characters in the current font. The ascent of a character is the distance from the top of a non-accent capital letter to the base line.

```
short FntAscent (
    void
);
```

Parameters

None.

Return Value

Returns character ascent (in pixels).

FntAverageCharWidth

Return the average character width in the current font.

```
short FntAverageCharWidth (
    void
);
```

Parameters

None.

Return Value

Returns the average character width (in pixels).

FntBaseLine

Return the distance from the top of character cell to the baseline for the current font.

```
short FntBaseLine (
    void
);
```

Parameters

None.

Return Value

Returns the baseline of the font (in pixels).

FntCharHeight

Return the character height, in the current font including accents and descenders.

```
short FntCharHeight (
    void
);
```

Parameters

None

Return Value

Height of the characters in the current font, expressed in pixels.

FntCharsInWidth

Find the number of characters in a string that fit within a passed width. Spaces at the end of a string are ignored and removed. Any characters after a carriage return are ignored and the string is considered truncated.

```
void FntCharsInWidth (
    CharPtr string,
    Int *stringWidthP,
    Int *stringLengthP,
    Boolean *fitWithinWidth
);
```

Parameters

string
Pointer to the char string.

stringWidthP
Maximum width to allow.

stringLengthP
Maximum characters to allow (assumes current Font).

fitWithinWidth
Set to TRUE if the string is considered truncated.

Return Value

When the call is completed, the information is updated as follows:
stringWidthP Set to the width of the chars allowed.
stringLengthP Set to the number of chars within the width.
fitWithinWidth TRUE if the string is considered truncated, FALSE if it isn't.

FntCharsWidth

Return the width of the specified character string. The Missing Character Symbol is substituted for any character which does not
exist in the current font.

```
short FntCharsWidth (
    CharPtr pChars,
    Word length
);
```

Parameters

pChars
    Pointer to a string of characters.

length
    Number of character in the string.

Return Value

Returns the width of the string, in pixels.

FntCharWidth

Return the width of the specified character. If the specified character does not exist within the current font, the Missing Character Symbol is substituted.

```
short FntCharWidth (
    char ch
);
```

Parameters

ch
    Character whose width is needed.

Return Value

Returns the width of the specified character (in pixels).

FntDescenderHeight

Return the height of a character's descender in the current font.
The height of a descender is the distance between the base line and the bottom of the character cell.

```
short FntDescenderHeight (
    void
);
```

Parameters

None.

Return Value

Returns the height of a descender, expressed in pixels.

FntGetFont

Return the Font ID of the current font.

```
FontID FntGetFont (
    void
);
```

Parameters

None.

Return Value

Returns FontID of the current font.

FntGetFontPtr

Return a pointer to the current font.

```
FontPtr FntGetFontPtr (
    void
);
```

Parameters

None.

Return Value

Returns the FontPtr of the current font.

FntLineHeight

Return the height of a line in the current font. The height of a line is the height of the character cell plus the space between lines (the external leading).

```
short FntLineHeight (
    void
);
```

Parameters

None.

Return Value

Returns the height of a line in the current font.

FntLineWidth

Return the width of the specified line of text, taking tab characters in to account. The function assumes that the characters passed are left-aligned and that the first character in the string is the first character drawn on a line. In other words, this routine doesn't work for characters that don't start at the beginning of a line.

```
short FntLineWidth (
    CharPtr pChars,
    Word length
);
```

Parameters

pChars
Pointer to a string of characters.

length
Number of character in the string.

Return Value

Returns the line width (in pixels).

FntProportionalFont

Indicate whether the current font is proportionally spaced or fixed width.

```
Boolean FntProportionalFont (
    void
);
```

Parameters

None.

Return Value

Returns TRUE if the current font is proportionally spaced, FALSE if it's fixed width.

FntSetFont

Set the current font.

```
FontID FntSetFont (
    FontID fontID
);
```

Parameters

fontID
ID of the font to make the active font.

Return Value

Returns ID of the current font before the change.

AbtShowAbout

Displays the info dialog box. The application name is picked up from either the tAIN resource of the application, or the name of the application database (which is assigned in the makefile).

```
void AbtShowAbout (
    ULong creator
);
```

Parameters

creator
Creator ID of this application.

Return Value

Returns nothing.

DayHandleEvent

Handle event in the specified control. This routine handles two type of events, penDownEvent and ctlEnterEvent.

```
Boolean DayHandleEvent (
    DaySelectorPtr pSelector,
    EventPtr pEvent
);
```

Parameters

pSelector
Pointer to control object (ControlType)

pEvent
Pointer to an EventType structure.

pError
Pointer to returned error code

Return Value

True if the event was handle or false if it was not.
Posts a daySelectEvent with info on whether to use the date.
A date is used if the user selects a day in the visible month.

Find

```
void Find (
    GoToParamsPtr goToP
);
```

WARNING: System Use Only!

UIInitialize

```
void UIInitialize (
    void
);
```

WARNING: System Use Only!

UIReset

```
void UIReset (
    void
);
```

WARNING: System Use Only!

SysAppLaunch

Launch the specified application with the given command line arguments, given a card number and database ID of an application resource database.

```
Err SysAppLaunch(
    UInt cardNo,
    LocalID dbID,
    UInt launchFlags,
    Word cmd,
    Ptr cmdPBP,
    DWord* resultP
);
```

Parameters

cardNo, dbID *cardNo* and *dbID*
identify the application.

launchFlags
Set to 0.

cmd
Action code.

cmdPBP
Action code parameter block.

resultP
Pointer to what's returned by the application's PilotMain routine.

Return Value

Returns 0 if no error, or one of sysErrParamErr, memErrNotEnoughSpace, sysErrOutOfOwnerIDs.

Remarks

Launching an application with all launch bits cleared makes the application a subroutine call from the point of view of the caller.

See Also

SysBroadcastActionCode, SysUIAppSwitch, SysCurAppDatabase

SysBatteryInfo

Retrieve settings for the batteries. Set set to FALSE to retrieve battery settings. (Applications should not change any of the settings).
WARNING: Use this function only to retrieve settings!

```
UInt SysBatteryInfo(
    Boolean set,
    UIntPtr warnThresholdP,
    UIntPtr criticalThresholdP,
    UIntPtr maxTicksP,
    SysBatteryKind* kindP,
    Boolean* pluggedIn
);
```

Parameters

set
If false, parameters with non-nil pointers are retrieved. Never set this parameter to TRUE.

warnThresholdP
Pointer to battery voltage warning threshold in volts*100, or nil.

criticalThresholdP
Pointer to the battery voltage critical threshold in volts*100, or nil.

maxTicksP
Pointer to the battery timeout, or nil.

kindP
Pointer to the battery kind, or nil.

pluggedInP
Pointer to pluggedIn return value, or nil.

Return Value

Returns the current battery voltage in volts*100.

Remarks

Call this function to make sure an upcoming activity won't be interrupted by a low battery warning. warnThresholdP and maxTicksP are the battery-warning voltage threshold and time out. If the battery voltage falls below the threshold, or the timeout expires, a lowBatteryChr key event is put on the queue. Normally, applications call SysHandleEvent which calls SysBatteryWarningDialog in response to this event.
criticalThresholdP is the battery voltage threshold. If battery voltage falls below this level, the system turns itself off without warning and doesn't turn on until battery voltage is above it again.

SysBroadcastActionCode

Send the specified action code and parameter block to the latest version of every UI application.

```
Err SysBroadcastActionCode (
    Word cmd,
    Ptr cmdPBP
);
```

Parameters

cmd
    Action code to send.

cmdPBP
    Action code parameter block to send.

Return Value

Returns 0 if no error, or one of the following errors:
sysErrParamErr, memErrNotEnoughSpace, sysErrOutOfOwnerIDs.

See Also

SysAppLaunch

SysCopyStringResource

Copy a resource string to a passed string.

```
void SysCopyStringResource (
    CharPtr string,
    UInt theID
);
```

Parameters

string
    String to copy the resource string to

theID
    Resource string ID

Return Value

Stores a copy of the resource string in string.

SysCurAppDatabase

Return the card number and database ID of the current application's resource database.

```
Err SysCurAppDatabase (
    UIntPtr cardNoP,
    LocalID* dbIDP
);
```

Parameters

cardNoP
    Pointer to the card number; 0 or 1.

dbIDB
    Pointer to the database ID.

Return Value

Returns 0 if no error, or SysErrParamErr if an error occurs.

See Also

SysAppLaunch, SysUIAppSwitch

SysFormPointerArrayToStrings

Form an array of pointers to strings in a block. Useful for setting the items of a list.

```
VoidHand SysFormPointerArrayToStrings (
    CharPtr c,
    Int stringCount
);
```

Parameters

c
    Pointer to packed block of strings, each terminated by NULL.

stringCount
    Count of strings in block.

Return Value

Unlocked handle to allocated array of pointers to the strings in the passed block. The returned array points to the strings in the passed packed block.

SysHandleEvent

Handle defaults for system events such as hard and soft key presses.

```
Boolean SysHandleEvent (
    EventPtr eventP
);
```

Parameters

eventP
    Pointer to an event.

Return Value

Returns TRUE if the system handled the event.

Remarks

Applications should call this routine immediately after calling EvtGetEvent unless they want to override the default system be-havior.
However, overriding the default system behavior is almost never appropriate for an application.

See Also

EvtProcessSoftKeyStroke, KeyRates

SysInsertionSort

Sort elements in an array according to the passed comparison function.
Only elements which are out of order move. Moved elements are moved to the end of the range of equal elements. If a large amount of elements are being sorted, try to use the quick sort (see SysQSort).
This the insertion sort algorithm: Starting with the second element, each element is compared to the preceding element. Each element not greater than the last is inserted into sorted position within those already sorted. A binary search for the insertion point is performed.
A moved element is inserted after any other equal elements.

```
void SysInsertionSort (
    Byte baseP,
    Int numOfElements,
    Int width,
    CmpFuncPtr comparF,
    Long other
);
```

Parameters

baseP
    Base pointer to an array of elements.

numOfElements
    Number of elements to sort (must be at least 2).

width
    Width of an element.

comparF
    Comparison function (see Remarks).

other
    Other data passed to the comparison function.

Return Value

Returns nothing.

Remarks

The comparison function (comparF) has this prototype:

```
int comparF (
    BytePtr A,
    BytePtr B,
    Long other
);
```

The function returns:
  > 0 if A > B
  < 0 if A< B
  0 if A = B

See Also

SysQSort

SysKeyboardDialog

Pop up the system keyboard if there is a field object with the focus.
The field object's text chunk is edited directly.

```
void SysKeyboardDialog (
    void
);
```

Parameters

None.

Return Value

Returns nothing. The field's text chunk is changed.

See Also

FrmSetFocus

SysQSort

Sort elements in an array according to the passed comparison function.
Equal records can be in any position relative to each other because a quick sort tends to scramble the ordering of records. As a result, calling SysQSort multiple times can result in a different order if the records are not completely unique. If you don't want this behavior, use the insertion sort instead (see SysInsertionSort).
To pick the pivot point, the quick sort algorithm picks the middle of three records picked from around the middle of all records. That
way, the algorithm can take advantage of partially sorted data.
These optimizations are built in:
The routine contains its own stack to limit uncontrolled recursion. When the stack is full, an insertion sort is used because it doesn't require more stack space.
An insertion sort is also used when the number of records is low. This avoids the overhead of a quick sort which is noticeable for small numbers of records.
If the records seem mostly sorted, an insertion sort is performed to move only those few records needing moving.

```
void SysQSort (
    Byte baseP,
    Int numOfElements,
    Int width,
    CmpFuncPtr comparF,
    Long other
);
```

Parameters

baseP
Base pointer to an array of elements.

numOfElements
Number of elements to sort (must be at least 2),

width
Width of an element.

comparF
Comparison function. See Remarks for SysInsertionSort.

other
Other data passed to the comparison function.

Return Value

Returns nothing.

See Also

SysInsertionSort

SysRandom

Return a random number anywhere from 0 to sysRandomMax.

```
Int SysRandom (
    ULong newSeed
);
```

Parameters

newSeed
    New seed value, or 0 to use existing seed.

Return Value

Returns a random number.

SysReset

Perform a soft reset and reinitialize the globals and the dynamic memory heap.

```
void SysReset (
    void
);
```

Parameters

None.

Return Value

No return value.

Remarks

This routine resets the system, reinitializes the globals area and all system managers, and reinitializes the dynamic heap. All database information is preserved. This routine is called when the user presses the hidden reset button on the device.
When running an application using the simulator, this routine looks for two data files that represent the memory of card 0 and
card 1. If these are found, the Palm OS memory image is created using them. If they are not found, they are created.
When running an application on the device, this routine simply looks for the memory cards at fixed locations.

SysSetAutoOffTime

Set the time out value in seconds for auto-power-off. Zero means never power-off.

```
UInt SysSetAutoOffTime (
    UInt seconds
);
```

Parameters

seconds
    Time out in seconds, or 0 for no time out

Return Value

Returns previous value of time out in seconds.

SysTaskDelay

Put the processor into doze mode for the specified number of ticks.

```
Err SysTaskDelay (
    Long delay
);
```

Parameters

delay
    Number of ticks to wait (see sysTicksPerSecond)

Return Value

Returns 0 if no error.

See Also

EvtGetEvent

SysUIAppSwitch

Try to make the current UI application quit and then launch the UI application specified by card number and database ID.

```
Err SysUIAppSwitch(
    UInt cardNo,
    LocalID dbID,
    Word cmd,
    Ptr cmdPBP
);
```

Parameters

cardNo
    Card number for the new application; currently only card 0 is valid.

dbID
    ID of the new application.

cmd
    Action code.

cmdPBP
    Action code parameter block.

Return Value

Returns 0 if no error.

See Also

SysAppLaunch

SysAppExit

```
Err SysAppExit (
    SysAppInfoPtr appInfoP,
    Ptr prevGlobalsP,
    Ptr globalsP
);
```

WARNING: System Use Only!

SysAppInfoPtr

```
SysAppInfoPtr SysCurAppInfoP (
    void
);
```

WARNING: System Use Only!

SysAppStartup

```
Err SysAppStartup (
    SysAppInfoPtr appInfoPP,
    Ptr prevGlobalsP,
    Ptr globalsP
);
```

WARNING: System Use Only!

SysBatteryDialog

```
void SysBatteryDialog (
    void
);
```

WARNING: System Use Only!

SysCardImageDeleted

```
void SysCardImageDeleted (
    UInt cardNo
);
```

WARNING: System Use Only!

SysCardImageInfo

```
Ptr SysCardImageInfo (
    UInt cardNo,
    ULongPtr sizeP
);
```

WARNING: System Use Only!

SysColdBoot

Perform a cold boot and reformat all RAM areas of both memory cards.

WARNING: System Use Only!

SysCurAppInfoP

```
SysCurAppInfoPtr SysCurrAppInfoP (
    void
);
```

WARNING: System Use Only!

SysDisableInts

```
Word SysDisableInts (
    void
);
```

WARNING: System Use Only!

SysDoze

```
void SysDoze (
    Boolean onlyNMI
);
```

WARNING: System Use Only!

SysGetTrapAddress

```
VoidPtr SysGetTrapAddress (
    UInt trapNum
);
```

WARNING: System Use Only!

SysInit

```
void SysInit (
    void
);
```

WARNING: System Use Only!

SysKernelInfo

```
Err SysKernelInfo (
    VoidPtr paramP
);
```

WARNING: System Use Only!

SysLaunchConsole

```
Err SysLaunchConsole (
    void
);
```

WARNING: System Use Only!

SysLibFind

```
Err SysLibFind (
    CharPtr nameP,
    UIntPtr refNumP
);
```

WARNING: System Use Only!

SysLibInstall

```
Err SysLibInstall (
    SysLibEntryProcPtr libraryP,
    UIntPtr refNumP
);
```

WARNING: System Use Only!

SysLibRemove

```
Err SysLibRemove (
    UInt refNum
);
```

WARNING: System Use Only!

SysLibTblEntry

```
SysLibTblEntryPtr SysLibTblEntry (
    UInt refNum
);
```

WARNING: System Use Only!

SysNewOwnerID

```
UInt SysNewOwnerID (
    void
);
```

WARNING: System Use Only!

SysPowerOn

```
void SysPowerOn (
    Ptr card0P,
    ULong card0Size,
    Ptr card1P,
    ULong card1Size,
    DWord sysCardHeaderOffset,
    Boolean reFormat
);
```

WARNING: System Use Only!

SysRestoreStatus

```
void SysRestoreStatus (
    Word status
);
```

WARNING: System Use Only!

SysSetA5

```
DWord SysSetA5 (
    DWord newValue
);
```

WARNING: System Use Only!

SysSetTrapAddress

```
Err SysSetTrapAddress (
    UInt trapNum,
    VoidPtr procP
);
```

WARNING: System Use Only!

SysSleep

```
void SysSleep (
    Boolean untilReset,
    Boolean emergency
);
```

WARNING: System Use Only!

SysUILaunch

```
void SysUILaunch (
    void
);
```

WARNING: System Use Only!

ErrDisplay

Display an error alert if error checking is set to partial or full.

```
void ErrDisplay (
    char* message
);
```

Parameters

message
    Error message text.

Return Value

No return value

Remarks

Call this routine to display an error message, source code filename, and line number. This routine is actually a macro that is compiled into the code only if the compiler define ERROR_CHECK_LEVEL is set to 1 or 2 (ERROR_CHECK_PARTIAL or ERROR_CHECK_FULL).

See Also

ErrFatalDisplayIf, ErrNonFatalDisplayIf, "Using the Error Manager"

ErrDisplayFileLineMsg

Display a nonexitable dialog with an error message. Do not allow the user to continue.

```
void ErrDisplayFileLineMsg(
    CharPtr filename,
    UInt lineno,
    CharPtr msg
);
```

Parameters

filename
    Source code filename.

lineno
    Line number in the source code file.

msg
    Message to display.

Return Value

Never returns.

Remarks

Called by ErrFatalDisplayIf and ErrNonFatalDisplayIf.
This function is useful when the application is already on the device and being tested by users.

See Also

ErrFatalDisplayIf, ErrNonFatalDisplayIf, ErrDisplay

ErrFatalDisplayIf

Display an error alert dialog if condition is TRUE and error checking is set to partial or full.

```
void ErrFatalDisplayIf (
    Boolean condition,
    char* message
);
```

Parameters

condition
    If TRUE, display the error.

message
    Error message text.

Return Value

No return value

Remarks

Call this routine to display a fatal error message, source code filename, and line number. The alert is displayed only if condition is true. The dialog is cleared only when the user resets the system by responding to the dialog.
This routine is actually a macro that is compiled into the code if the compiler define ERROR_CHECK_LEVEL is set to 1 or 2 (ERROR_CHECK_PARTIAL or ERROR_CHECK_FULL).

See Also

ErrNonFatalDisplayIf, ErrDisplay, "Using the Error Manager"

ErrNonFatalDisplayIf

Display an error alert dialog if condition is TRUE and error checking is set to full.

```
void ErrNonFatalDisplayIf (
    Boolean condition,
    char* message
);
```

Parameters

condition
    If TRUE, display the error.

message
    Error message text.

Return Value

No return value.

