FilterMeister

the graphics plug-in creator

Command Reference

FilterMeister Command Reference

Updated December 31, 2021
For FilterMeister 1.0.9h

Written by the many authors of the FilterMeister Wiki community project: Alex Hunter, Martijn Van Der Lee, Harry Heim, Kohan Ikin, Guessous Mehdi, J Duncan Suss, Scriveyn, Martin Koch, Eric von Bayer, Florian, Bob Oat, Pete, byRo, Ron C, Roberto Muscia and many more.

eBook versions formatted by Kohan Ikin

www.filtermeister.com

FilterMeister 1.0 Manual

Command Reference

Handlers

A *handler* is a special-purpose function which is invoked whenever a specified event or condition occurs. The parameters for a handler are passed implicitly via global variables. The value returned by a handler may vary, but is often a Boolean flag indicating whether or not the handler successfully completed its mission.

OnInitDialog

is an internal handler, not currently exposed to the Filter Designer. It is invoked when the plug-in is first loaded, and is responsible for initializing global variables, creating the initial filter dialog, and establishing the initial set of controls in the filter dialog.

OnCtl

is invoked whenever a control is activated which might require some response, such as initiating the image filtration process. This handler is also a catch-all for various other events.

OnFilterStart

is invoked whenever it is time to begin processing (or "filtering") the current image to produce the preview display or the final result image.

ForEveryTile

is invoked once for every tile in the image during filter processing (if **OnFilterStart** returns `false`).

ForEveryRow

is invoked once for every row in the current tile, if **ForEveryTile** returns `false`.

ForEveryPixel
is invoked once for every pixel in the current row, if **ForEveryRow** returns `false`.

R, G, B, A
Each of these handlers (R, G, B, or A) is invoked once for each pixel, if **ForEveryPixel** returns `false` for the pixel in question. There may be multiple instances of each R, G, B, or A handler, in which case`they will be called in sequence for each designated pixel.

OnFilterEnd
is called after the last pixel of the last row of the last tile in the current image has been processed, and is responsible for performing any post-filtering clean-up actions.

OnDivideCheck
is an internal handler not exposed to the Filter Designer. It is invoked immediately before each integer division instruction in the filter code to guard against the possibility of generating an integer division fault.

OnZeroDivide
is invoked whenever an integer division (or remainder) operation is about to attempt to divide by 0. The default OnZeroDivide handler forces the result of the operation to 0, for compatibility with Filter Factory.

OnDivideOverflow
is invoked whenever an integer division (or remainder) operation is about to produce a divide overflow error. The default OnDivideOverflow handler returns a result of 0x80000000 (*indefinite-Nan*) for the division or remainder operation.

onCtl

OnCtl() is one of the most useful handlers which doesn't do anything to the image pixels. It allows you to trap any event involving a user control and take appropriate action. It's commonly used to change the controls themselves, making for a dynamic user interface which adjusts itself depending on actions taken by the filter's user.

The basic format is:

```
OnCtl(n): {

    if (n == ?? && e == ?? ) {
        // ***  code goes here  ***
        // ***   to do things   ***
    }

    return false;

}  // end of OnCtl()
```

Where...

-- n specifies the number of the control which was acted upon to generate the event that sent the program into OnCtl(); i.e., if something happened with ctl[4] then n will be 4.

-- e is a global system variable which describes the type of event; the following FilterMeister Events (FME's) with their numerical values are recognized by the system:

- FME_UNKNOWN = 0
- FME_ZERODIVIDE = 1

- FME_DIVIDEOVERFLOW = 2
- FME_CLICKED = 3
- FME_DBLCLK = 4
- FME_PAGEUP = 5
- FME_PAGEDOWN = 6
- FME_LINEUP = 7
- FME_LINEDOWN = 8
- FME_MOUSEOVER = 9
- FME_MOUSEOUT = 10
- FME_MOUSEMOVE = 11
- FME_LEFTCLICKED_DOWN = 12
- FME_LEFTCLICKED_UP = 13
- FME_RIGHTCLICKED_DOWN = 14
- FME_RIGHTCLICKED_UP = 15
- FME_TIMER = 16
- FME_ZOOMCHANGED = 17 (not yet implemented)
- FME_KEYDOWN = 18
- FME_KEYUP = 19
- FME_VALUECHANGED = 20
- FME_SIZE = 21
- FME_ENTERSIZE = 22
- FME_EXITSIZE = 23
- FME_DRAWITEM = 24
- FME_CUSTOMEVENT = 25
- FME_COMBO_DROPDOWN = 26
- FME_COMBO_CLOSEUP = 27
- FME_CONTEXTMENU = 28
- FME_SETEDITFOCUS = 29
- FME_KILLEDITFOCUS = 30
- FME_PREVIEWDRAG = 31

* Proper coding style is to use the FME numerical values *only for debugging and testing*; in everyday code the symbolic constants should be used so the code is easier to read.

* While almost all of these are user actions, you can see that OnCtl() is also where you can trap for divide-by-zero and overflow problems. *(Wrong -- use OnZeroDivide or OnDivideOverflow instead. -Alex)*

* It's important to specify a value to test e against so you really do the correct thing(s) rather than make possibly invalid assumptions about what is happening; you can use logical operators to simultaneously test for several events which might be occurring with a given control.

Return Value

Use false as the return value if you want to process the image or preview as a result of the changes made in onCtl(). Otherwise return true. *(Not yet implemented; the return value, although required, is ignored for now.)*

Example:

```
OnCtl(n): {
  if (n == CTL_HELP ) {
    // ... open helpfile
    return true;
  else if (n == CTL_DO_FUNKY_STUFF){
    /* ... */
  else if (n == CTL_DO_MORE_FUNKY_STUFF){
    /* ... */
  }
return false;
}
```

Other notes

* Because the code in OnCtl() is executed when something is done with one of the controls, it appears in the program after the

controls have been defined. *(Not necessarily true. -Alex)*

* Variables can be defined which are local to the handler; these typically go right after the first line. Follow C conventions. Use the pre-declared global variables (or user-defined global variables, once they are implemented) for values needed outside OnCtl() or for values which are saved across filter invocations within a single host session. (See FMML Yahoo Group message #256 for these.)

OnFilterStart

Syntax

`OnFilterStart: <compound-statement></compound-statement>`

Return

`false` if filter processing is to continue with the `ForEveryTile` handler, or `true` to abort further filter processing (though this latter option is currently ignored in FM 0.4.20).

Description

The OnFilterStart handler is invoked at the start of each filter run, just before the `ForEveryTile`, `ForEveryRow`, `ForEveryPixel`, and `R, G, B, A` handlers are called. The OnFilterStart handler is a good place to perform one-time calculations at the start of a filter run, check for valid control inputs, valid image mode, etc.

Comment

The default OnFilterStart handler simply returns `false`, passing control to the `ForEveryTile` handler for every tile in the image or preview.

Example

You can test the current image mode in the OnFilterStart handler, and abort the invocation if the image mode is inappropriate. For example:

```
OnFilterStart:{
    if (imageMode != RGBMode) {
        ErrorOk("Not an RGB image.");
        doAction(CA_CANCEL); /* for now, until return true
works correctly */
        return true; /* abort filter processing */
    }
    return false; /* continue with ForEveryTile */
}
```

Note that we added a call to `doAction` to explicitly cancel the filter run, since return `true` does not yet correctly abort the filter run. Once this is fixed, the call to `doAction` can be removed.

See Also

handlers, **ForEveryTile**, **ForEveryRow**, **ForEveryPixel**, **OnFilterEnd**

ForEveryPixel

Syntax

`ForEveryPixel:`

Return

`true` if processing is complete for this pixel, else `false` if the `R, G, B, A` handlers should be called next for individual processing of each channel for this pixel.

Description

If the `ForEveryRow` handler returns `false`, then this handler will be called for every pixel in the current row, from left to right.

Default handler

The default `ForEveryPixel` handler simply returns `false`, passing control to the `R, G, B, A` handlers for every pixel.

Example

```
ForEveryPixel: {
    return true;   /* processsing complete for this pixel
*/
}
```

See Also

handlers, **ForEveryTile**, **ForEveryRow**

ForEveryRow

(not yet implemented)

Syntax

`ForEveryRow:`

Return

`true` if processing is complete for this row, else `false` if the `ForEveryPixel` handler should be called next for individual processing of each pixel in this row.

Description

If the `ForEveryTile` handler returns `false`. then this handler will be called for every row in the current tile, from top to bottom. This handler is a good place to call the `testAbort` or `updateProgress` function.

Default handler

The default `ForEveryRow` handler simply calls `updateProgress` to update the progress bar and test for an abort condition, and then returns `false`, passing control to the `ForEveryPixel` handler for every pixel in this row.

Example

```
ForEveryRow: {
    updateProgress(row, rows);    /* update progress bar
*/
```

```
    return true;    /* processsing complete for this row */
}
```

See Also

ForEveryTile, **ForEveryPixel**, **handlers**

ForEveryTile

Syntax

`ForEveryTile:`

Return

`true` if processing is complete for this tile, else `false` if the `ForEveryRow` handler should be called next for individual processing of each row in this tile.

Description

This handler is called for every tile in an image from left to right, then top to bottom.

Default handler

The default `ForEveryTile` handler simply returns `false`, passing control to the `ForEveryRow` handler for every row in this tile.

Example

```
ForEveryTile: {
    return true;   /* processsing complete for this tile
*/
}
```

See Also

ForEveryRow, ForEveryPixel, handlers

OnFilterEnd

Syntax

`OnFilterEnd: <compound-statement></compound-statement>`

Return

`false` if the default OnFilterEnd handler code is to be invoked, or `true` to prevent the default handler from being invoked (though this latter option is currently ignored in FM 0.4.20).

Description

The OnFilterEnd handler is invoked at the end of each filter run, after all calls to the `ForEveryTile`, `ForEveryRow`, `ForEveryPixel`, and `R, G, B, A` handlers. The OnFilterEnd handler is a good place to perform any necessary cleanup at the end of a filter run, such as resetting the progress bar indicator.

Comment

The default OnFilterEnd handler calls `playSoundWave(NULL)` to terminate any wave file that may still be playing.

Example

You can use the OnFilterEnd handler to reset the progress bar at the end of a filter run.

```
OnFilterEnd: {

    /* reset the progress bar to 0% */
    updateProgress(0, 100);

    /* allow the default OnFilterEnd handler to run also
*/
    return false;
}
```

See Also

handlers, OnFilterStart, ForEveryTile, ForEveryRow, ForEveryPixel

Structure

Page should describe the generic structure of a FM program. Are these sections named in any way?

```
// Section 1: language specifier
%ffp

// Section 2: identification & control definition
title: "Hello world"

ctl[0]: STANDARD

// Section 3: handlers
OnFilterStart:
{
    Info("Hello world");
}
```

I believe this order of sections is mandatory, so we'd better document it :)

Language Keywords

Flow control

break - Stops execution of the innermost looping or conditional block of code and continue processing from the statement immediately following this block.

case - Used in conjunction with the `switch` statement to specify a possible value to match against the variable specified in the `switch` statement.

continue - Stops execution of the innermost looping block of code, iterate in case of a `for` loop and continue the loop from the first statement in the looping block of code.

default - Used in conjunction with the `switch` statement to specify an alternative if none of the `case` statements match.

do - Used in conjunction with the `while` statement when testing after execution of the controlled block of code.

else - Used in conjunction with the `if` statement to execute a block of code when the Boolean expression given in the `if` statement does not evaluate to `true`.

for - Executes a block of code based on the value of a specified variable and its continual alteration at the end of each iteration.

if - Conditionally execute a block of code when a Boolean expression evaluates to `true`.

return - Return from a handler, optionally with a value to be passed along to the system.

sizeof - Returns the size in bytes of a specific type or variable.

switch - Executes any of a number of blocks of code or a consecutive number of block of code depending on whether the variable specified in the `switch` statement matches the value of a `case` statement.

while - Repeat a specific block of code as long as a specific Boolean expression evaluates to `true`.

Constants

false - Designates a state in which a value is considered logically false.

true - Designates a state in which a value is considered logically true.

NULL - Designates a pointer which points to nothing.

Type declaration

bool

char

const

double

dword

float

int

long

short

signed

void

word

switch example

```
switch (ctl(0))
{
  case 0:
    ... Your code
    break;

  case 1:
    ... Your code
    break;

  case 2:
    ... Your code
    break;

  case 3:
    ... Your code
    break;

  default:
    ... Your code
    break;
}
```

Dialog Functions

Built-in Controls

Particular FM controls have predefined indices, indicated by the constants below:

CTL_BACKGROUND

CTL_CANCEL

CTL_EDIT

CTL_FRAME

CTL_HOST

CTL_LOGO

CTL_OK

CTL_PREVIEW

CTL_PROGRESS

CTL_ZOOM

Notice: `CTL_USER_LAST` corresponds to the last available user control index. Check this value to ensure the index you use is not already reserved by a particular FM control.

User Control Properties

BITMAP - Creates a bitmap image

CHECKBOX - Creates a checkbox

COMBOBOX - Creates a pull-down list/combo box

EDIT - Creates a text editbox

FRAME - Creates a frame/border

GROUPBOX - Creates a groupbox for grouping radio buttons

ICON - Creates an icon image

IMAGE - Creates a transparent bitmap image

LISTBOX - Creates a listbox

METAFILE - Creates a vector image

MODIFY - Changes the properties of a predefined control

NONE - Removes a predefined user control

OWNERDRAW - Creates an actionable colored/transparent rectangle control

PUSHBUTTON - Creates a standard button

RADIOBUTTON - Creates a single radio button

RECT - Creates a rectangle

SCROLLBAR - Creates a scrollbar on its own

SLIDER - Creates a trackbar with a text label and numeric edit control

STANDARD - Creates a scrollbar with a text label and numeric edit control

STATICTEXT - Creates a text label

TAB - Creates a tab control for grouping controls into tab panels

TRACKBAR - Creates a trackbar (a kind of scrollbar with ticks)

User Control Functions

checkCtlFocus - Checks if a control currently has user focus.

clearCtlBuddyStyle - Clears the window style of labels and edit boxes on standard and slider controls.

clearCtlProperties - Clears a previously set property from a control.

createCtl - Creates a new user interface control at runtime.

createFont - Creates a font object to use with a user control.

ctlEnabled - Determines if a control is enabled.

deleteCtl - Deletes a user control.

deleteFont - Deletes a font object.

deleteCtlItem - Deletes an item from a Listbox, Combobox or Tab control.

deleteCtlItems - Deletes all items from a Listbox, Combobox or Tab control.

doAction - Performs one of several predefined actions.

enableCtl - Enables, disables & hides user controls.

getCtlClass - Returns the type of a specific user control.

getCtlColor - Returns the current color of a control.

getCtlCoord - Get the mouse coordinates of any control.

getCtlItemCount - Get the number of items in a Listbox, Combobox or Tab control

getCtlItemText - Get the text of an item in a Listbox, Combobox or Tab control.

getCtlPos - Get the current position and size of any control.

getCtlRange - Gets the minimum or maximum of the range of a user control.

getCtlTab - Gets the tab control or tab sheet that a user control has been assigned to.

getCtlText - Get the text associated with a user control.

getCtlVal - Returns the current value of a user control.

getPreviewCoordX - Returns the x coordinate in the image above which the mouse pointer is placed.

getPreviewCoordY - Returns the y coordinate in the image above which the mouse pointer is placed.

getPreviewCursor - Returns the resource number of the current preview cursor

mouseOverWhenInvisible - Enables & disables event triggering for disabled or invisible controls.

refreshCtl - Redraws or updates a certain control.

resetAllCtls - Resets the entire plugin UI to the default state when starting FilterMeister.

scrollPreview - Scrolls the preview window in a given position inside the full image.

setClickDrag - Changes how the user can move the image in the preview window.

setCtlAction - Sets the action performed when a user control is activated.

setCtlAnchor - Changes how a control is repositioned when the dialog is resized.

setCtlBuddyStyle - Sets the window style of labels and edit boxes on standard and slider controls.

setCtlColor - Changes the background color of the user control.

setCtlDivisor - Sets the number of decimal places a control appears to have.

setCtlEditSize - Changes the size of the text edit control of a slider.

setCtlFocus - Sets which control has the user focus.

setCtlFont - Sets the font of a user control.

setCtlFontColor - Changes the color of the user control's text.

setCtlFontSysColor - Changes the color of the user control's text to a defined system color.

setCtlGamma - Changes the curve of how quickly the slider changes values.

setCtlImage - Changes the image displayed by a control at runtime.

setCtlLineSize - Changes the control's line jump unit.

setCtlPageSize - Changes the control's page jump unit.

setCtlPixelPos - Resizes and repositions a control on the dialog interface (in pixels).

setCtlPos - Resizes and repositions a control on the dialog interface (in DBUs).

setCtlProperties - Sets the properties of a control.

setCtlRange - Changes the range of values a user control can return.

setCtlScripting - Enables & disables Photoshop scripting for a user control.

setCtlSysColor - Changes the background color of the user control to a defined system color.

setCtlTab - Assigns a control to the sheet of a tab control.

setCtlText - Sets the text property of the user control.

setCtlTextv - Sets the text property of the user control using printf-style formatting.

setCtlTheme - Switches XP/Vista themes on and off for a specific control. (Does not work correctly at the moment.)

setCtlTicFreq - Sets the frequency of tick marks in a TRACKBAR control.

setCtlToolTip - Sets the text that appears when the mouse hovers over the control.

setCtlVal - Sets the value associated with the control.

setPreviewCursor - Changes the cursor displayed when the mouse is over the preview window.

setZoom - Sets the zoom level of the filter preview image.

testAbort - Tests for a user-requested abort.

updateAnchors - Updates the anchors for all user controls.

updatePreview - Updates the filter preview image.

updateProgress - Updates the filter progress bar.

Message & Dialog Windows

chooseColor - Invokes the host application's default color picker dialog

Error - Displays an error box dialog with Cancel, Retry and Ignore buttons

ErrorOk - Displays an error box with an OK button

getOpenFileName - Displays the standard Open File dialog

getSaveFileName - Displays the standard Save File dialog

Info - Displays an information box dialog

msgBox - Displays a custom configured message box

Warn - Displays a warning box dialog

YesNo - Displays a message box containing Yes and No buttons

YesNoCancel - Displays a message box containing Yes, No and Cancel buttons

Dialog String Manipulation Functions

appendEllipsis - Returns a string with an ellipsis ("...") appended.

formatString - Returns a string with substitutions made for designated 2-character substrings

stripEllipsis - Returns a string with any trailing ellipsis removed.

Miscellaneous Dialog Functions

getAsyncKeyState - Determines whether a key is pressed or not.

getDisplaySettings - Finds the bit depth, resolution and refresh rate of the screen.

getSysColor - Gets the color of a specified display element.

HDBUsToPixels - Converts horizontal dialog box units to pixels.

playSoundWave - Plays a sound file.

playSoundWaveLoop - Plays a sound file repeatedly.

playSoundWaveSync - Plays a sound and waits until it finishes playing.

triggerEvent - Sets off a dialog, timer or internal event.

VDBUsToPixels - Converts vertical dialog box units to pixels.

Control drawing functions

Starting/stopping

You can draw in a number of different modes.

In the default mode, you draw directly to the control on screen.

startSetPixel - Start drawing on an **OWNERDRAW**

endSetPixel - Stop drawing

Meta

getSetPixelHeight - Returns the height of the control canvas or buffer in pixels. Only works between a startSetPixel* and endSetPixel* pair.

getSetPixelWidth - Returns the width of the control canvas or buffer in pixels. Only works between a startSetPixel* and endSetPixel* pair.

Pixels

getPixel - Read single pixel

setPixel - Set single pixel

Text

setFont - lets you choose the font name, its size, angle, boldness, and italics

setText - lets you draw a certain text with the chosen font, color and alignment

setTextv - Identical to **setText** but with support for formatted text.

Bitmaps

setBitmap - Draw a specified bitmap on the current ownerdraw/buffer canvas at the specified coordinates.

setBitmapTransparent - Draw a bitmap on the current ownerdraw/buffer canvas with a transparent color.

setBitmapStretch - Draw a stretched or shrunk bitmap on the current ownerdraw/buffer canvas.

setBitmapTile - Draw part of the specified bitmap on the current ownerdraw/buffer canvas at the specified coordinates.

setBitmapStretchTransparent - Draw a stretched or shrunk bitmap on the current ownerdraw/buffer canvas with a transparent color.

Shapes

setAngleArc - Draws the outlines of a pie segment of a circle.

setAngleArcFill - Draws a filled pie segment of a circle.

setEllipse - draws an ellipse

setEllipseFill - draws a filled ellipse

setFill - fills the whole ownerdraw control with a certain color

setLine - draws a line

setPenWidth - lets you change the broadness of drawn lines.

setRectangle - draws a rectangle

setRectFrame - draws a rectangle

setRectFill - draws a filled rectangle

setRectGradient - draws a gradient into the rectangle

Anti-aliased shapes

If you only need these shapes or non-anti-aliased shapes, these functions will be both faster and better quality than using SuperSampling mode.

setThinLineAA - Draw anti-aliassed line of one pixel width

setThinEllipseAA - draw anti-aliased lines and ellipses

Filter Dialog Window Functions

checkDialogFocus - Checks if the filter dialog currently has keyboard focus.

clearDialogEvent - Disables processing of init, cancel or keypress events.

createCircularRgn - Creates a simple circular region, for use with setDialogRegion.

createEllipticRgn - Creates an elliptical region, for use with setDialogRegion.

createPolyRgn - Creates an arbitrary polygon region object from a list of pairs of vertices.

createRectRgn - Creates a simple rectangular region, for use with setDialogRegion.

createRoundRectRgn - Creates a rectangular region with rounded corners, for use with setDialogRegion.

enableToolTipBalloon - Enables or disables speech balloon style tool tips for all controls.

getAppTheme - Checks if the host application has XP/Vista style themes enabled.

getDialogHandle - Gets the window handle of the filter dialog window.

getDialogHeight - Gets the current width of the dialog in vertical DBUs.

getDialogPos - Gets the position and size of the filter dialog window.

getDialogWidth - Gets the current width of the dialog in horizontal DBUs.

lockWindow - Enables / disables drawing in the filter dialog area.

refreshRgn - Redraws or updates a certain region of the filter dialog.

refreshWindow - Redraws or updates the filter dialog.

setDialogColor - Sets the background color of the filter dialog.

setDialogSysColor - Sets the background color of the filter dialog using defined system colors

setDialogDragMode - Sets whether the filter dialog can be dragged by the title bar and/or background.

setDialogEvent - Enables processing of init, cancel and keypress events.

setDialogGradient - Sets a gradient as the background color of the filter dialog.

setDialogImage - Changes the image used in the filter dialog background.

setDialogImageMode - Sets whether the dialog background image is tiled or stretched.

setDialogMinMax - Sets the minimum and maximum dimensions of the filter dialog.

setDialogPos - Sets the position and size of the filter dialog window.

setDialogRegion - Sets the clipping region (outline) of the filter dialog.

setDialogShowState - Shows and hides the main filter dialog window.

setDialogSizeGrip - Shows and hides the resizing size grip control.

setDialogStyle - Sets various styles of the dialog.

setDialogStyleEx - Sets various extended styles of the dialog.

setDialogText - Sets the caption in the title bar.

setDialogTextv - Sets the caption in the title bar with printf-style formatting.

setDialogTheme - Switches XP/Vista themes on and off. *(Does not work correctly at the moment.)*

Events or FME_ constants

All constants that begin by the FME_ prefix, are used to identify events, i.e actions done by the user.They must be used inside the OnCtl handler.

FME_CANCEL

FME_CHANGED

FME_CLICKED

FME_COMBO_CLOSEUP

FME_COMBO_DROPDOWN

FME_CONTEXTMENU

FME_CUSTOMEVENT

FME_DRAWITEM

FME_EXITSIZE

FME_INIT

FME_KEYDOWN

FME_KEYUP

FME_KILLEDITFOCUS (deprecated)

FME_KILLFOCUS

FME_LEFTCLICKED_DOWN

FME_LEFTCLICKED_UP

FME_MOUSEMOVE

FME_MOUSEOUT

FME_MOUSEOVER

FME_PREVIEWDRAG

FME_RIGHTCLICKED_DOWN

FME_RIGHTCLICKED_UP

FME_SETEDITFOCUS

FME_SETFOCUS (deprecated)

FME_SIZE

FME_TIMER

FME_VALUECHANGED (deprecated)

Constants

ALTERNATE

BLACKONWHITE

COLORONCOLOR

DIM_EXACT

DIM_STRETCHED

DIM_TILED

HALFTONE

IDABORT

IDCANCEL

IDIGNORE

IDNO

IDOK

IDRETRY

IDYES

MAXSTRETCHBLTMODE

RGN_AND

RGN_COPY

RGN_DIFF

RGN_MAX

RGN_MIN

RGN_OR

RGN_XOR

STRETCH_ANDSCANS

STRETCH_DELETESCANS

STRETCH_HALFTONE

STRETCH_ORSCANS

WHITEONBLACK

WINDING

See Also

Command Reference, **Image Functions**, **System Functions**, **C Runtime Functions**

Image Functions

Getting and Setting Pixels

iget

pget

pgetp

pgetr

pset

psetp

psetr

set_psetp_mode

src

src0 - for Adobe Premiere compatibility, identical to src

src1 - for Adobe Premiere compatibility, identical to src

srcp

tget

tgetp

tgetr

tset

Selections

haveMask - Boolean variable that is true when a non-rectangular area has been selected.

msk

isFloating - Boolean variable that is true when the selection is floating.

Notice: The test `imageWidth==filterRectWidth && imageHeight==filterRectHeight` allows to know if only a part of the whole picture has been selected.

Color Space Conversions

rgb2cmyk

rgb2hsl

rgb2iuv

rgb2lab

rgb2ycbcr

cmyk2rgb

hsl2rgb

iuv2rgb

lab2rgb

ycbcr2rgb

Blurs And Convolutions

cnv

cnv0 - for Adobe Premiere compatibility, identical to **cnv**

cnv1 - for Adobe Premiere compatibility, identical to **cnv**

cnvX

cnvY

xyzcnv

Effects

blend - blends two color values according to a variety of blend modes

contrast

gamma

grad2D

gray - converts RGB to a grayscale value using provided weight values

phaseshift

posterize

saturation

setGamma

solarize

tone

Mathematical Effects

cosineInterpolate

linearInterpolate

quickFill

quickMedian

sinbell

tri

tricos

Array and Cell Functions

allocArray - Allocate an Array.

allocArrayPad - Allocate an Array with padding.

cell_initialize - *(deprecated)*

cell_preserve - Preserve cell values across filter invocations.

copyArray - Copy one Array to another.

ffillArray - Fill an Array with a floating-point value.

fgetArray - Get the floating-point value stored in an Array element.

fillArray - Fill an Array with an integer value.

fputArray - Store a floating-point value into an Array element.

freeArray - Release storage when an Array is no longer needed.

get - Get the integer value stored in a cell.

getArray - Get the integer value stored in an Array element.

getArrayAddress - Get the address of the first element of an Array.

getArrayDim - Get the dimension(s) of an Array.

getArrayString - Get the string value stored in an Array element.

put - Store an integer value into a cell.

putArray - Store an integer value into an Array element.

putArrayString - Store a string value into an Array element.

set_array_mode - Set Array memory allocation mode.

Other Image Functions

Aval - Get the alpha component of an RGBA pixel value

Bval - Get the blue component of an RGB pixel value

Gval - Get the green component of an RGB pixel value

RGB

RGBA

Rval - Get the red component of an RGB pixel value

add

c2d

c2m

ctl

dif

fc2d

fc2m

fr2x

fr2y

getImageTitle - Get the title of the current image as a string

map

mix

mix1

mix2

r2x

r2y

rnd

rst

scl

sub

testAbort

bCircle

bRect

bRect2

bTriangle

egw

egm

pointer_to_buffer

set_edge_mode

See Also

Command Reference, Dialog Functions, System Functions, C Runtime Functions, Constants

System Functions

Registry Functions

setRegRoot - Sets the the current registry root key to either HKEY_LOCAL_MACHINE or HKEY_CURRENT_USER

getRegRoot - Gets the current registry root

setRegPath - Sets the the current registry path

getRegPath - Gets the the current registry path

putRegString - Stores a string value into the registry

getRegString - Fetches a C-style character string from the Windows Registry

enumRegValue - Enumerates the names of all values stored under the current key

deleteRegValue - Deletes specified value under the current registry key.

Filesystem Functions

getSpecialFolder - Finds the location of Windows-specific folders (eg current user's My Documents folder)

findFirstFile - Searches a directory for a file or subdirectory with a specific name

findNextFile - Finds the next file in a search started by findFirstFile

findClose - Closes a file search handle opened by findFirstFile

chdir - changes the working folder

getcwd - returns the current working folder

mkdir - creates a new folder

rmdir - deletes a folder

Memory Functions

allocHost - Allocates memory through the host application

lockHost - Gets a pointer to a memory block allocated by the host application

freeHost - Frees memory allocated by the host application

System Functions

getDisplaySettings - Determines the bit depth, resolution and refresh rate of the screen

getAsyncKeyState - Determines whether a key is up or down at the time the function is called

getSysMem - Returns some values about the system memory.

getWindowsVersion - Detects the currently running Windows version

setTimerEvent - Activates or deactivates one of 10 available timers

shellExec - Executes a Windows shell command

sleep - Delays execution of the filter for a specific period of time

Resource Functions

copyResToArray - Copies a file resource to an array.

getResAddress - Returns a pointer to the specified file resource.

getResSize - Returns the size of the file resource.

Locale Functions

getUserDefaultLCID - Returns the locale identifier for the user default locale

getSystemDefaultLCID - Returns the locale identifier for the system locale

getLocaleInfo - Retrieves information about a locale specified by identifier

DLL Access Functions

loadLib - Loads a DLL into memory

getLibFn - Looks up a function in a loaded DLL

callLib - Calls an **int**-valued CDECL or STDCALL function in a loaded DLL

callLibFmc - Calls an **int**-valued CDECL function in a loaded DLL, passing a pointer to the FM context record

fcallLib - Calls a **double**-valued CDECL or STDCALL function in a loaded DLL

fcallLibFmc - Calls a **double**-valued CDECL function in a loaded DLL, passing a pointer to the FM context record

freeLib - Releases a DLL from memory

CPU Register Access Functions

getCpuReg - Gets contents of a cpu register

Multithreading Functions

countProcessors - Returns the number of (virtual or physical) processors in your system.

triggerThread - Creates a worker thread to execute the **OnCtl** handler.

waitForThread - Waits for one (or all) worker threads to complete.

isThreadActive - Tests whether a specific (or any) worker thread is still active.

getThreadRetVal - Retrieves the exit code value of a completed worker thread.

terminateThread - Terminates a specified (or all) worker thread(s).

Critical Section Functions

createCriticalSection - Allocates and initializes a Critical Section.

enterCriticalSection - Enters a Critical Section, first waiting until no other worker thread is inside.

tryEnterCriticalSection - Enters a Critical Section immediately if available; returns **false** if the Critical Section is currently occupied by another worker thread.

leaveCriticalSection - Leaves a Critical Section, permitting another waiting worker thread to enter it.

deleteCriticalSection - Deletes a Critical Section and all its resources when no longer needed.

See Also

Command Reference, **Dialog Functions**, **Image Functions**, **C Runtime Functions**

C Runtime Functions

Besides the large number of graphics- or GUI-specific functions in FilterMeister, a large number of the standard C runtime functions are available to the developer. Almost all of the functions defined in the C99 standard are provided and even some common extensions are at your disposal.

The most notable omissions are the well-known `gets` and `scanf`. Since these require some sort of console-based input, which FilterMeister currently does not support, these cannot be implemented.

File I/O and file manipulation functions

clearerr - Clears the error indicator for a stream

fclose - Closes a stream

fcloseall - Closes all open streams

feof - Tests for end of file on a stream

ferror - Tests for error on a stream

fflush - Flushes a stream

fgetc - Reads a character from a stream

fgetpos - Get file position

fgets - Reads a string from a stream

flushall - Flushes all streams

fopen - Opens a stream

fputc - Writes a character to a stream

fputs - Writes a string to a stream

fread - Reads unformatted data from a stream

fseek - Moves file position to a given location

fsetpos - Sets the position indicator of a stream

ftell - Gets current file position

fwrite - Writes unformatted data to a stream

getc - Reads a character from a stream

printf - Writes formatted data to a message box.

putc - Writes a character to a stream

remove - Deletes a file

rename - Renames a file

rewind - Moves file pointer to beginning of stream

snprintf - Writes formatted data to string up to a specified length

sprintf - Writes formatted data to string

tmpfile - Creates a temporary file that is deleted on program exit

tmpnam - Creates a temporary file name

Data conversion functions

abs - Absolute value of integer

strtod - Converts string to double

strtol - Converts string to long

strtoul - Converts string to unsigned long

Memory management functions

calloc - Allocate memory block with zero fill

expand - Expands a memory block in place

free - Free memory block

malloc - Allocate memory block

msize - Returns the size of a memory block in bytes. *(Microsoft-specific function)*

realloc - Re-size memory block

sizeof - Returns the size of an object type in bytes

Memory manipulation functions

memcpy - Copies characters until character or number of characters has been copied

memchr - Return a pointer to the first occurrence, within a specified number of characters

memcmp - Compare a specified number of characters from two memory locations

memicmp - Compare a specified number of characters from two memory locations without regard to case *(Unix-specific function)*

memmove - Copy a specified number of characters from one memory location to another

memset - Set a region of memory to a specified character

String manipulation functions

strcat - Concatenate two strings

strchr - Find the first occurrence of a given character in a string

strcmp - Compare two strings

strcoll - Compare two strings using the current locale

strcpy - Copy one string to another

strcspn - Find any one of a set of characters in a string

strdup - Duplicate a string

strerror - Return string describing error code

stricmp - Compare two strings without regard to case. *(Microsoft-specific function)*

stricoll - Compare two strings using the current locale and without regard to case. *(Microsoft-specific function)*

strlen - Get the length of a string

strlwr - Convert a string to lower case. *(Microsoft specific function)*

strncat - Concatenate the specified number of characters from one string to another

strncmp - Compare the specified number of characters in two strings

strncoll - Compare the specified number of characters in two strings using the current locale. *(Microsoft-specific function)*

strncpy - Copy the specified number of characters from one string to another

strnicmp - Compare the specified number of characters in two strings without regard to case

strnicoll - Compare the specified number of characters in two strings using the current locale and without regard to case. *(Microsoft-specific function)*

strnset - Set the specified number of characters in a string to a given character

strpbrk - Find the first occurrence of a character from one string in another

strrchr - Find the last occurrence of a character in a string

strrev - Reverse a string

strset - Set all characters in a string to a given character

strspn - Find the first substring from a given character set in a string

strstr - Find the first occurrence of a string in another string

strtok - Find the next token in a string

strupr - Convert a string to upper case. *(Microsoft specific function)*

strxfrm - Transforms a string according to the current locale

Floating-point math

ceil - Floating point ceiling(x)

chop - Truncate floating point value towards 0.0.

exp - Floating point exponent(x)

fabs - Floating point absolute value(x)

floor - Floating point floor(x)

fmax - Find the maximum of two floating point numbers

fmin - Find the minimum of two floating point numbers

fmod - Floating point remainder (x/y)

fsin - Floating point sine(x)

hypot - Euclidean distance function, calculate the hypothenuse or a right-angled triangle.

ldexp - The value of x * 2**n

log - Natural log(x)

log10 - Log base 10 of x

max - Max of(a, b)

min - Min of(a, b)

modf - Split x into integral and fractional parts, each with the same sign as x.

pow - The value of x**y

rand - Generate a pseudo random number

round - Round floating point value to nearest or even integral value. ***(Not yet implemented.)***

sin - Implemented as **fsin**.

sqr - Integer square root(x)

sqrt - Floating point square root(x)

srand - Set random number seed

Process management and control

abort - Abort the current process

Time and date functions

clock - Returns the processor time since the beginning of execution

time - Returns the current calendar time

strdate - Converts the current date to a string

strtime - Converts the current time to a string

See Also

Command Reference, **Dialog Functions**, **Image Functions**, **System Functions**

Constants

Image constants

These constants contain information about the image or preview being processed.

D - Maximum angle, always "1024". Datatype "int".

dmax - *(deprecated)* Identical to **D**. Maximum angle, always "1023". Datatype "int".

imageHRes - The horizontal DPI (Dots Per Inch) setting of the source image. Datatype "double".

imageVRes - The vertical DPI (Dots Per Inch) setting of the source image. Datatype "double".

imageWidth - Width of the original image in pixels, or width of the floating selection. Datatype "int".

pmax - *(deprecated)* Identical to **Z**. Number of available planes (channels) in the image.

imageHeight - Height of the original image in pixels, or width of the floating selection. Datatype "int".

wholeWidth - Width of the original image in pixels, regardless of floating selection. Datatype "int".

wholeHeight - Height of the original image in pixels, regardless of floating selection. Datatype "int".

X - Width of the *currently processed* image in pixels: on the run it's the (preview zoomratio dependent) PROXY IMAGE width but it's the original image width when effect is actually applied. Datatype "int".

xmax - *(deprecated)* Identical to **X**. Width of the currently processed image in pixels.

x_end - Last x pixel coordinate visible inside the preview window / of the currently processed image tile.

x_start - First x pixel coordinate visible inside the preview window / of the currently processed image tile.

Y - Height of the *currently processed* image in pixels: on the run it's the (preview zoomratio dependent) PROXY IMAGE height but it's the original image height when effect is actually applied. Datatype "int".

ymax - *(deprecated)* Identical to **Y**. Height of the currently processed image in pixels.

y_end - Last y pixel coordinate visible inside the preview window / of the currently processed image tile.

y_start - First y pixel coordinate visible inside the preview window / of the currently processed image tile.

Z - Number of available channels. Number of color channels + one option channel if image is not the background layer. Datatype "int".

zmax - *(deprecated)* Identical to **Z**. Number of available channels in the image.

Processing constants

These constants contain information about how the image is being processed.

bgColor - The host application's current background color as an RGB triple stored in an integer

DESIGNTIME - true if the filter is running in the FilterMeister development environment, false if a compiled filter plugin.

doingProxy - true if the filter is running in a preview dialog window, and false if the filter is applying the effect to the image in the host program.

doingScripting - true if the filter is running via a script / smart filter, false otherwise

fgColor - The host application's current foreground color as an RGB triple stored in an integer

filterCase - The type of data being filtered (eg Flat with no selection, Flat with a selection, etc)

filterInstallDir- full directory path to where the plugin is installed

filterUniqueID - a unique id / GUID for the plugin, for use by plugins that support scripting

FMC__TARGET- set to 32 for 32-bit plugins, and 64 for 64-bit plugins. Only available from FM1.0 Beta9g MT4 onwards.

haveMask - Boolean variable that is true when a non-rectangular area has been selected.

hostSerialNumber - the serial number of the host application, if supported by the host program

hostSig - a value that can sometimes be used to identify the host program the plugin is running in (eg particular Photoshop versions)

imageMode - The mode of the image being filtered (eg Bitmap, Grayscale, RGB, CMYK etc)

isFloating - Boolean variable that is true when the selection is floating.

planes - the number of planes/channels in the image, including alpha & mask planes

planesWithoutAlpha - the number of color planes/channels in an image (excludes alpha & mask planes)

platformdata - a value that can sometimes be used to identify a particular combination of host program & operating system

samplingSupport - indicates whether the host supports non-1:1 sampling for the proxy preview

scaleFactor - an integer representing the current zoom/scale of the preview proxy window (eg 1 for 100%, 3 for 33%, 5 for 20%)

zoomFactor - a integer between 1 and 16 to indicate the current zoom factor of the proxy preview window, or 0 if the proxy zoom factor has not yet been set. (In the current implementation, the zoomFactor variable is essentially the same as the built-in scaleFactor variable.)

Events or FME_ constants

All constants that begin by the FME_ prefix, are used to identify events, i.e actions done by the user. They must be used inside the OnCtl handler.

FME_CANCEL

FME_CHANGED

FME_CLICKED

FME_COMBO_CLOSEUP

FME_COMBO_DROPDOWN

FME_CONTEXTMENU

FME_CUSTOMEVENT

FME_DRAWITEM

FME_EXITSIZE

FME_INIT

FME_KEYDOWN

FME_KEYUP

FME_KILLEDITFOCUS (deprecated)

FME_KILLFOCUS

FME_LEFTCLICKED_DOWN

FME_LEFTCLICKED_UP

FME_MOUSEMOVE

FME_MOUSEOUT

FME_MOUSEOVER

FME_PREVIEWDRAG

FME_RIGHTCLICKED_DOWN

FME_RIGHTCLICKED_UP

FME_SETEDITFOCUS

FME_SETFOCUS (deprecated)

FME_SIZE

FME_TIMER

FME_VALUECHANGED (deprecated)

Filtermeister Limits

MAX_LABEL_SIZE - max number of chars in a control label or dropdown list

MAX_TOOLTIP_SIZE - max number of chars in a tooltip

MAX_SOURCE_CODE_SIZE - max number of chars allowed by the source editor

MAX_DLITS - size of floating-point/string literal pool (in 8 byte units)

MAX_LOCALS - max number of local variables in a handler or user-defined function

MAX_TEMPS - max number of compiler-generated temporaries per handler or user-defined function

MAX_CASE_LABELS - max number of case labels per switch statement

N_CELLS - number of anonymous get/put cells

N_CTLS - max number of controls (user-defined and system reserved)

CTL_LAST_USER - index of last available user-defined control

C Miscellaneous Constants

These constants are specified in the C standard which FilterMeister aims to support; as such, they are available for developers.

_MAX_DIR - Maximum size in characters of a directory name (e.g., "adobe/photoshop/plug-ins").

_MAX_DRIVE - Maximum size in characters of a drive name (e.g., "C:").

_MAX_EXT - Maximum size in characters of a file extension (e.g., ".8bf").

_MAX_FNAME - Maximum size in characters of a file name (e.g., "myplugin").

_MAX_PATH - Maximum size in characters of a path name (e.g., "adobe/photoshop/plug-in/").

CLOCKS_PER_SEC

EDOM

ERANGE

EXIT_FAILURE

EXIT_SUCCESS

NULL - Constant for the integer value 0, representing a pointer to nothing.

RAND_MAX - Maximum value generated by the integer random number generator.

C Mathematical Constants

These constants are specified in the C standard which FilterMeister aims to support; especially in the library "math.h".

M_E - Same value as `exp(1.0)`

M_LOG2E - Same value as `1.0/log(2.0)`

M_LOG10E - Same value as `1.0/log(10.0)`

M_LN2 - Same value as `log(2.0)`

M_LN10 - Same value as `log(10.0)`

M_PI - The numerical value of the ratio of the circumference of a circle to its diameter (approximately 3.14159). Usually represented by the Greek letter Pi.

M_PI_2 - Same value as `M_PI/2.0`

M_PI_4 - Same value as `M_PI/4.0`

M_1_PI - Same value as `1.0/M_PI`

M_2_PI - Same value as `2.0/M_PI`

M_2_SQRTPI - Same value as `2.0/sqrt(M_PI)`

M_SQRT2 - Same value as `sqrt(2.0)`

M_SQRT1_2 - Same value as `1.0/sqrt(2.0)`

Events or FME_ constants

All constants that begin by the FME_ prefix, are used to identify events, i.e actions done by the user. They must be used inside the OnCtl handler.

FME_CANCEL

FME_CHANGED

FME_CLICKED

FME_COMBO_CLOSEUP

FME_COMBO_DROPDOWN

FME_CONTEXTMENU

FME_CUSTOMEVENT

FME_DRAWITEM

FME_EXITSIZE

FME_INIT

FME_KEYDOWN

FME_KEYUP

FME_KILLEDITFOCUS (deprecated)

FME_KILLFOCUS

FME_LEFTCLICKED_DOWN

FME_LEFTCLICKED_UP

FME_MOUSEMOVE

FME_MOUSEOUT

FME_MOUSEOVER

FME_PREVIEWDRAG

FME_RIGHTCLICKED_DOWN

FME_RIGHTCLICKED_UP

FME_SETEDITFOCUS

FME_SETFOCUS (deprecated)

FME_SIZE

FME_TIMER

FME_VALUECHANGED (deprecated)

All global, session persistent variables that are available in FilterMeister

```
int i0, i1, i2, i3, i4, i5, i6, i7, i8, i9;
int j0, j1, j2, j3, j4, j5, j6, j7, j8, j9;
int k0, k1, k2, k3, k4, k5, k6, k7, k8, k9;
double x0, x1, x2, x3, x4, x5, x6, x7, x8, x9;
double y0, y1, y2, y3, y4, y5, y6, y7, y8, y9;
double z0, z1, z2, z3, z4, z5, z6, z7, z8, z9;
char str0[256];
char str1[256];
char str2[256];
char str3[256];
char str4[256];
char str5[256];
char str6[256];
char str7[256];
char str8[256];
char str9[256];
```

They are initialized in DoStart (see the Photoshop SDK). They are usually initialized to zero, but don't count on that. They may have random values in other hosts than Photoshop. Initialize them in OnFilterStart when the filter is executed for the first time.

__fgetArray

Requires FM 1.0 Beta 9.1 (February 2009) or newer

Syntax

```
double __fgetArray(int nr, int x, int y, int z)
```

Arguments

nr
: Number of the array. Values from 0 to 99 are accepted.

x, y, z
: x, y, and z coordinates of a cell in the array. If you allocated a one-dimensional array, set y and z to zero. If you allocated a two-dimensional array, set z to zero.

Return

Returns the double floating-point value that was stored at the specified coordinates in the array. *Warning: If the specified coordinates lie outside the array, the plug-in may crash.*

Description

This function lets you read a floating-point value from an array. This is a fast version of **fgetArray** without any error checking or memory checking. As it does not do any border checking, it may produce error messages or even crashes if not used properly.

Example

See **allocArray** for an example of how to allocate arrays and use the similar **fgetArray** function.

See Also

___getArray, ___fputArray, allocArray, freeArray, putArray, getArrayDim, copyArray

__fputArray

Requires FM 1.0 Beta 9.1 (February 2009) or newer

Syntax

```
double __fputArray (int nr, int x, int y, int z, double
dval)
```

Arguments

nr
: Number of the array. Values from 0 to 99 are accepted.

x, y, z
: x, y, and z coordinates of a cell in the array. If you allocated a one-dimensional array, set y and z to zero. If you allocated a two-dimensional array, set z to zero.

dval
: Floating-point value that will be stored at the specified coordinates in the array.

Return

Returns 1 if the function succeeded, or 0 if there are more than 4 bytes per array cell. *Warning: If the specified coordinates lie outside the array, the plug-in may crash.*

Description

Lets you store a floating-point value in an array. This is a fast version of **fputArray** without any error checking or memory checking. As it does not do any border checking, it may produce error messages or even crashes if not used properly.

When storing a value into an Array with byte-size 2, the value will be converted to 16-bit half. For an Array of byte-size 4, the value is converted to a 32-bit float. If the byte-size is not 2, 4, or 8, a value of 0 is returned to indicate failure.

Example

See **fgetArray**.

See Also

__putArray, **__fputArray**, **allocArray**, **freeArray**, **putArray**, **getArrayDim**, **copyArray**

__getArray

Requires FM 1.0 Beta 9.1 (February 2009) or newer

Syntax

```
int __getArray(int nr, int x, int y, int z)
```

Arguments

nr
Number of the array. Values from 0 to 99 are accepted.

x, y, z
x, y, and z coordinates of a cell in the array. If you allocated a one-dimensional array, set y and z to zero. If you allocated a two-dimensional array, set z to zero.

Return

Returns the value that was stored at the specified coordinates in the array. *Warning: If the specified coordinates lie outside the array, the plug-in may crash.*

Description

Reads a value from an array. This is a fast version of **getArray** without any error checking or memory checking. As it does not do any border checking, it may produce error messages or even crashes if not used properly.

Example

See allocArray for an example of how to allocate arrays and use the similar getArray function.

See Also

__putArray, __fgetArray, allocArray, freeArray, putArray, getArrayDim, copyArray

__putArray

Requires FM 1.0 Beta 9.1 (February 2009) or newer

Syntax

```
int __putArray (int nr, int x, int y, int z, int val)
```

Arguments

nr
: Number of the array. Values from 0 to 99 are accepted.

x, y, z
: x, y, and z coordinates of a cell in the array. If you allocated a one-dimensional array, set y and z to zero. If you allocated a two-dimensional array, set z to zero.

val
: Value that will be stored at the specified coordinates in the array.

Return

Returns 1 if the function succeeded, or 0 if there are more than 4 bytes per array cell. *Warning: If the specified coordinates lie outside the array, the plug-in may crash.*

Description

Lets you store a value in an array. This is a fast version of **putArray** without any error checking or memory checking. As it does not do any border checking, it may produce error messages or even crashes if not used properly.

When storing a value into an Array with byte-size 1, the value will be clamped to the range [0,255]. For an Array of byte-size 2, the value is clamped to [0,65535]. If the byte-size is not 1, 2, or 4, a value of 0 is returned to indicate failure.

Example

See **allocArray** for an example of how to allocate arrays and use the similar **putArray** function.

See Also

___putArray, **___fputArray**, **allocArray**, **freeArray**, **putArray**, **getArrayDim**, **copyArray**

abort

Syntax

`void abort(void)`

Description

Causes the program to terminate processing immediately.

All open files are flushed and closed but reserved memory is not automatically freed so this function must be used with caution when allocating memory.

Comments

Depending on why you want to use the Abort function, you might be better to use `doAction(CA_CANCEL);` instead.

Abort can cause Photoshop to crash in some test builds of FilterMeister (eg FM 1.0g beta). Changing the **abort_mode** value in 64-bit plug-ins may help avoid this crash.