Remarks

Call this routine to display a nonfatal error message, source code filename, and line number. The alert is displayed only if condition is true. The alert dialog is cleared when the user selects to continue (or resets the system).
This routine is actually a macro that is compiled into the code only if the compiler define ERROR_CHECK_LEVEL is set to 2 (ERROR_CHECK_FULL).

See Also

ErrFatalDisplayIf, ErrDisplay, "Using the Error Manager"

ErrThrow

Cause a jump to the nearest Catch block.

```
void ErrThrow (
    Long err
);
```

Parameters

err
    Error code.

Return Value

Never returns.

Remarks

Use the macros ErrTry, ErrCatch, and ErrEndCatch in conjunction with this function.

See Also

ErrFatalDisplayIf, ErrNonFatalDisplayIf, ErrDisplay,
"Using the Error Manager"

PrefGetAppPreferences

Return a copy of an application's preferences.

```
Boolean PrefGetAppPreferences (
    ULong type,
    Int version,
    VoidPtr prefs,
    Word prefsSize
);
```

Parameters

type
    Application creator type.

version
    Version number of the application.

prefs
    Pointer to a buffer to hold preferences.

prefsSize
    Size of the buffer passed.

Return Value

Returns FALSE if the preference resource was not found or the preference resource contains the wrong version number.

Remarks

The content and format of an application preference is application-dependent.

See Also

PrefSetPreferences

PrefGetPreferences

Return a copy of the system preferences.

```
void PrefGetPreferences (
    SystemPreferencesPtr p
);
```

Parameters

p
    Pointer to system preferences.

Return Value

Returns nothing. Stores the system preferences in p.

Remarks

The p parameter points to a memory block allocated by the caller that is filled in by this function. This function is often called in StartApplication to get localized settings.

See Also

PrefSetPreferences

PrefOpenPreferenceDB

Return a handle to the system preference database.

```
DmOpenRef PrefOpenPreferenceDB (
    void
);
```

Parameters

Nothing.

Return Value

Returns the handle, or 0 if an error results.

See Also

PrefGetPreferences, PrefSetPreferences

PrefSetAppPreferences

Save an application's preferences in the preferences database.

```
void PrefSetAppPreferences (
    ULong type,
    Int version,
    VoidPtr prefs,
    Word prefsSize
);
```

Parameters

type
    Application creator type.

version
    Version number of the application.

prefs
    Pointer to a buffer holding preferences.

prefsSize
    Size of the buffer passed.

Return Value

Nothing.

Remarks

The content and format of an application preference is application-dependent.

See Also

PrefGetPreferences

PrefSetPreferences

Set the system preferences.

```
void PrefSetPreferences (
    SystemPreferencesPtr p
);
```

Parameters

p
    Pointer to system preferences.

Return Value

Returns nothing.

See Also

PrefGetPreferences

FindDrawHeader

Draw the header line that separates, by database, the list of found items.

```
Boolean FindDrawHeader (
    FindParamsPtr params,
    CharPtr title
);
```

Parameters

params
    Handle of FindParamsPtr.

title
    Description of the database (for example Memos)

Return Value

Returns TRUE if Find screen is filled up. Applications should exit from the search if this occurs.

FindGetLineBounds

Returns the bounds of the next available line for displaying a match in the Find Results dialog.

```
void FindGetLineBounds (
    FindParamsPtr params,
    RectanglePtr r
);
```

Parameters

params
    Handle of FindParamsPtr.

r
    Pointer to a structure to hold the bounds of the next results line.

Return Value

Returns nothing.

FindSaveMatch

Saves the record and position within the record of a text search match. This information is saved so that it's possible to later navigate to the match.

```
void FindSaveMatch (
    FindParamsPtr params,
    UInt recordNum,
    Word pos,
    UInt fieldNum,
    DWord appCustom,
    UInt dbCardNo,
    LocalID rdbID
);
```

Parameters

params
    Handle of FindParamsPtr.

recordNum
    Record index.

pos
    Offset of the match string from start of record.

appCustom
    Extra data the application can save with a match.

dbCardNo
    Car number of the database that contains the match.

dbID
    Local ID of the database that contains the match.

Return Value

Returns TRUE if the maximum number of displayable items has been exceeded

Remarks

Called by application code when it gets a match.

FindStrInStr

Perform a case-blind partial word search for a string in another string. This function assumes that the string to find is in lowercase characters.

```
void FindStrInStr (
    CharPtr strToSearch,
    CharPtr strToFind,
    WordPtr posP
);
```

Parameters

strToSearch
    String to search.

strToFind
    String to find.

posP
    Pointer to the offset in the search string of the match.

Return Value

Returns TRUE if the string was found.

EvtAddEventToQueue

Add an event to the event queue.

```
void EvtAddEventToQueue (
    EventPtr event
);
```

Parameters

event
    Pointer to the structure that contains the event.

error
    Pointer to any error encountered by this function.

Return Value

Returns nothing.

EvtCopyEvent

Copy an event.

```
void EvtCopyEvent (
    EventPtr source,
    EventPtr dest
);
```

Parameters

source
    Pointer to the structure containing the event to copy.

dest
    Pointer to the structure to copy the event to.

Return Value

Returns nothing.

EvtDequeuePenPoint

Get the next pen point out of the pen queue (called by the recognizers).

```
Err EvtDequeuePenPoint(
    PointType* retP
);
```

Parameters

retP
    Return point.

Return Value

Always returns 0.

Remarks

Called by a recognizer that wishes to extract the points of a stroke.
Returns the point (-1, -1) at the end of a stroke.
Before calling this routine, you must call EvtDequeuePenStrokeInfo.

See Also

EvtDequeuePenStrokeInfo

EvtDequeuePenStrokeInfo

Initiate the extraction of a stroke from the pen queue.

```
Err EvtDequeuePenStrokeInfo(
    PointType* startPtP,
    PointType* endPtP
);
```

Parameters

startPtP
    Start point returned here.

startPtP
    End point returned here.

Return Value

Always returns 0.

Remarks

Called by the system function EvtGetSysEvent. This routine must be called before EvtDequeuePenPoint is called.
Subsequent calls to EvtDequeuePenPoint return points at the starting point in the stroke and including the end point. After the end point is returned, the next call to EvtDequeuePenPoint returns the point -1, -1.

See Also

EvtDequeuePenPoint

EvtEnableGraffiti

Set Graffiti enabled or disabled.

```
void EvtEnableGraffiti (
    Boolean enable
);
```

Parameters

enable
    TRUE to enable Graffiti, FALSE to disable Graffiti.

Return Value

Returns nothing.

EvtEnqueueKey

Place keys into the key queue.

```
Err EvtEnqueueKey (
    UInt ascii,
    UInt keycode,
    UInt modifiers
);
```

Parameters

ascii
    ascii code of key.

keycode
    Virtual key code of key.

modifiers
    Modifiers for key event.

Return Value

Returns 0 if successful, or evtErrParamErr if an error occurs.

Remarks

Called by the keyboard interrupt routine and the Graffiti and Soft-Keys recognizers. Note that because both interrupt- and noninterrupt-level code can post keys into the queue, this routine disables interrupts while the queue header is being modified.
Most keys in the queue take only 1 byte if they have no modifiers and no virtual key code, and are 8-bit ASCII. If a key event in the queue has modifiers or is a non-standard ascii code, it takes up to 7 bytes of storage and has the following format:
evtKeyStringEscape 1 byte
ASCII code 2 bytes
virtual key code 2 bytes
modifiers 2 bytes

EvtFlushKeyQueue

Flush all keys out of the key queue.

```
Err EvtFlushKeyQueue (
    void
);
```

Parameters

None.

Return Value

Always returns 0.

Remarks

Called by the system function EvtSetPenQueuePtr.

EvtFlushNextPenStroke

Flush the next stroke out of the pen queue.

```
Err EvtFlushNextPenStroke (
    void
);
```

Parameters

None

Return Value

Always returns 0.

Remarks

Called by recognizers that need only the start and end points of a stroke. If a stroke has already been partially dequeued (by EvtDequeuePenStrokeInfo) this routine finishes the stroke de-queueing. Otherwise, this routine flushes the next stroke in the queue.

See Also

EvtDequeuePenPoint

EvtFlushPenQueue

Flush all points out of the pen queue.

```
Err EvtFlushPenQueue (
    void
);
```

Parameters

None

Return Value

Always returns 0.

Remarks

Called by the system function EvtSetKeyQueuePtr.

See Also

EvtPenQueueSize

EvtGetEvent

Return the next available event.

```
void EvtGetEvent (
    EventPtr event,
    Long timeout
);
```

Parameters

event
    Pointer to the structure to hold the event returned.

timeout
    Max amount of ticks to wait before an event is returned (-1 means wait indefinitely).

Remarks

Pass timeout= -1 in most instances. When running on the device, this makes the CPU go into doze mode until the user provides input. For applications that do animation, pass timeout >= 0.

Return Value

Returns nothing.

EvtGetPen

Return the current status of the pen.

```
void EvtGetPen(
    Sword *pScreenX,
    Sword *pScreenY,
    Boolean *pPenDown
);
```

Parameters

pScreenX
    x location relative to display.

pScreenY
    y location relative to display.

pPenDown
    TRUE or FALSE.

Return Value

Returns nothing.

Remarks

Called by various UI routines.

See Also

KeyCurrentState

EvtGetPenBtnList

Return a pointer to the silk-screen button array.

```
PenBtnInfoPtr asm EvtGetPenBtnList(
    UIntPtr numButtons
);
```

Parameters

numButtons
    Pointer to the variable to contain the number of buttons in the array.

Return Value

Returns a pointer to the array.

Remarks

The array returned contains the bounds of each silk-screened button and the ASCII code and modifiers byte to generate for each button.

See Also

EvtProcessSoftKeyStroke

EvtKeyQueueEmpty

Return TRUE if the key queue is currently empty.

```
Boolean EvtKeyQueueEmpty (
    void
);
```

Parameters

None.

Return Value

Returns TRUE if the key queue is currently empty, otherwise returns FALSE.

Remarks

Called by key manager to determine if it should enqueue auto-repeat keys.

EvtKeyQueueSize

Return the size of the current key queue in bytes.

```
ULong EvtKeyQueueSize (
    void
);
```

Parameters

None.

Return Value

Returns size of queue in bytes.

Remarks

Called by applications that wish to see how large the current key queue is.

EvtPenQueueSize

Return the size of the current pen queue in bytes.

```
ULong EvtPenQueueSize (
    void
);
```

Parameters

None.

Return Value

Returns size of queue in bytes.

Remarks

Call this function to see how large the current pen queue is.

EvtKeyQueueEmpty

Return TRUE if the key queue is currently empty.

```
Boolean EvtKeyQueueEmpty (
    void
);
```

Parameters

None.

Return Value

Returns TRUE if the key queue is currently empty, otherwise returns FALSE.

Remarks

Called by key manager to determine if it should enqueue auto-repeat keys.

EvtKeyQueueSize

Return the size of the current key queue in bytes.

```
ULong EvtKeyQueueSize (
    void
);
```

Parameters

None.

Return Value

Returns size of queue in bytes.

Remarks

Called by applications that wish to see how large the current key queue is.

EvtPenQueueSize

Return the size of the current pen queue in bytes.

```
ULong EvtPenQueueSize (
    void
);
```

Parameters

None.

Return Value

Returns size of queue in bytes.

Remarks

Call this function to see how large the current pen queue is.

EvtProcessSoftKeyStroke

Translate a stroke in the system area of the digitizer and enqueue the appropriate key events in to the key queue.

Err EvtProcessSoftKeyStroke(PointType* startPtP, PointType* endPtP)

Parameters

startPtP
    Start point of stroke.

endPtP
    End point of stroke.

Return Value

Returns 0 if recognized, -1 if not recognized.

See Also

EvtGetPenBtnList, GrfProcessStroke

EvtResetAutoOffTimer

Reset the auto-off timer to assure that the device doesn't automatically power off during a long operation without user input (for example, serial port activity).

```
Err EvtResetAutoOffTimer (
    void
);
```

Parameters

None.

Return Value

Always returns 0.

Remarks

Called by SerialLinkMgr, Can be called periodically by other managers.

See Also

SysSetAutoOffTime

EvtWakeup

Force the event manager to wake up and send a nilEvent to the current application.

```
Err EvtWakeup (
    void
);
```

Parameters

None.

Return Value

Always returns 0.

Remarks

Called by interrupt routines, like the sound manager and alarm manager.

EvtDequeueKeyEvent

```
Err EvtDequeueKeyEvent (
    EventPtr eventP
);
```

WARNING: System Use Only!

EvtEnqueuePenPoint

```
Err EvtEnqueuePenPoint (
    PointType* ptP
);
```

WARNING: System Use Only!

EvtGetSysEvent

```
void EvtGetSysEvent (
    EventPtr eventP,
    Long timeout
);
```

WARNING: System Use Only!

EvtInitialize

```
void EvtInitialize (
    void
);
```

WARNING: System Use Only!

EvtSetKeyQueuePtr

```
Err EvtSetKeyQueuePtr (
    Ptr keyQueueP,
    ULong size
);
```

WARNING: System Use Only!

EvtSetPenQueuePtr

```
Err EvtSetPenQueuePtr (
    Ptr penQueueP,
    ULong size
);
```

WARNING: System Use Only!

EvtSysInit

```
Err EvtSysInit (
    void
);
```

WARNING: System Use Only!

FtrGet

Get a feature.

```
Err FtrGet (
    DWord creator,
    UInt featureNum,
    DWordPtr valueP
);
```

Parameters

creator
    Creator type, should be same as the application that owns this feature.

featureNum
    Feature number of the feature.

valueP
    Value of the feature is returned here.

Return Value

Returns 0 if no error, or ftrErrNoSuchFtr or ftrErrInternalError if an error occurs.

Remarks

The value of the feature is application-dependent.

See Also

FtrSet

FtrGetByIndex

Get a feature by index.
Until the caller gets back ftrErrNoSuchFeature, it should pass indices for each table (ROM, RAM) starting at 0 and incrementing.

```
Err FtrGetByIndex (
    UInt index,
    Boolean romTable,
    DWordPtr creatorP,
    UIntPtr numP,
    DWordPtr valueP
);
```

Parameters

index
    Index of feature.

romTable
    If TRUE, index into ROM table; otherwise, index into RAM table.

creatorP
    Feature creator is returned here.

numP
    Feature number is returned here.

valueP
    Feature value is returned here.

Return Value

Returns 0 if no error, or ftrErrInternalError or ftrErrNoSuchFeature if an error occurs.

Remarks

This routine is normally only used by shell commands. Most applications do not need it.

FtrSet

Set a feature.

```
Err FtrSet (
    DWord creator,
    UInt featureNum,
    DWord newValue
);
```

Parameters

creator
    Creator type, should be same as the application that owns this feature.

featureNum
    Feature number of the feature.

newValue
    New value.

Return Value

Returns 0 if no error, or ftrErrNoSuchFeature, memErrChunkLocked, memErrInvalidParam, or memErrNotEnoughSpace if an error occurs.

Remarks

The value of the feature is application-dependent.

See Also

FtrGet

FtrUnregister

Unregister a feature.

```
Err FtrUnregister (
    DWord creator,
    UInt featureNum
);
```

Parameters

creator
    Creator type, should be same as the application that owns the creator.

featureNum
    Feature number of the feature.

Return Value

Returns 0 if no error, or ftrInternalError, ftrErrNoSuchFeature, memErrChunkLocked, memErrInvalidParam, or memErrNotEnoughSpace if an error occurs.

FtrInit

```
Err FtrInit (
    void
);
```

WARNING: This function for System use only

StrAToI

Converts a string to an integer.

```
Int StrAToI (
    CharPtr str
);
```

Parameters

str
    String to convert.

Return Value

Returns the integer.

Remarks

Use this function instead of the standard atoi routine.

StrCat

Concatenate one string to another.

```
CharPtr StrCat (
    CharPtr dst,
    CharPtr src
);
```

Parameters

Two string pointers.

Return Value

Returns a pointer to the destination string.

Remarks

Use this function instead of the standard strcat routine.

StrCaselessCompare

Compare two strings with case and accent insensitivity.

```
Int StrCaselessCompare (
    CharPtr s1,
    CharPtr s2
);
```

Parameters

Two string pointers.

Return Value

Returns 0 if the two strings match, or non-zero if they don't.

Remarks

Use this function instead of the standard stricmp routine. Use it to find strings but not sort them because it ignores case and accents.

See Also

StrCompare

StrChr

Look for a character within a string.

```
CharPtr StrChr (
    CharPtr str,
    Int chr
);
```

Parameters

str
    String to search.

chr
    Character to search for.

Return Value

Returns a pointer to the first occurrence of character in str, or NULL if not found.

Remarks

Use this function instead of the standard strchr routine. This routine does not correctly find a '\0' character.

See Also

StrStr

StrCompare

Compare two strings.

```
Int StrCompare (
    CharPtr s1,
    CharPtr s2
);
```

Parameters

Two string pointers.

Return Value

Returns 0 if the strings match.
Returns a positive number if s1 > s2.
Returns a negative number if s1 < s2.

Remarks

This function is case sensitive. Use it to sort strings but not to find them.
Use this function instead of the standard strcmp routine.

See Also

StrCaselessCompare

StrCopy

Copy one string to another.

```
CharPtr StrCopy (
    CharPtr dst,
    CharPtr src
);
```

Parameters

Two string pointers.

Return Value

Returns a pointer to the destination string.

Remarks

Use this function instead of the standard strcpy routine.
This function does not return overlapping strings.

StrIToA

Convert an integer to ASCII.

```
CharPtr StrIToA (
    CharPtr s,
    Long i
);
```

Parameters

s
    String pointer to store results.

i
    Integer to convert.

Return Value

Returns a pointer to the result string.

See Also

StrAToI, StrIToH

StrIToH

Convert an integer to hexadecimal ASCII.

```
CharPtr StrIToH (
    CharPtr s,
    ULong i
);
```

Parameters

s
: String pointer to store results.

i
: Integer to convert.

Return Value

Returns the string pointer s.

See Also

StrIToA

StrLen

Compute the length of a string.

```
UInt StrLen (
    CharPtr src
);
```

Parameters

src
    String pointer

Return Value

Returns the length of the string.

Remarks

Use this function instead of the standard strlen routine.

StrStr

Look for a substring within a string.

```
CharPtr StrStr (
    CharPtr str,
    CharPtr token
);
```

Parameters

str
    String to search.

token
    String to search for.

Return Value

Returns a pointer to the first occurrence of token in str, or NULL if not found.

Remarks

Use this function instead of the standard strstr routine.

See Also

StrChr

StrToLower

Convert all the characters in a string to lowercase.

```
CharPtr StrToLower (
    CharPtr dst,
    CharPtr src
);
```

Parameters

Two string pointers.

Return Value

Returns a pointer to the destination string.

Remarks

This function doesn't convert accented characters.

DateAdjust

Return a new date +/- the days adjustment.

```
void DateAdjust (
    DatePtr dateP,
    Long adjustment
);
```

Parameters

dateP
    A DateType structure with the date to be adjusted (see DateTime.h).

adjustment
    The adjustment in seconds.

Return Value

Changes dateP to contain the new date.

Remarks

This function is useful for advancing a day or week and not worrying about month and year wrapping. If the time is advanced out of bounds, it is cut at the bounds surpassed.