When using Ctrl+F (Photoshop repeat last filter) condition is ignored with `doAction(CA_CANCEL);` This doesn't happen with `abort();` At least for this purpose, `abort();` seems to work more reliably than `doAction(CA_CANCEL);`

See Also

doAction, **abort_mode**

abort_mode

Syntax

```
int abort_mode
```

Description

The abort mode variable determines how the filter will terminate execution when the **abort** function is called.

Details

Set "abort_mode" to one of the following values during initial execution of your filter:

0 - abort() throws an exception which is caught and handled at the main execution level. This was the original FM 1.0 implementation, and requires Structured Exception Handling (SEH) -- which is not yet available in FM64.

1 - abort() sets gResult = userCanceledErr, and returns 0; this is the workaround suggested by Harry. This abort mode does not stop the filter immediately, but lets the filter run to the end and then discards the processing. The filter developer can use "abort(); return true;" to make it stop immediately.

2 - abort() displays a message box "Processing aborted...", then longjmp's back to the most recent setjmp.

3 - abort() silently longjmp's back to the most recent setjmp. This is an attempt to simulate the original SEH implementation, but won't work in all cases.

-1 - abort() uses the default implementation for the current execution target (mode 0 for x32, mode 3 for x64; the latter will revert to mode 0 once x64 supports SEH).

Comments

abort_mode is only available in FM1.0 Beta 9g MT4 and newer.

Example

```
%fml-2.0

OnFilterStart: {

    // This will cause abort to act as
    // a Cancel click in compiled
    // plug-ins and display an error
    // message during filter
    // development.

    abort_mode = 2;
    if (DESIGNTIME) abort_mode = 1;
    return true;
}
```

See Also

abort

abs

Syntax

```
int abs(int number)
```

Arguments

number
    Any integer number.

Return

The absolute value of the supplied argument.

Description

Returns the absolute value of an integer number. When supplied with a negative number, this function will return a positive number of equal distance from zero.

Example

```
%ffp

OnFilterStart:
{
  Info("The absolute value of -10 is %d", abs(-10));
}
```

See Also

fabs

add

Syntax

```
int add(int a, int b, int c)
```

Arguments

a
    Any integer.
b
    Any integer.
c
    Any integer.

Return

The lower integer value of c and (a+b).

Description

This function adds 'a' to 'b', then compares the result with 'c', and returns the lower of the two values. Please note that this function has been retained for compatibility with Filter Factory, and that **min**(a+b,c) will give the same result and compute faster.

Example

```
// sets 'p' to 1, because 3+2=5,
// and 5>1 !
int p = add(3,2,1);
```

See Also

sub,**min**, **max**

allocArray

Syntax

```
int allocArray (int nr, int X, int Y, int Z, int bytes)
```

Arguments

nr
: Number of the Array. Values from 0 to 99 are allowed.

X, Y, Z
: Amount of X, Y and Z cells in the three-dimensional array. If you want to allocate a one-dimensional array simply set Y and Z to zero. If you want to allocate a two-dimensional array, please set Z to zero.

bytes
: Size in bytes of each cell of the array. Only values of 1, 2, 4 and 8 are allowed.

Return

Returns a value of 1 if the allocation succeeded and a value of zero if it failed.

Description

This function lets you allocate up to 100 different arrays of a user-definable size, with up to three dimensions and with up to 8 bytes per cell. 1 byte arrays have values from 0 to 255, 2 byte arrays have values from 0 to 65535, 4 byte arrays can use integer or float values and 8 byte arrays use double values.

Note that FilterMeister does not support standard C-language arrays. For this purpose FilterMeister offers you alternative functions like **allocArray**, **putArray** and **getArray**. Or you can dynamically allocate memory with the **malloc** function.

These 100 arrays are internally managed by FilterMeister and automatically freed when FilterMeister or a FM plug-in exits. So you don't necessarily need to free them yourself with **freeArray**, but it is recommended to do so if you don't need an array all the time.

By default, arrays are allocated by calling the host application's buffer allocation API, if it exists. This permits the host application to coordinate its memory requirements with those of the plug-in filter. If the host application does not support the buffer allocation API, or if you call **set_array_mode**(0), then FilterMeister will allocate arrays from the C runtime heap instead.

Basically these array functions are a replacement for the less flexible tile buffers. FilterMeister will support defining arrays as it is usually done in C in future. In the meantime please use the Array functions.

Example

```
%ffp

// This code works for 8-bit
// images and 16-bit images

ForEveryTile:
{

  int bitMultiply = (imageMode > 9 ? 128 : 1);

  // Allocate Array Nr. 5 and
  // make it the same size and
  // bitdepth as the image
  if (allocArray(5, X, Y, Z, imageMode > 9 ? 2 : 1 )) {

    // Store the image data
    // in the array
    for (y=y_start; y<y_end; y++) {

      updateProgress(y,y_end);

      for (x = x_start; x < x_end; x++) {

        for (z = 0; z < Z; z++) {
          putArray(5, x, y, z, src(x,y,z));
        }

      }
    }

    // Read the image data from
    // the array and write it
    // back to the image as a
    // negative

    for (y = y_start; y < y_end; y++) {

      updateProgress(y, y_end);

      for (x = x_start; x < x_end; x++) {

        for (z = 0; z < Z; z++) {
          pset(x, y, z, 255 * bitMultiply - getArray(5, x,
y, z) );
        }

      }
    }


    // Free the array
    freeArray(5);
  }
  else
    ErrorOk ("Array allocation failed");

  return true;
}
```

See Also

allocArrayPad, freeArray, getArray, fgetArray, putArray, fputArray, getArrayDim, copyArray, fillArray, set__array__mode

allocArrayPad

Syntax

```
int allocArrayPad (int nr, int X, int Y, int Z, int bytes,
int padding)
```

Arguments

nr
: Number of the Array. Values from 0 to 99 are allowed.

X, Y, Z
: Amount of X, Y and Z cells in the three-dimensional array. If you want to allocate a one-dimensional array simply set Y and Z to zero. If you want to allocate a two-dimensional array, please set Z to zero.

bytes
: Size in bytes of each cell of the array. Only values of 1, 2, 4 and 8 are allowed.

padding
: Size of the padding.

Return

Returns a value of 1 if the allocation succeeded and a value of zero if it failed.

Description

allocArrayPad works just like **allocArray** with the difference that there is an additional parameter for the padding size. For example, if you use allocArrayPad(0,200,200,1,1,3), which sets the padding size to 3, you can read and write values at the coordinates (-3,-3) and (202,202).

See Also

allocArray, freeArray, getArray, putArray, getArrayDim, copyArray, set__array__mode

allocHost

Requires FM 1.0 Beta 9.1 (February 2009) or newer

Syntax

```
int allocHost(int size)
```

Arguments

size
The size of the memory block to allocate in bytes.

Return

A bufferID for the allocated memory block, or NULL if the memory couldn't be allocated (or the host program doesn't support allocated memory blocks).

Description

Allocates a block of memory through the host application. Since some graphics programs like Photoshop manage memory and allocate large chunks of memory for themselves, you might want to use this in preference to the system memory functions like malloc and calloc.

Example

```
int bufferID = allocHost(100);
if (bufferID == NULL) {
  Warn("Could not allocate memory");
}
else {
  char* memptr = lockHost(bufferID);
  sprintf(memptr, "Message goes here!");
  Info(memptr);
  freeHost(bufferID);
}
```

See Also

lockHost, freeHost

appendEllipsis

Syntax

```
char* appendEllipsis(char* s)
```

Arguments

s
The text that will have an ellipsis appended.

Description

Returns string s with an ellipsis ("...") appended.

Example

```
%fml
ctl[4]: STATICTEXT, Text=""

OnFilterStart: {
  strcpy(str0, "Waiting");
  strcpy(str1, appendEllipsis(str0));
  setCtlText(4, str1);
  return true;
}
```

See Also

formatString, stripEllipsis

Aval

Syntax

```
int Aval(int rgba)
```

Arguments

rgba
A 32-bit quadruple holding values for the three color channels, plus the alpha channel, each as eight bits.

Return

A value in the range 0 to 255 inclusive.

Description

The value returned is the value for the alpha channel, extracted from the quadruple.

See Also

Rval,**Gval**,**Bval**

bCircle

Syntax

`int bCircle(int x, int y, int centerx, int centery, int radius)`

Arguments

x, y
: Current x and y coordinates in the image

centerx, centery
: Center coordinates of the circle

radius
: Radius of the circle

Return

If the supplied x and y values lie within the defined circle a value of 1 will be returned, otherwise 0.

Description

Lets you easily draw a circle on the image.

Example

```
%ffp

ctl(0): "Center X", range=(0,255), val=128
ctl(1): "Center Y", range=(0,255), val=128
ctl(2): "Radius", range=(0,128), val=64

ForEveryTile:
```

```
{

  int Draw;

  for (y=y_start; y<y_end; y++) {
    updateProgress(y,y_end);
    for (x=x_start; x<x_end; x++) {
      Draw = bCircle(x,y,ctl(0),ctl(1),ctl(2));
      for (z=0; z<Z; z++) {
        if (Draw) pset(x,y,z,255);
      }
    }
  }

  return true;
}
```

See Also

bRect, **bRect2**, **bTriangle**

bgColor

Syntax

`int bgColor`

Description

The currently chosen background color value in the host application (eg Photoshop®, Paint Shop Pro®).

Example

```
%fml
ctl[0]: OWNERDRAW, Size=(50,50), Pos=(240,3)
ctl[1]: OWNERDRAW, Size=(50,50), Pos=(300,3)
ctl[2]: STATICTEXT, Text="Foreground", Pos=(240, 55)
ctl[3]: STATICTEXT, Text="Background", Pos=(300, 55)

OnFilterStart: {
  setCtlColor(0, fgColor);
  setCtlColor(1, bgColor);
  return true;
}
```

See Also

fgColor, setCtlColor

BITMAP

Syntax

```
ctl[n]: BITMAP(Class Specific Properties), Other Properties
```

Description

The class BITMAP allows the filter developer to place a bitmap in the dialog window. By default, this user control is not actionable.

Class Specific Properties

CENTERIMAGE
: Scales the image to original size and centers the image within the control.

NOTIFY
: Makes the user control actionable and activates tooltip.

Other Properties

Action
: The action that the plug-in should take when the bitmap is clicked on.

Image
: The path to the image that should be used for the bitmap

Val
: Assigns a value to the bitmap, but only when it is disabled. *(default = 0)*

Example

```
ctl[0]: BITMAP, Image="Logo.bmp"
ctl[1]: BITMAP(CENTERIMAGE, NOTIFY), Image =
"C:\\Images\\OK_Button.bmp", Action=APPLY
```

Notes

Once the bitmap is actionable, its value definitions are lost. The reason is that an action returns a specific value and overwrites (once the mouse button is clicked over the user control) the user control's value.

Currently, only BMP files are supported. Transparency is supported by using the class **IMAGE**.

When standalone filters are created, images are not embedded by default. You can use the Embed: function to embed the image into the standalone filter file.

The image file should be present in the active directory or in any of the directories set in the PATH or FM_PATH variables (check your AUTOEXEC.BAT file).

See Also

ICON, **IMAGE**, **METAFILE**

blend

Syntax

```
int blend (int a, int b, int z, int m, int r)
```

Arguments

a
: Bottom Color Value from 0 to 255

b
: Top Color Value from 0 to 255

z
: Color Channel Value z. Only needed for the Dissolve, Threshold and Threshold 2 blend modes.

m
: Blend Mode (0 = Normal, 1 =Dissolve, 2 =Threshold, 3 = Threshold 2, 4 = Multiply, 5 = Screen, 6 = Overlay, 7= Soft Light, 8 =Hard Light, 9 = Dodge, 10 = Burn, 11 = Darken, 12 = Lighten, 13 = Exclusion, 14 = Difference, 15 = NegDif 1, 16 = NegDif 2, 17 = Subtract, 18 = Add, 19 = Expose)

r
: blending ratio from 0 (bottom) to 255 (top)

Return

The blended color value is returned.

Description

Blends two color values according to 20 different blending modes. There's also a parameter for the blending ratio. Currently works only with 8-bit color values.

Example

```
%ffp
// This example blends the image
// with a mirrored version
// of itself.

ctl(0): "Ratio", range=(0,255), val=128
ctl(1): combobox(vscroll), action=preview,
        color=#FFFFFF, fontcolor=#0000ff,
        size=(50,200), text="Normal\n"
        "Dissolve\nThreshold\n"
        "Threshold 2\nMultiply\n"
        "Screen\nOverlay\n"
        "Soft Light\n"
        "Hard Light\n"
        "Dodge\nBurn\nDarken\n"
        "Lighten\nExclusion\n"
        "Difference\nNegDif 1\n"
        "NegDif 2\nSubtract\n"
        "Add\nExpose",
        val=0

ForEveryTile: {

  for (y=y_start; y < y_end; y++) {

    updateProgress(y,y_end);

    for (x = x_start; x < x_end; x++) {
      for (z = 0; z < Z; z++) {

        pset(x, y, z, blend( src(x,y,z), src(X-x,Y-y,z),
z, ctl(1), ctl(0) ) );
```

```
      }
    }
  }

  return true;

}
```

bRect

Syntax

`int fm_bRect(int x, int y, int centerx, int centery, int radiusx, int radiusy)`

Arguments

x, y
    Current x and y coordinates in the image
centerx, centery
    Center coordinates of the circle
radiusx, radiusy
    Horizontal and vertical radius of the rectangle

Return

If the supplied x and y values lie within the defined rectangle a value of 1 will be returned, otherwise 0.

Description

Lets you easily draw a rectangle on the image.

Example

```
%ffp

ctl(0): "Center X", range=(0,255), val=128
ctl(1): "Center Y", range=(0,255), val=128
ctl(2): "Radius X", range=(0,128), val=64
ctl(3): "Radius Y", range=(0,128), val=64

```

```
ForEveryTile:
{

  int Draw;

  setCtlRange(0, 0, X);
  setCtlRange(1, 0, Y);
  setCtlRange(2, 0, X);
  setCtlRange(3, 0, Y);

  for (y=y_start; y<y_end; y++) {

    updateProgress(y, y_end);

    for (x=x_start; x<x_end; x++) {

      Draw = bRect(x, y, ctl(0), ctl(1), ctl(2), ctl(3));

      for (z=0; z<Z; z++) {
        if (Draw) pset(x,y,z,255);
      }
    }
  }

  return true;
}
```

See Also

bCircle, bRect2, bTriangle

bRect2

Syntax

`int bRect2(int x, int y, int topx, int topy, int bottomx, int bottomy)`

Arguments

x, y
: Current x and y coordinates in the image

topx, topy
: Coordinates of the top left corner of the rectangle

bottomx, bottomy
: Coordinates of the bottom right corner of the rectangle

Return

If the supplied x and y values lie within the defined rectangle a value of 1 will be returned, otherwise 0.

Description

Lets you easily draw a rectangle on the image.

Example

```
%ffp

ctl(0): "Top X",val=64
ctl(1): "Top Y",val=64
ctl(2): "Bottom X",val=128
ctl(3): "Bottom Y",val=128

```

```
ForEveryTile:
{
  int Draw;

  for (y=y_start; y<y_end; y++) {

    updateProgress(y,y_end);

    for (x=x_start; x<x_end; x++) {

      Draw = bRect2(x, y, ctl(0), ctl(1), ctl(2), ctl(3));

      for (z=0; z<Z; z++) {
        if (Draw) pset(x,y,z,255);
      }
    }
  }

  return true;
}
```

See Also

bCircle, **bRect**, **bTriangle**

bTriangle

Syntax

`int bTriangle(int x, int y, int centerx, int centery, int radius)`

Arguments

x, y
Current x and y coordinates in the image

centerx, centery
Center coordinates of the triangle

radius
Radius of the triangle

Return

If the supplied x and y values lie within the defined triangle a value of 1 will be returned, otherwise 0.

Description

Lets you easily draw a triangle on the image.

Example

```
%ffp

ctl(0): "X",range=(0,255),val=150
ctl(1): "Y",range=(0,255),val=100
ctl(2): "Radius",range=(0,128),val=64

ForEveryTile:
```

```
{

  int Draw;

  for (y=y_start; y<y_end; y++) {

    updateProgress(y,y_end);

    for (x=x_start; x<x_end; x++) {

      Draw = bTriangle (x, y, ctl(0), ctl(1), ctl(2));

      for (z=0; z<Z; z++) {
        if (Draw) pset(x,y,z,255);
      }
    }
  }

  return true;
}
```

See Also

bRect, **bRect**, **bRect2**

Bval

Syntax

```
int Bval(int rgb)
```

Arguments

rgb
: Either a 32-bit RGB triple or a 32-bit RGBA quadruple; in either case, the red, green and blue channels are represented as eight bit values, as is the alpha channel in the RGBA form.

Return

A value in the range 0 to 255 inclusive.

Description

The return value represents the value of the blue channel, extracted from the triple (or quadruple).

Example

```
blue = Bval(fgColor);
// Gives the blue channel value from the current
foreground color
```

See Also

Aval, **Gval**, **Rval**

c2d

Syntax

```
int c2d(int x, int y)
```

Arguments

x
: An integer pixel x-coordinate.

y
: An integer pixel y-coordinate.

Return

An integer in the range -511 to 512.

Description

Pixels are usually addressed by their cartesian coordinates [x,y], but FilterMeister also allows the use of polar coordinates, and provides a set of functions for conversion between the two systems. Polar coordinates are expressed as [d,m], where 'd' represents the 'direction' to the pixel from the origin [0,0], and 'm' is the 'magnitude' of the distance from the origin. The c2d() function returns the polar coordinate direction 'd' for the pixel at [x,y], relative to the top left corner of the image. A 'd' value of 0 represents the direction to the right along the x-axis (ie y=0); a value of 256 represents the direction downward along the y-axis (ie x=0); a value of 512 represents the direction to the left along the x-axis (where y=0); and a value of -256 represents the upward direction on the y-axis (where x=0). Naturally, intermediate values represent the intermediate angles.

Example

```
%ffp

ForEveryTile:
{

for (y = y_start; y < y_end; ++y)
{
  for (x = x_start; x < x_end; ++x)
  {
    for (z = 0; z < Z; ++z)
    {
      pset(x, y, z, (256 * abs( c2d(x-X/2, y-Y/2) )) /
512);
    }
  }
}

return true;
```

See Also

c2m, **r2x**, **r2y**

c2m

Syntax

```
int c2m(int x, int y)
```

Arguments

x
: An integer pixel x-coordinate.

y
: An integer pixel y-coordinate.

Return

An integer giving the distance from the center of the image to the pixel at coordinates [x,y].

Description

Pixels are usually addressed by their cartesian coordinates [x,y], but FilterMeister also allows the use of polar coordinates, and provides a set of functions for conversion between the two systems. Polar coordinates are expressed as [d,m], where 'd' represents the 'direction' to the pixel from the origin and 'm' is the 'magnitude' of the distance from the origin. The c2m() function returns the polar coordinate magnitude 'm' for the pixel at [x,y], relative to the top left corner of the image.

Example

```
%ffp

ForEveryTile: {
  for (y=y_start; y < y_end; ++y) {
    for (x=x_start; x < x_end; ++x) {
      for (z=0; z < Z; ++z) {

        pset(x, y, z, (256 * (c2m(x-X/2, y-Y/2)))/128);

      }
    }
  }

  return true;
}
```

See Also

c2d, **r2x**, **r2y**

callLib

Requires FM 1.0 Beta 9.0 (April 2008) or newer

Syntax

```
??? callLib(void *fnptr, ...)
```

Arguments

fnptr
: A pointer (obtained with **getLibFn**) to the DLL function to call.

...
: The other parameters required by the DLL function (varies depending on the function).

Return

Returns the return value of the called DLL function, which varies depending on the function called.

Description

Calls a function in a DLL that has previously been loaded with **loadLib** and **getLibFn**.

Example

```
// This code loads the user32.dll
// DLL included with Windows and
// uses it to display a YES/NO
// Message Box.
```

```
int* lib_user32;
int* functionPointer;
int returnval;

// Load the DLL library
lib_user32 = loadLib("user32");
if (lib_user32) {

  // Get the function in the DLL
  functionPointer = getLibFn(lib_user32, "MessageBoxA");
  if (functionPointer) {

    // Call the function
    strcpy(str0, "The window text is here");
    strcpy(str1, "Caption Text");
    returnval = callLib(functionPointer, NULL, str0, str1,
MB_YESNO);

    // Process return value
    if (returnval == IDYES)
      msgBox(MB_OK, "Yes!", "Yes was clicked");
    if (returnval == IDNO)
      msgBox(MB_OK, "No :(", "No was clicked");

  }
  else msgBox(MB_OK, "Error", "Function wasn't loaded");

  // Free the library DLL
  freeLib(lib_user32);

}
else msgBox(MB_OK, "Error", "DLL was not loaded");
```

See Also

loadLib, **getLibFn**, **freeLib**

callLibFmc

Requires FM 1.0 Beta 9.0 (April 2008) or newer

Syntax

```
??? callLibFmc(void *fnptr, ...)
```

Arguments

fnptr
: A pointer (obtained with **getLibFn**) to the DLL function to call.

...
: The other parameters required by the DLL function (varies depending on the function).

Return

Returns the return value of the called DLL function, which varies depending on the function called.

Description

Calls a function in a DLL that has previously been loaded with **loadLib** and **getLibFn**, except that the first argument (fnptr) is replaced by a pointer to the FMcontext record (fmcp) that you can then access from your DLL function. The structure of FMcontext can be found in the FilterMeister source code in the AfhFMcontext.h file. As the context record can change between FilterMeister versions, it is important that your DLL is designed to use the correct version of the context record for your plugin.

Example

Example of called functions in a DLL:

```
__declspec(dllexport)
int MyForEveryTile(FMcontext *fmcp, int X, int Y) { return
false; }

__declspec(dllexport)
double MyFloat(FMcontext *fmcp, int a, double x, double y)
{
  return a*(x - y);
}
```

To invoke these functions from FM:

```
int g_hMyDll = loadLib("MyDll");
if (!g_hMyDll) ERROR...

int g_pfnMyForEveryTile = getLibFn(g_hMyDll,
"MyForEveryTile");
if (!g_pfnMyForEveryTile) ERROR...

int g_pfnMyFloat = getLibFn(g_hMyDll, "MyFloat");
if (!g_pfnMyFloat) ERROR...

int iRes = callLibFmc(g_pfnMyForEveryTile, 100, 200);

double fRes = fcallLibFmc(g_pfnMyFloat, 10, 1.2, 3.14);
```

See Also

loadLib, **getLibFn**, **fcallLib**, **fcallLibFmc**, **freeLib**

calloc

Syntax

```
void* calloc(int number, int size)
```

Arguments

number
: The number of elements to be reserved.

size
: The size of each of the elements to be reserved, measured in bytes.

Return

A pointer to the allocated memory.

Description

Reserves memory for an array of **number** elements, each **size** bytes in size. Each byte of memory allocated using this function is initialized to 0. If you do not require this, **malloc** may be preferred for speed.

If the memory could be allocated, a pointer to the first element in the reserved memory block is returned.

If the memory could not be allocated (e.g., due to memory shortage or a high degree of memory fragmentation), a NULL value is returned instead.

Any memory reserved by use of this function must be manually deallocated by means of the **free** function; failure to do so will result in memory leakage and may ultimately crash the system.

Example

```
%ffp

OnFilterStart:
{
    // Allocate a string for 255 characters.
    char* buffer_1 = calloc(255, 1);

    free(buffer_1);
}
```

See Also

free, **malloc**, **realloc**

ceil

Syntax

```
double ceil(double number)
```

Arguments

number
: Any double or float number.

Return

The rounded value.

Description

Returns the smallest integral value greater than or equal to **number**.

Example

```
%ffp

OnFilterStart:
{
    Info("Rounding 2.345 to ceiling gives %f",
ceil(2.345));
    Info("Rounding -2.345 to ceiling gives %f",
ceil(-2.345));
}
```

See Also

chop, **floor**, **iceil**, **round**, **C Runtime Functions**

cell_initialize

Note: this function is deprecated.

Syntax

`cell_initialize(int n)`

Arguments

n
    0 or 1

Description

`cell_initialize` is now deprecated. It is recommended that **cell_preserve** be substituted whenever `cell_initialize` is encountered in old code. See **cell_preserve** for details of usage.

See Also

cell_preserve, **get**, **put**

cell_preserve

Syntax

`cell_preserve(int n)`

Arguments

n
: 0 or 1

Description

FilterMeister has a small internal buffer of 1024 integer items which can be accessed by means of the `get` and `put` functions. They provide the simplest means for storing integer data since they require no variable to be declared.

By default, the items in the buffer are initialized to zero at the end of the executing code block. Calling the `cell_preserve(1)` function changes this behavior so the buffer values are stored between separate handlers, making them ideal for transporting information between them.

Calling the `cell_preserve(0)` function restores the default behavior, meaning that the buffer values will be reset to zero at the end of the currently executing handler.

Comment

Note: `cell_preserve` was originally introduced as a FilterMeister function with the name `cell_initialize`. The `cell_initialize` form is now deprecated, and it is recommended that the `cell_preserve` form be substituted whenever old code is updated. `cell_initialize` took the same parameter values and performed identically.

Example

```
%ffp

OnFilterStart:
{
    // remember values in buffer
    cell_preserve(1);

    put(10, 0);
    Info("The value of buffer position 0 is %d", get(0));
}
```

See Also

get, **put**

chdir

Syntax

```
int chdir(string path)
```

Arguments

path
The path of the directory/folder you want to change to.

Return

Returns 0 if the system successfully changed to the new folder, non-zero otherwise.

Description

Changes the current working directory to the specified directory/folder on the user's filesystem. Remember to use double backslashes in the path.

Example

```
if (chdir("c:\\abc\\mynewfolder") != 0) {
  msgBox(MB_OK, "Successful", "Successfully changed the
current working directory.");
}
else msgBox(MB_OK | MB_ICONWARNING, "Error", "The system
was unable to change to the given folder.");
```

See Also

getSpecialFolder, **rmdir**, **getcwd**

CHECKBOX

Syntax

```
ctl[n]: CHECKBOX(Class Specific Properties), Other
Properties
```

Description

The checkbox acts as a toggle for two or three states. You should use this control when the user has a choice of enabling or disabling some function (eg disabling previewing, enabling a part of your filter's algorithm). The three state option should be used when the state can become "partially enabled". A common use for checkboxes is to toggle between horizontal and vertical orientation, for example in a blurring, or scanline filter.

Class Specific Properties

3STATE
: Allows the checkbox to returns one of three values (third state is a grayed check)

BORDER
: Draws a border around the checkbox.

BOTTOM
: Aligns text at the bottom of the checkbox's text area.

CENTER
: Centers text within the checkbox's text area.

CLIENTEDGE
: Draws a 3D-border around the checkbox.

FLAT
: Gives the checkbox a flat, 2D appearance.

LEFT
: Left-aligns text within the checkbox's text area. *(default)*

LEFTTEXT
Places the text label on the left of the checkbox (same as RIGHTBUTTON).
MODALFRAME
Draws a 3D-socket around the checkbox.
MULTILINE
Allows word-wrapping within the checkbox's text area.
PUSHLIKE
Makes the checkbox appear as a depressable pushbutton.
RIGHT
Right-aligns text within the checkbox's text area.
RIGHTBUTTON
Places the text label on the left of the checkbox (same as LEFTTEXT).
STATICEDGE
Draws a 3D-border around the checkbox.
TOP
Aligns text at the top of the checkbox's text area.
VCENTER
Vertically-centers text within the checkbox's text area.

Other Properties

Color
Sets text background color in plain English format *(default = transparent)*
FontColor
Sets font color in hexadecimal format *(default = #FFFFFF)*
Text
Defines the text label next to the checkbox *(default = no text)*

Comment

If you are using the class-specific property 3STATE, these are the values a function like ctl(n) returns (also applies if the class-specific property PUSHLIKE is defined):

0
: not checked

1
: checked (black check in the checkbox)

2
: checked (gray check in the checkbox)

There is a bug in Windows XP that causes checkboxes to show a black background, even if a background image hasn't been defined for the dialog. Currently, the only solution is to use the Pushlike property, define another color using the Color property, or simulate a checkbox using your own code and bitmaps.

Example

```
ctl[5]: CHECKBOX, "Horizontal/Vertical Lines",
FontColor=Darkblue
ctl[7]: CHECKBOX(3STATE), "White/Black/Gray"
```

checkCtlFocus

Requires FM 1.0 Beta 8 (Mar 2007) or newer

Syntax

```
int checkCtlFocus(int n)
```

Arguments

n
: The number of the control to check for focus

Return

Returns true if the given control has focus, false otherwise.

Description

Checks if a given control currently has the user's focus (ie is currently highlighted and responding to keyboard controls after recently being clicked on).

Example

```
ctl[0]: STANDARD, "Control 0"
ctl[1]: STANDARD, "Control 1"
ctl[8]: STATICTEXT, "Which control has focus?"

OnCtl(n): {
  if (checkCtlFocus(0)) {
    setCtlText(8, "Control 0 has focus");
  }
  else if (checkCtlFocus(1)) {
    setCtlText(8, "Control 1 has focus");
  }
  else {
    setCtlText(8, "");
  }
  return true;
}
```

See Also

setCtlFocus

checkDialogFocus

Syntax

```
int checkDialogFocus()
```

Return

True if the dialog has keyboard focus, false otherwise.

Description

Checks if the dialog has keyboard focus.

Example

```
%fml
ctl[4]: STATICTEXT, Text=""

OnCtl(n): {

  // Move the mouse over the
  // preview to display if the
  // dialog is in focus or not.

  if (checkDialogFocus()) {
    setCtlText(4, "Has focus");
  }
  else setCtlText(4, "Out of focus.");

  return false;
}
```

chooseColor

Syntax

```
int chooseColor(int initialColor, string promptString, ...)
```

Arguments

initialColor
: Specifies the initial or default color for the color picker, as an RGB triple.

promptString
: Specifies the prompt string for the color picker. This string may contain printf-style format descriptors, which will be expanded using the succeeding arguments.

...
: Variable number of arguments of varying types, should correspond to the format descriptors in promptString.

Return

The chosen color stored as an RGB triple, or -1 if the user cancels the color picker.

Description

chooseColor invokes the host application's default color picker dialog to allow the user to select an RGB color.

Example

```
int color = 0;

// Displays the color dialog with white as the default
```

```
color
color = chooseColor( RGB ( 255,255,255 ), "Please choose a
color:" );

// Display the RGB values of the chosen color
Info( "The following color was chosen: RGB (%d, %d, %d)",
Rval( color ), Gval( color ), Bval( color ) );
```

chop

Syntax

```
double chop(double number)
```

Arguments

number
Any double or float number.

Return

The truncated value.

Description

Returns the value of **number** truncated, towards 0.0, to an integral value.

Example

```
%ffp

OnFilterStart:
{
    Info("Chopping 2.543 towards 0.0 gives %f",
chop(2.543));
    Info("Chopping -2.345 towards 0.0 gives %f",
chop(-2.345));
}
```

See Also

ceil, **floor**, **ichop**, **round**, **C Runtime Functions**

clearCtlBuddyStyle

Syntax

```
int clearCtlBuddyStyle(int n, int buddy, int flags)
```

Arguments

n
: The number of the STANDARD or SLIDER control to modify

buddy
: Set to **1** to modify the edit box, or to **0** to modify the text label

flags
: The style flags to clear from this control.

Style Flags

ES_CENTER	Centers the text in the edit box when editing
ES_LEFT	Left-aligns the text in the edit box when editing
ES_LOWERCASE	Forces all letters to be lowercase in the editbox
ES_NUMBER	Only allow numbers to be typed into the editbox
ES_RIGHT	Right-aligns the text in the edit box when editing
ES_UPPERCASE	Forces all letters to be uppercase in the editbox
SS_BLACKFRAME	Adds a black rectangle around the text label
SS_BLACKRECT	Replaces the label with a blackbox

SS_ETCHEDFRAME	Adds an etched frame to the text label
SS_ETCHEDHORZ	Adds an etched frame to the text label
SS_GRAYFRAME	Adds a gray rectangle around the text label
SS_GRAYRECT	Replaces the label with a gray box
SS_WHITEFRAME	Adds a white rectangle around the text label
SS_WHITERECT	Replaces the label with a white box
WS_BORDER	Enables the single border outline of the editbox

Return

Returns false if control number n is out of range or unused. Otherwise, returns the result of the internal SetWindowPos function (true if succeeded, false otherwise).

Description

Clears the given window style of the labels and edit boxes from a **STANDARD** or **SLIDER** control.

Comment

This function must be followed by **refreshCtl** or **refreshWindow**, or the changes may not take effect.

Due to a bug, **refreshCtl** and **refreshWindow** may affect the text label rendering differently.

Example

```
%ffp

ctl(0): PUSHBUTTON, "Make Changes", Size=(60, *)
ctl(1): PUSHBUTTON, "Clear Changes", Size=(60, *), Pos=
(*,20)
ctl[4]: STANDARD, "Example"

OnCtl(n):{

  if (n==0 && e==FME_CLICKED) {

    // Make editbox Uppercase &
    // Right-aligned

    setCtlBuddyStyle(4, 1, WS_BORDER | ES_UPPERCASE |
ES_RIGHT);
    setCtlBuddyStyle(4, 2, SS_GRAYFRAME);
    refreshWindow();
  }

  if (n==1 && e==FME_CLICKED) {

    // Clear previous styles

    clearCtlBuddyStyle(4, 1, WS_BORDER | ES_UPPERCASE |
ES_RIGHT);
    clearCtlBuddyStyle(4, 2, SS_GRAYFRAME);
    refreshWindow();
  }

  return false;
}
```

See Also

refreshCtl, **refreshWindow**, **setCtlBuddyStyle**, **STANDARD**, **SLIDER**

clearCtlProperties

Syntax

```
int clearCtlProperties(int n, int props)
```

Arguments

n
: Number of the control to clear properties from

props
: The properties to clear from the control.

Return

Returns an integer with the previous properties before they were cleared by the function, or -1 if the control number is out of range or if the control doesn't exist.

Description

This function clears a property (ie a style or special characteristic) from a control that was previously set.

Example

```
// Set control #4 to trigger
// events whenever it changes
setCtlProperties(4, CTP_TRACK);

// Stop control #4 triggering
// events (ie clear the
// CTP_TRACK property)
clearCtlProperties(4, CTP_TRACK);
```

See Also

mkdir, **chdir**, **rmdir**

clearDialogEvent

Syntax

```
int clearDialogEvent(int state)
```

Arguments

state
: An integer of bitwise flags of events to disable. Set to **1** to disable Init events, **2** to disable Cancel events, **4** to disable Keypress events.

Return

Always returns true

Description

Disables certain dialog events from being processed / triggered in your own FilterMeister code. You might use this function to stop keypresses from being processed by your code, or to let FilterMeister process all cancel events internally.

Comment

Note that clearDialogEvent can only disable / deregister / clear events. To enable an event & process it, you must use **setDialogEvent**.

There appears to be a bug in Beta 9g and possibly newer FilterMeister versions that prevents **FME_CANCEL** events from being processed in code, even if you use this function to enabled Cancel events.

Example

```
ctl[0]: CHECKBOX, "Enable Init Events"
ctl[1]: CHECKBOX, "Enable Cancel Events"
ctl[2]: CHECKBOX, "Enable Keypress Events"
ctl[4]: STATICTEXT, ""
ctl[5]: STATICTEXT, ""


OnCtl(n): {

  if (e == FME_INIT) {
    Info("The Init event was intercepted.");
    return true;
  }

  if (e == FME_CANCEL) {
    Info("Cancel event was intercepted!");
    return true;
  }

  if (e == FME_KEYDOWN) {
    printf("Key down!");
    return true;
  }


  if (n >= 0 && n <= 2 && e == FME_CLICKED) {

    int statevalue = ctl(0)*1 + ctl(1)*2 + ctl(2)*4;

    // Enable events we turned on
    setDialogEvent( statevalue );
    sprintf(str1, "setDialogEvent(%d) called",
statevalue);
```

```
    setCtlText(4, str1);

    // Clear events that we
    // turned off
    // (Use bitwise XOR)
    clearDialogEvent( statevalue ^ 7);
    sprintf(str2, "clearDialogEvent(%d) called",
statevalue ^ 7);
    setCtlText(5, str2);
  }

  return false;
}
```

See Also

setDialogEvent, **FME_INIT**, **FME_CANCEL**, **FME_KEYDOWN**, **FME_KEYUP**

clearerr

Syntax

```
void clearerr(int filepointer)
```

Arguments

filepointer
    file pointer as returned by **fopen** and such.

Description

Clears the end-of-file and error indicators for the given file. As long as the error indicator is set, all file operations will return an error until **clearerr** or rewind is called.

See Also

ferror

clock

Syntax

```
int clock()
```

Return

The number of clock ticks since the plug-in was started.

Description

Returns the number of clock ticks elapsed since the plug-in was started. You can use this function to measure small differences in time between two events, such as for performance testing in your plug-in code.

See Also

time

cmyk2rgb

Syntax

`int cmyk2rgb(int c, int m, int y, int k, int z)`

Arguments

c
: Cyan value

m
: Magenta value

y
: Yellow value

z
: Determines which value is returned. z=0 for R (Red), z=1 for G (Green), z=2 for B (Blue)

Return

Returns the R, G, B value from 0 to 255 depending on the value of z

Description

Lets you convert CMYK color values to RGB color values.

Example

```
%ffp

ctl(0): "Adjust C", Range=(-255,255), val=0
ctl(1): "Adjust M", Range=(-255,255), val=0
ctl(2): "Adjust Y", Range=(-255,255), val=0
ctl(3): "Adjust K", Range=(-255,255), val=0

ForEveryTile:{

  int r,g,b,cyan,mag,yel,k;

  for (y= y_start; y < y_end; y++) {

    if (updateProgress(y, y_end)) abort();

    for (x = x_start; x < x_end; x++) {

      r = src(x,y,0);
      g = src(x,y,1);
      b = src(x,y,2);

      cyan = rgb2cmyk(r,g,b,0);
      mag = rgb2cmyk(r,g,b,1);
      yel = rgb2cmyk(r,g,b,2);
      k = rgb2cmyk(r,g,b,3);

      // Do the CMYK adjustment
      cyan = cyan + ctl(0);
      mag = mag + ctl(1);
      yel = yel + ctl(2);
      k = k + ctl(3);

      pset( x, y, 0, cmyk2rgb(cyan,mag,yel,k,0) );
      pset( x, y, 1, cmyk2rgb(cyan,mag,yel,k,1) );
      pset( x, y, 2, cmyk2rgb(cyan,mag,yel,k,2) );
    }
  }

  return true;
}
```

See Also

rgb2cmyk

cnv

Syntax

```
int cnv(int m11, int m12, int m13, int m21, int m22, int
m23, int m31, int m32, int m33, int d)
```

Arguments

m11
: An integer weighting for the pixel at [x-1,y-1]

m12
: An integer weighting for the pixel at [x,y-1]

m13
: An integer weighting for the pixel at [x+1,y-1]

m21
: An integer weighting for the pixel at [x-1,y]

m22
: An integer weighting for the pixel at [x,y]

m23
: An integer weighting for the pixel at [x+1,y]

m31
: An integer weighting for the pixel at [x-1,y+1]

m32
: An integer weighting for the pixel at [x,y+1]

m33
: An integer weighting for the pixel at [x+1,y+1]

d
: An integer divisor - usually the sum of the nine weighting values

Return

An integer in the range 0 to 255

Description

This function is valid only in the RGBA and ForEveryPixel handlers; the **xyzcnv** function performs the same operation in the ForEveryTile handler. The cnv function performs a basic convolution operation using a 3 by 3 matrix of image pixels and a corresponding matrix of weightings. The parameters to the function call represent the values of the weighting matrix, presented in row order - in other words, in the designation used here the first digit after the 'm' represents the row number and the second the column. The divisor 'd' should usually be given as the sum of the nine weightings. The cnv function sums the values for the nine pixels centered on the current [x,y] position and channel, multiplying each pixel value by the corresponding weighting from the matrix; the value returned is then that sum divided by the divisor 'd'.

Example

```
%ffp

// Simple 3x3 Gaussian blur
R,G,B,A :  cnv(1,2,1, 2,4,2, 1,2,1, 16)
```

See Also

xyzcnv

cnvX

Syntax

```
int cnvX(int k, int off, int d, function* pGetf, int x, int
y, int z)
```

Arguments

k
: The kernel radius. Must be a positive number.

off
: The starting index within the anonymous array of **put**/**get** cells where the kernel coefficients are stored. The number of coefficients is n=k*2+1, where 'k' is the kernel radius.

d
: The denominator by which the convolution sum will be divided.

pGetf
: The FM built-in function that will be called to fetch values from an image buffer at the designated coordinates. Must be 'src', 'tget', 't2get', or 'pget'.

x
: The x-coordinate at which the (center of the) kernel is to be applied.

y
: The y-coordinate at which the (center of the) kernel is to be applied.

z
: The channel number to which the kernel is applied.

Return

The integer result obtained by summing the products of the n kernel coefficients with n image pixels in the X direction, and dividing the sum by 'd'.

Description

Applies a 1-D convolution to the image in the horizontal / x direction. That is, cnvX convolves pixels at co-these ordinates:

(x-k,y), (x-k+1,y), ... (x-1,y), (x,y), (x+1,y), ... (x+k-1,y), (x+k,y)

Example

```
// 3-pixel radius blur in the x-direction

ForEveryTile: {

  // Convolution coefficients
  // in first 7 cells
  put(1, 0);
  put(2, 1);
  put(3, 2);
  put(4, 3);
  put(3, 4);
  put(2, 5);
  put(1, 6);

  int val;

  for (z = 0; z < 3; z++) {
    for (y = y_start; y < y_end; y++) {
      for (x = x_start; x < x_end; x++) {

        val = cnvX(3, 0, 16, src, x, y, z);
        pset(x, y, z, val);
      }
    }
  }

  return true;
}
```

See Also

cnv, cnvY, xyzcnv

cnvY

Syntax

```
int cnvY(int k, int off, int d, function* pGetf, int x, int y, int z)
```

Arguments

k
: The kernel radius. Must be a positive number.

off
: The starting index within the anonymous array of **put**/**get** cells where the kernel coefficients are stored. The number of coefficients is n=k*2+1, where 'k' is the kernel radius.

d
: The denominator by which the convolution sum will be divided.

pGetf
: The FM built-in function that will be called to fetch values from an image buffer at the designated coordinates. Must be 'src', 'tget', 't2get', or 'pget'.

x
: The x-coordinate at which the (center of the) kernel is to be applied.

y
: The y-coordinate at which the (center of the) kernel is to be applied.

z
: The channel number to which the kernel is applied.

Return

The integer result obtained by summing the products of the n kernel coefficients with n image pixels in the Y direction, and dividing the sum by 'd'.

Description

Applies a 1-D convolution to the image in the vertical / y direction. That is, cnvY convolves pixels at co-these ordinates:

(x,y-k), (x,y-k+1), ... (x,y-1), (x,y), (x,y+1), ... (x,y+k-1), (x,y+k)

Example

```
// 3-pixel radius blur in the y-direction

ForEveryTile: {

  // Convolution coefficients
  // in first 7 cells
  put(1, 0);
  put(2, 1);
  put(3, 2);
  put(4, 3);
  put(3, 4);
  put(2, 5);
  put(1, 6);

  int val;

  for (z = 0; z < 3; z++) {
    for (y = y_start; y < y_end; y++) {
      for (x = x_start; x < x_end; x++) {

        val = cnvY(3, 0, 16, src, x, y, z);
        pset(x, y, z, val);
      }
    }
  }

  return true;
}
```

See Also

cnv, **cnvX**, **xyzcnv**

COMBOBOX

Syntax

```
ctl[n]: COMBOBOX(Class Specific Properties), Other
Properties
```

Description

Comboboxes are good for drop-down lists, also known as "pull-down menus". The items in the listbox are specified in the Text string, separated with the new-line escape sequence (\n). Each item has a unique integer value, starting at 0 and increasing.

Class Specific Properties

DISABLENOSCROLL
: Used in conjunction with HSCROLL or VSCROLL; if the item amount is less than needed to require scrolling, the scrollbar is disabled (instead of it being removed).

EXTENDEDUI
: List drops down when right and down arrow keys are pressed. By default, arrow keys select the next item without dropping down the list.

INTEGRALHEIGHT
: The height of the listbox is resized according to the items' height. *(default)*

LOWERCASE
: Items in the combobox are displayed in lowercase characters.

NOINTEGRALHEIGHT
: The height of the listbox is resized according to the Size property, even if items are partially displayed.

SORT

Sorts the items in alphabetical order. The values of the items are recomputed - the top item is always 0 and continues with 1, 2, etc.

UPPERCASE
Items in the combobox are displayed in uppercase characters.

VSCROLL
If necessary, a vertical scrollbar is activated.

Other Properties

Text
Defines the combobox's text contents. *(default = no text)*

Val
Assigns a value to the combobox and activates the item. *(default = -1)*

Example

```
ctl[0]: COMBOBOX(VSCROLL, DISABLENOSCROLL),
        Text="U.S.A.\nGermany\nRussia\n"
        "Brazil\nSpain\nEgypt", Val=1,
        Size=(*,60)
```

COMBOBOX usage example

```
%ffp

ctl[0]: COMBOBOX(VSCROLL),
        "Algorithm 1\nAlgorithm 2\n"
        "Algorithm 3\Algorithm 4", Val=0,
        Size=(130,60), Action=PREVIEW

ForEveryTile:
{
  // Loop through all rows
  for (y = y_start; y < y_end; y++)
  {
    // Loop through all columns
    for (x = x_start; x < x_end; x++)
    {
      switch (ctl(0))
      {
        case 0: //Algorithm 1
        ... Your code
        break;

        case 1: //Algorithm 2
        ... Your code
        break;

        case 2: //Algorithm 3
        ... Your code
        break;

        case 3: //Algorithm 4
        ... Your code
        break;
      }
    }
  }

  // Tile has been completely processed
  return true;
}
```

Comment

When using the INTEGRALHEIGHT and Size=(x,y) properties, the y value refers to the height of the list displayed after the combobox is clicked. The height is automatically rounded-off to display a whole number of textlines. You cannot extend the height of the combobox beyond the length of the combobox list (FilterMeister will round down values that are too high). This behaviour can be overridden using the NOINTEGRALHEIGHT property.

See Also

LISTBOX

contrast

Syntax

```
int contrast(int pixel, int contrast)
```

Arguments

pixel
: The pixel color value the contrast algorithm will be applied to.

contrast
: The amount of contrast to apply on a range of -127 to 128, with 0 meaning no contrast adjustment.

Return

The new pixel color value after contrast is applied.

Description

Applies a simple contrast effect to a given pixel value.

Example

```
ctl[0]: "Contrast", Range=(-127,128), Val=0

ForEveryTile: {
  for (y = y_start; y < y_end; y++) {
    for (x = x_start; x < x_end; x++) {
       for (z=0; z < 3; z++) {
          pset(x,y,z, contrast(src(x,y,z), ctl(0)));
       }
    }
  }
  return true;
}
```

See Also

blend, **gamma**, **gray**, **saturation**

copyArray

Syntax

```
int copyArray (int src, int dest)
```

Arguments

src
: Number of the source array whose data will be copied. Values from 0 to 99 are allowed.

dest
: Number of the destination array to which the data from the source array will be copied. Values from 0 to 99 are allowed.

Return

Returns zero if it failed and 1 if it succeeded.

Description

Lets you very quickly create a duplicate of the source array. If the source array doesn't exist, this function will fail. If the destination array doesn't exist, it will be automatically created. The destination array will automatically be adjusted to the size of the source array. Data that was stored in the destination array will be overwritten.

See Also

allocArray, freeArray, getArray, putArray, getArrayDim

copyResToArray

Requires FM 1.0 Beta 8.5 (Nov 2007) or newer

Syntax

```
bool copyResToArray(string restype, string resname, int arraynr)
```

Arguments

restype
: The type of the resource (probably "FMDATA" in most cases).

resname
: The name of the resource to copy into an array.

arraynr
: The number of the built-in FM array to copy the resource into

Return

Returns true if copying the resource into the array succeeded, false otherwise.

Description

Copies a file that has been embedded as a resource in the plugin DLL into a built-in FM array.

You can embed any kind of file into a plugin using the Other embed type, e.g. Embed: Other="Test.txt". The file will be embedded as an "FMDATA" resource type in the plugin 8BF DLL file, and can be verified with third-party programs like Resource Hacker.

From FM 1.0 Beta 9.1 (Feb 2009) onwards, if copyResToArray can't find a resource with the given name, it will also try looking for a file with the same name as the resource.

Example

```
%fml

Title:    "Test copyResToArray"
Category: "FM Example Code"
Embed:    Other="test.txt"

// Make a control to display text on the interface
ctl[10]: STATICTEXT, Size=(100, 40), Text="This text will
be replaced"

OnFilterStart: {

  bool loaded = false;

  // Load the test.txt file embedded in the plugin into
Array #0
  loaded = copyResToArray("FMDATA", "test.txt", 0);

  // If the resource loaded, copy the (max) first 255
characters
  // into an internal string variable, so it correctly
ends
  // with a zero byte to mark the end of string
  if (loaded) {
    strncpy(str1, getArrayAddress(0), 255);
  }
  else {
    strncpy(str1, "Could not find test.txt file", 255);
  }

  // Set control #10 to be labelled with the contents of
str1
  setCtlText(10, str1);

  return true;
}
```

See Also

getArrayAddress

cosineInterpolate

Syntax

```
int cosineInterpolate(int v1, int v2, double x)
```

Arguments

v1
: The first value to interpolate between

v2
: The second value to interpolate between

x
: The point between the two values to interpolate at, a floating value between 0.0 and 1.0.

Return

The integer result of interpolating between the two values.

Description

Interpolates between two values according to a cosine function. If you have values at two known points, you can estimate (interpolate) the value somewhere between those two points using this function. This is useful if you need to estimate a pixel value "between" the actual pixels, for example when zooming into an image. Cosine interpolation can create a slightly crisper/sharper result than linear interpolation.