DateDaysToDate

Return the date, given days.

```
void DateDaysToDate (
    ULong days,
    DatePtr dateP
);
```

Parameters

days
    Days since 1/1/1904.

dateP
    Pointer to DateType structure (returned).

Return Value

Returns nothing, stores the date in dateP.

See Also

TimAdjust, DateToDays

DateSecondsToDate

Return the date given seconds.

```
void DateSecondsToDate (
    ULong seconds,
    DatePtr dateP
);
```

Parameters

seconds
    Seconds since 1/1/1904.

dateP
    Pointer to DateType structure (returned).

Return Value

Returns nothing; stores the date in dateP.

DateToAscii

Convert the time passed to an ASCII string in the passed DateFormatType.

```
void DateToAscii(
    Byte months,
    Byte days,
    Word years,
    DateFormatType dateFormat,
    CharPtr pString
);
```

NOTE: Handles the long and short forms of the date formats.

Parameters

months
    Months (1-12).

days
    Days (1-31).

years
    Years (for example 1995).

dateFormat
    Long or short DateFormatType.

pString
    Pointer to string which gets the result. Must be of length dateStringLength for standard formats or longDateStrLength for long date formats.

Return Value

Returns nothing; stores the result in pString.

See Also

TimeToAscii, DateToDOWDMFormat

DateToDays

Return the date in days since 1/1/1904.

```
ULong DateToDays (
    DateType date
);
```

Parameters

date
    DateType structure.

Return Value

Returns the days since 1/1/1904.

See Also

TimAdjust, DateDaysToDate

DateToDOWDMFormat

Convert the date passed to an ASCII string.

```
void DateToDOWDMFormat(
    Byte months,
    Byte days,
    Word years,
    DateFormatType dateFormat,
    CharPtr pString
);
```

Parameters

months
    Month (1-12).

days
    Day (1-31).

years
    Years (for example 1995).

dateFormat
    False to use AM and PM.

pString
    Pointer to string which gets the result. The string must be of length timeStringLength.

Return Value

Returns nothing; stores ASCII string in pString.

See Also

DateToAscii

DayOfMonth

Return the day of a month on which the specified date occurs (for example, dom2ndTue).

```
UInt DayOfMonth (
    UInt month,
    UInt day,
    UInt year
);
```

Parameters

month
    Month (1-12).

day
    Day (1-31).

year
    Year (ex: 1995).

Return Value

Returns the day of the month as a DayOfWeekType, see DateTime.h.

DayOfWeek

Return the day of the week.

```
UInt DayOfWeek (
    UInt month,
    UInt day,
    UInt year
);
```

Parameters

month
    Month (1-12).

day
    Day (1-31).

year
    Year (ex: 1995).

Return Value

Returns the day of the week (Sunday = 0, Monday = 1, etc.).

DaysInMonth

Return the number of days in the month.

```
UInt DaysInMonth (
    UInt month,
    UInt year
);
```

Parameters

month
    Month (1-12).

year
    Year (for example, 1995).

Return Value

Returns the number of days in the month for that year.

SelectDay

Display a form showing a date and allow the user to select a different date.

```
Boolean SelectDay (
    int *month,
    int *day,
    int *year,
    CharPtr title
);
```

Parameters

month
    Month selected.

day
    Day selected.

year
    Year selected.

title
    String title for the dialog.

Return Value

Returns true if the OK button was pressed. In that case, the parameters passed are changed.

TimAdjust

Return a new date, +/- the time adjustment.

```
void TimAdjust(
    DateTimePtr dateTimeP,
    Long adjustment
);
```

Parameters

dateTimeP
    A DateType structure (see DateTime.h).

adjustment
    The adjustment in seconds.

Return Value

Returns nothing. Changes dateTimeP to the new date and time.

Remarks

This function is useful for advancing a day or week and not worrying about month and year wrapping. If the time is advanced out of bounds it is cut at the bounds surpassed.

See Also

DateAdjust

TimDateTimeToSeconds

Return the date and time in seconds since 1/1/1904.

```
ULong TimDateTimeToSeconds (
    DateTimePtr dateTimeP
);
```

Parameters

dateTimeP
    A DateType structure (see DateTime.h).

Return Value

The time in seconds since 1/1/1904.

See Also

TimSecondsToDateTime

TimGetSeconds

Return seconds since 1/1/1904.

```
ULong TimGetSeconds (
    void
);
```

Parameters

None.

Return Value

Returns the number of seconds.

See Also

TimSetSeconds

TimGetTicks

Return the tick count since the last reset.

```
ULong TimGetTicks (
    void
);
```

Parameters

None.

Return Value

Returns the tick count.

Remarks

The tick count does not advance while the device is in sleep mode.

TimSecondsToDateTime

Return the date and time, given seconds.

```
void TimSecondsToDateTime(
    ULong seconds,
    DateTimePtr dateTimeP
);
```

Parameters

seconds
    Seconds to advance from 1/1/1904.

dateTimeP
    A DateTimeType structure that's filled by the function.

Return Value

Returns nothing. Stores the date and time given seconds since 1/1/1904 in dateTimeP.

See Also

TimDateTimeToSeconds

TimSetSeconds

Return seconds since 1/1/1904.

```
void TimSetSeconds (
    ULong seconds
);
```

Parameters

seconds
    Place to return the seconds since 1/1/1904.

Return Value

Returns nothing; modifies seconds.

See Also

TimGetSeconds

TimeToAscii

Convert the time passed to an ASCII string.

```
void TimeToAscii(
    Byte hours,
    Byte minutes,
    TimeFormatType timeFormat,
    CharPtr pString
);
```

Parameters

hours
    Hours (0-23).

minutes
    Minutes (0-59).

timeFormat
    False to use AM and PM.

pString
    Pointer to string which gets the result. Must be of length timeStringLength.

Return Value

Returns nothing. Stores pointer to the text of the current selection in pString.

See Also

DateToAscii

TimGetAlarm

```
ULong TimGetAlarm (
    void
);
```

WARNING: System use only!

TimHandleInterrupt

```
void TimHandleInterrupt (
    Boolean periodicUpdate
);
```

Warning: System use only!

TimInit

```
Err TimInit (
    void
);
```

Warning: System use only!

TimSetAlarm

```
ULong TimSetAlarm (
    ULong alarmSeconds
);
```

Warning: System use only!

FplAdd

Add two floating-point numbers (returns a + b).

```
FloatType FplAdd (
    FloatType a,
    FloatType b
);
```

Parameters

a, b The floating-point numbers.

Return Value

Returns the normalized floating-point result of the addition.

FplAToF

Convert a zero-terminated ASCII string to a floating-point number.
The string must be in the format : [-]x[.]yyyyyyyy[e[-]zz]

```
FloatType FplAToF (
    char* s
);
```

Parameters

s
    Pointer to the ASCII string.

Return Value

Returns the floating-point number.

See Also

FplFToA

FplBase10Info

Extract detailed information on the base 10 form of a floating-point number: the base 10 mantissa, exponent, and sign.

```
Err FplBase10Info (
    FloatType a,
    ULong* mantissaP,
    Int* exponentP,
    Int* signP
);
```

Parameters

a
    The floating-point number.

mantissaP
    The base 10 mantissa (return value).

exponentP
    The base 10 exponent (return value).

signP
    The sign, 1 or -1 (return value).

Return Value

Returns an error code, or 0 if no error.

Remarks

The mantissa is normalized so it contains at least kMaxSignificantDigits significant digits when printed as an integer value.

FplDiv

Divide two floating-point numbers (result = dividend/divisor).

```
FloatType FplDiv (
    FloatType dividend,
    FloatType divisor
);
```

Parameters

dividend
    Floating-point dividend.

divisor
    Floating-point divisor.

Return Value

Returns the normalized floating-point result of the division.

FplFloatToLong

Convert a floating-point number to a long integer.

```
Long FplFloatToLong (
    FloatType f
);
```

Parameters

f
    Floating-point number to be converted.

Return Value

Returns the long integer.

See Also

FplLongToFloat, FplFloatToULong

FplFloatToULong

Convert a floating-point number to an unsigned long integer.

```
ULong FplFloatToULong (
    FloatType f
);
```

Parameters

f
    Floating-point number to be converted.

Return Value

Returns an unsigned long integer.

See Also

FplLongToFloat, FplFloatToLong

FplFree

Release all memory allocated by the floating-point initialization.

```
void FplFree(
    void
);
```

Parameters

None.

Return Value

Returns nothing.

Remarks

Applications must call this routine after they've called other functions that are part of the float manager.

See Also

FplInit

FplFToA

Convert a floating-point number to a zero-terminated ASCII string in exponential format : [-]x.yyyyyyyye[-]zz

```
Err FplFToA (
    FloatType a,
    char* s
);
```

Parameters

a
    The floating-point number.

s
    Pointer to buffer to contain the ASCII string.

Return Value

Returns an error code, or 0 if no error.

See Also

FplAToF

FplInit

Initialize the floating-point conversion routines.
Allocate space in the system heap for fpl globals.
Initialize the tenPowers array in the globals area to the powers of 10 from -99 to +99 in floating-point format.

```
Err FplInit(
    void
);
```

Parameters

None.

Return Value

Returns an error code, or 0 if no error.

Remarks

Applications must call this routine before calling any other fpl function.

See Also

FplFree

FplLongToFloat

Convert a long integer to a floating-point number.

```
FloatType FplLongToFloat (
    Long x
);
```

Parameters

x
    A long integer.

Return Value

Returns the floating-point number.

FplMul

Multiply two floating-point numbers.

```
FloatType FplMul(
    FloatType a,
    FloatType b
);
```

Parameters

a, b The floating-point numbers.

Return Value

Returns the normalized floating-point result of the multiplication.

FplSub

Subtract two floating-point numbers (returns a - b).

```
FloatType FplSub (
    FloatType a,
    FloatType b
);
```

Parameters

a, b The floating-point numbers.

Return Value

Returns the normalized floating-point result of the subtraction.

AlmGetAlarm

Return the alarm date/time in seconds since 1/1/1904 and the caller-defined alarm reference value for the given application.

```
ULong AlmGetAlarm (
    UInt cardNo,
    LocalID dbID,
    DWordPtr refP
);
```

Parameters

cardNo
    Storage card number of the application.

dbID
    Local ID of the application.

refP
    Pointer to location for the alarm's reference value.

Return Value

Alarm seconds since 1/1/1904; if no alarm is active for the application, 0 is returned for the alarm seconds and the reference value is undefined.

AlmSetAlarm

Set or cancel an alarm for the given application.

```
Err AlmSetAlarm (
    UInt cardNo,
    LocalID dbID,
    DWord ref,
    ULong alarmSeconds,
    Boolean quiet
);
```

Parameters

cardNo
    Storage card number of the application.

dbID
    Local ID of the application.

ref
    Caller-defined value to be passed with notifications.

alarmSeconds
    Alarm date/time in seconds since 1/1/1904, or 0 to cancel the current alarm (if any).

quiet
    Reserved for future upgrade (set to zero).

Return Value

0 No error.
almErrMemory Insufficient memory.
almErrFull Alarm table is full.

Remarks

If an alarm for this application has already been set, it is replaced with the new alarm. Action code notifications are sent after the alarm is triggered and can be used by the application to set the next alarm.

AlmAlarmCallback

```
void AlmAlarmCallback (
    void
);
```

WARNING: This function for use by system software only.

AlmCancelAll

```
void AlmCancelAll (
    Boolean enable
);
```

WARNING: This function for use by system software only.

AlmDisplayAlarm

```
void AlmDisplayAlarm (
    Boolean displayOnly
);
```

WARNING: This function for use by system software only.

AlmEnableNotification

```
void AlmEnableNotificatio(
    Boolean enable
);
```

WARNING: This function for use by system software only.

AlmInit

```
Err AlmInit (
    void
);
```

WARNING: This function for use by system software only.

SndDoCmd

Send a sound manager command to a specified sound channel.
NOTE: Passing NIL for the channel pointer causes the command to be sent to the shared sound channel.

```
Err SndDoCmd (
    VoidPtr chanP,
    SndCommandPtr cmdP,
    Boolean noWait
);
```

Parameters

chanP
    Pointer to sound channel. Present implementation doesn't support multiple channels. Must be zero.

cmdP
    Pointer to a SndCommandType structure which contains command parameters.

noWait
    0 = await completion
    !0 = immediate return (asynchronous) asynchronous mode is not presently supported

Return Value

0 No error.
sndErrBadParam Invalid parameter.
sndErrBadChannel Invalid channel pointer.
sndErrQFull Sound queue is full.

SndGetDefaultVolume

Return default sound volume levels.

```
void SndGetDefaultVolume (
    UIntPtr alarmAmpP,
    UIntPtr sysAmpP,
    UIntPtr defAmpP
);
```

Parameters

alarmAmpP
    Pointer to storage for alarm amplitude.

sysAmpP
    Pointer to storage for system sound amplitude.

defAmpP
    Pointer to storage for master amplitude.

Return Value

Returns nothing.

Remarks

Any pointer arguments may be passed as NULL. In that case, the corresponding setting is not returned.

SndPlaySystemSound

Play a standard system sound.

```
void SndPlaySystemSound (
    SndSysBeepType beepID
);
```

Parameters

beepID
    ID of system sound to play.

Return Value

Returns nothing.

SndSetDefaultVolume

Set the default sound volume levels.

```
void SndSetDefaultVolume (
    UIntPtr alarmAmpP,
    UIntPtr sysAmpP,
    UIntPtr defAmpP
);
```

Parameters

alarmAmpP
    Pointer to alarm amplitude (0-sndMaxAmp).

sysAmpP
    Pointer to system sound amplitude (0-sndMaxAmp).

defAmpP
    Pointer to master amplitude (0-sndMaxAmp).

Return Value

Returns nothing.

Remarks

Any pointer arguments may be passed as NULL. In that case, the corresponding setting are not affected.

SndInit

```
Err SndInit(
    void
);
```

WARNING: This function for use by system software only.

PenCalibrate

Set the calibration of the pen.

```
Err PenCalibrate (
    PointType* digTopLeftP,
    PointType* digBotRightP,
    PointType* scrTopLeftP,
    PointType* scrBotRightP
);
```

Parameters

digTopLeftP
    Digitizer output from top-left coordinate.

digBotRightP
    Digitizer output from bottom-right coordinate.

scrTopLeftP
    Screen coordinate near top-left corner.

scrBotRightP
    Screen coordinate near bottom-right corner.

Return Value

Returns 0 if no error.

Remarks

Called by Preferences application when calibrating pen.

See Also

PenResetCalibration

PenResetCalibration

Reset the calibration in preparation for calibrating the pen again.

```
Err PenResetCalibration (
    void
);
```

Parameters

None.

Return Value

Always returns 0.

Remarks

Called by Preferences application before capturing points when calibrating the digitizer.

See Also

PenCalibrate
WARNING: The digitizer is off after calling this routine and must be calibrated again!!!

PenClose

```
Err PenClose (
    void
);
```

WARNING: This function for use by system software only.

PenGetRawPen

```
Err PenGetRawPen (
    PointType* penP
);
```

See Instead EvtDequeuePenPoint

WARNING: This function for use by system software only.

PenOpen

```
Err PenOpen (
    void
);
```

WARNING: This function for use by system software only.

PenSleep

```
Err PenSleep (
    void
);
```

WARNING: This function for use by system software only.

PenRawToScreen

```
Err PenRawToScreen (
    PointType* penP
);
```

WARNING: This function for use by system software only.

PenScreenToRaw

```
Err PenScreenToRaw (
    PointType* penP
);
```

WARNING: This function for use by system software only.

PenWake

```
Err PenWake (
    void
);
```

WARNING: This function for use by system software only.

KeyCurrentState

Return bit field with bits set for each key that is currently depressed.

```
DWord KeyCurrentState (
    void
);
```

Parameters

void

Return Value

DWord with bits set for keys that are depressed. See keyBitPower, keyBitPageUp, keyBitPageDown, etc., in KeyMgr.h.

Remarks

Called by applications that need to poll the keys.

See Also

KeyRates

KeyRates

Get or set the key repeat rates.

```
Err KeyRates (
    Boolean set,
    WordPtr initDelayP,
    WordPtr periodP,
    WordPtr doubleTapDelayP,
    BooleanPtr queueAheadP
);
```

Parameters

set
    If TRUE, settings are changed; if FALSE, current settings are returned.

initDelayP
    Initial delay in ticks for a auto-repeat event.

periodP
    Auto-repeat rate specified as period in ticks.

doubleTapDelayP
    Max double-tap delay in ticks.

queueAheadP
    If TRUE, auto-repeating keeps queueing up key events if the queue has keys in it. If FALSE, auto-repeat does not enqueue keys unless the queue is already empty.

Return Value

Returns 0 if no error.

See Also

KeyCurrentState

KeyBootKeys

```
DWord KeyBootKeys (
    void
);
```

WARNING: This function for use by system software only.

KeyHandleInterrupt

```
ULong KeyHandleInterrupt(
    Boolean periodic,
    DWord status
);
```

WARNING: This function for use by system software only.

KeyInit

```
Err KeyInit (
    void
);
```

WARNING: This function for use by system software only.

KeyResetDoubleTap

```
Err KeyResetDoubleTap (
    void
);
```

WARNING: This function for use by system software only.

KeySleep

```
Err KeySleep (
    Boolean untilReset,
    Boolean emergency
);
```

WARNING: This function for use by system software only.

KeyWake

```
Err KeyWake (
    void
);
```

WARNING: This function for use by system software only.

GrfAddMacro

Add a macro to the macro list.

```
Err GrfAddMacro (
    CharPtr nameP,
    BytePtr macroDataP,
    Word dataLen
);
```

Parameters

nameP
    Name of macro.

macroDataP
    Data of macro.

dataLen
    Size of macro data in bytes.

Return Value

Returns 0 if no error; returns grfErrNoMacros, grfErrMacroPtrTooSmall, dmErrNotValidRecord, dmErrWriteOutOfBounds if an error occurs.

See Also

GrfGetMacro, GrfGetMacroName, GrfDeleteMacro

GrfAddPoint

Add a point to the Graffiti point buffer.

```
Err GrfAddPoint (
    PointType* ptP
);
```

Parameters

ptP
    Pointer to point.

Return Value

Returns 0 if no error; returns grfErrPointBufferFull if an error occurs.

See Also

GrfFlushPoints

GrfCleanState

Remove any temporary shifts from the dictionary state.

```
Err GrfCleanState (
    void
);
```

Parameters

None

Return Value

Returns 0 if no error, or grfErrNoDictionary if an error occurs.

See Also

GrfInitState

GrfDeleteMacro

Delete a macro from the macro list.

```
Err GrfDeleteMacro (
    Word index
);
```

Parameters

index
    Which macro to delete.

Return Value

Returns 0 if no error, or grfErrNoMacros, grfErrMacroNotFound if an error occurs.

See Also

GrfAddMacro

GrfFindBranch

Locate a branch in the Graffiti dictionary by flags.

```
Err GrfFindBranch (
    Word flags
);
```

Parameters

flags
    Flags of the branch we're searching for.

Return Value

Returns 0 if no error, or grfErrNoDictionary or grfErrBranchNotFound if an error occurs.

See Also

GrfCleanState, GrfInitState

GrfFilterPoints

Filter the points in the Graffiti point buffer.

```
Err GrfFilterPoints (
    void
);
```

Parameters

None.

Return Value

Always returns 0.

See Also

GrfMatch

GrfFlushPoints

Dispose of all points in the Graffiti point buffer.

```
Err GrfFlushPoints (
    void
);
```

Parameters

None.

Return Value

Always returns 0.

See Also

GrfAddPoint

GrfGetAndExpandMacro

Look up and expand a macro in the current macros.

```
Err GrfGetAndExpandMacro(
    CharPtr nameP,
    BytePtr macroDataP,
    WordPtr dataLenP
);
```

Parameters

nameP
    Name of macro to look up.

macroDataP
    Macro contents returned here.

dataLenP
    On entry, size of macroDataP buffer; on exit, number of bytes in macro data.

Return Value

Returns 0 if no error, or grfErrNoMacros or grfErrMacroNotFound if an error occurs.

See Also

GrfAddMacro, GrfGetMacro

GrfGetGlyphMapping

Look up a glyph in the dictionary and return the text.

```
Err GrfGetGlyphMapping(
    Word glyphID,
    WordPtr flagsP,
    void* dataPtrP,
    WordPtr dataLenP,
    WordPtr uncertainLenP
);
```

Parameters

glyphID
    Glyph ID to lookup.

flagsP
    Returned dictionary flags.

dataPtrP
    Where returned text goes.

dataLenP
    On entry, size of dataPtrP; on exit, number of bytes returned.

uncertainLenP
    Return number of uncertain characters in text.

Return Value

Returns 0 if no error, or grfErrNoDictionary or grfErrNoMapping if an error occurs.

See Also

GrfMatch

GrfGetMacro

Look up a macro in the current macros.

```
Err GrfGetMacro(
    CharPtr nameP,
    BytePtr macroDataP,
    WordPtr dataLenP
);
```

Parameters

nameP
    Name of macro to lookup.

macroDataP
    Macro contents returned here.

dataLenP
    On entry: size of macroDataP buffer. On exit: number of bytes in macro data.

Return Value

Returns 0 if no error or grfErrNoMacros, grfErrMacroNotFound.

See Also

GrfAddMacro

GrfGetMacroName

Look up a macro name by index.

```
Err GrfGetMacroName (
    Word index,
    CharPtr nameP
);
```

Parameters

index
    Index of macro.

nameP
    Name returned here.

Return Value

Returns 0 if no error, or grfErrNoMacros or grfErrMacroNotFound if an error occurs.

See Also

GrfAddMacro, GrfGetMacro

GrfGetNumPoints

Return the number of points in the point buffer.

```
Err GrfGetNumPoints (
    WordPtr numPtsP
);
```

Parameters

numPtsP
    Returned number of points.

Return Value

Always returns 0.

See Also

GrfAddPoint

GrfGetPoint

Return a point out of the Graffiti point buffer.

```
Err GrfGetPoint (
    Word index,
    PointType* pointP
);
```

Parameters

index
    Which point to get.

pointP
    Returned point.

Return Value

Returns 0 if no error, or grfErrBadParam if an error occurs.

See Also

GrfAddPoint, GrfGetNumPoints

GrfGetState

Returns the current shift state of Graffiti.

```
Err GrfGetState(
    Boolean* capsLockP,
    Boolean* numLockP,
    WordPtr tempShiftP,
    Boolean* autoShiftedP
);
```

Parameters

capsLockP
    Returns TRUE if caps lock on.

numLockP
    Returns TRUE if num lock on.

tempShiftP
    Current temporary shift.

autoShiftedP
    Returns TRUE if shift not set by the user.

Return Value

Always returns 0.

See Also

GrfSetState

GrfInitState

Reinitialize the Graffiti dictionary state.

```
Err GrfInitState (
    void
);
```

Parameters

None.

Return Value

Always returns 0.

See Also

GrfGetState, GrfSetState

GrfMatch

Recognize the current stroke in the Graffiti point buffer and return with the recognized text.

```
Err GrfMatch (
    WordPtr flagsP,
    void* dataPtrP,
    WordPtr dataLenP,
    WordPtr uncertainLenP,
    GrfMatchInfoPtr matchInfoP
);
```

Parameters

flagsP
    Glyph flags are returned here.

dataPtrP
    Return text is placed here.

dataLenP
    Size of dataptr on exit; number of characters returned on exit.

uncertainLenP
    Return number of uncertain characters.

matchInfoP
    Array of grfMaxMatches, or nil.

Return Value

Returns 0 if no error, or grfErrNoGlyphTable, grfErrNoDictionary, or grfErrNoMapping if an error occurs.

See Also

GrfAddPoint, GrfFlushPoints

GrfMatchGlyph

Recognize the current stroke as a glyph.

```
Err GrfMatchGlyph (
    GrfMatchInfoPtr matchInfoP,
    Word maxUnCertainty,
    Word maxMatches
);
```

Parameters

matchInfoP
    Pointer to array of matches to fill in.

maxUnCertainty
    Maximum number of errors to tolerate.

maxMatches
    Size of matchInfoP array.

Return Value

Returns 0 if no error, or grfErrNoGlyphTable if an error occurs.

See Also

GrfMatch

GrfProcessStroke

Translate a stroke to keyboard events using Graffiti.

```
Err GrfProcessStroke (
    PointType* startPtP,
    PointType* endPtP,
    Boolean upShift
);
```

Parameters

startPtP
    Start point of stroke.

endPtP
    End point of stroke.

upShift
    Set to TRUE to feed an artificial upshift into the engine.

Return Value

0 if recognized.

Remarks

Called by SysHandleEvent when a pen-up is detected in the writing area. This routine recognizes the stroke and sends the recognized characters into the key queue. It also flushes the stroke out of the pen queue after recognition.

See Also

SysHandleEvent

GrfSetState

Set the current shift state of Graffiti.

```
Err GrfSetState(
    Boolean capsLock,
    Boolean numLock,
    Boolean upperShift
);
```

Parameters

capsLock
    Set to TRUE to turn on caps lock.

numLock
    Set to TRUE to turn on num lock.

upperShift
    Set to TRUE to put into upper shift.

Return Value

Always returns 0.

See Also

GrfGetState

SysShortCutListDialog

Pop up the Graffiti ShortCut list as a field object with the focus.

```
void SysGrfShortCutListDialog (
    void
);
```

Parameters

event
    Pointer to an EventType structure.

Return Value

The field's text chunk is changed.

See Also

GrfGetMacro, GrfGetMacroName

GrfFieldChange

```
Err GrfFieldChange(
    Boolean resetState,
    UIntPtr characterToDelete
);
```

WARNING: System Use Only.

GrfFree

```
Err GrfFree(
    void
);
```

WARNING: System Use Only.

GsiEnable

Enable or disable the Graffiti-shift state indicator.

```
void GsiEnable (
    Boolean enableIt
);
```

Parameters

enableIt
    TRUE to enable, FALSE to disable.

Return Value

Returns nothing.

Remarks

Enabling the indicator makes it visible, disabling it makes the insertion point invisible.

GsiEnabled

Return TRUE if the Graffiti-shift state indicator is enabled, or FALSE if it's disabled.

```
Boolean GsiEnabled (
    void
);
```

Parameters

None.

Return Value

TRUE if enabled, FALSE if not.

GsiInitialize

Initialize the global variables used to manage the Graffiti-shift state indicator.

```
void GsiInitialize (
    void
);
```

Parameters

None.

Return Value

Returns nothing.

GsiSetLocation

Set the display-relative position of the Graffiti-shift state indicator.

```
void GsiSetLocation (
    short x,
    short y
);
```

Parameters

x
    Coordinate of left side of the indicator.

y
    Coordinate of top of the indicator.

Return Value

Returns nothing.

Remarks

The indicator is not redrawn by this routine.

GsiSetShiftState

Set the Graffiti-shift state indicator.

```
void GsiSetShiftState (
    Word lockFlags,
    Word tempShift
);
```

Parameters

lockFlags
    glfCapsLock or glfNumLock.

tempShift
    The current temporary shift.

Return Value

Returns nothing.

Remarks

This function affects only the state of the UI element, not the underlying Graffiti engine.

See Also

GrfSetState

MemCardInfo

Return information about a memory card.

```
Err MemCardInfo (
    UInt cardNo,
    CharPtr cardNameP,
    CharPtr manufNamP,
    UIntPtr versionP,
    ULongPtr crDateP,
    ULongPtr romSizeP,
    ULongPTr ramSizeP,
    ULongPtr freeBytesP
);
```

Parameters

cardNo
Card number.

cardNameP
Pointer to character array (32 bytes) or 0.

manufNameP
Pointer to character array (32 bytes) or 0.

versionP
Pointer to version variable, or 0.

crDateP
Pointer to creation date variable, or 0.

romSizeP
Pointer to ROM size variable, or 0.

ramSizeP
Pointer to RAM size variable, or 0.

freeBytesP
Pointer to free byte-count variable, or 0.

Return Value

Returns 0 if no error.

Remarks

Pass 0 for those variables that you don't want returned.

MemChunkFree

Dispose of a chunk.

```
Err MemChunkFree (
    VoidPtr chunkDataP
);
```

Parameters

chunkDataP
    Chunk data pointer.

Return Value

0 No error
memErrInvalidParam Invalid parameter

Remarks

Call this routine to dispose of a chunk, which is disposed of even if it's locked.

MemDebugMode

Return the current debugging mode of the memory manager.

```
Word MemDebugMode (
    void
);
```

Parameters

No parameters.

Return Value

Returns debug flags as described for MemSetDebugMode.

MemHandleDataStorage

Return true if the given handle is part of a data storage heap. If not, it's a handle in the dynamic heap.

```
Boolean MemHandleDataStorage (
    VoidHand h
);
```

Parameters

h
    Chunk handle.

Return Value

Returns true if the handle is part of a data storage heap.

Remarks

Called by Fields package routines to determine if they need to worry about data storage write-protection when editing a text field.

See Also

MemPtrDataStorage

MemHandleCardNo

Return the card number a chunk resides in.

```
UInt MemHandleCardNo (
    VoidHand h
);
```

Parameters

h
    Chunk handle.

Return Value

Returns the card number.

Remarks

Call this routine to retrieve which card number (0 or 1) a movable chunk resides on.

See Also

MemPtrCardNo

MemHandleFree

Dispose of a movable chunk.

```
Err MemHandleFree (
    VoidHand h
);
```

Parameters

h
    Chunk handle.

Return Value

Returns 0 if no error, or memErrInvalidParam if an error occurs.

Remarks

Call this routine to dispose of a movable chunk.

See Also

MemHandleNew

MemHandleHeapID

Return the heap ID of a chunk.

```
UInt MemHandleHeapID (
    VoidHand h
);
```

Parameters

h
    Chunk handle.

Return Value

Returns the heap ID of a chunk.

Remarks

Call this routine to get the heap ID of the heap a chunk resides in.

See Also

MemPtrHeapID

MemHandleLock

Lock a chunk and obtain a pointer to the chunk's data.

```
VoidPtr MemHandleLock (
    VoidHand h
);
```

Parameters

h
    Chunk handle.

Return Value

Returns a pointer to the chunk.

Remarks

Call this routine to lock a chunk and obtain a pointer to the chunk. MemHandleLock and MemHandleUnlock should be used in pairs.

See Also

MemHandleNew, MemHandleUnlock

MemHandleNew

Allocate a new movable chunk in the dynamic heap.

```
VoidHand MemHandleNew (
    ULong size
);
```

Parameters

size
    The desired size of the chunk.

Return Value

Returns handle to the new chunk, or 0 if unsuccessful.

Remarks

Allocates a movable chunk in the dynamic heap and returns a handle it. Use this call when allocating dynamic memory.

See Also

MemPtrFree, MemPtrNew, MemHandleFree

MemHandleResize

Resize a chunk.

```
Err MemHandleResize (
    VoidHandle h,
    ULong newSize
);
```

Parameters

h
    Chunk handle.

newSize
    The new desired size.

Return Value

0 No error.
memErrInvalidParam Invalid parameter passed.
memErrNotEnoughSpace Not enough free space in heap to grow chunk.
memErrChunkLocked Can't grow chunk because it's locked.

Remarks

Call this routine to resize a chunk. This routine is always successful when shrinking the size of a chunk, even if the chunk is locked. When growing a chunk, it first attempts to grab free space immediately following the chunk so that the chunk does not have to move. If the chunk has to move to another free area of the heap to grow, it must be movable and have a lock count of 0.

See Also

MemHandleNew, MemHandleSize

MemHandleSize

Return the requested size of a chunk.

```
ULong MemHandleSize (
    VoidHand h
);
```

Parameters

h
    Chunk handle.

Return Value

Returns the requested size of the chunk.

Remarks

Call this routine to get the size originally requested for a chunk.

See Also

MemHandleResize

MemHandleToLocalID

Convert a handle into a local chunk ID which is card relative.

```
LocalID MemHandleToLocalID (
    VoidHand h
);
```

Parameters

h
    Chunk handle.

Return Value

Returns Local ID, or nil (0) if unsuccessful.

Remarks

Call this routine to convert a chunk handle to a Local ID.

See Also

MemLocalIDToGlobal, MemLocalIDToLockedPtr

MemHandleUnlock

Unlock a chunk given a chunk handle.

```
Err MemHandleUnlock (
    VoidHand h
);
```

Parameters

h
    The chunk handle.

Return Value

0 No error.
memErrInvalidParam Invalid parameter passed

Remarks

Call this routine to decrement the lock count for a chunk.
MemHandleLock and MemHandleUnlock should be used in pairs.

See Also

MemHandleLock

MemHeapCheck

Check validity of a given heap.

```
Err MemHeapCheck (
    UInt heapID
);
```

Parameters

heapID
    ID of heap to check.

Return Value

Returns 0 if no error.

See Also

MemDebugMode, MemSetDebugMode

MemHeapCompact

Compact a heap.

```
Err MemHeapCompact (
    UInt heapID
);
```

Parameters

heapID
    ID of the heap to compact.

Return Value

Always returns 0.

Remarks

Call this routine to compact a heap and merge all free space. This routine attempts to move all movable chunks to the start of the heap and merge all free space in the center of the heap.
The system software calls this function at various times; for example, during memory allocation (if sufficient free space is not available) and during system reboot.

MemHeapDynamic

Return TRUE if the given heap is a dynamic heap.

```
Boolean MemHeapDynamic (
    UInt heapID
);
```

Parameters

heapID
    ID of the heap to be tested.

Return Value

Returns TRUE if dynamic, FALSE if not.

Remarks

Dynamic heaps are used for volatile storage, application stacks, globals, and dynamically allocated memory.

See Also

MemNumHeaps, MemHeapID

MemHeapFlags

Return the heap flags for a heap.

```
UInt MemHeapFlags (
    UInt heapID
);
```

Parameters

heapID
    ID of heap.

Return Value

Returns the heap flags.

Remarks

Call this routine to retrieve the heap flags for a heap. The flags can be examined to determine if the heap is ROM based or not. ROM-based heaps have the memHeapFlagReadOnly bit set.

See Also

MemNumHeaps, MemHeapID

MemHeapFreeBytes

Return the total number of free bytes in a heap and the size of the largest free chunk in the heap.

```
Err MemHeapFreeBytes (
    UInt heapID,
    ULongPtr freeP,
    ULongPtr maxP
);
```

Parameters

heapID
    ID of heap.

freeP
    Pointer to a variable of type ULong for free bytes.

maxP
    Pointer to a variable of type ULong for max free chunk size.

Return Value

Always returns 0.

Remarks

Call this routine to retrieve the total number of free bytes left in a heap and the size of the largest free chunk. This routine doesn't compact the heap but the caller may compact the heap explicitly before calling this routine to determine if an allocation will succeed or not.

See Also

MemHeapSize, MemHeapID, MemHeapCompact

MemHeapID

Return the heapID for a heap, given its index and the card number.

```
UInt MemHeapID (
    UInt cardNo,
    UInt heapIndex
);
```

Parameters

cardNo
    The card number, either 0 or 1.

heapIndex
    The heap index, anywhere from 0 to MemNumHeaps - 1.

Return Value

Returns the heap ID.

Remarks

Call this routine to retrieve the heap ID of a heap, given the heap index and the card number. A heap ID must be used to obtain information on a heap such as its size, free bytes, etc., and is also passed to any routines which manipulate heaps.

See Also

MemNumHeaps

MemHeapScramble

Scramble the given heap.

```
Err MemHeapScramble (
    UInt heapID
);
```

Parameters

heapID
    ID of heap to scramble.

Remarks

The system does multiple passes over the heap attempting to move each movable chunk. Useful during debugging.

Return Value

Always returns 0.

See Also

MemDebugMode, MemSetDebugMode

MemHeapSize

Return the total size of a heap including the heap header.

```
ULong MemHeapSize (
    UInt heapID
);
```

Parameters

heapID
    ID of heap.

Return Value

Returns the total size of the heap.

See Also

MemHeapFreeBytes, MemHeapID

MemLocalIDKind

Return whether or not a Local ID references a handle or a pointer.

```
LocalIDKind MemLocalIDKind (
    LocalID local
);
```

Parameters

local
    The Local ID to query

Return Value

Returns LocalIDKind, or a memIDHandle or memIDPtr (see MemoryMgr.h).

Remarks

This routine determines if the given Local ID is to a nonmovable (memIDPtr) or movable (memIDHandle) chunk.

MemLocalIDToGlobal

Convert a Local ID, which is card relative, into a global pointer in the designated card.

```
VoidPtr MemLocalIDToGlobal (
    LocalID local,
    UInt cardNo
);
```

Parameters

local
    The Local ID to convert.

cardNo
    Memory card the chunk resides in.

Return Value

Returns pointer or handle to chunk.

Remarks

This routine converts a Local ID back to a pointer or handle, given the card number that the chunk resides in.

See Also

MemLocalIDKind, MemLocalIDToLockedPtr

MemLocalIDToLockedPtr

Return a pointer to a chunk designated by Local ID and card number.
Note: If the Local ID references a movable chunk handle, this routine automatically locks the chunk before returning.

```
VoidPtr MemLocalIDToLockedPtr(
    LocalID local,
    UInt cardNo
);
```

Parameters

local
    Local chunkID.

cardNo
    Card number.

Return Value

Returns pointer to chunk, or 0 if an error occurs.

See Also

MemLocalIDToGlobal, MemLocalIDToPtr, MemLocalIDKind, MemPtrToLocalID, MemHandleToLocalID

MemLocalIDToPtr

Return pointer to chunk, given the Local ID and card number.

```
VoidPtr MemLocalIDToPtr(
    LocalID local,
    UInt cardNo
);
```

Parameters

local
    Local ID to query.

cardNo
    Card number the chunk resides in.

Return Value

Returns a pointer to the chunk or 0 if error.

Remarks

If the Local ID references a movable chunk and that chunk is not locked, this function returns zero to indicate an error.

See Also

MemLocalIDToGlobal, MemLocalIDToLockedPtr

MemMove

Move a range of memory to another range in the dynamic heap.

```
Err MemMove(
    VoidPtr dstP,
    VoidPtr srcP,
    ULong numBytes
);
```

Parameters

dstP
    Pointer to destination.

srcP
    Pointer to source.

numBytes
    Number of bytes to move.

Return Value

Always returns 0.