Example

This example performs a kind of cosine interpolated zoom operation.

```
%fml
ctl[0]: STANDARD, Text="Zoom", Val=100
ctl[1]: CHECKBOX, Text="Use Cosine Interpolation", Val=1

ForEveryTile: {
  for (y=0; y < Y; y++) {
    for (x=0; x < X; x++) {
      for (z=0; z < Z; z++) {

        double srcx = 100.0 * x / ctl(0);
        double srcy = 100.0 * y / ctl(0);

        int topleft  = src((int)floor(srcx),
(int)floor(srcy), z);
        int topright = src((int)ceil(srcx),
(int)floor(srcy), z);
        int lwrleft  = src((int)floor(srcx),
(int)ceil(srcy), z);
        int lwrright = src((int)ceil(srcx),
(int)ceil(srcy), z);

        int interpolatedtop, interpolatedlwr,
interpolated;

        if (ctl(1)) {
          interpolatedtop = cosineInterpolate(topleft,
topright, srcx - floor(srcx));
          interpolatedlwr = cosineInterpolate(lwrleft,
lwrright, srcx - floor(srcx));
          interpolated =
cosineInterpolate(interpolatedtop, interpolatedlwr, srcy -
floor(srcy));
        }
        else {
          interpolatedtop = linearInterpolate(topleft,
topright, srcx - floor(srcx));
          interpolatedlwr = linearInterpolate(lwrleft,
lwrright, srcx - floor(srcx));
          interpolated =
linearInterpolate(interpolatedtop, interpolatedlwr, srcy -
floor(srcy));
        }

        pset(x, y, z, interpolated);

      }
    }
  }
  return true;
}
```

Comment

To perform interpolation across an image, it is easier to use the **iget** function, which does the hard work for you.

See Also

iget, **linearInterpolate**

countProcessors

Requires FM 1.0 Beta 9d (June 2008) or newer

Syntax

`int countProcessors(void)`

Return

Returns the number of physical and/or logical processors on your system, as reported by the Windows GetSystemInfo API.

Description

Lets you know the number of processors you have at your disposal to perform useful work. The process count includes each separate physical cpu, or each core within a multicore cpu, or the two logical cpus within a Hyper-Threaded cpu.

Comments

It would be nice to have an API that gives additional info about the processors, such as which ones are physical, which are logical, which logical processors reside on the same H-T or multicore processor, etc.

Example

```
%ffp

OnFilterStart:
{
    int ncpu = countProcessors();

    Info ("This system has %d available processor%s.", 
ncpu, ncpu==1?"":"s");

    return false;  //continue processing
}
```

See Also

System Functions, triggerThread, waitForThread, isThreadActive, getThreadRetVal, terminateThread

createCircularRgn

Syntax

```
region createCircularRgn(int x, int y, int d)
```

Arguments

x
: x defines abscissa (in DBUs) of upper-left corner of the bounding box (not the circle's center!).

y
: y defines ordinate (in DBUs) of upper-left corner of the bounding box (not the circle's center!).

d
: d defines the circle's diameter

Return

Return a circular region (see comments)

Description

Allows to define a rounded circular visual region, that can be for instance handled by the functions **refreshRgn** or **setDialogRegion**.

Example

```
setDialogRegion( createCircularRgn(20, 20, 200) );
```

Comments

"region" corresponds to a particular FM type or object that might need to be better documented or defined. You can't directly cast it to an integer, despite it behaving like an integer.

createCircularRgn is a wrapper around the Windows Win32 built-in function CreateEllipticRgn, documented here: **Microsoft MSDN CreateEllipticRgn function**

See Also

createEllipticRgn, **createRectRgn**, **createRoundRectRgn**, **createCircularRgn**, **createPolyRgn**, **refreshRgn**, **setDialogRegion**

createCriticalSection

Requires FM 1.0 Beta 9d (June 2008) or newer

Syntax

```
int createCriticalSection(void)
```

Return

Returns a handle to a newly created Critical Section, or 0 if the Critical Section could not be created.

Description

This function creates a new Critical Section, which may be used to enforce mutually exclusive access to a resource or set of resources. At most one thread can "enter" a Critical Section at any given time. Any other threads wishing to enter the same Critical Section must wait until the first thread "leaves" the Critical Section. If you have more that one set of resources that need protection, you can create a separate Critical Section to guard each resource, thus allowing a higher degree of parallel processing, since a thread using one such resource does not lock out other threads from using other guarded resources at the same time.

For more information about Critical Sections, see the MSDN documentation about **[Critical Section Objects]**.

Example

```
%fml
```

```
// Sample filter creates 5 worker
// threads. Each thread is
// assigned the "next available"
// line "y" to process, until all
// lines have been processed.
// Since multiple threads are
// attempting to read and
// increment the "nextY" variable
// at the same time, we use a
// Critical Section to serialize
// access to "nextY".
//
// N.B. This example assumes
// global variables are
// implemented in such a manner
// that they are shared among all
// threads (which is not yet true
// as of FM 1.0 Beta 9e).

// Global int to hold next line
// to be processed
int nextY;

// Critical Section handle for
// guarding access to "nextY"
int csNEXTY;

OnCtl(n):{

  if (e==FME_CUSTOMEVENT && n==666) {

    // Perform function #666
    while(1) {

      // Need to serialize access
      // to "nextY"; else two
      // threads may try to
      // increment nextY at the
      // same time, with
      // indeterminate results.

      enterCriticalSection(csNEXTY);
        // "nextY" is a
        // shared global var
        int y = nextY++;
      leaveCriticalSection(csNEXTY);

      // exit thread if no more
      // lines to process
      if (y >= y_end) return true;

      // process this line y...
      for (int x = x_start; x < x_end; x++) {
        for (int z = 0; z < Z; z++) {

          // invert this pixel...
          pset(x,y,z, 255-src(x,y,z));
          // do other work here

        } //for z
      } //for x
    } //while(1)
  } //if FME_CUSTOMEVENT 666

  // other events not yet processed
  return false;
}


ForEveryTile: {

  // Create a Critical Section to
  // guard access to nextY

  csNEXTY = createCriticalSection();
  if (!csNEXTY) {
    ErrorOk("Failed to create Critical Section csNEXTY");
    return false;
  }

  // Set initial value of
  // "nextY" to y_start
  nextY = y_start;

  // Start 5 worker threads...
  for (int i=0; i < 5; i++)
    triggerThread(666,FME_CUSTOMEVENT,i);

  // Wait for  threads to finish
  waitForThread(0, INFINITE, 0);

  // Delete the Critical Section
  deleteCriticalSection(csNEXTY);

  // finished processing tile
  return true;
}
```

See Also

System Functions, enterCriticalSection, tryEnterCriticalSection, leaveCriticalSection, deleteCriticalSection

createCtl

Syntax

```
createCtl(int n, int c, string t, int x, int y, int w, int
h, int s, int sx, int p, int e)
```

Arguments

n
: Index number to give this control.

c
: Class of control to create.

t
: Text label for the control.

x
: X-axis position, the left-most position of the control (in DBUs)

y
: Y-axis position, the upper-most position of the control (in DBUs)

w
: Width of the control (in DBUs)

h
: Height of the control (in DBUs)

s
: Style parameters for the control.

sx
: Extended style parameters for this control.

p
: Properties to give this control.

e
: Enable level of the control.

Description

This function creates a new user control. Unlike the control definitions at the start of your source code, createCtl can create controls at runtime from within your filter code. In conjunction with the **deleteCtl** function, for example, you can dynamically create and destroy user controls.

The class c should be one of the following predefined values: CC_STANDARD, CC_SCROLLBAR, CC_TRACKBAR, CC_CHECKBOX, CC_PUSHBUTTON, CC_GROUPBOX, CC_RADIOBUTTON, CC_LISTBOX, CC_COMBOBOX, CC_OWNERDRAW, CC_STATICTEXT, CC_FRAME, CC_RECT, CC_BITMAP, CC_IMAGE, CC_ICON.

For x, y, w, and h, a value of -1 means use the default value.

The style of the user control differs from control to control, but you can try using:

CC_CHECKBOX	use s=BS_CHECKBOX (2), BS_AUTOCHECKBOX (3), BS_3STATE (5), or BS_AUTO3STATE (6)
CC_GROUPBOX	use s=BS_GROUPBOX (7)
CC_RADIOBUTTON	use s=BS_RADIOBUTTON (4) or BS_AUTORADIOBUTTON (9)
CC_COMBOBOX	use s=CBS_SIMPLE (1), CBS_DROPDOWN (2), or CBS_DROPDOWNLIST (3)
CC_OWNERDRAW	use s=BS_OWNERDRAW (11)
CC_FRAME	use s=SS_BLACKFRAME (7), SS_GRAYFRAME (8), SS_WHITEFRAME (9), or SS_ETCHEDFRAME (18)
CC_RECT	use s=SS_BLACKRECT (4), SS_GRAYRECT (5), or SS_WHITERECT (6) for black, gray,

	or white fill
CC_STATICTEXT	use s=SS_LEFT (0) for left-aligned text, SS_CENTER (1) for center-aligned text, SS_RIGHT (2) for right-aligned text, or SS_LEFTNOWORDWRAP (12) for left-aligned text with no word wrap. This corresponds to the LEFT, CENTER, RIGHT, and LEFTNOWORDWRAP styles in control definitions.
rest of the user controls	use s=0

The extended style of the control may differ depending on class type, but generally you can set the value to 0 (for default settings), WS_EX_DLGMODALFRAME (1) for a MODALFRAME-like look, WS_EX_CLIENTEDGE (0x200) for a sunken 3D-look, and WS_EX_STATICEDGE (0x20000) for a 3D border. *(NB: check this??)*

The enable level can be any of 0 (for invisible and disabled), 1 (for visible but disabled), or 3 (for visible and enabled).

Example

```
createCtl(0, CC_PUSHBUTTON, "Oh boy", 200, 20, 40, 30, 0,
1, 0, 3);
createCtl(4, CC_CHECKBOX, "Disrupt color", -1, -1, -1, -1,
3, 0, 0, 3);
```

See Also

clearCtlProperties, **deleteCtl**, **setCtlProperties**

createEllipticRgn

Syntax

```
region createEllipticRgn(int x1, int y1, int x2, int y2)
```

Arguments

x1
: The x coordinate (in DBUs) of the upper-left corner of the bounding box of the ellipse.

y1
: The y coordinate (in DBUs) of the upper-left corner of the bounding box of the ellipse.

x2
: The x coordinate (in DBUs) of the lower-right corner of the bounding box of the ellipse.

y2
: The y coordinate (in DBUs) of the lower-right corner of the bounding box of the ellipse.

Return

Return an elliptic region (see comments)

Description

Allows to define an elliptic visual region, that can be for instance handled by the functions **refreshRgn** or **setDialogRegion**

Comments

"region" correspond to a particular FM type or object that might need to be better documented or defined. You can't directly cast it to an integer, despite it behaves like an integer.

Example

```
setDialogRegion( createEllipticRgn(20, 20, 300, 200) );
```

See Also

createCircularRgn, **createRectRgn**, **createRoundRectRgn**, **createCircularRgn**, **createPolyRgn**, **refreshRgn**, **setDialogRegion**

createFont

Requires FM 1.0 Beta 9.1 (Feb 2009) or newer

Syntax

```
bool createFont(int i, int size, bool bold, bool italic,
char *fontname)
```

Arguments

i
: The index number of the font (from 0 - 31) to create.

size
: The size (height) of the font, in logical units.

bold
: Set to true if the font should be bold, false otherwise

italic
: Set to true if the font should be in italics, false otherwise

fontname
: Pointer to a string with the font name, as embedded in the font file

Return

Returns true if creating the font succeeded, false otherwise.

Description

Creates a font that can be used in your plug-in user interface with **setCtlFont**. The font must already be installed on the user's computer, so it is best to choose fonts that come installed with Windows. You can use up to 32 different fonts, and the fonts are deleted automatically when FM exits. if a font already exists at the given index number, it is deleted and replaced with a pointer to the newly created font.

Comment

Behind the scenes, this function is mostly a wrapper around the [CreateFont] Win32 GDI API function.

Example

```
%fml

// Create some text to display on the interface
ctl[10]: STATICTEXT, Size=(100, 40), Text="The Text
Appears Here"


OnFilterStart: {

  // Create font #1 as Segoe Print in height 40
  createFont(1, 40, false, false, "Segoe Print");

  // Set control #10 to use our newly created Segoe Print
40
  setCtlFont(10, 1);

  return true;
}
```

See Also

setCtlFont, setCtlFontColor, deleteFont

createPolyRgn

Syntax

```
region createEllipticRgn(int fill_mode,int x1, int y1, int x2, int y2, ..., int xn, int yn)
```

Arguments

fill_mode
Use either the predefined constant ALTERNATE or WINDING to set this parameter. Then FM will either fill or not overlapping areas.

x1,y1, x2,y2, ..., xn,yn
At least three coordinate pairs must be defined.

Return

Return a polygonal region (see comments).

Comments

"region" correspond to a particular FM type or object that might need to be better documented or defined. You can't directly cast it to an integer, despite it behaves like an integer.

Example

```
setDialogRegion( createPolyRgn( WINDING, 6, 50, 45, 6,
100, 30, 95, 87, 36, 102 ) );
```

See Also

createCircularRgn, createEllipticRgn, createRectRgn, createRoundRectRgn, createCircularRgn, refreshRgn, setDialogRegion

createPopupMenu

Syntax

```
int createPopupMenu();
```

Return

A pointer to the created menu object.

Description

Start a new popup menu.

Comment

You'll need to use **insertMenuItem** to add menu items and **trackPopupMenu** to display it. After using the menu, you must destroy it using **destroyMenu**.

Example

```
%ffp

ctl[0]: PUSHBUTTON, "Click Me!"

OnCtl(n): {

  if (n==0 && e == FME_CLICKED){
    int menu=0;

    menu = createPopupMenu();

    insertMenuItem(menu, 1, "Do This",MFS_ENABLED , NULL);
    insertMenuItem(menu, 2, "Do That",MFS_ENABLED |
MFS_DEFAULT, NULL);
    insertMenuItem(menu, 3, "Do Nothing",MFS_ENABLED,
NULL);
    Info("Selection: %d", trackPopupMenu (menu, 1, 0,0,0)
);

    destroyMenu(menu);
  }

  return false;
}
```

See Also

insertMenuItem, **trackPopupMenu**, **destroyMenu**

createRectRgn

Syntax

```
region createRectRgn(int x1, int y1, int x2, int y2)
```

Arguments

x1
: The x coordinate (in DBUs) of the upper-left edge of the rectangle.

y1
: The y coordinate (in DBUs) of the upper-left edge of the rectangle.

x2
: The x coordinate (in DBUs) of the lower-right edge of the rectangle.

y2
: The y coordinate (in DBUs) of the lower-right edge of the rectangle.

Return

Return a rectangular region (see comments)

Description

Allows to define a rectangular visual region, that can be for instance handled by the functions **refreshRgn** or **setDialogRegion**

Comments

"region" correspond to a particular FM type or object that might need to be better documented or defined. You can't directly cast it to an integer, despite it behaves like an integer.

Example

```
refreshRgn( createRectRgn(215,30,410,250) );
```

See Also

createRoundRectRgn, **createCircularRgn**, **createEllipticRgn**, **createPolyRgn**, **refreshRgn**, **setDialogRegion**

createRoundRectRgn

Syntax

```
region createRoundRectRgn(int x1, int y1, int x2, int y2, int w, int h)
```

Arguments

x1
: The x coordinate (in DBUs) of the upper-left edge of the rectangle.

y1
: The y coordinate (in DBUs) of the upper-left edge of the rectangle.

x2
: The x coordinate (in DBUs) of the lower-right edge of the rectangle.

y2
: The y coordinate (in DBUs) of the lower-right edge of the rectangle.

w
: Width of the ellipse for the rounded corners

h
: Height of the ellipse for the rounded corners

Return

Return a rounded rectangular region (see comments)

Description

Allows to define a rounded rectangular visual region, that can be for instance handled by the functions **refreshRgn** or setDialogRegion

Comments

"region" correspond to a particular FM type or object that might need to be better documented or defined. You can't directly cast it to an integer, despite it behaves like an integer.

Example

```
refreshRgn( createRoundRectRgn(215,30,410,250, 10,20) );
```

See Also

createRectRgn, createCircularRgn, createEllipticRgn, createPolyRgn, refreshRgn, setDialogRegion

ctl

Syntax

`int ctl(int id)`

Arguments

id
: A numeric identifier for a control.

Return

The current value of the control specified. Max range: from -9999999 (7 digits) to +99999999 (8 digits).

Description

Returns the current value of the user control specified by the `id` argument. There are max. 250 ctl's possible: ctl(0) through ctl(249).

Example

```
%ffp

ctl(0): STANDARD

OnFilterStart:
{
    Info("The value of control 0 is %d", ctl(0));
    return false;
}
```

ctlEnabled

Syntax

```
int ctlEnabled(int n)
```

Arguments

n
Control number whose state will be queried.

Return

Returns 0 if the control is invisible, 1 if the control is disabled or 3 if the control is visible and enabled.

Description

This function lets you check the state of a control. Without this function you would have to store the control states in an array or in the put/get cells which would be less convenient. While you can check the state of a control with this function, you can set it with the **enableCtl** function.

See Also

enableCtl

ctlEnabledAs

Requires FM 1.0 Beta 9.1 (Feb 2009) or newer

Syntax

```
int ctlEnabledAs(int n)
```

Arguments

n
: Control number whose state will be queried.

Return

Returns 0 if the control is invisible, 1 if the control is disabled or 3 if the control is visible and enabled.

Description

This function is closely related to **ctlEnabled**. Whereas ctlEnabled returns the state of the control itself, ctlEnabledAs returns the enabled state depending on the state(s) of any parent controls the specified control might have.

If a control is visible (state 3) by itself, but is part of a tab control (the parent) which is invisible (state 0), this control will not be drawn. ctlEnabled would return state 3 (visible), ctlEnabledAs would return 0 (invisible).

This function will return the "lowest" state of all states of the control itself and all it's parent controls, in order from low to high: 0 (invisible), 1 (disabled), 3 (visible and enabled).

See Also

ctlEnabled

deleteCriticalSection

Requires FM 1.0 Beta 9d (June 2008) or newer

Syntax

```
bool deleteCriticalSection(int hCS)
```

Arguments

hCS
: Specifies the handle of the Critical Section to be deleted, as obtained from a call to **createCriticalSection**.

Return

This function returns **true** immediately after deleting the specified Critical Section. It returns **false** if hCS is zero.

Description

This function deletes a specified Critical Section and releases all system resources that were allocated to it. Any thread can delete the Critical Section, but it must be currently unowned by any thread. After a Critical Section is deleted, it can no longer be used as a synchronization object until it is recreated by a call to **createCriticalSection**.

For more information about Critical Sections, see the MSDN documentation about **[Critical Section Objects]**.

Comments

One need not normally test the return value of **deleteCriticalSection**. This is merely a check to make sure that hCS is non-zero so the **DeleteCriticalSection** Win32 API won't cause a memory access violation.

Example

See the **createCriticalSection example**.

See Also

System Functions, **createCriticalSection**, **enterCriticalSection**, **tryEnterCriticalSection**, **leaveCriticalSection**

deleteCtl

Syntax

```
void deleteCtl(int n);
```

Arguments

n

The index number of the control you want to delete.

Description

This function simply deletes the user control with the index n.

Example

```
deleteCtl(5);
deleteCtl(CTL_CANCEL);
```

See Also

createCtl

deleteCtlItem

Syntax

```
int deleteCtlItem(int n, int itemnum)
```

Arguments

n
: The number of the listbox/combobox control to delete an item from

itemnum
: The number of the item to delete from the listbox/combobox/tab control

Return

Returns false if the control number is out of range, not in use, or not a LISTBOX, COMBOBOX or TAB control. Otherwise, returns true if the function succeeded, false otherwise.

Description

Deletes an item in a LISTBOX, COMBOBOX or a TAB control.

Example

```
%fml
ctl[0]: LISTBOX(VSCROLL),
Text="Harry\nLarry\nSally\nCarrie"
ctl[3]: PUSHBUTTON, Text="Delete 1st Person", Size=(100,*)
ctl[5]: PUSHBUTTON, Text="Delete 2nd Person", Size=(100,*)
ctl[7]: PUSHBUTTON, Text="Reset Everyone", Size=(100,*)
ctl[11]: STATICTEXT, Text="People Left: ", Pos=(*, 100)
```

```
OnCtl(n): {

  if (n==3 && e == FME_CLICKED) {
    deleteCtlItem(0, 0);
  }

  if (n==5 && e == FME_CLICKED) {
    deleteCtlItem(0, 1);
  }

  if (n==7 && e == FME_CLICKED) {
    setCtlText(0, "Harry\nLarry\nSally\nCarrie");
  }

  return false;
}

OnFilterStart: {
  setCtlTextv(11, "People Left: %d", getCtlItemCount(0));
  return false;
}
```

See Also

getCtlItemCount, **deleteCtlItems**, **setCtlText**, **setCtlTextv**, **COMBOBOX**, **LISTBOX**, **TAB**

deleteCtlItems

Warning: This function causes memory access violations, do not use.

Syntax

`int deleteCtlItems(int n)`

Arguments

n
: The number of the listbox/combobox/tab control to delete all items from

Return

Returns false if the control number is out of range, not in use, or not a **LISTBOX**, **COMBOBOX** or **TAB** control.

Description

Deletes all items in a **LISTBOX**, **COMBOBOX** or **TAB** control.

Comment

Please do not use this function. Note that there is currently a bug in this function that causes memory access violations, resulting in a crash in the host application when used with listboxes & comboboxes. As a workaround, use `setCtlText(n,"")` instead to reset the text contents to zero, which is the same as what FilterMeister does internally (or at least, is meant to).

Example

```
%fml
ctl[0]: LISTBOX(VSCROLL),
        Text="Harry\nLarry\n"
        "Sally\nCarrie"
ctl[4]: PUSHBUTTON, Text="Delete Everyone", Size=(100,*)
ctl[7]: PUSHBUTTON, Text="Add Everyone", Size=(100,*)

OnCtl(n): {

  if (n==4 && e == FME_CLICKED) {
    // Due to a bug, this next
    // line causes a memory
    // access violation
    //deleteCtlItems(0);

    // As a workaround, do this
    setCtlText(0, "");
  }

  if (n==7 && e == FME_CLICKED) {
    setCtlText(0, "Harry\n"
    "Larry\nSally\nCarrie");
  }

  return false;
}
```

See Also

deleteCtlItem, setCtlText, COMBOBOX, LISTBOX, TAB

deleteFont

Requires FM 1.0 Beta 9.1 (Feb 2009) or newer

Syntax

```
int deleteFont(int n)
```

Arguments

n
: The number of the font to delete (from memory), in the range 0 - 31.

Return

Returns true if the font was successfully removed from FM memory, false otherwise.

Description

Deletes a font object reference from memory in FilterMeister. The font must first have been created/assigned using the **createFont** function.

Comment

Behind the scenes, FilterMeister just uses the **[DeleteObject]** function from the Win32 API.

Example

```
// Delete the object for font #10
deleteFont(10);
```

See Also

createFont, setCtlFont

deleteRegValue

Syntax

```
int deleteRegValue(char *szValueName[, varargs]...)
```

Arguments

szValueName
: String with the name of the value to delete. May contain printf-style formatting codes as well as FM !-codes.

varargs
: A list of optional arguments used to perform printf-style formatting on the szValueName string.

Return

Returns ERROR_SUCCESS if the operation was successful, otherwise it returns one of the following integer error codes:

ERROR_SUCCESS	(==0) no error
ERROR_FILE_NOT_FOUND	key or value name not found
ERROR_MORE_DATA	buffer wasn't big enough (e.g., getRegString, getRegData)
ERROR_NO_MORE_ITEMS	index >= # of values or subkeys (enumRegValue, enumRegSubKey)
ERROR_INVALID_FUNCTION	bad top-level key, etc
ERROR_INVALID_DATA	wrong data type or size (or size > 2048)
ERROR_BADDB	registry database is corrupt
ERROR_BADKEY	registry key is invalid

ERROR_CANTOPEN	registry key could not be opened
ERROR_CANTREAD	registry key could not be read
ERROR_CANTWRITE	registry key could not be written
ERROR_REGISTRY_CORRUPT	registry is corrupt
ERROR_REGISTRY_IO_FAILED	input/output to registry failed
ERROR_KEY_DELETED	Illegal operation attempted on a Registry key which has been marked for deletion.
ERROR_KEY_HAS_CHILDREN	cannot delete a key with subkeys (Windows NT)

Description

Deletes a specified value under the current registry key.

See Also

getRegRoot, **setRegPath**

DESIGNTIME

Syntax

`bool DESIGNTIME`

Description

Use the DESIGNTIME system variable to test if the plug-in is running inside the FilterMeister editor (true) or as a stand-alone compiled plug-in (false).