Remarks

Handles overlapping ranges.
For operations where the destination is in a data heap, see DmSet, DmWrite, and related functions.

MemNumCards

Return the number of memory card slots in the system, not all slots need to be populated.

```
UInt MemNumCards (
    void
);
```

Parameters

None.

Return Value

Returns number of slots in the system.

MemNumHeaps

Return the number of heaps available on a particular card.

```
UInt MemNumHeaps (
    UInt cardNo
);
```

Parameters

cardNo
    The card number; either 0 or 1.

Return Value

Number of heaps available including ROM- and RAM-based heaps.

Remarks

Call this routine to retrieve the total number of heaps on a memory card. The information can be obtained by calling MemHeapSize,
MemHeapFreeBytes, and MemHeapFlags on each heap using its heapID. The heapID is obtained by calling MemHeapID with the card number and the heap index which can be any value from 0 to MemNumHeaps.

MemNumRAMHeaps

Return the number of RAM heaps in the given card.

```
UInt MemNumRAMHeaps (
    UInt cardNo
);
```

Parameters

cardNo
    The card number.

Return Value

Returns the number of RAM heaps.

See Also

MemNumCards

MemPtrCardNo

Return the card number (0 or 1) a nonmovable chunk resides on.

```
UInt MemPtrCardNo (
    VoidPtr chunkP
);
```

Parameters

chunkP
    Pointer to the chunk.

Return Value

Returns the card number.

See Also

MemHandleCardNo

MemPtrDataStorage

Return TRUE if the given pointer is part of a data storage heap; if not, it is a pointer in the dynamic heap.

```
Boolean MemPtrDataStorage (
    VoidPtr p
);
```

Parameters

p
    Pointer to a chunk.

Return Value

Returns true if the chunk is part of a data storage heap.

Remarks

Called by Fields package to determine if it needs to worry about data storage write-protection when editing a text field.

See Also

MemHeapDynamic

MemPtrFree

Macro to dispose of a chunk.

```
Err MemPtrFree (
    VoidPtr p
);
```

Parameters

p
    Pointer to a chunk.

Return Value

Returns 0 if no error or memErrInvalidParam (Invalid parameter).

Remarks

Call this routine to dispose of a nonmovable chunk.

MemPtrHeapID

Return the heap ID of a chunk.

```
UInt MemPtrHeapID (
    VoidPtr p
);
```

Parameters

chunkP
    Pointer to the chunk.

Return Value

Returns the heap ID of a chunk.

Remarks

Call this routine to get the heap ID of the heap a chunk resides in.

MemPtrToLocalID

Convert a pointer into a card-relative local chunk ID.

```
LocalID MemPtrToLocalID (
    VoidPtr chunkP
);
```

Parameters

chunkP
    Pointer to a chunk.

Return Value

Returns the local ID of the chunk.

Remarks

Call this routine to convert a chunk pointer to a Local ID.

See Also

MemLocalIDToPtr

MemPtrNew

Allocate a new nonmovable chunk in the dynamic heap.

```
VoidPtr MemPtrNew (
    ULong size
);
```

Parameters

size
    The desired size of the chunk.

Return Value

Returns pointer to the new chunk, or 0 if unsuccessful.

Remarks

This routine allocates a nonmovable chunk in the dynamic heap and returns a pointer to the chunk. Applications can use it when allocating dynamic memory.

MemPtrRecoverHandle

Recover the handle of a movable chunk, given a pointer to its data.

```
VoidHand MemPtrRecoverHandle (
    VoidPtr p
);
```

Parameters

p
    Pointer to the chunk.

Return Value

Returns the handle of the chunk, or 0 if unsuccessful.

Remarks

Don't call this function for pointers in ROM or non-movable data chunks.

MemPtrResize

Resize a chunk.

```
Err MemPtrResize (
    VoidPtr p,
    ULong newSize
);
```

Parameters

p
    Pointer to the chunk.

newSize
    The new desired size.

Return Value

Returns 0 if no error, or memErrNotEnoughSpace memErrInvalidParam, or memErrChunkLocked if an error occurs.

Remarks

Call this routine to resize a locked chunk. This routine is always successful when shrinking the size of a chunk. When growing a chunk, it attempts to use free space immediately following the chunk.

See Also

MemPtrSize, MemHandleResize

MemSet

Set a memory range in a dynamic heap to a specific value.

```
Err MemSet (
    VoidPtr dstP,
    ULong numBytes,
    Byte value
);
```

Parameters

dstP
    Pointer to the destination.

numBytes
    Number of bytes to set.

value
    Value to set.

Return Value

Always returns 0.

Remarks

For operations where the destination is in a data heap, see DmSet, DmWrite, and related functions.

MemSetDebugMode

Set the debugging mode of the memory manager.

```
Err MemSetDebugMode (
    Word flags
);
```

Parameters

flags
    Debug flags.

Remarks

Provide one (or none) of the following flags:
memDebugModeCheckOnChange
memDebugModeCheckOnAll
memDebugModeScrambleOnChange
memDebugModeScrambleOnAll
memDebugModeFillFree
memDebugModeAllHeaps
memDebugModeAllHeaps
memDebugModeRecordMinDynHeapFree

Return Value

Returns 0 if no error, or -1 if an error occurs.

MemPtrSize

Return the size of a chunk.

```
ULong MemPtrSize (
    VoidPtr p
);
```

Parameters

p
    Pointer to the chunk.

Return Value

The requested size of the chunk.

Remarks

Call this routine to get the original requested size of a chunk.

MemPtrUnlock

Unlock a chunk given a pointer to the chunk.

```
Err MemPtrUnlock (
    VoidPtr p
);
```

Parameters

p
    Pointer to a chunk.

Return Value

0 if no error, or memErrInvalidParam if an error occurs.

Remarks

A chunk must not be unlocked more times than it was locked.

See Also

MemHandleLock

MemStoreInfo

Return information on either the RAM store or the ROM store for a memory card.

```
Err MemStoreInfo (
    UInt cardNo,
    UInt storeNumber,
    UIntPtr versionP,
    UIntPtr flagsP,
    CharPtr nameP,
    ULongPtr crDateP,
    ULongPtr bckUpDateP,
    ULongPtr heapListOffsetP,
    ULongPtr initCodeOffset1P,
    ULongPtr initCodeOffset2P,
    LocalID* databaseDirIDP
);
```

Parameters

cardNo
    Card number, either 0 or 1.

storeNumber
    Store number; 0 for ROM, 1 for RAM.

versionP
    Pointer to version variable, or 0.

flagsP
    Pointer to flags variable, or 0.

nameP
    Pointer to character array (32 bytes) or 0.

crDateP
    Pointer to creation date variable, or 0.

bckUpDateP
    Pointer to backup date variable, or 0.

heapListOffsetP
    Pointer to heapListOffset variable, or 0.

initCodeOffset1P
    Pointer to initCodeOffset1 variable, or 0.

initCodeOffset2P
    Pointer to initCodeOffset2 variable, or 0.

databaseDirIDP
    Pointer to database directory chunk ID variable, or 0.

Return Value

Returns 0 if no error, or memErrCardNoPresent, memErrRAMOnlyCard, or memErrInvalidStoreHeader if an error occurs.

Remarks

Call this routine to retrieve any or all information on either the RAM store or the ROM store for a card. Pass 0 for variables that you don't wish returned.

MemCardFormat

```
Err MemCardFormat (
    UInt cardNo,
    CharPtr cardNameP,
    CharPtr manufNameP,
    CharPtr ramStoreNameP
);
```

WARNING: This function for use by system software only.

MemChunkNew

```
VoidPtr MemChunkNew (
    UInt heapID,
    ULong size,
    UInt attributes
);
```

WARNING: This function for use by system software only.

MemHandleFlags

```
UInt MemHandleFlags (
    VoidHand h
);
```

WARNING: This function for use by system software only.

MemHandleLockCount

```
UInt MemHandleLockCount (
    VoidHand h
);
```

WARNING: This function for use by system software only.

MemHandleOwner

```
UInt MemHandleOwner (
    VoidHand h
);
```

WARNING: This function for use by system software only.

MemHandleResetLock

```
Err MemHandleResetLock (
    VoidHand h
);
```

WARNING: This function for use by system software only.

MemHandleSetOwner

```
Err MemHandleSetOwner (
    VoidHand h,
    UInt owner
);
```

WARNING: This function for use by system software only.

MemHeapFreeByOwnerID

```
Err MemHeapFreeByOwnerID (
    UInt heapID,
    UInt ownerID
);
```

WARNING: This function for use by system software only.

MemHeapInit

```
Err MemHeapInit(
    UInt heapID,
    Int numHandles,
    Boolean initContents
);
```

WARNING: This function for use by system software only.

MemInit

```
Err MemInit (
    void
);
```

Warning: This function for use by system software only.

MemInitHeapTable

```
Err MemInitHeapTable (
    UInt cardNo
);
```

WARNING: This function for use by system software only.

MemKernelInit

```
Err MemKernelInit(
    void
);
```

WARNING: This function for use by system software only.

MemPtrFlags

```
UInt MemPtrFlags (
    VoidPtr chunkDataP
);
```

WARNING: This function for use by system software only.

MemPtrOwner

```
UInt MemPtrOwner (
    VoidPtr chunkDataP
);
```

WARNING: This function for use by system software only.

MemPtrResetLock

```
Err MemPtrResetLock (
    VoidPtr chunkP
);
```

WARNING: This function for use by system software only.

MemPtrSetOwner

```
Err MemPtrSetOwner (
    VoidPtr chunkP,
    UInt owner
);
```

WARNING: This function for use by system software only.

MemSemaphoreRelease

```
Err MemSemaphoreRelease (
    Boolean writeAccess
);
```

Warning: This function for use by system software only.

MemSemaphoreReserve

```
Err MemSemaphoreReserve (
    Boolean writeAccess
);
```

Warning: This function for use by system software only.

MemStoreSetInfo

```
Err MemStoreSetInfo (
    UInt cardNo,
    UInt storeNumber,
    UIntPtr versionP,
    UIntPtr flagsP,
    CharPtr nameP,
    ULongPtr crDateP,
    ULongPtr bckUpDateP,
    ULongPtr heapListOffsetP,
    ULongPtr initCodeOffset1P,
    ULongPtr initCodeOffset2P,
    LocalID* databaseDirIDP
);
```

Warning: This function for use by system software only.

DmArchiveRecord

Mark a record as archived by leaving the record's chunk around and setting the delete bit for the next sync.

```
Err DmArchiveRecord (
    DmOpenRef dbR,
    UInt index
);
```

Parameters

dbR
DmOpenRef to open database.

index
Which record to archive.

Return Value

Returns 0 if no error or dmErrIndexOutOfRange or dmErrReadOnly if an error occurs.

Remarks

Marks the delete bit in the database header for the record but does not dispose of the record's data chunk.

See Also

DmRemoveRecord, DmDetachRecord, DmNewRecord, DmDeleteRecord

DmAttachRecord

Attach an existing chunk ID handle to a database as a record.

```
Err DmAttachRecord (
    DmOpenRef dbR,
    UIntPtr atP,
    Handle newH,
    Handle* oldHP
);
```

Parameters

dbR
DmOpenRef to open database.

atP
Pointer to index where new record should be placed.

newH
Handle of new record.

oldHP
Pointer to return old handle if replacing existing record.

Return Value

Returns 0 if no error, or dmErrIndexOutOfRange, dmErrMemError, dmErrReadOnly, dmErrRecordInWrongCard, memErrChunkLocked, memErrInvalidParam, or memErrNotEnoughSpace if an error occurs.

Remarks

Given the handle of an existing chunk, this routine makes that chunk a new record in a database and sets the dirty bit. The parameter atP points to an index variable. If oldHP is nil, the new record is inserted at index *atP and all following record indices are shifted down. If *atP is greater than the number of records currently in the database, the new record is appended to the end and the index of it returned in *atP. If oldHP is not nil, the new record replaces an existing record at index *atP and the handle of the old record is returned in *oldHP so that the application can free it or attach it to another database.
Useful for cutting and pasting between databases.

See Also

DmDetachRecord, DmNewRecord, DmNewHandle

DmAttachResource

Attach an existing chunk ID to a resource database as a new resource.

```
Err DmAttachResource (
    DmOpenRef dbR,
    VoidHand newH,
    ULong resType,
    Int resID
);
```

Parameters

dbR
DmOpenRef to open database.

newH
Handle of new resource's data.

resType
Type of the new resource.

resID
ID of the new resource.

Return Value

Returns 0 if no error, or dmErrIndexOutOfRange, dmErrMemError, dmErrReadOnly, dmErrRecordInWrongCard, memErrChunkLocked, memErrInvalidParam, or memErrNotEnoughSpace if an error occurs.

Remarks

Given the handle of an existing chunk with resource data in it, this routine makes that chunk a new resource in a resource database.
The new resource will have the given type and ID.

See Also

DmDetachResource, DmRemoveResource, DmNewHandle, DmNewResource

DmCloseDatabase

Close a database.

```
Err DmCloseDatabase (
    DmOpenRef dbR
);
```

Parameters

dbR
Database access pointer.

Return Value

Returns 0 if no error or dmErrInvalidParam if an error occurs.

Remarks

This routine doesn't unlock any records in the database which have been left locked, so the application should be careful not to leave records locked. When performance is not an issue, call DmResetRecordStates before closing the database in order to unlock all records and clear the busy bits.

See Also

DmOpenDatabase, DmDeleteDatabase, DmOpenDatabaseByTypeCreator

DmCreateDatabase

Create a new database on the specified card with the given name, creator, and type.

```
Err DmCreateDatabase (
    UInt cardNo,
    CharPtr nameP,
    ULong creator,
    ULong type,
    Boolean resDB
);
```

Parameters

cardNo
The card number to create the database on.

nameP
Name of new database, up to 31 ASCII bytes long.

creator
Creator of the database.

type
Type of the database.

resDB
If true, create a resource database.

Return Value

Returns 0 if no error, or dmErrInvalidDatabaseName, dmErrAlreadyExists, memErrCardNotPresent, dmErrMemError, memErrChunkLocked, memErrInvalidParam, memErrInvalidStoreHeader, memErrNotEnoughSpace, or memErrRAMOnlyCard if an error occurs.

Remarks

Call this routine to create a new database on a specific card. This routine doesn't check for a database with the same name, so check for it yourself. Once created, the database ID can be retrieved by calling DmFindDatabase and the database opened using the database ID. To create a resource database instead of a record-based database, set the resDB boolean to TRUE.

See Also

DmCreateDatabaseFromImage, DmOpenDatabase, DmDeleteDatabase

DmCreateDatabaseFromImage

Call to create an entire database from a single resource that contains an image of the database; usually, make this call from an application's reset action code during boot.

```
Err DMCreateDatabaseFromImage (
    Ptr bufferP
);
```

Parameters

bufferP
Pointer to locked resource containing database image.

Return Value

Returns 0 if no error

Remarks

Use this function to create the default database for an application.

See Also

DmCreateDatabase, DmOpenDatabase

DmDatabaseInfo

Retrieve information about a database.

```
Err DmDatabaseInfo (
    UInt cardNo,
    LocalID dbID,
    CharPtr nameP,
    UIntPtr attributesP,
    UIntPtr versionP,
    ULongPtr crDateP,
    ULongPtr modDateP,
    ULongPtr bckUpDateP,
    ULongPtr modNumP,
    LocalID* appInfoIDP,
    LocalID* sortInfoIDP,
    ULongPtr typeP,
    ULongPtr creatorP
);
```

Parameters

cardNo
Which card number database resides on.

dbID
Database ID of the database.

nameP
Pointer to 32-byte character array for returning the name, or nil.

attributesP
Pointer to return attributes variable, or nil.

versionP
Pointer to new version, or nil.

crDateP
Pointer to return creation date variable, or nil.

modDateP
Pointer to return modification date variable, or nil.

bckUpDateP
Pointer to return backup date variable, or nil.

modNumP
Pointer to return modification number variable, or nil.

appInfoIDP
Pointer to return appInfoID variable, or nil.

sortInfoIDP
Pointer to return sortInfoID variable, or nil.

typeP
Pointer to return type variable, or nil.

creatorP
Pointer to return creator variable, or nil.

Return Value

Returns 0 if no error, or dmErrInvalidParam if an error occurs.

Remarks

Call this routine to retrieve any or all information about a database.
This routine accepts nil for any return variable parameter pointer you don't want returned.

See Also

DmSetDatabaseInfo, **DmDatabaseSize**, **DmOpenDatabaseInfo**, **DmFindDatabase**, **DmGetNextDatabaseByTypeCreator**

DmDatabaseSize

Retrieve size information on a database.

```
Err DmDatabaseSize (
    UInt cardNo,
    ChunkID dbID,
    ULongPtr numRecordsP,
    ULongPtr totalBytesP,
    ULongPtr dataBytesP
);
```

Parameters

cardNo
Which card number database resides on.

dbID
Database ID of the database.

numRecordsP
Pointer to return numRecords variable, or nil.

totalBytesP
Pointer to return totalBytes variable, or nil.

dataBytesP
Pointer to return dataBytes variable, or nil.

Return Value

Returns 0 if no error, or dmErrMemError if an error occurs.

Remarks

Call this routine to retrieve the size of a database. Any of the return data variable pointers can be nil.
The total number of records is returned in *numRecordsP.
The total number of bytes used by the database including the overhead is returned in *totalBytesP.
The total number of bytes used to store just each record's data, not including overhead, is returned in *dataBytesP.

See Also

DmDatabaseInfo, DmOpenDatabaseInfo, DmFindDatabase, DmGetNextDatabaseByTypeCreator

DmDeleteDatabase

Delete a database and all its records.

```
Err DmDeleteDatabase (
    UInt cardNo,
    LocalID dbID
);
```

Parameters

cardNo
Card number the database resides on.

dbID
Database ID.

Return Value

Returns 0 if no error, or dmErrCantFind, dmErrCantOpen, memErrChunkLocked, dmErrDatabaseOpen, dmErrROMBased, memErrInvalidParam, or memErrNotEnoughSpace if an error occurs.

Remarks

Call this routine to delete a database. This routine accepts a database ID as a parameter. To determine the database ID, call either DmFindDatabase or DmGetDatabase with a database index.

See Also

DmDeleteRecord, DmRemoveRecord, DmRemoveResource, DmCreateDatabase, DmGetNextDatabaseByTypeCreator, DmFindDatabase

DmDeleteRecord

Delete a record's chunk from a database but leave the record entry in the header and set the delete bit for the next sync.

```
Err DmDeleteRecord (
    DmOpenRef dbR,
    UInt index
);
```

Parameters

dbR
DmOpenRef to open database.

index
Which record to delete.

Return Value

Returns 0 if no error, or dmErrIndexOutOfRange, dmErrReadOnly, or memErrInvalidParam if an error occurs.

Remarks

Marks the delete bit in the database header for the record and disposes of the record's data chunk. Does not remove the record entry from the database header, but simply sets the localChunkID of the record entry to nil.

See Also

DmDetachRecord, DmRemoveRecord, DmArchiveRecord, DmNewRecord

DmDetachRecord

Detach and orphan a record from a database but don't delete the record's chunk.

```
Err DmDetachRecord (
    DmOpenRef dbR,
    UInt index,
    Handle* oldHP
);
```

Parameters

dbR
DmOpenRef to open.

index
Index of the record to detach.

oldHP
Pointer to return handle of the detached record.