Example

If you want to create a plug-in without a dialog window, you can use the following code. The use of DESIGNTIME helps to keep the FilterMeister editor open whilst creating the plug-in.

```
OnFilterStart: {
    if (!DESIGNTIME) {
        doAction(CA_APPLY);
    }
    return true;
}
```

destroyMenu

Syntax

`void destroyMenu(int hMenu)`

Arguments

hMenu
: Handle to the menu to be destroyed

Description

Destroys the given menu object.

Example

```
%ffp

ctl[0]: PUSHBUTTON, "Click Me!"

OnCtl(n): {

  if (n==0 && e == FME_CLICKED){

    int menu = 0;
    menu = createPopupMenu();

    insertMenuItem(menu, 1, "Do This", MFS_ENABLED ,
NULL);
    insertMenuItem(menu, 2, "Do That", MFS_ENABLED |
MFS_DEFAULT, NULL);
    insertMenuItem(menu, 3, "Do Nothing", MFS_ENABLED,
NULL);
```

```
    Info("Selection: %d", trackPopupMenu(menu, 1, 0, 0, 0)
);

    destroyMenu(menu);
  }

  return false;
}
```

See Also

createPopupMenu, insertMenuItem, trackPopupMenu

dif

Syntax

`int dif(int a, int b)`

Arguments

a
: Any integer.

b
: Any integer

Return

The absolute difference of the integers a and b.

Description

This function returns the absolute difference of the two arguments, and is equivalent to the function abs(a-b). (An absolute difference is computed by subtracting one argument from the other and ignoring the sign of the result).

Example

```
// sets 'p' to 127
int p = dif(128,255);
```

See Also

abs

doAction

Syntax

```
doAction(int action)
```

Arguments

action
The action to be performed (preview, apply, cancel etc)

Description

Performs one of several predefined control actions specified by its argument action, where action may have one of the following predefined symbolic constant values:

Symbolic Constant	**Control Action**
CA_PREVIEW	Updates the proxy preview window.
CA_APPLY	Applies the filter to the original source image and exits the plug-in.
CA_CANCEL	Exits the plug-in filter without modifying the original source image.
CA_EDIT	Enters or exits source code editing mode (ignored in standalone filters).
CA_ABOUT	Displays the ABOUT dialog box.
CA_RESET	Resets all controls to their initial values.
CA_NONE	Performs no action.

Example

```
// Display the plug-in about box
doAction( CA_ABOUT );
```

doingProxy

Syntax

`bool doingProxy`

Description

doingProxy is True if the filter is running, and False if the filter is applying the effect to the image in the host program.

With the doingProxy function you can check if the filter is processing the proxy image ('preview image') or the original image (layer) in the host program.

Since it sometimes can be useful to let the filter process the preview image and the final image differently, the doingProxy constant can be used to seperate them in your code. You might e.g. want to add guide lines in your preview image and not in the final image, or you might prefer to add a watermark to the final image and not in the preview (e.g. in case you create a demo version of your plug-in).

Example

```
if (doingProxy) {
  // execute this if filter is running
}
else {
  // execute this if filter is
  // applying effect to host image
}
```

See Also

Constants

doingScripting

Syntax

`bool doingScripting`

Description

doingScripting is **`true`** if the filter is running via Photoshop's scripting mechanisms, and **`false`** if the filter is applying the effect to the image in the host program (with or without a dialog).

With the doingScripting variable you can check if the filter is running in scripting mode (ie without a UI dialog window, with parameters passed directly from Photoshop).

Example

```
if (doingScripting) {
  // get scripting parameters here
}
else {
  if (doingProxy) {
    // Code for managing UI dialog
  }
}

// Image processing code here
```

See Also

Constants, doingProxy

EDIT

Syntax

```
ctl[n]: EDIT(Class Specific Properties), Other Properties
```

Description

The EDIT class creates a text edit control on the filter dialog window.

Class Specific Properties

AUTOHSCROLL
: Allows the text entered to extend longer than the width of the control.

AUTOVSCROLL
: Allows the text entered to extend longer than the height of the control.

CENTER
: Centers the text in the edit control

LEFT
: Aligns the text to the left of the control

LOWERCASE
: Converts any text typed into the control into lowercase.

MULTILINE
: Extends the text control to multiple lines and allows newline characters to be entered.

NOHIDESEL
: Doesn't hide the selected text when the control loses focus.

NUMBER
: Restricts the text that can be entered into the control to numbers only.

PASSWORD

Masks the text typed into the control, displaying them as asterisks.

READONLY

Prevents the text in the control from being edited.

RIGHT

Aligns the text to the right of the control

UPPERCASE

Converts any text typed into the control into uppercase.

Example

```
ctl[0]: EDIT(RIGHT,UPPERCASE), Pos=(50,100)
ctl[1]: EDIT(RIGHT,NUMBER), Pos=(50,120), size=(15,10),
"0"

OnCtl(n): {
  int r;

  if (n==1) {
    r = (int)strtod(getCtlText(1), 0);
  }
}
```

Notes

This control is only available as of FilterMeister 1.0 Beta 8.5.

Note that the Lowercase, Uppercase and Number modifiers do not prevent excluded characters from being pasted into the control.

When using the Multiline property, remember that native Windows controls use \r\n as a line ending, not just \n on its own.

egm

NB: Not available in current FM versions

Syntax

```
int egm(int edge_a, int edge_b, int value)
```

Arguments

edge_a
: Bottom edge value

edge_b
: Top edge value

value
: Value that will be edge wrapped.

Return

The edge-mirrored value will be returned.

Description

This function returns the *value* untouched if it lies between *bottom* and *top*. If it lies outside *bottom* and *top*, the *value* will be edge mirrored to lie between *bottom* and *top*. For example, an input value of 21 will be returned as 19 if edge_a = 10 and edge_b = 20.

Example

```
%ffp

ctl(0): "Bottom", Range=(0,255), Val=20
ctl(1): "Top", Range=(0,255), Val=100

ForEveryTile:
{
  setCtlRange(0,0,min(X,Y));
  setCtlRange(1,0,min(X,Y));

  for (y=y_start; y < y_end; y++) {
    updateProgress(y,y_end);
    for (x=x_start; x < x_end; x++) {
      for (z=0; z < Z; z++) {

        pset(x, y, z, src( egm(ctl(0),ctl(1),x) ,
egm(ctl(0),ctl(1),y) , z) );

      }
    }
  }

  return true;
}
```

See Also

set_edge_mode, **egw**

egw

Syntax

```
int egw (int bottom, int top, int value)
```

Arguments

bottom
: Bottom edge value

top
: Top edge value

value
: Value that will be edge wrapped.

Return

The edge-wrapped value will be returned.

Description

This function returns the *value* untouched if it lies between *bottom* and *top*. If it lies outside *bottom* and *top*, the *value* will be edge wrapped to lie between *bottom* and *top*. For example, a value of 21 would be returned as 11 if bottom=10 and top=20 or returned as 12 if bottom=10 and top=19.

Example

```
%ffp

ctl(0): "Bottom", Range=(0,255), Val=20
ctl(1): "Top", Range=(0,255), Val=100

```

```
ForEveryTile:
{
  setCtlRange(0,0,min(X,Y));
  setCtlRange(1,0,min(X,Y));

  for (y=y_start; y < y_end; y++) {
    updateProgress(y,y_end);
    for (x=x_start; x < x_end; x++) {
      for (z=0; z < Z; z++) {

        pset(x, y, z, src( egw(ctl(0),ctl(1),x) ,
egw(ctl(0),ctl(1),y) , z) );

      }
    }
  }

  return true;
}
```

See Also

set_edge_mode, **egw**

enableCtl

Syntax

```
int enableCtl(int n, int level);
```

Arguments

n
: Control number whose state will be set.

level
: State of the control. Set 3 for visible and enabled, 0 for invisible or 1 for disabled.

Return

Returns the previous state of the control. Usually this value isn't needed.

Description

Sets the state of a control.

Example

```
enableCtl(0, 1); //disables a visible user control
enableCtl(5, 0); //makes user control 5 invisible and
disabled
```

See Also

ctlEnabled

enableToolTipBalloon

Requires FM 1.0 Beta 9.1 (Feb 2009) or newer

Syntax

```
bool enableToolTipBalloon(bool enable)
```

Arguments

enable
TRUE if you want balloon tooltips, FALSE if you want normal (box) tooltips.

Return

Returns the previous state.

Description

Use this function to set the style of tooltips.

endSetPixel

Syntax

```
void endSetPixel(int ctl)
```

Arguments

ctl
The control number of the control you were drawing on using Control drawing functions.

Description

Stops drawing to an **OWNERDRAW** control and display it on screen.

Example

```
ctl(1): OWNERDRAW(drawitem), Pos=(300,50), size=(100, 100)
startSetPixel(1);
setRectFill( 10, 20, 30, 40, RGB(255, 0, 0));
endSetPixel(1);
```

See Also

startSetPixel, **setPixel**, **setRectFill**

enterCriticalSection

Requires FM 1.0 Beta 9d (June 2008) or newer

Syntax

```
bool enterCriticalSection(int hCS)
```

Arguments

hCS
: Specifies the handle of the Critical Section to be entered, as returned by a call to **createCriticalSection**.

Return

This function returns **true** when access to this Critical Section is granted to the calling thread, or **false** if hCS is 0.

Description

This function waits for ownership of the specified Critical Section. If another thread currently owns this Critical Section, then this function waits until the owning thread leaves the Critical Section, at which point one of the threads waiting to enter the Critical Section is granted access. If no other thread currently owns the Critical Section, the calling thread is granted immediate access.

For more information about Critical Sections, see the MSDN documentation about **[Critical Section Objects]**.

Comments

One need not normally test the return value of **enterCriticalSection**. This is merely a check to make sure that hCS is non-zero so the **EnterCriticalSection** Win32 API won't cause a memory access violation.

Example

See the **createCriticalSection example**.

See Also

System Functions, **createCriticalSection**, **tryEnterCriticalSection**, **leaveCriticalSection**, **deleteCriticalSection**

enumRegValue

Syntax

```
int enumRegValue(int index, lpsz szValueName, int
maxValueNameLen, int *piType, int *pcbData)
```

Arguments

index
: is the zero-based index for the next value name to be retrieved

szValueName
: is a character string buffer to receive the value name

maxValueNameLen
: is the size of the szValueName buffer in bytes (including room for the terminating null character)

piType
: is the address of an int variable which will receive a code indicating the type of data associated with this value (REG_DWORD for 32-bit integer data, REG_SZ for character string data, REG_BINARY for byte sequence data)

pcbData
: is the address of an int variable which will receive the size in bytes of the data associated with this value

Return

Returns ERROR_SUCCESS if the operation was successful, otherwise it returns one of the following integer error codes:

ERROR_SUCCESS (==0)	no error
ERROR_FILE_NOT_FOUND	key or value name not found

ERROR_MORE_DATA	buffer wasn't big enough (e.g., getRegString, getRegData)
ERROR_NO_MORE_ITEMS	index >= # of values or subkeys (enumRegValue, enumRegSubKey)
ERROR_INVALID_FUNCTION	bad top-level key, etc
ERROR_INVALID_DATA	wrong data type or size (or size > 2048)
ERROR_BADDB	registry database is corrupt
ERROR_BADKEY	registry key is invalid
ERROR_CANTOPEN	registry key could not be opened
ERROR_CANTREAD	registry key could not be read
ERROR_CANTWRITE	registry key could not be written
ERROR_REGISTRY_CORRUPT	registry is corrupt
ERROR_REGISTRY_IO_FAILED	input/output to registry failed
ERROR_KEY_DELETED	Illegal operation attempted on a Registry key which has been marked for deletion.
ERROR_KEY_HAS_CHILDREN	cannot delete a key with subkeys (Windows NT)

Description

Enumerates the names of all values stored under the current key.

Example

```
// fetches installed fonts from
// the registry and loads them
// into a combobox (modified
// example, original code by Alex
// Hunter)

%fml

ctl[0]: COMBOBOX, "Style 1\nStyle 2\nStyle 3",
     Pos=(*,10), Size=(99,50),
     Val=0, Action=PREVIEW
ctl[1]: pushbutton, "Populate Font List", size=(80,15),
pos=(*,35)

int iRetVal;
int iDataType;
int iDataLen;

OnCtl(n): {

  if (n==1 && e== FME_CLICKED) {

  // clear the ComboBox control
  setCtlText(0, "");

  //save current registry path
  getRegPath(str9, 256);
  Info("path=%s",str9);

  setRegRoot(HKEY_LOCAL_MACHINE);
  setRegPath("SOFTWARE\\Microsoft\\Windows
NT\\CurrentVersion\\Fonts");

  //save current registry path
  getRegPath(str0, 256);
  Info("path=%s",str0);

  for (i=0, iRetVal=ERROR_SUCCESS; iRetVal ==
ERROR_SUCCESS; i++) {
    iRetVal = enumRegValue(i, str0, 256, &iDataType,
&iDataLen);

    if (i < 21) { Info("i=%i, iRetVal=%i, value=%s", i,
iRetVal, str0); }

    if (iRetVal == ERROR_SUCCESS) {
      setCtlItemText(0,i,str0);
    }
  }

  // restore registry path to
  // original value
  setRegPath(str9);
}

  return false;
}
```

See Also

getRegRoot, setRegRoot, getRegPath, setRegPath

Error

Syntax

```
int Error(string promptString, ...)
```

Arguments

promptString
: Specifies the prompt string for the error message window. This string may contain printf-style format descriptors, which will be expanded using the succeeding arguments.

...
: Variable number of arguments of varying types, should correspond to the format descriptors in promptString.

Return

IDCANCEL, IDRETRY or IDIGNORE depending on which button the user clicks.

Description

This function displays an error box containing a text string, a CANCEL, a RETRY and an IGNORE button. Clicking the CANCEL button returns the value IDCANCEL, clicking on RETRY returns the value IDRETRY and clicking on IGNORE the value IDIGNORE.

Example

```
Error ( "The setting %d of the scrollbar %d\nis not
allowed", ctl(i), i );
```

```
if ( Error ( "I'm not in the mood" ) == IDIGNORE)
   Warn( "If you ignore me, I'll\ndeinstall !H" );
```

See Also

msgBox

ErrorOk

Syntax

```
int ErrorOk(string promptString, ...)
```

Arguments

promptString
: Specifies the prompt string for the error message window. This string may contain printf-style format descriptors, which will be expanded using the succeeding arguments.

...
: Variable number of arguments of varying types, should correspond to the format descriptors in promptString.

Return

IDOK once the user has clicked the Ok button.

Description

This function displays an error box containing a text string, and an OK button. Clicking the OK button returns the value IDOK.

Example

```
ErrorOk ( "Image is too small!!" );
```

See Also

Error, **msgBox**

exp

Syntax

```
double exp(double x)
```

Arguments

x
: Double-precision floating point value.

Return

Exponential of x.

Description

Returns the exponential value of parameter x.

expand

Syntax

```
int expand(int pointer, int size)
```

Arguments

pointer
    Pointer to the block of memory to expand
size
    The new size of the memory block, in bytes

Return

A pointer to the resized block of memory, or NULL if there was an error resizing the memory block.

Description

Resizes a block of memory to a new, larger size. expand cannot be used to make a memory block smaller (use **realloc** for this instead). expand can only be used to expand memory "in place" - that is, without moving the memory block to another position in memory. For this reason, expand is not always able to give you the maximum expansion that you ask for, so be sure to check the size of the new block by using **msize**.

See Also

malloc, **realloc**, **msize**, **free**

fabs

Syntax

```
double fabs(double number)
```

Arguments

number
Any double or float number.

Return

The absolute value of the supplied argument.

Description

Returns the absolute value of a double or float number. When supplied with a negative number, this function will return a positive number of equal distance from zero.

Example

```
%ffp

OnFilterStart:
{
    Info("The absolute value of -1.23 is %f", abs(-1.23));
}
```

See Also

abs

false

Description

`false` is a Boolean constant representing a logical state of "falsehood". In FilterMeister, any numeric value of 0 or ±0.0, the NUL character constant, and any null pointer value may represent a state of "falsehood". The integral numeric value 0 is reserved as the *canonical* representation of "false". Thus, the Boolean constant `false` has a numeric value of 0 when evaluated in a numeric context.

Note that *any* zero or null value represents "falsehood", so the integer constant 0, the floating point constants 0.0, -0.0, 0.0L, and -0.0L, the pointer constant `NULL`, and the character constant '\0' (NUL) will all evaluate as false in a Boolean or logical context. A null string ("" or "\0"), however, will evaluate as true, since the value of a character string is its address, and by convention no object in FM may be allocated at address 0.

Comment

A common cause of mistakes in C or FM programming is to assume that any true expression has the value 1 (or `true`). It is bad programming style, and a frequent cause of subtle errors, to code, for example:

```
if (flag==true) break;
```

since this code will *not* break if the value of `flag` is 77, which also represents truth. Unless the programmer is specifically testing for the value 1, the preferred idiom is:

```
if (flag) break;
```

which will break when `flag` contains *any* true value (i.e., any value but 0, ±0.0, '\0', or `NULL`), and not just in the case that `flag` has the value 1.

Note that this caution does not apply to the Boolean constant `false`, since the canonical numeric value of `false` is 0. The only other representation of falsity is a null pointer or a NUL character, and the numeric value of these is always 0 in FM. Thus the following code fragment, while poor style, will always break correctly when `flag` has the value 0, ±0.0, `false`, '\0', or `NULL`.

```
if (flag==false) break;
```

Nevertheless, the preferred coding style is:

```
if (!flag) break;
```

Example

```
%ffp

OnFilterStart:{
    bool flag = false;

    Info("The integer value of false is %d", flag);

    if (flag) Info("This should not appear");

    return false;
}
```

This snippet will display a single message box with the message:

```
The integer value of false is 0
```

See Also

true, **Constants**

fc2d

Syntax

```
double fc2d(double x, double y)
```

Arguments

x
: A double-precision floating-point pixel x-coordinate.

y
: A double-precision floating-point pixel y-coordinate.

Return

A double-precision floating-point value in the range -511 to 512.

Description

Pixels are usually addressed by their cartesian coordinates [x,y], but FilterMeister also allows the use of polar coordinates, and provides a set of functions for conversion between the two systems. Polar coordinates are expressed as [d,m], where 'd' represents the 'direction' to the pixel from the origin [0,0], and 'm' is the 'magnitude' of the distance from the origin. The c2d() function returns the polar coordinate direction 'd' for the pixel at [x,y], relative to the top left corner of the image. A 'd' value of 0 represents the direction to the right along the x-axis (ie y=0); a value of 256 represents the direction downward along the y-axis (ie x=0); a value of 512 represents the direction to the left along the x-axis (where y=0); and a value of -256 represents the upward direction on the y-axis (where x=0). Naturally, intermediate values represent the intermediate angles.

Comment

Note that there is a bug in fc2d, so that it doesn't give exactly the same results as c2d.

Example

```
%ffp

ForEveryTile:
{

for (y=y_start; y<y_end; ++y)
{
  for (x=x_start; x<x_end; ++x)
  {
    for (z=0; z<Z; ++z)
    {
      pset(x, y, z, (256 * fabs(fc2d(x-X/2, y-Y/2)))/512);
    }
  }
}

return true;
```

See Also

c2m, **r2x**, **r2y**

fc2m

Syntax

```
double fc2m(double x, double y)
```

Arguments

x
: A double floating-point pixel x-coordinate.

y
: A double floating-point pixel y-coordinate.

Return

An double floating-point value giving the distance from the center of the image to the pixel at coordinates [x,y].

Description

Pixels are usually addressed by their cartesian coordinates [x,y], but FilterMeister also allows the use of polar coordinates, and provides a set of functions for conversion between the two systems. Polar coordinates are expressed as [d,m], where 'd' represents the 'direction' to the pixel from the origin and 'm' is the 'magnitude' of the distance from the origin. The c2m() function returns the polar coordinate magnitude 'm' for the pixel at [x,y], relative to the top left corner of the image.

Example

```
%ffp

ForEveryTile: {
  for (y=y_start; y < y_end; ++y) {
    for (x=x_start; x < x_end; ++x) {
      for (z=0; z < Z; ++z) {

        pset(x, y, z, (256 * (fc2m(x-X/2, y-Y/2)))/128);

      }
    }
  }

  return true;
}
```

See Also

c2d, fc2d, r2x, r2y

fCallLib

Requires FM 1.0 Beta 9.0 (April 2008) or newer

Syntax

```
double fCallLib(void *fnptr, ...)
```

Arguments

fnptr
: A pointer (obtained with **getLibFn**) to the DLL function to call.

...
: The other parameters required by the DLL function (varies depending on the function).

Return

Returns the return value of the called DLL function, which varies depending on the function called.

Description

Calls a function in a DLL that has previously been loaded with **loadLib** and **getLibFn**.

Comment

In most cases, you would use the **callLib** function. Only use fCallLib if the DLL function you are calling returns a double value.

See Also

callLib, **loadLib**, **getLibFn**, **freeLib**

fcallLibFmc

Requires FM 1.0 Beta 9.0 (April 2008) or newer

Syntax

```
??? fcallLibFmc(void *fnptr, ...)
```

Arguments

fnptr
: A pointer (obtained with **getLibFn**) to the DLL function to call.

...
: The other parameters required by the DLL function (varies depending on the function).

Return

Returns the return double value of the called DLL function, which varies depending on the function called.

Description

Calls a function in a DLL that has previously been loaded with **loadLib** and **getLibFn**, except that the first argument (fnptr) is replaced by a pointer to the FMcontext record (fmcp) that you can then access from your DLL function. The structure of FMcontext can be found in the FilterMeister source code in the AfhFMcontext.h file. As the context record can change between FilterMeister versions, it is important that your DLL is designed to use the correct version of the context record for your plugin.

Comment

In most cases, you would use the **callLib** or **callLibFmc** functions. Only use fCallLibFmc if the DLL function you are calling returns a double value, and if you need access to the FilterMeister Context Record to access FM functions and variables from your DLL.

Example

Example of called functions in a DLL:

```
__declspec(dllexport)
int MyForEveryTile(FMcontext *fmcp, int X, int Y) { return
false; }

__declspec(dllexport)
double MyFloat(FMcontext *fmcp, int a, double x, double y)
{
  return a*(x - y);
}
```

To invoke these functions from FM:

```
int g_hMyDll = loadLib("MyDll");
if (!g_hMyDll) ERROR...

int g_pfnMyForEveryTile = getLibFn(g_hMyDll,
"MyForEveryTile");
if (!g_pfnMyForEveryTile) ERROR...

int g_pfnMyFloat = getLibFn(g_hMyDll, "MyFloat");
if (!g_pfnMyFloat) ERROR...

int iRes = callLibFmc(g_pfnMyForEveryTile, 100, 200);

double fRes = fcallLibFmc(g_pfnMyFloat, 10, 1.2, 3.14);
```

See Also

loadLib, getLibFn, callLibFmc, fcallLib, fcallLibFmc, freeLib

fclose

Syntax

```
int fclose(int filepointer)
```

Arguments

filepointer
A pointer to a file/stream previously opened with **fopen**.

Return

Returns 0 if the file is closed successfully, and EOF if it encounters any problems.

Description

Closes a file or stream, writing any data remaining in the writing buffer to the file and freeing the file for use by other programs.

Example

```
int IMG_FILE;
if (IMG_FILE = fopen("d:\\FM_image0.fmi", "wb"))
{
  // Write out the src pixels as raw data
  for (z=0; z<Z; z++)
  for (y=0; y<Y; y++)
  for (x=0; x<X; x++)
    fputc(src(x,y,z), IMG_FILE);
}
else
  ErrorOk("Cannot write image file\nDrive is either full
or write-protected!");

if (fclose(IMG_FILE))
  ErrorOk("Cannot close image file!");
```

See Also

fopen, fcloseall

fcloseall

Syntax

`int fcloseall()`

Return

Always returns 0.

Description

Closes all opened files and streams, writing any data remaining in the writing buffers to the files and freeing the files for use by other programs.

Example

```
int inFile, outFile;
if (inFile = fopen("d:\\source.bmp", "rb")) {
  if (outFile = fopen("d:\\destination.bmp", "wb")) {
    while (i = fgetc(inFile)) {
      fputc(i, outFile);
    }
  }
  fcloseall();
}
```

See Also

fopen, **fclose**

feof

Syntax

```
int feof(int filepointer)
```

Arguments

filepointer
Pointer to a file as returned by **fopen** and such.

Return

non-zero value if end-of-file, 0 otherwise.

Description

Tests the end-of-file indicator for the specified file. If the file is at the end-of-file, then it returns a nonzero value. If it is not at the end of the file, then it returns zero.

ferror

Syntax

```
int ferror(int filepointer)
```

Arguments

filepointer
Pointer to a file such as created by **fopen**.

Return

Zero if no error is set for the specified file, non-zero value otherwise.

Description

Tests the error indicator for the specified file. If the error indicator is set, then it returns a nonzero value. If the error indicator is not set, then it returns zero.

See Also

clearerr

ffillArray

Requires FM 1.0 Beta 9.0c (28 May 2008) or newer

Syntax

```
bool ffillArray(int nr, double dval)
```

Arguments

nr
: Number of the array. Values from 0 to 99 are allowed.

dval
: Floating-point value to fill the Array

Return

Returns **true** if successful, or **false** if an error occurred (e.g., incorrect value for nr or byte-size of Array).