Return Value

Returns 0 if no error or dmErrReadOnly (database is marked read only), dmErrIndexOutOfRange (index out of range), memErrChunkLocked, memErrInvalidParam, or memErrNotEnoughSpace if an error occurs.

Remarks

This routine detaches a record from a database by removing its entry from the database header and returns the handle of the record's data chunk in *oldHP. Unlike DmDeleteRecord, this routine removes any traces of the record including its entry in the database header.

See Also

DmAttachRecord, DmRemoveRecord, DmArchiveRecord, DmDeleteRecord

DmDetachResource

Detach a resource from a database and return the handle of the resource's data.

```
Err DmDetachResource (
    DmOpenRef dbR,
    Int index,
    VoidHand* oldHP
);
```

Parameters

dbR
DmOpenRef to open database.

index
Index of resource to detach.

oldHP
Pointer to return handle of the detached record.

Return Value

Returns 0 if no error, or dmErrCorruptDatabase, dmErrIndexOutOfRange, dmErrReadOnly, memErrChunkLocked, memErrInvalidParam, or memErrNotEnoughSpace if an error occurs.

Remarks

This routine detaches a resource from a database by removing its entry from the database header and returns the handle of the resource's data chunk in *oldHP.

See Also

DmAttachResource, DmRemoveResource

DmFindDatabase

Return the database ID of a database by card number and name.

```
LocalID DmFindDatabase (
    UInt cardNo,
    CharPtr nameP
);
```

Parameters

cardNo
Number of card to search.

nameP
Name of the database to look for.

Return Value

Returns the database ID, or 0 if not found.

See Also

DmGetNextDatabaseByTypeCreator, DmDatabaseInfo, DmOpenDatabase

DmFindRecordByID

Return the index of the record with the given unique ID.

```
Err DmFindRecordByID (
    DmOpenRef dbR,
    ULong uniqueID,
    UIntPtr indexP
);
```

Parameters

dbR
Database access pointer.

uniqueID
Unique ID to search for.

indexP
Return index.

Return Value

Returns 0 if found, otherwise dmErrUniqueIDNotFound.

See Also

DmQueryRecord, DmGetRecord, DmRecordInfo

DmFindResource

Search the given database for a resource by type and ID, or by pointer if it is non-nil.

```
Int DmFindResource (
    DmOpenRef dbR,
    ULong resType,
    Int resID,
    VoidHand findResH
);
```

Parameters

dbR
Open resource database access pointer.

resType
Type of resource to search for.

resID
ID of resource to search for.

findResH
Pointer to locked resource, or nil.

Return Value

Returns index of resource in resource database, or -1 if not found.

Remarks

Use this routine to find a resource in a particular resource database by type and ID or by pointer. It is particularly useful when you want to search only one database for a resource and that database is not the topmost one.
If findResH is nil, the resource is searched for by type and ID.
If findResH is not nil, resType and resID are ignored and the
index of the given locked resource is returned.
Once the index of a resource is determined, it can be locked down and accessed by calling DmGetResourceIndex.

See Also

DmGetResource, **DmSearchResource**, **DmResourceInfo**, **DmGetResourceIndex**, **DmFindResourceType**

DmFindResourceType

Search the given database for a resource by type and type index.

```
Int DmFindResourceType (
    DmOpenRef dbR,
    ULong resType,
    Int typeIndex
);
```

Parameters

dbR
Open resource database access pointer.

resType
Type of resource to search for.

typeIndex
Index of given resource type.

Return Value

Index of resource in resource database, or -1 if not found.

Remarks

Use this routine to retrieve all the resources of a given type in a resource database. By starting at typeIndex 0 and incrementing
until an error is returned, the total number of resources of a given type and the index of each of these resources can be determined.
Once the index of a resource is determined, it can be locked down and accessed by calling DmGetResourceIndex.

See Also

DmGetResource, DmSearchResource, DmResourceInfo, DmGetResourceIndex, DmFindResource

DmFindSortPosition

Return where a record is or should be.
Useful to find an existing record or find where to insert a record.
Uses a binary search.

```
UInt DmFindSortPosition(
    DmOpenRef dbR,
    VoidPtr newRecord,
    DmComparF *compar,
    Int other
);
```

Parameters

dbR
Database access pointer.

newRecord
Pointer to the new record.

compar
Comparison function (see remarks).

other
Any value the application wants to pass to the comparison function.

Return Value

Returns the position where the record should be inserted. The position should be viewed as between the record returned and the record before it. Note that the return value may be one greater than the number of records.

Remarks

compar, the comparison function, accepts two arguments, elem1 and elem2, each a pointer to an entry in the table. The comparison
function compares each of the pointed-to items (*elem1 and *elem2), and returns an integer based on the result of the comparison.
If the items compar returns
*elem1 < *elem2 an integer < 0
*elem1 == *elem2 0
*elem1 > *elem2 an integer > 0

See Also

DmQuickSort, DmInsertionSort

DmGetAppInfoID

Return the Local ID of the application info block.

```
LocalID DmGetAppInfoID (
    DmOpenRef dbR
);
```

Parameters

dbR
    Database access pointer.

Return Value

Returns Local ID of the application info block

See Also

DmDatabaseInfo, DmOpenDatabase

DmGetDatabase

Return the database header ID of a database by index and card number.

```
LocalID DmGetDatabase (
    UInt cardNo,
    UInt index
);
```

Parameters

cardNo
Which card number.

index
Index of database.

Return Value

Returns the database ID, or 0 if an invalid parameter passed.

Remarks

Call this routine to retrieve the database ID of a database by index.
The index should range from 0 to DmNumDatabases-1. This routine is useful for getting a directory of all databases on a card.

See Also

DmOpenDatabase, DmNumDatabases, DmDatabaseInfo, DmDatabaseSize

DmGetLastErr

Return error code from last data manager call.

```
Err DmGetLastErr (
    void
);
```

Parameters

None

Return Value

Error code from last unsuccessful data manager call.

Remarks

Use this routine to determine why a data manager call failed. In particular, calls like DmGetRecord return 0 only if unsuccessful, so calling DmGetLastErr is the only way to determine why they failed.
Note that DmGetLastErr does not always reflect the error status of the last data manager call. Rather, it reflects the error status of data manager calls that don't return an error code. For some of those calls, the saved error code value is not set to 0 when the call is successful.
For example, if a call to DmOpenDatabaseByTypeCreator returns null for database reference (that is, it fails), DmGetLastErr returns
something meaningful; otherwise, it returns the error value of some previous data manager call.
Only the following data manager functions currently affect the value returned by DmGetLastErr: DmFindDatabase, DmOpenDatabaseByTypeCreator, DmOpenDatabase, DmNewRecord, DmQueryRecord, DmGetRecord, DmQueryNextInCategory, DmPositionInCategory, DmSeekRecordInCategory, DmResizeRecord, DmGetResource, DmGet1Resource, DmNewResource, DmGetResourceIndex.

DmGetNextDatabaseByTypeCreator

Return a database header ID and card number given the type and/or creator. This routine searches all memory cards for a match.

```
Err DmGetNextDatabaseByTypeCreator (
    Boolean newSearch,
    DmSearchStatePtr stateInfoP,
    ULong type,
    ULong creator,
    Boolean onlyLatestVers,
    UIntPtr cardNoP,
    LocalID* dbIDP
);
```

Parameters

newSearch
True if starting a new search.

stateInfoP
If newSearch is false, this must point to the same data used for the previous invocation.

type
Type of database to search for, pass 0 as a wildcard.

creator
Creator of database to search for, pass 0 as a wildcard.

onlyLatestVers
If true, only latest version of each database with a given type and creator is returned.

cardNoP
On exit, the cardNo of the found database.

dbIDP
Database Local ID of the found database.

Return Value

0 No error.
dmErrCantFind No matches found.

Remarks

To start the search, pass TRUE for newSearch. To continue a search where the previous one left off, pass FALSE for newSearch.
When continuing a search, stateInfoP must point to the same structure passed during the previous invocation.
If the type parameter is nil, this routine can be called successively to return all databases of the given creator. If the creator parameter is nil, this routine can be called successively to return all databases of the given type.
If the onlyLatestVers parameter is set, only the latest version of each database with a given creator/type pair is returned.
If you're searching for the latest version and either type or creator is nil (wildcard), this routine returns the

index of the next database which matches the search criteria. This database can't have been superseded by a newer version of that database with the same type and creator.

See Also

DmGetDatabase, **DmFindDatabase**, **DmDatabaseInfo**, **DmOpenDatabaseByTypeCreator**, **DmDatabaseSize**

DmGetRecord

Return a handle to a record by index and mark the record busy.

```
VoidHand DmGetRecord (
    DmOpenRef dbR,
    UInt index
);
```

Parameters

dbR
DmOpenRef to open database.

index
Which record to retrieve.

Return Value

Handle to record data.

Remarks

Returns handle to given record and sets the busy bit for the record.
If another call to DmGetRecord for the same record is attempted before the record is released, an error is returned.
If the record is ROM-based (pointer accessed), this routine makes a fake handle to it and store this handle in the DmAccessType structure.
DmReleaseRecord should be called as soon as the caller is done viewing or editing the record.

See Also

DmSearchRecord, DmFindRecordByID, DmRecordInfo, DmReleaseRecord, DmQueryRecord

DmGetResource

Search all open resource databases and return a handle to a resourcegiven the resource type and ID.

```
VoidHand DmGetResource (
    ULong type,
    Int ID
);
```

Parameters

type
The resource type.

ID
The resource ID.

Return Value

Returns pointer to resource data, or nil if unsuccessful.

Remarks

Searches all open resource databases starting with the most recently opened one for a resource of the given type and ID. If found, the resource handle is returned. The application should call DmReleaseRecord as soon as it's done accessing the resource data to avoid fragmenting the heap.

See Also

DmGet1Resource, DmReleaseResource

DmGetResourceIndex

Return a handle to a resource by index.

```
VoidHand DmGetResourceIndex (
    DmOpenRef dbR,
    Int index
);
```

Parameters

dbR
Access pointer to open database.

index
Index of resource to lock down.

Return Value

Handle to resource data, or nil if unsuccessful.

See Also

DmFindResource, DmFindResourceType, DmSearchResource

DmGet1Resource

Search the most recently opened resource database and return a handle to a resource given the resource type and ID.

```
VoidHand DmGet1Resource (
    ULong type,
    Int ID
);
```

Parameters

type
The resource type.

ID
The resource ID.

Return Value

Returns a pointer to resource data, or nil if unsuccessful.

Remarks

Searches the most recently opened resource database for a resource of the given type and ID. If found, the resource handle is returned.
The application should call DmReleaseRecord as soon as it's done accessing the resource data in order to avoid fragmenting the heap.

See Also

DmGetResource, DmReleaseResource

DmInsertionSort
Sort records in a database.

```
Err DmInsertionSort (
    DmOpenRef dbR,
    DmComparF *compar,
    Int other
);
```

Parameters

dbR
Database access pointer.

compar
Comparison function (see below).

other
Any value the application wants to pass to the comparison function.

Return Value

Returns 0 if no error or dmErrReadOnly if read only database.

Remarks

Deleted records are placed last in any order. All others are sorted according to the passed comparison function. Only records which are out of order move. Moved records are moved to the end of the range of equal records. If a large amount of records are being sorted, try to use the quick sort.
The following insertion sort algorithm is used: Starting with the second record, each record is compared to the preceding record. Each record not greater than the last is inserted into sorted position within those already sorted. A binary insertion is performed. A moved record is inserted after any other equal records.
compar, the comparison function, accepts two arguments, *elem1 and * elem2, each a pointer to an entry in the table. The comparison function compares each of the pointed-to items (*elem1 and *elem2), and returns an integer based on the result * of the comparison.
If the items compar returns *elem1 < *elem2 an integer < 0
*elem1 == *elem2 0
*elem1 > *elem2 an integer > 0

Return Value

Returns 0 if no error or dmErrInvalidParam.

Remarks

Called by SysAppLaunch (see Part 1) to move an application data-base
is launching out of the system list and into the application's
list.

See Also

DmFindSortPosition, **DmQuickSort**

DmMoveCategory

Move all records in a category to another category.

```
Err DmMoveCategory (
    DmOpenRef dbR,
    UInt toCategory,
    UInt fromCategory,
    Boolean dirty
);
```

Parameters

dbR
    DmOpenRef to open database.

toCategory
    Category to which to retrieve records.

fromCategory
    Category from which to retrieve records.

dirty
    If TRUE, set the dirty bit.

Return Value

Returns 0 if successful, or dmErrReadOnly if read-only database.

Remarks

If dirty is TRUE, the moved records are marked as dirty.

DmMoveRecord

Move a record from one index to another.

```
Err DmMoveRecord (
    DmOpenRef dbR,
    UInt from,
    UInt to
);
```

Parameters

dbR
    DmOpenRef to open database.

from
    Index of record to move.

to
    Where to move the record.

Return Value

Returns 0 if no error or one of dmErrIndexOutOfRange, dmErrReadOnly, memErrChunkLocked, memErrInvalidParam, or memErrNotEnoughSpace if an error occurs.

Remarks

Insert the record at the "to" index and move other records down. The "to" position should be viewed as an insertion position. Note that this value may be one greater than the index of the last record in the database.

DmNewHandle

Attempt to allocate a new chunk in the same data heap or card as the database header of the passed database access pointer. If there is not enough space in that data heap, tries other heaps.

```
VoidHand DmNewHandle (
    DmOpenRef dbR,
    ULong size
);
```

Parameters

dbR
DmOpenRef to open database.

size
Size of new handle.

Return Value

Returns the chunkID of new chunk, or 0 if not enough space.

Remarks

Allocates a new handle of the given size. Ensures that the new handle is in the same memory card as the given database. This guarantees that you can attach the handle to the database as a record obtain and save its LocalID in the appInfoID or sortInfoID fields of the header.

DmNextOpenDatabase

Return DmOpenRef to next open database for the current task.

```
DmOpenRef DmNextOpenDatabase (
    DmOpenRef currentP
);
```

Parameters

currentP
Current database access pointer or nil.

Return Value

DmOpenRef to next open database, or nil if there are no more.

Remarks

Call this routine successively to get the DmOpenRefs of all open databases.
Pass nil for currentP to get the first one. This routine would not normally be called by applications but is useful for system information.

See Also

DmOpenDatabaseInfo, DmDatabaseInfo

DmNextOpenResDatabase

Return access pointer to next open resource database in the search chain.

```
DmOpenRef DmNextOpenResDatabase (
    DmOpenRef dbR
);
```

Parameters

dbR
Database reference, or 0 to start search from the top.

Return Value

Pointer to next open resource database.

Remarks

Returns pointer to next open resource database. To get a pointer to the first one in the search chain, pass nil for dbR. This first database is the first and only one searched when DmGet1Resource is called.

DmNewRecord

Return a handle to a new record in the database and mark the record busy.

VoidHand **DmNewRecord (**
    DmOpenRef dbR,
    UIntPtr atP,
    ULong size
);

Parameters

dbR
    DmOpenRef to open database.

atP
    Pointer to index where new record should be placed.

size
    Size of new record.

Return Value

Pointer to record data, or 0 if error.

Remarks

Allocates a new record of the given size, and returns a handle to the record data.
The parameter atP points to an index variable. The new record is inserted at index *atP and all following record indices are shifted down. If *atP is greater than the number of records currently in the database, the new record is appended to the end and its index is returned in *atP.
Both the busy and dirty bits are set for the new record and a unique ID is automatically created.

See Also

DmAttachRecord, **DmRemoveRecord**, **DmDeleteRecord**

DmNewResource

Allocate and add a new resource to a resource database.

```
VoidHand DmNewResource (
    DmOpenRef dbR,
    ULong resType,
    Int resID,
    ULong size
);
```

Parameters

dbR
DmOpenRef to open database.

resType
Type of the new resource.

resID
ID of the new resource.

size
Desired size of the new resource.

Return Value

Returns a handle to new resource, or nil if unsuccessful.

Remarks

Allocates a memory chunk for a new resource and adds it to the given resource database. The new resource has the given type and ID. If successful, the application should call DmReleaseResource as soon as it finishes initializing the resource.

See Also

DmAttachResource, DmRemoveResource

DmNumDatabases

Determine how many databases reside on a memory card.

```
UInt DmNumDatabases (
    UInt cardNo
);
```

Parameters

cardNo
Number of the card to check.

Return Value

Returns the number of databases found.

Remarks

This routine is helpful for getting a directory of all databases on a card. The routine DmGetDatabase accepts an index from 0 to DmNumDatabases -1 and returns a database ID by index.

See Also

DmGetDatabase

DmNumRecords

Return the number of records in a database.

```
UInt DmNumRecords (
    DmOpenRef dbR
);
```

Parameters

dbR
    DmOpenRef to open database.

Return Value

Returns the number of records in a database.

See Also

DmNumRecordsInCategory, DmRecordInfo, DmSetRecordInfo

DmNumRecordsInCategory

Return the number of records of a specified category in a database.

```
UInt DmNumRecordsInCategory (
    DmOpenRef dbR,
    UInt category
);
```

Parameters

dbr
    DmOpenRef to open database.

category
    Category.

Return Value

Returns the number of records.

See Also

DmNumRecords, DmQueryNextInCategory, DmPositionInCategory, DmSeekRecordInCategory, DmMoveCategory

DmNumResources

Return the total number of resources in a given resource database.

```
UInt DmNumResources (
    DmOpenRef dbR
);
```

Parameters

dbR
DmOpenRef to open database.

Return Value

Returns the total number of resources in the given database.

DmOpenDatabase

Open a database and return a reference to it.

```
DmOpenRef DmOpenDatabase (
    UInt cardNo,
    LocalID dbID,
    UInt mode
);
```

Parameters

cardNo
Which card number database resides on.

dbID
The database ID of the database.

mode
Which mode to open database in (see below).

Return Value

Returns DmOpenRef to open database, or 0 if unsuccessful.

Remarks

Call this routine to open a database for reading or writing. The mode parameter can be one or more of the following constants ORed together:

dmModeReadWrite Read-write access.
dmModeReadOnly Read-only access.
dmModeLeaveOpen Leave database open even after application quits.
dmModeExclusive Don't let anyone else open it.

This routine returns a DmOpenRef which must be used to access particular records in a database. If unsuccessful, 0 is returned and the cause of the error can be determined by calling DmGetLastErr.

See Also

DmCloseDatabase, **DmCreateDatabase**, **DmFindDatabase**, **DmOpenDatabaseByTypeCreator**, **DmDeleteDatabase**

DmOpenDatabaseByTypeCreator

Open the most recent revision of a database with the given type and creator.

DmOpenRef **DmOpenDatabaseByTypeCreator(**
    ULong type,
    ULong creator,
    UInt mode
);

Parameters

type
    Type of database.

creator
    Creator of database.

mode
    Open mode (see remarks for DmOpenDatabase).

Return Value

DmOpenRef to open database, or 0 if unsuccessful.

See Also

DmCreateDatabase, **DmOpenDatabase**, **DmOpenDatabaseInfo**, **DmCloseDatabase**

DmOpenDatabaseInfo

Retrieve information about an open database.

```
Err DmOpenDatabaseInfo (
    DmOpenRef dbR,
    LocalIDPtr dbIDP,
    UIntPtr openCountP,
    UIntPtr modeP,
    UIntPtr cardNoP,
    BooleanPtr resDBP
);
```

Parameters

dbR
DmOpenRef to open database.

dbIDP
Pointer to return dbID variable, or nil.

openCountP
Pointer to return openCount variable, or nil.

modeP
Pointer to return mode variable, or nil.

cardNoP
Pointer to return card number, or nil.

resDBP
Pointer to return resDB Boolean, or nil.

Return Value

0 No error.
dmErrInvalidParam Invalid parameter passed.

Remarks

This routine retrieves information about an open database. Any nil return parameter pointers are ignored.

See Also

DmDatabaseInfo

DmPositionInCategory

Return a position of a record within the specified category.

```
UInt DmPositionInCategory (
    DmOpenRef dbR,
    UInt index,
    UInt category
);
```

Parameters

dbR
DmOpenRef to open database.

index
Index of the record.

category
Category to search.

Return Value

Returns the position (zero-based).

Remarks

If the record is ROM-based (pointer accessed) this routine makes a fake handle to it and stores this handle in the DmAccessType structure.

See Also

DmQueryNextInCategory, DmSeekRecordInCategory, DmMoveCategory

DmQueryNextInCategory

Return a handle to the next record in the specified category for reading only (does not set the busy bit).

VoidHand **DmQueryNextInCategory (**
    DmOpenRef dbR,
    UIntPtr indexP,
    UInt category
);

Parameters

dbR
    DmOpenRef to open database.

indexP
    Index of a known record (often retrieved with DmPositionInCategory).

category
    Which category to query.

Return Value

Returns a handle to the record following a known record.

See Also

DmNumRecordsInCategory, DmPositionInCategory, DmSeekRecordInCategory,

DmQueryRecord

Return a handle to a record for reading only (does not set the busy bit).

```
VoidHand DmQueryRecord (
    DmOpenRef dbR,
    UInt index
);
```

Parameters

dbR
DmOpenRef to open database.

index
Which record to retrieve.

Return Value

Returns record handle, or 0 if record is out of range or deleted.

Remarks

Returns handle to given record. Use this routine only when viewing the record. This routine successfully returns a handle to the record even if the record is busy.
If the record is ROM-based (pointer accessed) this routine returns the fake handle to it.

DmQuickSort

Sort records in a database.

```
Err DmQuickSort(
    const DmOpenRef dbR,
    DmComparF *compar,
    Int other
);
```

Parameters

dbR
Database access pointer

compar
Comparison function (see remarks)

other
Any value the application wants to pass to the comparison function.

Return Value

Returns 0 if no error or DmErrReadOnly if an error occurred.

Remarks

Deleted records are placed last in any order. All others are sorted according to the passed comparison function.
compar, the comparison function, accepts two arguments, elem1 and elem2, each a pointer to an entry in the table. The comparison function compares each of the pointed-to items (*elem1 and *elem2), and returns an integer based on the result of the comparison.
If the items compar returns
*elem1 < *elem2 an integer < 0
*elem1 == *elem2 0
*elem1 > *elem2 an integer > 0

See Also

DmFindSortPosition, DmInsertionSort

DmRecordInfo

Retrieve the record information as stored in the database header.

```
Err DmRecordInfo (
    DmOpenRef dbR,
    UInt index,
    UBytePtr attrP,
    ULongPtr uniqueIDP,
    LocalID* chunkIDP
);
```

Parameters

dbR
DmOpenRef to open database.

index
Index of record.

attrP
Pointer to return attribute variable, or nil.

uniqueIDP
Pointer to return unique ID variable, or nil.

chunkIDP
Pointer to return Local ID variable, or nil.

Return Value

Returns 0 if no error or dmErrIndexOutOfRange if an error occurred.

Remarks

Retrieves information about a record. Any of the return variable pointers can be nil.

See Also

DmNumRecords, DmSetRecordInfo, DmQueryNextInCategory

DmResourceInfo

Retrieve information on a given resource.

```
Err DmResourceInfo (
    DmOpenRef dbR,
    Int index,
    ULongPtr resTypeP,
    IntPtr resIDP,
    LocalID* chunkLocalIDP
);
```

Parameters

dbR
DmOpenRef to open database.

index
Index of resource to get info on.

resTypeP
Pointer to return resType variable, or nil.

resIDP
Pointer to return resID variable, or nil.

chunkLocalIDP
Pointer to return chunkID variable, or nil.

Return Value

Returns 0 if no error or dmErrIndexOutOfRange if an error occurred.

Remarks

Use this routine to retrieve all or a portion of the information on a particular resource. Any or all of the return variable pointers can be nil. The type and ID of the resource are returned in *resTypeP and *resIDP. The Memory Manager Local ID of the resource data is returned in *chunkLocalIDP.

See Also

DmGetResource, DmGet1Resource, DmSetResourceInfo, DmFindResource, DmFindResourceType

DmReleaseRecord

Clear the busy bit for the given record and set the dirty bit if dirty is true.

```
Err DmReleaseRecord (
    DmOpenRef dbR,
    UInt index,
    Boolean dirty
);
```

Parameters

dbR
DmOpenRef to open database.

index
Which record to unlock.

dirty
If TRUE, set the dirty bit.

Return Value

Returns 0 if no error or dmErrIndexOutOfRange if an error occurred.

Remarks

Call this routine when you finished modifying or reading a record that you've called DmGetRecord on. It sets the dirty bit for the record if the dirty parameter is set.

See Also

DmGetRecord

DmReleaseResource

Release a resource acquired with DmGetResource.

```
Err DmReleaseResource (
    VoidHand resourceH
);
```

Parameters

resourceH
Handle to resource.

Return Value

Returns 0 if no error.

Remarks

Marks a resource as being no longer needed by the application.

See Also

DmGet1Resource, DmGetResource

DmRemoveRecord

Remove a record from a database and dispose of its data chunk.

```
Err DmRemoveRecord (
    DmOpenRef dbR,
    UInt index
);
```

Parameters

dbR
DmOpenRef to open database.

index
Index of the record to remove.

Return Value

Returns 0 if no error, or dmErrCorruptDatabase, dmErrIndexOutOfRange, dmErrReadOnly, memErrChunkLocked, memErrInvalidParam, or memErrNotEnoughSpace if an error occurs.

Remarks

Disposes of a the record's data chunk and removes the record's entry from the database header.

See Also

DmDetachRecord, DmDeleteRecord, DmArchiveRecord, DmNewRecord

DmRemoveResource

Delete a resource from a resource database.

```
Err DmRemoveResource (
    DmOpenRef dbR,
    Int index
);
```

Parameters

dbR
DmOpenRef to open database.

index
Index of resource to delete.

Return Value

Returns 0 if no error or dmErrCorruptDatabase, dmErrIndexOutOfRange, dmErrReadOnly, memErrChunkLocked, memErrInvalidParam, or memErrNotEnoughSpace if an error occurs.

Remarks

This routine disposes of the memory manager chunk that holds the given resource and removes its entry from the database header.

See Also

DmDetachResource, DmRemoveResource, DmAttachResource

DmRemoveSecretRecords

Remove all secret records.

```
Err DmRemoveSecretRecords (
    DmOpenRef dbR
);
```

Parameters

dbR
DmOpenRef to open database.

Return Value

Returns 0 if no error or dmErrReadOnly (read-only database) if an error occurred.

See Also

DmRemoveRecord, DmRecordInfo, DmSetRecordInfo

DmResetRecordStates

Unlock all records in a database and clear all busy bits.

```
Err DmResetRecordStates (
    DmOpenRef dbR
);
```

Parameters

dbR
    DmOpenRef to open database.

Return Value

Returns 0 if no error or dmErrROMBased if an error occurred.

Remarks

This routine unlocks all records in a database and clears all busy bits. It can optionally be called before closing a database to ensure that the records are all unlocked. For performance reasons, the data manager does not call DmResetRecordStates automatically when closing a database.
This routine automatically allocates the record in another data heap if the current heap is too full.

DmResizeRecord

Resize a record by index.

```
VoidHand DmResizeRecord (
    DmOpenRef dbR,
    UInt index,
    ULong newSize
);
```

Parameters

dbR
DmOpenRef to open database.

index
Which record to retrieve.

newSize
New size of record.

Return Value

Pointer to resized record, or nil if unsuccessful.

Remarks

This routine reallocates the record in another heap of the same memory card if the current heap is not big enough. If this happens, the handle changes, so be sure to use the return handle to access the resized resource.

DmResizeResource

Resize a resource and return the new handle.

```
VoidHand DmResizeResource (
    VoidHand resourceH,
    ULong newSize
);
```

Parameters

resourceH
Handle to resource.

newSize
Desired new size of resource.

Return Value

Returns a handle to newly-sized resource or nil if unsuccessful.

Remarks

Resizes the resource and returns new handle. If necessary in order to grow the resource, this routine will reallocate it in another heap on the same memory card that it is currently in.
The handle may change if the resource had to be reallocated in a different data heap because there was not enough space in its present data heap.

DmSearchRecord

Search all open record databases for a record with the handle passed.

```
Int DmSearchRecord (
    VoidHand recH,
    DmOpenRef* dbRP
);
```

Parameters

recH
Record handle.

dbRP
Pointer to return variable of type DmOpenRef.

Return Value

Returns the index of the record and database access pointer; if not found, index will be -1 and *dbRP will be 0.

See Also

DmGetRecord, DmFindRecordByID, DmRecordInfo

DmSearchResource

Search all open resource databases for a resource by type and ID, or by pointer if it is non-nil.

```
Int DmSearchResource (
    ULong resType,
    Int resID,
    VoidHand resH,
    DmOpenRef* dbRP
);
```

Parameters

resType
Type of resource to search for.

resID
ID of resource to search for.

resH
Pointer to locked resource, or nil.

dbRP
Pointer to return variable of type DmOpenRef.

Return Value

Returns the index of the resource, stores DmOpenRef in dbRP.

Remarks

This routine can be used to find a resource in all open resource databases by type and ID or by pointer. If resH is nil, the resource is searched for by type and ID. If resH is not nil, resType and resID is ignored and the index of the resource handle is returned. On return *dbRP contains the access pointer of the resource database that the resource was eventually found in. Once the index of a resource is determined, it can be locked down and accessed by calling DmGetResourceByIndex.

See Also

DmGetResource, **DmFindResourceType**, **DmResourceInfo**, **DmGetResourceIndex**, **DmFindResource**

DmSeekRecordInCategory

Return the index of the record at the offset from the passed record index. (The offset parameter indicates the number of records to move forward or backward; the value for backward is negative.)

```
Err DmSeekRecordInCategory (
    DmOpenRef dbR,
    UIntPtr indexP,
    Int offset,
    Int direction,
    UInt category
);
```

Parameters

dbR
DmOpenRef to open database.

index
Pointer to the returned index.

offset
Offset of the passed record index.

direction
dmSeekForward or dmSeekBackward.

category
Category ID.

Return Value

Returns 0 if no error or dmErrIndexOutOfRange or dmErrSeekFailed if an error occurred.

See Also

DmNumRecordsInCategory, DmQueryNextInCategory, DmPositionInCategory, DmMoveCategory

DmSet

Check the validity of the chunk pointer for a record and makes sure that writing the record does not exceed the chunk bounds.

```
Err DmSet (
    VoidPtr recordP,
    ULong offset,
    ULong bytes,
    Byte value
);
```

Parameters

recordP
Pointer to locked data record (chunk pointer).

offset
Offset within record to start writing.

bytes
Number of bytes to write.

value
Byte value to write.

Return Value

Returns 0 if no error or dmErrNotValidRecord or dmErrWriteOutOfBounds if an error occurred.

Remarks

Must be used to write to data manager records because the data storage area is write-protected.

See Also

DmWrite

DmSetDatabaseInfo

Set information about a database.

```
Err DmSetDatabaseInfo (
    UInt cardNo,
    LocalID dbID,
    CharPtr nameP,
    UIntPtr attributesP,
    UIntPtr versionP,
    ULongPtr crDateP,
    ULongPtr modDateP,
    ULongPtr bckUpDateP,
    ULongPtr modNumP,
    LocalID* appInfoIDP,
    LocalID* sortInfoIDP,
    ULongPtr typeP,
    ULongPtr creatorP
);
```

Parameters

cardNo
Card number the database resides on.

dbID
Database ID of the database.

nameP
Pointer to 32-byte character array for new name, or nil.

attributesP
Pointer to new attributes variable, or nil.

versionP
Pointer to new version, or nil.

crDateP
Pointer to new creation date variable, or nil.

modDateP
Pointer to new modification date variable, or nil.

bckUpDateP
Pointer to new backup date variable, or nil.

modNumP
Pointer to new modification number variable, or nil.

appInfoIDP
Pointer to new appInfoID variable, or nil.

sortInfoIDP
Pointer to new sortInfoID variable, or nil.

typeP
Pointer to new type variable, or nil.

creatorP
Pointer to new creator variable, or nil.

Return Value

Returns 0 if no error or dmErrInvalidParam if an error occurred.

Remarks

When this call changes appInfoID or sortInfoID, the old chunkID (if any) is marked as an orphan chunk and the new chunk ID is unorphaned. Consequently, you shouldn't replace an existing appInfoID or sortInfoID if that chunk has already been attached to another database.
Call this routine to set any or all information about a database except for the card number and database ID. This routine sets the new value for any non-nil parameter.

See Also

DmDatabaseInfo, **DmOpenDatabaseInfo**, **DmFindDatabase**, **DmGetNextDatabaseByTypeCreator**

DmSetRecordInfo

Set record information stored in the database header.

```
Err DmSetRecordInfo (
    DmOpenRef dbR,
    UInt index,
    UBytePtr attrP,
    ULongPtr uniqueIDP
);
```

Parameters

dbR
DmOpenRef to open database.

index
Index of record.

attrP
Pointer to new attribute variable, or nil.

uniqueIDP
Pointer to new unique ID variable, or nil.

Return Value

Returns 0 if no error or dmErrIndexOutOfRange or dmErrReadOnly if an error occurred.

Remarks

Set information about a record.

See Also

DmNumRecords, DmRecordInfo

DmSetResourceInfo

Set information on a given resource.

```
Err DmSetResourceInfo (
    DmOpenRef dbR,
    Int index,
    ULongPtr resTypeP,
    IntPtr resIDP
);
```

Parameters

dbR
DmOpenRef to open database.

index
Index of resource to set info for.

resTypeP
Pointer to new resType, or nil.

resIDP
Pointer to new resID, or nil.

Return Value

Returns 0 if no error or dmErrIndexOutOfRange or dmErrReadOnly if an error occurred.

Remarks

Use this routine to set all, or a portion of the information on a particular resource. Any or all of the new info pointers can be nil. If not nil, the type and ID of the resource are changed to *resTypeP and *resIDP. Normally, the unique ID for a record is automatically created by the Data Manager when a record is created using DmNewRecord, so an application would not typically change the unique ID.

DmStrCopy

Check the validity of the chunk pointer for the record and make sure that writing the record will not exceed the chunk bounds.

```
Err DmStrCopy (
    VoidPtr recordP,
    ULong offset,
    CharPtr srcP
);
```

Parameters

recordP
Pointer to Data Record (chunk pointer).

offset
Offset within record to start writing.

srcP
Pointer to 0-terminated string.

Return Value

Returns 0 if no error or dmErrNotValidRecord or dmErrWriteOutOfBounds if an error occurred.

See Also

DmWrite, DmSet

DmWrite

Must be used to write to data manager records because the data storage area is write-protected. This routine checks the validity of the chunk pointer for the record and makes sure that the write will not exceed the chunk bounds.

```
Err DmWrite (
    VoidPtr recordP,
    ULong offset,
    VoidPtr srcP,
    ULong bytes
);
```

Parameters

recordP
Pointer to locked data record (chunk pointer).

offset
Offset within record to start writing.

srcP
Pointer to data to copy into record.

bytes
Number of bytes to write.

Return Value

Returns 0 if no error or dmErrNotValidRecord or dmErrWriteOutOfBounds if an error occurred.

See Also

DmSet

DmWriteCheck

Check the parameters of a write operation to a data storage chunk before actually performing the write.

```
Err DmWriteCheck(
    VoidPtr recordP,
    ULong offset,
    ULong bytes
);
```

Parameters

recordP
Locked pointer to recordH.

offset
Offset into record to start writing.

bytes
Number of bytes to write.

Return Value

Returns 0 if no error or dmErrNotValidRecord or dmErrWriteOutOfBounds if an error occurred.

DmMoveOpenDBContext

```
Err DmMoveOpenDBContext (
    DmOpenRef* dstHeadP,
    DmOpenRef dbR
);
```

Warning: System Use Only!

SerClearErr

Reset the serial port's line error status.

```
Err SerClearErr (
    UInt refNum
);
```

Parameters

refNum
    The serial library reference number.

Return Value

0 No error.

Remarks

Other serial manager functions, such as SerReceive, immediately return with the error code serErrLineErr if any line errors are pending. It is therefore important to check the result of serial manager function calls and call SerClearErr in acknowledgment if line error(s) occur.

SerClose

Release the serial port previously acquired by SerOpen.

```
Err SerClose (
    UInt refNum
);
```

Parameters

refNum
    Serial library reference number.

Return Value

0 No error.
serErrNotOpen The port wasn't open.
serErrStillOpen The port is still held open by someone else.

Remarks

Releases the serial port and shuts down serial port hardware if the open count has reached 0. SerClose may be called only if the return value fromSerOpen was 0 (zero) or serErrAlreadyOpen.
Open serial ports consume more energy from the device's batteries; it's therefore essential to keep a port open only as long as necessary.

See Also

SerOpen

SerGetSettings

Fill in SerSettingsType structure with current serial port attributes.

```
Err SerGetSettings (
    UInt refNum,
    SerSettingsPtr settingsP
);
```

Parameters

refNum
    Serial library reference number.

settingsP
    Pointer to SerSettingsType structure to be filled in.

Return Value

0 No error.
serErrNotOpen The port wasn't open.

Remarks

The information returned by this call includes the current baud rate, CTS timeout, handshaking options, data format options. See the definition of the SerSettingsType structure for more details.

See Also

SerSend

SerGetStatus

Return the pending line error status for errors which have been detected since the last time SerClearErr was called.

```
Word SerGetStatus (
    UInt refNum,
    BooleanPtr ctsOnP,
    BooleanPtr dsrOnP
);
```

Parameters

refNum
    The serial library reference number.

ctsOnP
    Pointer to location for storing a Boolean value.

dsrOnP
    Pointer to location for storing a Boolean value.

Return Value

Any combination of the following constants bitwise or'ed together:
serLineErrorParity Parity error.
serLineErrorHWOverrun Hardware overrun.
serLineErrorFraming Framing error.
serLineErrorBreak Break signal detected.
serLineErrorHShake Line hand-shake error.
serLineErrorSWOverrun Software overrun.

Remarks

When another serial manager function returns an error code of serErrLineErr, SerGetStatus can be used to find out the specific nature of the line error(s). The values returned via ctsOnP and dsrOnP are not meaningful in the present version of the software.

See Also

SerClearErr

SerOpen

Acquire and open a serial port with given baud rate and default settings.

```
Err SerOpen (
    UInt refNum,
    UInt port,
    ULong baud
);
```

Parameters

refNum
    Serial library reference number.

port
    Port number.

baud
    Baud rate.