Description

Function for filling an Array with a certain floating-point value; e.g., for initializing all elements of the Array to -0.5. The byte-size of the Array must be 2, 4, or 8. If the byte-size is 2 or 4, dval will be converted from double to half (16-bit) or float (32-bit), resp., format before storing it in the Array.

Example

```
%ffp
OnFilterStart:{
  allocArray (0,100,0,0,4);
  ffillArray (0, -0.5);
  Info ("The Array was filled with the value: %g",
fgetArray(0, rnd(0,99) ,0,0) );
  freeArray(0);
  return false;
}
```

See Also

fillArray, **allocArray**, **freeArray**, **fgetArray**, **fputArray**, **getArrayDim**, **copyArray**

fflush

Syntax

```
int fflush(int filepointer)
```

Arguments

filepointer
A pointer to a file/stream previously opened with **fopen**.

Return

Returns 0 if the file is flushed successfully, and EOF if it encounters any problems.

Description

Writes all remaining data in the writing buffer of the specified filepointer to the associated file.

The main use of this function is security; ensuring the data is stored in the file in order to protect against crashes. Use this function if your file format allows processing of the partial file.

Example

```
int IMG_FILE;
if (IMG_FILE = fopen("d:\\FM_image0.fmi", "wb")) {
  // Write out the src pixels as raw data
  for (z = 0; z < Z; ++z) {
    for (y = 0; y < Y; ++y) {
      for (x = 0; x < X; ++x) {
        fputc(src(x, y, z), IMG_FILE);
      }
    }
    fflush(IMG_FILE); // ensure complete channel is stored
before next channel is written.
  }
}
else
  ErrorOk("Cannot write image file\nDrive is either full
or write-protected!");

if (fclose(IMG_FILE))
  ErrorOk("Cannot close image file!");
```

See Also

flushall

fgColor

Syntax

`int fgColor`

Description

The currently chosen foreground color value in the host application (ie Photoshop®, Paint Shop Pro®).

Example

```
%fml
ctl[0]: OWNERDRAW, Size=(50,50), Pos=(240,3)
ctl[1]: OWNERDRAW, Size=(50,50), Pos=(300,3)
ctl[2]: STATICTEXT, Text="Foreground", Pos=(240, 55)
ctl[3]: STATICTEXT, Text="Background", Pos=(300, 55)

OnFilterStart: {
  setCtlColor(0, fgColor);
  setCtlColor(1, bgColor);
  return true;
}
```

See Also

bgColor, **setCtlColor**

fgetArray

Requires FM 1.0 Beta 9.0 (Apr 2008) or newer

Syntax

```
double fgetArray (int nr, int x, int y, int z)
```

Arguments

nr
: Number of the array. Values from 0 to 99 are accepted.

x, y, z
: x, y, and z coordinates of a cell in the array. If you allocated a one-dimensional array, set y and z to zero. If you allocated a two-dimensional array, set z to zero.

Return

Returns the double floating-point value that was stored at the specified coordinates in the array. If the specified coordinates lie outside the array or the index nr is invalid, the value 0.0 will be returned. The byte-size of the Array must be 2 (half), 4 (float), or 8 (double); otherwise 0.0 will be returned.

Description

This function lets you read a floating-point value from an array.

Example

See allocArray

See Also

allocArray, freeArray, getArray, fgetArray, putArray, fputArray, getArrayDim, copyArray, ffillArray

fgetc

Syntax

```
int fgetc(int * fileptr)
```

Arguments

fileptr
Pointer to a file opened using **fopen**.

Return

The byte (unsigned) read from the file if successful, otherwise -1.

Description

Reads a single byte from the file referenced by **fileptr**.

See Also

fopen, **fputc**, **getc**, **fgets**, **fread**

fgetpos

Syntax

```
int fgetpos(int *fileptr, int *pos)
```

Arguments

fileptr
  Pointer to a file opened using **fopen**.

pos
  Pointer to the current position in the file.

Return

Returns 0 if successful, otherwise non-zero and sets errno to the relevant error value. The file position pointer is stored in the **pos** parameter.

Description

Gets the current position in the file. You can use the position pointer retrieved by this function in future calls to **fsetpos**.

See Also

fopen, **fputs**, **fread**, **fsetpos**, **fclose**

fgets

Syntax

```
char * fgets(char * s, int n, int * fileptr)
```

Arguments

s
: Pointer to a string which will contain the bytes read.

n
: Maximum number of bytes to read.

fileptr
: Pointer to a file opened using **fopen**.

Return

A pointer to string **s** if successful, otherwise NULL.

Description

Reads up to **n** number of bytes from the file referenced by **fileptr** into string **s**.

This function will add a NULL to the end of the string, so make sure to make **n** less than the memory allocated for string **s**.

See Also

fopen, **fputs**, **fgetc**, **fread**

fillArray

Syntax

```
bool fillArray (int nr, int val)
```

Arguments

nr
: Number of the Array. Values from 0 to 99 are allowed.

val
: Value to fill array

Return

Returns **true** if successful, or **false** if an error occurred (e.g., incorrect value for nr or byte-size of Array).

Description

Function for filling an array with a certain byte, short, integer, or double value, e.g. for initializing all cells of the array to zero. If the byte-size of the Array is 1, 2, or 4, then val is converted to an 8-bit, 16-bit, or 32-bit integer as needed. If the byte-size of the Array is 8, then val is converted to a 64-bit double floating-point value (not a 64-bit integer!) for storage in the Array.

Comments

Note that the use of fillArray to store a 64-bit floating-point value in the Array is deprecated; use **ffillArray** instead.

Example

```
%ffp

OnFilterStart:{

  allocArray (0,100,0,0,1);
  fillArray (0, 112);
  Info ("The array was filled with the value: %d",
getArray(0, rnd(0,99) ,0,0) );
  freeArray(0);

  return false;
}
```

See Also

ffillArray, **allocArray**, **freeArray**, **getArray**, **putArray**, **getArrayDim**, **copyArray**

filterCase

Syntax

`int filterCase`

Description

The type of data being filtered: Flat with no selection (1), Flat with a selection (2), Floating (3), Layer with editable transparency and no selection (4), Layer with editable transparency and a selection (5), Layer with preserved transparency and no selection (6), or Layer with preserved transparency and a selection (7).

A zero indicates that the host did not set this variable, and the plug-in should look at the **haveMask** and **isFloating** variables to determine the filter case.

Example

```
%ffp

OnFilterStart: {
  switch(filterCase) {
    case 0:
      Info ("Unknown filterCase");
      break;
    case 1:
      Info ("Flat with no selection");
      break;
    case 2:
      Info ("Flat with a selection");
      break;
    case 3:
      Info ("Floating");
      break;
    case 4:
      Info ("Layer with editable transparency and no
selection");
      break;
    case 5:
      Info ("Layer with editable transparency and a
selection");
      break;
    case 6:
      Info ("Layer with preserved transparency and no
selection");
      break;
    case 7:
      Info ("Layer with preserved transparency and a
selection");
      break;
    default:
      Info ("Unknown filterCase");
      break;
  }
  return false;
}
```

filterInstallDir

Syntax

```
int filterInstallDir
```

Description

The full path name of the directory from which your filter was loaded (which is presumably the directory in which it was installed). This can be useful for locating files (such as HTML help files) which were installed as separate resources in your filter's installation directory.

Comment

Note that at filter design time, filterInstallDir will contain the name of the directory from which FilterMeister itself was loaded, not the directory from which your target filter will ultimately be loaded at run-time.

Example

```
Info(filterInstallDir);
```

filterUniqueID

Syntax

`string filterUniqueID`

Description

The UniqueID string value (up to 36 characters) used to identify your plug-in uniquely to the scripting system. The UniqueID is needed for scripting so that Photoshop® and other programs can distinguish between your plug-in and other plug-ins. Two different plug-ins with the same UniqueID will be treated as the same plug-in by Photoshop, so make sure your UniqueID is really unique, otherwise another plug-in may be executed instead of yours.

Comment

If you do not define UniqueID at the top of your code, FM will generate a random UniqueID every time it is launched.

Example

```
%ffp

// UniqueID: "d5326b0c-7c78-26c7-97bc-00c0d1465278"

OnFilterStart:{
  Info ("%s",filterUniqueID);
  return false;
}
```

findClose

Requires FM 1.0 Beta 6 (May 2005) or newer

Syntax

```
int findClose(int *searchHandle)
```

Arguments

searchHandle
A search handle from a previous call to findFirstFile / findNextFile.

Return

Returns 0 if the function failed, or non-zero if the function succeeded.

Description

Closes a file search handle opened with findFirstFile.

Comment

Internally, findClose is a simple wrapper around the built-in Windows **[Win32 FindClose API]** function.

Example

```
%ffp

OnFilterStart: {
  int Handle, Attribute;

  // Search for 8bf files in the filterInstallDir
  snprintf(str1, 255, "%s\\*.8bf", filterInstallDir);

  // Optionally, search the C: root instead
  //strcpy(str1,"C:\\");

  Handle = findFirstFile (str1, str9, &Attribute);
  if (Attribute != FILE_ATTRIBUTE_DIRECTORY)
    Info ("%s - %d",str9, Attribute);

  if (Handle != INVALID_HANDLE_VALUE){
    while (findNextFile(Handle, str9, &Attribute) != 0) {
      if (Attribute != FILE_ATTRIBUTE_DIRECTORY)
        Info ("%s - %d",str9, Attribute);
    }
  }

  findClose(Handle);

  return false;
}
```

See Also

findFirstFile, findNextFile

findFirstFile

Requires FM 1.0 Beta 6 (May 2005) or newer

Syntax

```
int findFirstFile(char* lpFileName, char* foundItem, int
*dwFileAttributes)
```

Arguments

lpFileName
: The path to the directory to start looking for files. This can be appended with a wildcard to search for specific files.

foundItem
: A string where the name of the found file will be stored.

dwFileAttributes
: An integer where the attributes of the located file will be stored (see below for attributes list).

File Attributes

FILE_ATTRIBUTE_ARCHIVE	A file or directory that is an archive file or directory.
FILE_ATTRIBUTE_COMPRESSED	A file or directory that is compressed.
FILE_ATTRIBUTE_DEVICE	This value is reserved for system use.
FILE_ATTRIBUTE_DIRECTORY	The handle that identifies a directory.
FILE_ATTRIBUTE_ENCRYPTED	A file or directory that is encrypted.
FILE_ATTRIBUTE_HIDDEN	The file or directory is hidden. It

	is not included in an ordinary directory listing.
FILE_ATTRIBUTE_INTEGRITY_STREAM	The directory or user data stream is configured with integrity (only supported on ReFS volumes).
FILE_ATTRIBUTE_NORMAL	A file that does not have other attributes set. This attribute is valid only when used alone.
FILE_ATTRIBUTE_NOT_CONTENT_INDEXED	The file or directory is not to be indexed by the content indexing service.
FILE_ATTRIBUTE_NO_SCRUB_DATA	The user data stream not to be read by the background data integrity scanner (AKA scrubber).
FILE_ATTRIBUTE_OFFLINE	The file data is physically moved to offline storage.
FILE_ATTRIBUTE_READONLY	A file that is read-only.
FILE_ATTRIBUTE_RECALL_ON_DATA_ACCESS	The file or directory is not fully present locally. For a file that means that not all of its data is on local storage (e.g. it may be sparse with some data still in remote storage).
FILE_ATTRIBUTE_RECALL_ON_OPEN	The file or directory has no physical representation on the local system; the item is virtual.
FILE_ATTRIBUTE_REPARSE_POINT	A file that is a symbolic link.
FILE_ATTRIBUTE_SPARSE_FILE	A file that is a sparse file.
FILE_ATTRIBUTE_SYSTEM	A file or directory that the operating system uses a part of,

	or uses exclusively.
FILE_ATTRIBUTE_TEMPORARY	A file that is being used for temporary storage.
FILE_ATTRIBUTE_VIRTUAL	This value is reserved for system use.

Return

Returns INVALID_HANDLE_VALUE if the search failed or could not find any files. Otherwise it returns a search handle that can be used in subsequent calls to **findNextFile** or **findClose**.

Description

Searches a directory for a file or subdirectory matching the given name or wildcard.

Example

```
%ffp

OnFilterStart: {

  int Handle, Attribute;

  // Search for 8bf files in the filterInstallDir
  snprintf(str1, 255, "%s\\*.8bf", filterInstallDir);

  // Optionally, search the C: root instead
  //strcpy(str1,"C:\\");

  Handle = findFirstFile (str1, str9, &Attribute);
  if (Attribute != FILE_ATTRIBUTE_DIRECTORY)
    Info ("%s - %d",str9, Attribute);
  if (Handle != INVALID_HANDLE_VALUE){
    while (findNextFile(Handle, str9, &Attribute) != 0) {
      if (Attribute != FILE_ATTRIBUTE_DIRECTORY)
        Info ("%s - %d",str9, Attribute);
    }
  }

  findClose(Handle);

  return false;
}
```

See Also

findNextFile, **findClose**

findNextFile

Requires FM 1.0 Beta 6 (May 2005) or newer

Syntax

```
int findNextFile(int* hFindFile, char* foundItem, int
*dwFileAttributes)
```

Arguments

hFindFile
: Search handle obtained from a previous call to findFirstFile

foundItem
: A string where the name of the found file will be stored.

dwFileAttributes
: An integer where the attributes of the located file will be stored (see below for attributes list).

File Attributes

FILE_ATTRIBUTE_ARCHIVE	A file or directory that is an archive file or directory.
FILE_ATTRIBUTE_COMPRESSED	A file or directory that is compressed.
FILE_ATTRIBUTE_DEVICE	This value is reserved for system use.
FILE_ATTRIBUTE_DIRECTORY	The handle that identifies a directory.
FILE_ATTRIBUTE_ENCRYPTED	A file or directory that is encrypted.
FILE_ATTRIBUTE_HIDDEN	The file or directory is hidden. It is not included in an ordinary

	directory listing.
FILE_ATTRIBUTE_INTEGRITY_STREAM	The directory or user data stream is configured with integrity (only supported on ReFS volumes).
FILE_ATTRIBUTE_NORMAL	A file that does not have other attributes set. This attribute is valid only when used alone.
FILE_ATTRIBUTE_NOT_CONTENT_INDEXED	The file or directory is not to be indexed by the content indexing service.
FILE_ATTRIBUTE_NO_SCRUB_DATA	The user data stream not to be read by the background data integrity scanner (AKA scrubber).
FILE_ATTRIBUTE_OFFLINE	The file data is physically moved to offline storage.
FILE_ATTRIBUTE_READONLY	A file that is read-only.
FILE_ATTRIBUTE_RECALL_ON_DATA_ACCESS	The file or directory is not fully present locally. For a file that means that not all of its data is on local storage (e.g. it may be sparse with some data still in remote storage).
FILE_ATTRIBUTE_RECALL_ON_OPEN	The file or directory has no physical representation on the local system; the item is virtual.
FILE_ATTRIBUTE_REPARSE_POINT	A file that is a symbolic link.
FILE_ATTRIBUTE_SPARSE_FILE	A file that is a sparse file.
FILE_ATTRIBUTE_SYSTEM	A file or directory that the operating system uses a part of, or uses exclusively.

FILE_ATTRIBUTE_TEMPORARY	A file that is being used for temporary storage.
FILE_ATTRIBUTE_VIRTUAL	This value is reserved for system use.

Return

Returns INVALID_HANDLE_VALUE if the search failed or could not find any files. Otherwise it returns a search handle that can be used in subsequent calls to **findNextFile** or **findClose**.

Description

Searches a directory for another file or subdirectory matching the given name or wildcard associated with the search handle from a previous call to findFirstFile.

Example

```
%ffp

OnFilterStart: {

  int Handle, Attribute;

  // Search for 8bf files in the filterInstallDir
  snprintf(str1, 255, "%s\\*.8bf", filterInstallDir);

  // Optionally, search the C: root instead
  //strcpy(str1,"C:\\");

  Handle = findFirstFile (str1, str9, &Attribute);
  if (Attribute != FILE_ATTRIBUTE_DIRECTORY)
    Info ("%s - %d",str9, Attribute);
  if (Handle != INVALID_HANDLE_VALUE){
    while (findNextFile(Handle, str9, &Attribute) != 0) {
      if (Attribute != FILE_ATTRIBUTE_DIRECTORY)
        Info ("%s - %d",str9, Attribute);
    }
  }

  findClose(Handle);

  return false;
}
```

See Also

findFirstFile, **findClose**

floor

Syntax

```
double floor(double number)
```

Arguments

number
: Any double or float number.

Return

The rounded value.

Description

Returns the largest integral value smaller than or equal to **number**.

Example

```
%ffp

OnFilterStart:
{
    Info("Rounding 2.345 to floor gives %f",
floor(2.345));
    Info("Rounding -2.345 to floor gives %f",
floor(-2.345));
}
```

See Also

ceil

flushall

Syntax

```
int flushall()
```

Return

Returns the number of files flushed.

Description

Writes all remaining data in the writing buffer of all opened files and streams.

The main use of this function is security; ensuring the data is stored in the file in order to protect against crashes. Use this function if your file format allows processing of the partial file.

Example

```
int FMI_FILE;
if (FMI_FILE = fopen("d:\\FM_image0.fmi", "wb")) {
  // Write out the src pixels as raw data
  for (z = 0; z < Z; ++z) {
    for (y = 0; y < Y; ++y) {
      for (x = 0; x < X; ++x) {
        fputc(src(x, y, z), IMG_FILE);
      }
    }
    flushall(); // ensure complete channel is stored
before next channel is written.
  }
}
```

```
else
  ErrorOk("Cannot write image file\nDrive is either full
or write-protected!");

if (fclose(IMG_FILE))
  ErrorOk("Cannot close image file!");
```

See Also

fflush

fmax

Requires FM 1.0 Beta 6 (May 2005) or newer

Syntax

```
double fmax(double a, double b, ...)
```

Arguments

a
: Any floating-point double value.

b
: Any floating-point double value.

...
: Any number of floating-point double values.

Return

The higher value of all given parameters.

Description

Returns the largest of the given values.

Example

```
double p = fmax(1.0, 2.0);  // sets p to 2.0
```

See Also

add, **fmin**, **max**, **min**, **sub**

FMC_TARGET

Requires FM version 1.0 Beta9g MT4 or newer

Syntax

`int FMC_TARGET`

Description

Integer constant value set by FilterMeister to 32 for 32-bit plugins, and 64 for 64-bit plugins. Only available from FM1.0 Beta9g MT4 onwards.

Example

```
%fml

ForEveryTile:
{
  Info("This is a %d-bit plugin", FMC_TARGET);
  return true;
}
```

FME_CANCEL

Description

FME_CANCEL event is triggered when the user exits the plug-in without applying the effect.

It also works for the ESC key and the x button in the titlebar.

To activate this event you need to use `Dialog: cancelevent` or `setDialogEvent(2)`.

Comments

Since Beta 8.7 the event `n== CTL_CANCEL && e==FME_CLICKED` is almost equivalent to the event `e==FME_CANCEL`

Example

```
%ffp
Dialog: cancelevent

OnCtl(n): {
    if (e==FME_CANCEL) {
         Info("You have just exited the plug-in");
    }
    return false;
}
```

See Also

FME__INIT, **setDialogEvent**

FME_DRAWITEM

Description

Event that is triggered when an **OWNERDRAW** control needs to be redrawn.

FME_INIT

Description

The FME_INIT event is triggered before the FM dialog is displayed. To activate this event you need to use `Dialog: initevent`.

You can use the previous value to check if the plug-in is executed for the first time in the host. In that case `previous==false`.

Comment

1) Using the instruction `setDialogEvent(1)` allows also to enable the FME_INIT event. However, it isn't recommended to use that by default, as it won't work for the very first launch of the plug-in.

2) For your convenience, this event is also triggered each time you recompile the code of your plug-in.

Example

```
%ffp
Dialog: initevent

OnCtl(n): {
  if (e==FME_INIT) {
    if (!previous)
        Info("plug-in launched for first time");
    else
        Info("plug-in just launched or recompiled");
  }
  return false;
}
```

See Also

FME__CANCEL, **setDialogEvent**

FME_KEYDOWN

Description

FME_KEYDOWN event is triggered when the user presses a keyboard button.

The *n* value of the events contains the VK code of the pressed key.

To activate the FME_KEYDOWN event you need use "Dialog: keyevents" or setDialogEvent(4) in your code.

If you want to check for mouse buttons, Shift, Ctrl, Alt etc. you need to use getAsyncKeyState() or getAsyncKeyStateF().

This event is only triggered once per keypress; it is not triggered again if you hold the key.

Example

```
%ffp
Dialog: keyevents

OnCtl(n): {
    if (e == FME_KEYDOWN) {
         Info("You just pressed key %d", n);
    }
    return false;
}
```

See Also

FME__KEYUP, getAsyncKeyState, getAsyncKeyStateF, setDialogEvent, VK codes

FME_KEYUP

Description

FME_KEYUP event is triggered when the user releases a keyboard button.

The *n* value of the events contains the VK code of the released key.

To activate the FME_KEYDOWN event you need use "Dialog: keyevents" or **setDialogEvent(4)** in your code.

If you want to check for mouse buttons, Shift, Ctrl, Alt etc. you need to use **getAsyncKeyState** or **getAsyncKeyStateF**.

Example

```
%ffp
Dialog: keyevents

OnCtl(n): {
    if (e == FME_KEYUP) {
         Info("You just released key %d", n);
    }
    return false;
}
```

See Also

FME__KEYDOWN, **getAsyncKeyState**, **getAsyncKeyStateF**, **setDialogEvent**, **VK codes**

FME_MOUSEMOVE

Returned Parameters

n
: Currently the mousemove event is only supported for the preview. So n will be always equal to CTL_PREVIEW.

Description

This event is triggered for every move of the mouse pointer above the preview. To get the coordinates of the mouse pointer you have to use the **getPreviewCoordX** and **getPreviewCoordY** functions.

Comment

No action will be performed after this event was processed. So if you want to update the preview you have to add `doAction(CA_PREVIEW)` to the code that processed this event in the OnCtl handler.

Example

```
%ffp

ctl(10):statictext,"Please move the mouse over the 
preview!",size=(150,*)

OnCtl(n):{

   if (n == CTL_PREVIEW && e == FME_MOUSEMOVE){
       setCtlTextv ( 10, "%d, %d", getPreviewCoordX(), 
getPreviewCoordY() );
```

```
    }

    return false;
}
```

See Also

FME__MOUSEOVER, **FME__MOUSEOUT**, **getPreviewCoordX**, **getPreviewCoordY**

FME_MOUSEOUT

Returned Parameters

n
: The number of the control which triggered this event.

Description

The mouseout event will be triggered when the mouse leaves the screen space used by any control (numbered **n**) which has the MOUSEOVER properties specified in its definition.

Comment

No action will be performed after this event was processed. So if you want to update the preview you have to add `doAction(CA_PREVIEW)` to the code that processed this event in the OnCtl handler.

Example

```
%ffp

ctl(0): statictext(MOUSEOVER), "Move the mouse over me"

OnCtl(n):{

   if (n == 0 && e == FME_MOUSEOVER){
       setCtlTextv (0, "Now move off me");
   }

   if (n == 0 && e == FME_MOUSEOUT){
       setCtlTextv (0, "Now move over me again");
```

```
    }

    return false;
}
```

See Also

FME_MOUSEMOVE, **FME_MOUSEOVER**

FME_MOUSEOVER

Returned Parameters

n

The number of the control which triggered this event.

Description

The mouseover event will be triggered when the mouse enters the screen space used by any control (numbered **n**) which has the MOUSEOVER properties specified in it's definition. It will only be triggered once for every time spend hovering over the control, in order to trigger an event with every movement over the control, use the **FME_ MOUSEMOVE** event.

Comment

No action will be performed after this event was processed. So if you want to update the preview you have to add *doAction(CA_PREVIEW)* to the code that processed this event in the OnCtl handler.

Example

```
%ffp

ctl(0): statictext(MOUSEOVER), "Move the mouse over me"

OnCtl(n):{

   if (n == 0 && e == FME_MOUSEOVER){
       setCtlTextv (0, "Now move off me");
   }

   if (n == 0 && e == FME_MOUSEOUT){
       setCtlTextv (0, "Now move over me again");
   }

   return false;
}
```

See Also

FME_MOUSEMOVE, FME_MOUSEOUT

fmin

Requires FM 1.0 Beta 6 (May 2005) or newer

Syntax

```
double fmin(double a, double b, ...)
```

Arguments

a
: Any floating-point double value.

b
: Any floating-point double value.

...
: Any number of floating-point double values.

Return

The lower value of all given parameters.

Description

Returns the least of the given values.

A common use for **fmin** is to truncate a variable to a certain upper boundary.

Example

```
// sets p to 1.0
double p = fmin(1.0, 2.0);
```

See Also

add, **fmax**, **max**, **min**, **sub**

fmod

Syntax

```
double fmod(double x, double y)
```

Arguments

x
: Value to be divided.

y
: Divider.