Return Value

0 No error.
serErrAlreadyOpen Port was open. Enables port sharing by "friendly" clients (not recommended).
serErrBadParam Invalid parameter.
memErrNotEnoughSpace Insufficient memory.

Remarks

Acquires the serial port, powers it up, and prepares it for operation.
To obtain the serial library reference number, call SysLibFind with "Serial Library" as the library name. This reference number must be passed as a parameter to all serial manager functions. The device currently contains only one serial port with port number 0 (zero). The baud rate is an integral baud value (for example - 300, 1200, 2400, 9600, 19200, 38400, 57600, etc.). The Palm OS device has been tested at the standard baud rates in the range of 300 - 57600 baud.
Baud rates through 1 Mbit are theoretically possible. Use CTS hand-shaking at baud rates above 19200 (see SerSetSettings).
An error code of 0 (zero) or serErrAlreadyOpen indicates that the port was successfully opened. If the port is already open when SerOpen is called, the port's open count is incremented and an error code of serErrAlreadyOpen is returned. This ability to open the serial port multiple times is provided for use by cooperating tasks which need to share the serial port. Other tasks must refrain
from using the port if serErrAlreadyOpen is returned and close it by calling SerClose.

See Also

SerClose

SerReceive

Receive a stream of bytes.

```
Err SerReceive (
    UInt refNum,
    VoidPtr bufP,
    ULong bytes,
    Long timeout
);
```

Parameters

refNum
    The serial library reference number.

bufP
    Pointer to the buffer for receiving data.

bytes
    Number of bytes desired.

timeout
    Interbyte time out in system ticks (-1 = forever)

Return Value

0 No error. Requested number of bytes was received.
serErrTimeOut Interbyte time out exceeded while waiting for the next byte to arrive.
serErrLineErr Line error occurred (see SerClearErr and SerGetStatus).

Remarks

SerReceive blocks until all the requested data has been received or an error occurs. Because this call returns immediately without any data if line errors are pending, it is important to acknowledge the detection of line errors by calling SerClearErr. If you just need to retrieve all or some of the bytes which are already in the receive queue, call SerReceiveCheck first to get the count of bytes presently in the receive queue.

SerReceiveCheck

Return the count of bytes presently in the receive queue.

```
Err SerReceiveCheck(
    UInt refNum,
    ULongPtr numBytesP
);
```

Parameters

refNum
    Serial library reference number.

numBytesP
    Pointer to location for returning the byte count.

Return Value

0 No error.
serErrLineErr Line error pending (see SerClearErr and SerGetStatus).

Remarks

Because this call does not return the byte count if line errors are pending, it is important to acknowledge the detection of line errors
by calling SerClearErr.

See Also

SerReceiveWait

SerReceiveFlush

Discard all data presently in the receive queue and flush bytes coming into the serial port. Clear the saved error status.

```
void SerReceiveFlush (
    UInt refNum,
    Long timeout
);
```

Parameters

refNum
    Serial library reference number.

timeout
    Interbyte time out in system ticks (-1 = forever).

Return Value

Returns nothing.

Remarks

SerReceiveFlush blocks until a time out occurs while waiting for the next byte to arrive.

SerReceiveWait

Wait for at least bytes bytes of data to accumulate in the receive queue.

```
Err SerReceiveWait (
    UInt refNum,
    ULong bytes,
    Long timeout
);
```

Parameters

refNum
    Serial library reference number.

bytes
    Number of bytes desired.

timeout
    Interbyte time out in system ticks (-1 = forever).

Return Value

0 No error.
serErrTimeOut Interbyte time out exceeded while waiting for next byte to arrive.
serErrLineErr Line error occurred (see SerClearErrr and SerGetStatus).

Remarks

This is the preferred method of waiting for serial input, since it blocks the current task and allows switching the processor into a more energy-efficient state.
SerReceiveWait blocks until the desired number of bytes accumulate in the receive queue or an error occurs. The desired number
of bytes must be less than the current receive queue size. The default queue size is 512 bytes. Because this call returns immediately if line errors are pending, it is important to acknowledge the detection of line errors by calling SerClearErr.

See Also

SerReceiveCheck, SerSetReceiveBuffer

SerSend

Send a stream of bytes to the serial port.

```
Err SerSend (
    UInt refNum,
    VoidPtr bufP,
    ULong size
);
```

Parameters

refNum
    The serial library reference number.

bufP
    Pointer to the data to send.

size
    Size (in number of bytes) of the data to send.

Return Value

0 No error.
serErrTimeOut Handshake time out (such as waiting for CTS to become asserted.)

Remarks

In the present implementation, SerSend blocks until all data is transferred to the UART or a time out error (if CTS handshaking is enabled) occurs. Future implementations may queue up the request and return immediately, performing transmission in the background.
If your software needs to detect when all data has been transmitted, see SerSendWait.
This routine observes the current CTS time out setting if CTS hand-shaking is enabled (see SerGetSettings and SerSend).

SerSendWait

Wait until the serial transmit buffer empties.

```
Err SerSendWait (
    UInt refNum,
    Long timeout
);
```

Parameters

refNum
    The serial library reference number.

timeout
    Reserved for future enhancements.

Set to (-1) for compatibility.

Return Value

0 No error.
serErrTimeOut Handshake time out (such as waiting for CTS to become asserted).

Remarks

SerSendWait blocks until all data is transferred or a time-out error (if CTS handshaking is enabled) occurs. This routine observes the current CTS timeout setting if CTS handshaking is enabled (see SerGetSettings and SerSend).

SerSetReceiveBuffer

Replace the default receive queue. To restore the original buffer, pass bufSize = 0.

```
Err SerSetReceiveBuffer(
    UInt refNum,
    VoidPtr bufP,
    UInt bufSize
);
```

Parameters

refNum
    Serial library reference number.

bufP
    Pointer to buffer to be used as the new receive queue.

bufSize
    Size of buffer, or 0 to restore the default receive queue.

Return Value

Returns 0 if successful.

Remarks

The specified buffer needs to contain 32 extra bytes for serial manager overhead (its size should be your application's requirement
plus 32 bytes). The default receive queue must be restored before the serial port is closed. To restore the default receive queue, call
SerSetReceiveBuffer passing 0 (zero) for the buffer size. The serial manager does not free the custom receive queue.

SerSetSettings

Set the serial port settings; that is, change its attributes.

```
Err SerSetSettings (
    UInt refNum,
    SerSettingsPtr settingsP
);
```

Parameters

refNum
    Serial library reference number.

settingsP
    Pointer to the filled in SerSettingsType structure.

Return Value

0 No error.
serErrNotOpen The port wasn't open.
serErrBadParam Invalid parameter.

Remarks

The attributes set by this call include the current baud rate, CTS time out, handshaking options, and data format options. See the definition of the SerSettingsType structure for more details.

See Also

SerGetSettings

SerSleep

```
Err SerSleep (
    UInt refNum
);
```

WARNING: This function for use by system software only.

SerWake

```
Err SerWake (
    UInt refNum
);
```

WARNING: This function for use by system software only.

SerReceiveISP

```
Boolean SerReceiveISP (
    void
);
```

WARNING: This function for use by system software only.

SlkClose

Close down the serial link manager.

```
Err SlkClose (
    void
);
```

Parameters

None.

Return Value

0 No error.
slkErrNotOpen The serial link manager was not open.

Remarks

When the open count reaches zero, this routine frees resources allocated by serial link manager.

SlkCloseSocket

Closes a socket previously opened with SlkOpenSocket.
WARNING: The caller is responsible for closing the communications library used by this socket, if necessary.

```
Err SlkCloseSocket (
    UInt socket
);
```

Parameters

socket
    The socket ID to close.

Return Value

0 No error.
slkErrSocketNotOpen The socket was not open.

Remarks

SlkCloseSocket frees system resources the serial link manager allocated for the socket. It does not free resources allocated and passed by the client, such as the buffers passed to SlkSetSocketListener; this is the client's responsibility. The caller is also responsible for closing the communications library used by this socket.

See Also

SlkOpenSocket, SlkSocketRefNum

SlkFlushSocket

Flush the receive queue of the communications library associated with the given socket.

```
Err SlkFlushSocket (
    UInt socket,
    Long timeout
);
```

Parameters

socket
    Socket ID.

timeout
    Interbyte time out in system ticks.

Return Value

0 No error.
slkErrSocketNotOpen The socket was not open.

SlkOpen

Initialize the serial link manager.

```
Err SlkOpen (
    void
);
```

Parameters

None.

Return Value

0 No error.
slkErrAlreadyOpen No error.

Remarks

Initializes the serial link manager, allocating necessary resources.
Return codes of 0 (zero) and slkErrAlreadyOpen both indicate success. Any other return code indicates failure.
slkErrAlreadyOpen informs the client that someone else is also using the serial link manager. If the serial link manager was successfully opened by the client, the client needs to call SlkClose when it finishes using the serial link manager.

SlkOpenSocket

Open a serial link socket and associate it with a communications library.
The socket may be a known static socket or a dynamically assigned socket.

```
Err SlkOpenSocket (
    UInt libRefNum,
    UIntPtr socketP,
    Boolean staticSocket
);
```

Parameters

libRefNum
    Communications library reference number for socket.

socketP
    Pointer to location for returning the socket ID.

staticSocket
    If true, *socketP contains the desired static socket number to open. If false, any free socket number is assigned dynamically and opened.

Return Value

0 No error.
slkErrOutOfSockets No more sockets can be opened.

Remarks

The communications library must already be initialized and opened (see SerOpen). When finished using the socket, the caller must call SlkCloseSocket to free system resources allocated for the socket.
For information about well-known static socket ID's, see The Serial Link Protocol.

SlkReceivePacket

Receive and validate a packet for a particular socket or for any socket. Check for format and checksum errors.

```
Err SlkReceivePacket(
    UInt socket,
    Boolean andOtherSockets,
    SlkPktHeaderPtr headerP,
    void* bodyP,
    UInt bodySize,
    Long timeout
);
```

Parameters

socket
    The socket ID.

andOtherSockets
    If true, ignore actual dest in packet header.

headerP
    Pointer to the packet header buffer (size of SlkPktHeaderType).

bodyP
    Pointer to the packet client data buffer.

bodySize
    Size of the client data buffer (maximum client data size which may be accommodated).

timeout
    Maximum number of system ticks to wait for beginning of a packet (-1) means wait forever.

Return Value

0 No error.
slkErrSocketNotOpen The socket was not open.
slkErrTimeOut Timed out waiting for a packet.
slkErrWrongDestSocket The packet being received had an unexpected destination.
slkErrChecksum Invalid header checksum or packet CRC-16.
slkErrBuffer Client data buffer was too small for packet's client data.
If andOtherSockets is FALSE, this routine returns with an error code unless it gets a packet for the specific socket.
If andOtherSockets is TRUE, this routine returns successfully if it sees any incoming packet from the communications library used by socket.

Remarks

You may request to receive a packet for the passed socket ID only, or for any open socket which does not have a socket listener. The parameters also specify buffers for the packet header and client data, and a timeout. The time out indicates how long the receiver should wait for a packet to begin arriving before timing out. If a packet is received for a socket with a registered socket listener, it will be dispatched via its socket listener procedure. On success, the packet header buffer and packet client data buffer is filled in with the actual size of the packet's client data in the packet header's bodySize field.

SlkSendPacket

Send a serial link packet via the serial output driver.

```
Err SlkSendPacket(
    SlkPktHeaderPtr headerP,
    SlkWriteDataPtr writeList
);
```

Parameters

headerP
    Pointer to the packet header structure with client information filled in (see remarks).

writeList
    List of packet client data blocks (see remarks).

Return Value

0 No error.
slkErrSocketNotOpen The socket was not open.
slkErrTimeOut Handshake time out.

Remarks

SlkSendPacket stuffs the signature, client data size, and the checksum fields of the packet header. The caller must fill in all other packet header fields. If the transaction ID field is set to 0 (zero), the serial link manager automatically generates and stuffs a new non-zero transaction ID. The array of SlkWriteDataType structures enables the caller to specify the client data part of the packet as a list of non-contiguous blocks. The end of list is indicated by an array element with the size field set to 0 (zero). This call blocks until the entire packet is sent out or until an error occurs.

SlkSetSocketListener

Register a socket listener for a particular socket.

```
Err SlkSetSocketListener (
    UInt socket,
    SlkSocketListenPtr socketP
);
```

Parameters

socket
    Socket ID.

socketP
    Pointer to a SlkSocketListenType structure.

Return Value

0 No error.
slkErrBadParam Invalid parameter.
slkErrSocketNotOpen The socket was not open.

Remarks

Called by applications to set up a socket listener.
Since the serial link manager does not make a copy of the SlkSocketListenType structure, but instead saves the passed pointer to it, the structure may not be an automatic variable (that is, local variable allocated on the stack). The SlkSocketListenType structure may be a global variable in an application or a locked chunk allocated from the dynamic heap. The SlkSocketListenType structure specifies pointers to the socket listener procedure and the data buffers for dispatching packets destined for this socket. Pointers to two buffers must be specified: the packet header buffer (size of SlkPktHeaderType), and the packet body (client data) buffer. The packet body buffer must be large enough for the largest expected client data size. Both buffers may be application global variables or locked chunks allocated from the dynamic heap. The socket listener procedure is called when a valid packet is received for the socket. Pointers to the packet header buffer and the packet body buffer are passed as parameters to the socket listener procedure.
Note: The application is responsible for freeing the SlkSocketListenType structure or the allocated buffers when the socket is closed. The serial link manager doesn't do it.

SlkSocketRefNum

Get the reference number of the communications library associated with a particular socket.

```
Err SlkSocketRefNum (
    UInt socket,
    UIntPtr refNumP
);
```

Parameters

socket
    The socket ID.

refNumP
    Pointer to location for returning the communications library reference number.

Return Value

0 No error.
slkErrSocketNotOpen The socket was not open.

SlkSocketSetTimeout

Set the interbyte packet receive time out for a particular socket.

```
Err SlkSocketSetTimeout (
    UInt socket,
    Long timeout
);
```

Parameters

socket
    Socket ID.

timeout
    Interbyte packet receive time out in system ticks.

Return Value

0 No error.
slkErrSocketNotOpen The socket was not open.

SlkSysPktDefaultResponse

```
Err SlkSysPktDefaultResponse (
    SlkPktHeaderPtr headerP,
    void* bodyP
);
```

WARNING: This function for use by system software only.

SlkProcessRPC

```
Err SlkProcessRPC (
    SlkPktHeaderPtr headerP,
    void* bodyP
);
```

WARNING: This function for use by system software only.

PsrClose

Close the PAD server.

```
Err PsrClose(
    void
);
```

Parameters

None.

Return Value

0 No error.

Remarks

This routine frees resources allocated by the PAD server. It should be called when the PAD server client is finished using PAD server and only if the call to PsrInit was successful.
The routine must be called by the client when finished with the session if the call to PsrInit was successful.

PsrGetCommand

Receive a command.

```
Err PsrGetCommand(
    DmOpenRef refDBP,
    VoidPtr* cmdPP,
    VoidHand* cmdBufHP,
    WordPtr rcvdCmdLenP,
    BytePtr tidP,
    BytePtr remoteSocketP
);
```

Parameters

refDBP
    Database reference for allocating a command buffer, or 0 (zero) for none.

cmdPP
    Pointer to location for storing a pointer to the internal command buffer.

cmdBufHP
    Pointer to location for storing a handle of the command buffer allocated from a data storage heap.

rcvdCmdLenP
    Pointer to location for storing the size (in number of bytes) of the received command.

tidP
    Pointer to location for storing the transaction ID of the command.

remoteSocketP
    Pointer to location for storing the remote socket ID (the source socket).

Return Value

0 No error.
psrErrUserCan Cancelled by user (Cancel callback returned non-zero).
psrErrParam Invalid parameter.
psrErrBlockFormat Invalid command data format detected (severe protocol error).
psrErrTimeOut Timed out waiting for command.

Remarks

PsrGetCommand blocks until a command is received, a time-out error occurs, or the Cancel callback (see PsrInit) returns non-zero. On success, the command is in the buffer, referenced either by *cm-dPP or by *cmdBufHP. In the first case (cmdPP), the command will be in a Pad Server internal buffer in the dynamic heap. This buffer must be treated as read-only. In the second case (cmdBufHP), the internal buffer was not big enough to contain the entire command (such as when writing a large record), and a data heap chunk was allocated by PAD server via DmNewHandle (provided that a valid refDBP was specified). The caller inherits ownership of this chunk and is responsible for freeing it if it is not needed (it can be resized, attached to a database, deleted, etc.).

PsrInit

Initialize the PAD server.

```
Err PsrInit (
    Byte serverSocket,
    PsrUserCanProcPtr canProcP,
    DWord userRef,
    Int cmdWaitSec
);
```

Parameters

serverSocket
    Socket ID of an open Serial Link socket.

canProcP
    Pointer to the Cancel callback procedure or 0 (zero) if none.

userRef
    Any DWord(32-bit) parameter to be passed to the Cancel callback procedure.

cmdWaitSec
    Number of seconds to wait for command; 0 = default; -1 = forever.

Return Value

0 No error.
psrErrInUse PAD server is in use.
psrErrMemory Insufficient memory to initialize PAD server.

Remarks

This routine initializes the PAD server, allocating any necessary resources.
Return code of 0 (zero) indicates success; any other return code indicates failure. If the PAD server was successfully opened by the client, the client needs to call PsrClose when it has finished using the PAD server. If specified, the cancel callback procedure is called periodically. If the cancel callback procedure returns non-zero, the current PAD server request aborts and returns immediately with an error code of psrErrUserCan.

PsrSendReply

Send a response to the workstation.

```
Err PsrSendReply (
    Byte remoteSocket,
    Byte refTID,
    PmSegmentPtr segP,
    Int segCount
);
```

Parameters

remoteSocket
    Remote socket ID.

refTID
    Transaction ID of the response (should be same as that returned by the matching PsrGetCommand call).

segP
    Pointer to array of response data segments.

segCount
    Number of reply data segments in the array.

Return Value

0 No error.
psrErrParam Invalid ID parameter(s).
psrErrSizeErr Sum of the response data segments exceeded PADP block size limit.
psrErrTooManyRetries Maximum retry count was exceeded but acknowledgment wasn't received. (connection is presumed lost).
psrErrTimeOut Transmission handshake time out (connection is presumed lost).
psrErrUserCan Cancelled by user (cancel callback returned non-zero).

Remarks

PsrSendReply blocks until the entire response data block is successfully delivered to the workstation, lost connection is detected, or the cancel callback (see PsrInit) returns non-zero. For convenience, the response data block is specified as a list of data segments via an array of PmSegmentType structures. The PmSegmentType structure allows selective specification of word alignment for each data segment. Any bytes inserted as the result of word alignment are set to 0 (zero) in the resulting response block.

Crc16CalcBlock

Calculate the 16-bit CRC of a data block using the table lookup method.

```
Word Crc16CalcBlock (
    VoidPtr bufP,
    UInt count,
    Word crc
);
```

Parameters

bufP
    Pointer to the data buffer.

count
    Number of bytes in the buffer.

crc
    Seed crc value.

Return Value

A 16-bit CRC for the data buffer.