Return

Remainder of x/y.

Description

Performs division x/y and returns the remainder of the operation.

fopen

Syntax

```
int fopen(string filename, string mode)
```

Arguments

filename
: The full pathname of the file to open. Use double backslashes on Windows machines.

mode
: The mode to open the file in, see below for details.

Return

Returns a pointer to the file stream that can be used in later file operations, if the file was opened successfully. Returns NULL if the file could not be opened.

Description

Opens a file or a stream, allowing it to be read from or written to. The mode can be any of a number of settings, including:

r	Opens the file for reading.
w	Opens the file for writing, overwriting any currently existing file.
a	Opens the file in append mode, adding data to end of file or creating new file if doesn't exist.
rb	Open a file in binary mode for reading only.
wb	Open or create a file in binary mode for writing only.
r+	Opens the file for reading and writing.

w+	Opens the file for reading and writing, overwriting any currently existing file.

You can also specify whether the file should be opened in text or binary mode by adding 't' or 'b' to the mode string respectively. Text mode causes the program to interpret newlines differently depending on the machine it is operating on (Unix, Windows and Mac all have different newline endings). Binary mode is recommended for most operations on data files.

Comments

If you repeatedly have problems opening a file, and you are sure that it exists and isn't in use by other programs, check that you have used double backslashes in the filename (like in the example above). Otherwise character sequences such as \t or \n may be interpreted as tabs & newlines respectively... which is not what you want to happen.

Example

```
int IMG_FILE;
if (IMG_FILE = fopen("d:\\FM_image0.fmi", "wb")) {
  // Write out the src pixels
  // as raw data
  for (z=0; z<Z; z++)
  for (y=0; y<Y; y++)
  for (x=0; x<X; x++)
    fputc(src(x,y,z), IMG_FILE);
}
else
  ErrorOk("Cannot write image file\nDrive is either full
or write-protected!");
```

```
if (fclose(IMG_FILE))
  ErrorOk("Cannot close image file!");
```

See Also

fclose

formatString

Syntax

```
string formatString(string s)
```

Arguments

s
: A string containing 2-character formatting substrings and HTML entity codes

Return

Returns string s with substitutions made.

Description

Returns string s with the following substitutions made for the designated 2-character substrings (all of which begin with "!" or "&")

FM deals with 4 different escape characters, which are substituted in 3 separate phases:

escape character '\'

Escape code sequences beginning with '\' are handled by the parser, only during the parsing of string constants (not string variables). FM performs the usual C-language substitutions: '\t' for TAB, '\n' for new-line, '\r' for carriage-return, '\"' for a double-quote, etc.

The '\' char itself is represented by the escape code '\\'.

escape character '%'

Escape codes beginning with '%' are handled by the next phase, only during processing by built-in functions that take a variable number of arguments (such as setDialogTextv). FM calls the C-library routine sprintf (or some equivalent) which replaces the %-escape codes with formatted representations of the optional arguments according to the C-library conventions: "%d" for a decimal number argument, "%s" for a character string argument, etc.

The '%' char itself is represented by the escape code "%%".

escape characters '!' and '&'

Escape codes beginning with these characters are processed by the formatString routine during the final pass.

Two-character escape codes begining with '!' are substituted with an internal FM text string value: "!A" is replaced with the contents of the built-in string filterAuthorText (which contains the value of the Author: keyword field), "!C" is replaced by the contents of the filterCategoryText string, etc.

The '!' char itself is represented by the escape code "!!".

Escape codes beginning with '&' and ending with ';' are HTML entity references. FM replaces the escape code sequence """ with a double-quote character, "€" with the euro character, etc.

The '&' char itself is represented by the escape code "&&" or "&".

In summary, if your character string will be processed by any one of the above 3 passes, the escape character(s) for that pass can be represented by doubling the escape character in question. (If your string will NOT be processed by some particular pass, then the escape character(s) for that pass should NOT be doubled.)

! escape sequences

substring	is replaced by
&&	"&" (i.e., a verbatim ampersand)
!!	"!" (i.e., a verbatim exclamation point)
!A	text specified by the Author key, obtained from the filterAuthorText string
!a	text specified by the About key, obtained from the filterAboutText string
!C	text specified by the Category key, obtained from the filterCategoryText string
!c	text specified by the Copyright key, obtained from the filterCopyrightText string
!D	text specified by the Description key, obtained from the filterDescriptionText string
!d	text specified by the propTitle image property, obtained from the documentTitleText string ***(not yet implemented)***
!F	text specified by the Filename key, obtained from the filterFilenameText string
!f	the current Filter Case (e.g., "Flat image, no selection"), obtained from the filterCaseText string
!H	the name of the Host application (e.g., "Adobe Photoshop"), obtained from the filterHostText string
!h	the height of the image in pixels, obtained from the imageHeight variable
!I	the directory in which the plug-in filter was installed, from the filterInstallDir string ***(not yet implemented)***
!M	the current Image Mode (e.g., "RGB Color"), obtained

	from the filterImageModeText string
!m	the current image mode as a decimal integer, obtained from the imageMode variable
!O	text specified by the Organization key, obtained from the filterOrganizationText string
!S	the current host serial number string, obtained from the hostSerialString string ***(not yet implemented)***
!s	the current host serial number as a decimal value, from the hostSerialNumber variable ***(not yet implemented)***
!T	text specified by the Title key, obtained from the filterTitleText string
!t	text specified by the Title key as above, with any trailing ellipsis removed
!U	text specified by the URL key, obtained from the filterURLText string
!V	text specified by the Version key, obtained from the filterVersionText string
!w	the width of the image in pixels, obtained from the imageWidth variable
!X	the Host architecture of the host system running the plug-in (either "32" for a 32-bit system or "64" for a 64-bit system) ***(implemented in FM mt5c05)***
!x	the Target architecture the plug-in was compiled for (eg "32" for a 32-bit plug-in, "64" for a 64-bit plug-in) ***(implemented in FM mt5c05)***
!Y	the current year in decimal (e.g., for use in a Copyright message) ***(implemented in 1.0 Beta 9.0c)***
!z	the current proxy preview zoom factor (1 - 16), from the zoomFactor variable

Caution: Other substrings beginning with "!" are reserved for future use.

HTML entities

formatString also substitutes ISO character codes for the following HTML entity names.

Entity name	**ISO**	**Character meaning**
Common Punctuation		
&quot;	"	double quote
&amp;	&	ampersand
&apos;	'	apostrophe
&hellip;	…	horizontal ellipsis
&iquest;	¿	inverted question mark
&iexcl;	¡	inverted exclamation mark
&Exclam;	‼	double exclamation mark
&sect;	§	section mark
&para;	¶	paragraph mark
&pilcrow;		pilcrow (unfilled paragraph mark)
&bull;	•	bullet
&bull2;		bullet style 2
&ibull2;		inverse bullet style 2
Currency and Commerce Symbols		
&curren;	¤	currency sign
&cent;	¢	cent sign
&pound;	£	pound sign
&yen;	¥	Yen sign
&euro;	€	Euro sign
&copy;	©	copyright
&reg;	®	registered trademark
&trade;	™	trademark

&permil;	‰	per mille
&brvbar;	¦	broken vertical bar
&ordf;	ª	feminine ordinal
&ordm;	º	masculine ordinal
&male;	♂	male symbol
&female;	♀	female symbol
Science, Logic, and Mathematics Symbols		
&lt;	<	less than
&gt;	>	greater than
&le;	<=	less or equal (substitute for ≤)
&ge;	>=	greater or equal (substitute for ≥)
&ne;	!=	not equal (substitute for ≠)
&colone;	:=	(used for colon-equal)
&Colon;	::	(used for double colon)
&deg;	°	degree sign
&prime;	′	prime, minutes, feet
&Prime;	″	double prime, seconds, inches
&not;	¬	logical not
&plusmn;	±	plus-or-minus sign
&minus;	–	minus sign (en-dash used as substitute)
&times;	×	multiplication symbol (times)
&divide;	:	Division sign
&lowast;	∗	low asterisk (asterisk used as substitute)
&sdot;	·	dot operator
&middot;	·	vertically centered dot
&empty;	∅	Empty set
&sim;	~	similar to (tilde used as substitute)
&frac14;	¼	fraction 1/4

&frac12;	½	fraction 1/2
&frac34;	¾	fraction 3/4
&frasl;	⁄	fraction slash (use /)
&oline;	‾	overline (use macron)
&micro;	µ	micro
&fnof;	ƒ	f with hook
&image;	ℑ	imaginary part (blackletter I)
&real;	ℜ	real part (blackletter R)
&weierp;	℘	Weierstrass powerset (script P)
Markup-significant and Internationalization Characters		
&nbsp;		non-breaking space
&shy;		soft hyphen (never breaks in FM, so never displays)
&mdash;	—	em dash
&ndash;	–	en dash
&emsp;		em-space (use normal space)
&ensp;		en-space (use normal space)
&thinsp;		thin space (use normal space)
&zwnj;		zero width non-joiner (no char)
&zwj;		zero width joiner (no char)
Spacing Accents, Diaereses and other Diacritical Marks		
&acute;	´	acute accent
&uml;	¨	umlaut (diaeresis)
&cedil;	¸	spacing cedilla
&circ;	ˆ	modifier letter circumflex accent
&tilde;	˜	small tilde
&macr;	¯	spacing macron (overhead line)
Uppercase Accented Letters and Ligatures		
&Agrave;	À	A grave

&Aacute;	Á	A acute
&Acirc;	Â	A circumflex
&Atilde;	Ã	A tilde
&Auml;	Ä	A umlaut
&Aring;	Å	A ring
&AElig;	Æ	A-E ligature
&Ccedil;	Ç	C cedilla
&Egrave;	È	E grave
&Eacute;	É	E acute
&Ecirc;	Ê	E circumflex
&Euml;	Ë	E umlaut
&Igrave;	Ì	I grave
&Iacute;	Í	I acute
&Icirc;	Î	I circumflex
&Iuml;	Ï	I umlaut
&Ntilde;	Ñ	N tilde
&Ograve;	Ò	O grave
&Oacute;	Ó	O acute
&Ocirc;	Ô	O circumflex
&Otilde;	Õ	O tilde
&Ouml;	Ö	O umlaut
&Oslash;	Ø	slashed O
&OElig;	Œ	O-E ligature
&Scaron;	Š	capital S with caron
&Ugrave;	Ù	U grave
&Uacute;	Ú	U acute
&Ucirc;	Û	U circumflex
&Uuml;	Ü	U umlaut

&Yacute;	Ý	Y acute
&Yuml;	Ÿ	capital Y with diaeresis
&Zcaron;	Ž	capital Z with caron (not official)
&ETH;	Ð	capital letter Eth
&THORN;	Þ	capital letter Thorn
Lowercase Accented Letters and Ligatures		
&szlig;	ß	s-z ligature
&agrave;	à	a grave
&aacute;	á	a acute
&acirc;	â	a circumflex
&atilde;	ã	a tilde
&auml;	ä	a umlaut
&aring;	å	a ring
&aelig;	æ	a-e ligature
&ccedil;	ç	c cedilla
&egrave;	è	e grave
&eacute;	é	e acute
&ecirc;	ê	e circumflex
&euml;	ë	e umlaut
&igrave;	ì	i grave
&iacute;	í	i acute
&icirc;	î	i circumflex
&iuml;	ï	i umlaut
&ntilde;	ñ	n tilde
&ograve;	ò	o grave
&oacute;	ó	o acute
&ocirc;	ô	o circumflex
&otilde;	õ	o tilde

&ouml;	ö	o umlaut
&oslash;	ø	slashed o
&oelig;	œ	small o-e ligature
&scaron;	š	small s with caron
&ugrave;	ù	u grave
&uacute;	ú	u acute
&ucirc;	û	u circumflex
&uuml;	ü	u umlaut
&yacute;	ý	y acute
&yuml;	ÿ	y umlaut
&zcaron;	ž	small z with caron (not official)
&eth;	ð	small letter eth
&thorn;	þ	small letter thorn
Subscripts, Superscripts and Footnotes		
&sup1;	¹	superscript 1
&sup2;	²	superscript 2
&sup3;	³	superscript 3
&dagger;	†	dagger
&Dagger;	‡	double dagger
Various Quotation Marks		
&lsquo;	‘	left single quotation mark
&rsquo;	’	right single quotation mark
&sbquo;	‚	single low-9 quotation mark
&bdquo;	„	double low-9 quotation mark
&ldquo;	“	left double quotation mark
&rdquo;	”	right double quotation mark
&lsaquo;	‹	left single angle quotation
&rsaquo;	›	right single angle quotation

&laquo;	«	left angle quote
&raquo;	»	right angle quote
Subset of Available Greek Characters		
&Alpha;	A	Alpha (use latin capital A)
&Beta;	B	Beta (use latin capital B)
&Epsilon;	E	Epsilon (use latin capital E)
&Zeta;	Z	Zeta (use latin capital Z)
&Eta;	H	Eta (use latin capital H)
&Iota;	I	Iota (use latin capital I)
&Kappa;	K	Kappa (use latin capital K)
&Mu;	M	Mu (use latin capital M)
&Nu;	N	Nu (use latin capital N)
&Omicron;	O	Omicron (use latin capital O)
&Rho;	P	Rho (use latin capital P)
&Tau;	T	Tau (use latin capital T)
&Upsilon;	Y	Upsilon (use latin capital Y)
&Chi;	X	Chi (use latin capital X)
&beta;	β	beta (use sz-ligature; yecch!)
&mu;	µ	mu (use µ)
&omicron;	o	omicron (use small latin o)
OEM Arrows, Line Drawing and Graphics		
&larr;	←	leftwards arrow
&uarr;	↑	upwards arrow
&rarr;	→	rightwards arrow
&crarr;	↲	carriage-return arrow (composed by two other graohics)
&varr;	↕	up/down (vertical) arrow (not official)
&ltrif;	◂	left-pointing triangle, filled
&blk14;	░	lightly shaded box

&boxv;	│	box drawing: vertical
&boxvl;	┤	box drawing: vertical and left
&boxdl;	┐	box drawing: down and left
&boxur;	└	box drawing: up and right
&boxhu;	┴	box drawing: horizontal and up
&boxhd;	┬	box drawing: horizontal and down
&boxvr;	├	box drawing: vertical and right
&boxh;	─	box drawing: horizontal
&boxvh;	┼	box drawing: vertical and horizontal
&boxul;	┘	box drawing: up and left
&boxdr;	┌	box drawing: down and right
&twonotes;		two beamed 16th notes

Notes

You do not usually need to call formatString() explicitly, since it is implicitly applied in the following cases:

- when displaying the text label for a dialog control,
- when displaying the items in a **LISTBOX** or **COMBOBOX** control,
- when displaying the dialog caption in the title bar,
- when displaying text in a Message Box via the **msgBox**(), **printf**(), **Info**(), **Warn**(), **Error**(), **ErrorOk**(), **YesNo**(), or **YesNoCancel**() built-in functions,
- when displaying the prompt in the **chooseColor**() color-picker dialog box,
- when specifying the registry path and key names in the Windows Systems Registry access functions,
- when displaying the About text in the About dialog box.

Example

For example, the default Dialog text is "!t (!M, !f)", which formatString() translates to something like "My Filter Title (RGB Color, Flat image, no selection)" at run time.

See Also

appendEllipsis, **stripEllipsis**

fputArray

Requires FM 1.0 Beta 9.0 (Apr 2008) or newer

Syntax

```
int fputArray (int nr, int x, int y, int z, double dval)
```

Arguments

nr
: Number of the array. Values from 0 to 99 are accepted.

x, y, z
: x, y, and z coordinates of a cell in the array. If you allocated a one-dimensional array, set y and z to zero. If you allocated a two-dimensional array, set z to zero.

dval
: Floating-point value that will be stored at the specified coordinates in the array.

Return

Returns 0 for failure (invalid index nr, or invalid byte-size of Array), and 1 for success.

Description

Lets you store a floating-point value in an array. When storing a value into an Array with byte-size 2, the value will be converted to a 16-bit half. For an Array of byte-size 4, the value is converted to a 32-bit float. If the byte-size is not 2, 4, or 8, a value of 0 is returned to indicate failure.)

Example

See **fgetArray**

See Also

allocArray, **freeArray**, **getArray**, **fgetArray**, **putArray**, **fillArray**, **ffillArray**, **getArrayDim**, **copyArray**

fputc

Syntax

```
int fputc(int c, int * fileptr)
```

Arguments

c
: An integer containing a byte value.

fileptr
: Pointer to a file opened using **fopen**.

Return

The value of **c** if successful, otherwise -1.

Description

Write a single byte, contained in variable **c** to the file referenced by **fileptr**.

See Also

fopen, **fgetc**, **putc**, **fputs**

fputs

Syntax

```
int fputs(char * s, int * fileptr)
```

Arguments

s
: Pointer to a string which contains the bytes to be written.

fileptr
: Pointer to a file opened using **fopen**.

Return

A positive number if successful, otherwise a negative number.

Description

Writes the contents of string **s** to the file referenced by **fileptr**.

See Also

fopen, **fgets**, **fputc**, **fwrite**

fr2x

Syntax

```
double fr2x(double d, double m)
```

Arguments

d
: A double floating-point value for the 'direction' of a pixel.

m
: A double floating-point value for the 'magnitude' of a pixel.

Return

A double floating-point value giving the cartesian x coordinate for the pixel whose polar coordinates are [d,m].

Description

Pixels are usually addressed by their cartesian coordinates [x,y], but FilterMeister also allows the use of polar coordinates, and provides a set of functions for conversion between the two systems. Polar coordinates are expressed as [d,m], where 'd' represents the 'direction' to the pixel from the image's center point, and 'm' is the 'magnitude' of the distance from the center. The fr2x() function takes a pair of polar coordinates as arguments, and returns the cartesian x coordinate of the corresponding pixel.

See Also

c2d, c2m, r2y

fr2y

Syntax

```
double fr2y(double d, double m)
```

Arguments

d
: A double floating-point value for the 'direction' of a pixel.

m
: A double floating-point value for the 'magnitude' of a pixel.

Return

A double floating-point value giving the cartesian y coordinate for the pixel whose polar coordinates are [d,m].

Description

Pixels are usually addressed by their cartesian coordinates [x,y], but FilterMeister also allows the use of polar coordinates, and provides a set of functions for conversion between the two systems. Polar coordinates are expressed as [d,m], where 'd' represents the 'direction' to the pixel from the image's center point, and 'm' is the 'magnitude' of the distance from the center. The fr2y() function takes a pair of polar coordinates as arguments, and returns the cartesian y coordinate of the corresponding pixel.

See Also

c2d, **c2m**, **fr2x**, **r2y**

FRAME

Syntax

```
ctl[n]: FRAME(Class Specific Properties), Other Properties
```

Description

This user control class draws a frame in the dialog window. By default, this user control is not actionable.

Class Specific Properties

BLACK
Defines the border's color as black. *(default)*

ETCHED
Gives the border a 3D-look.

ETCHEDHORZ
Makes a single horizontal line with a 3D appearance.

ETCHEDVERT
Makes a single vertical line with a 3D appearance.

GRAY
Defines the border's color as gray.

NOTIFY
Makes the user control actionable and activates tooltip.

WHITE
Defines the border's color as white.

Other Properties

Val
Assigns a value to the frame, but only when it is disabled. *(default = 0)*

Example

```
ctl[0]: FRAME
ctl[1]: FRAME(ETCHED), Val=45, Disabled
ctl[2]: FRAME(GRAY), Pos=(320,60), Size=(160,60)
```

Notes

Once the frame user control is actionable, its value definitions are lost. The reason is that an action returns a specific value and overwrites (once the mouse button is clicked over the user control) the user control's value.

fread

Syntax

```
int fread(void* buffer, int size, int numitems, void* file)
```

Arguments

buffer
: A pointer to the data buffer where the data read from the file will be stored.

size
: The size of the data blocks you want to read from the file (eg if reading 32-bit integers, you might set this to 4 bytes).

numitems
: The number of blocks of data you want to read from the file (eg number of integers you want to read from the file in a row).

file
: The pointer/handle to the already opened file you want to read from.

Return

Returns the number of items that were successfully read from the file.

Description

Reads data from a previously opened file or stream.

Comments

The size and numitems parameters can be confusing to use. It's often easier to set the size parameter to the number of bytes you want to read from the file, and keep the numitems parameter set to 1.

See Also

fopen, **fread**, **fseek**, **fsetpos**, **fwrite**, **fclose**

free

Syntax

`free(void* buffer)`

Arguments

buffer
: A pointer to a previously allocated memory block.

Description

Frees the block of memory previously allocated specified by the pointer **buffer**.

When specifying a NULL pointer, this function does nothing.

Example

```
%ffp

OnFilterStart: {

    // Allocate a string for
    // 255 characters.
    char* buffer_1 = malloc(255);

    free(buffer_1);
}
```

See Also

calloc, malloc, realloc

freeArray

Syntax

```
int freeArray (int nr)
```

Arguments

nr
: Number of the array. Values from 0 to 99 are accepted.

Return

Always returns a value of 1

Description

Lets you delete one of the 100 available arrays and free the memory that is used for it.

Example

See **allocArray**

See Also

allocArray, **getArray**, **putArray**, **getArrayDim**, **copyArray**

freeHost

Requires FM 1.0 Beta 9.1 (February 2009) or newer

Syntax

```
int freeHost(int bufferID)
```

Arguments

bufferID
: The ID of the memory buffer to be freed.

Return

Always returns true.

Description

Frees a memory block previously allocated by the host application by using the **allocHost** function.

Example

```
int bufferID = allocHost(100);
if (bufferID == NULL) {
  Warn("Could not allocate memory");
}
else {
  char* memptr = lockHost(bufferID);
  sprintf(memptr, "Message goes here!");
  Info(memptr);
  freeHost(bufferID);
}
```

See Also

allocHost, lockHost

freeLib

Requires FM 1.0 Beta 9.0 (April 2008) or newer

Syntax

```
bool freeLib(void *handle)
```

Arguments

handle
Handle (as obtained from **loadLib**) to the DLL to be freed.

Returns

Returns a non-zero value (ie true) if the library was freed successfully, or zero / false if the function failed.

Description

Frees a DLL from memory. Behind the scenes, this function is just a wrapper around the **[FreeLibrary]** Win32 API function.

Example

```
// This code loads the user32.dll
// DLL included with Windows and
// uses it to display a YES/NO
// Message Box.

int* lib_user32;
int* functionPointer;
int returnval;
```

```
// Load the DLL library
lib_user32 = loadLib("user32");
if (lib_user32) {

  // Get the function in the DLL
  functionPointer = getLibFn(lib_user32, "MessageBoxA");
  if (functionPointer) {

    // Call the function
    strcpy(str0, "The window text is here");
    strcpy(str1, "Caption Text");
    returnval = callLib(functionPointer, NULL, str0, str1,
MB_YESNO);

    // Process return value
    if (returnval == IDYES)
      msgBox(MB_OK, "Yes!", "Yes was clicked");
    if (returnval == IDNO)
      msgBox(MB_OK, "No :(", "No was clicked");

  }
  else msgBox(MB_OK, "Error", "Function wasn't loaded");

  // Free the library DLL
  freeLib(lib_user32);

}
else msgBox(MB_OK, "Error", "DLL was not loaded");
```

See Also

loadLib, getLibFn, callLib

fseek

Syntax

```
int fseek(FILE * file, long int offset, int origin)
```

Arguments

file
: A pointer to the file that you are working with (obtained with **fopen**).

offset
: The offset (usually in bytes, for binary files) that you want to seek to in the file, from the given origin.

origin
: Designates whether to seek from the start (== 0), current position (== 1), or end (== 2) of the file.

Return

Zero if the operation succeeded, non-zero otherwise.

Description

Seeks within a file to the given position.

See Also

fopen, **fread**, **fseek**, **fsetpos**, **fwrite**, **fclose**

fsetpos

Syntax

```
int fsetpos(FILE * file, fpos_t * position)
```

Arguments

file
: A pointer to the file that you are working with (obtained with fopen).

position
: A 64-bit pointer to the new position in the file you want to move to (obtained with fgetpos).

Return

Returns zero if the operation completed successfully, non-zero otherwise.

Description

Changes the current read/write position in the file to the new position, given by position. Not really intended for moving to random positions in a file, unless you've already obtained pointers to those positions previously using fgetpos. For random file seeking, you should use fseek instead.

Note that position is actually a 64-bit pointer, which FilterMeister doesn't yet support. If you need to use this, you should allocate your own memory for the pointer, or use the fact that the k0 & k1 variables are allocated next to each other in memory.

Comments

Due to a bug in early versions of FilterMeister, fsetpos was sometimes used in place of fseek for seeking. It isn't recommended to do this, but if you need to use this workaround, here is more information on using fsetpos instead of fseek from the FMML FilterMeister Mailing List:

> *Where you have:*
>
> ```
> // fsetpos(SETTINGS_FILE, &mybyte);
> // seek forward 37 bytes
> ```
>
> *you might like to try this instead:*
>
> ```
> k0 = 37;
> k1 = 0;
> fsetpos(SETTINGS_FILE, &k0);
> ```
>
> *Why are we doing things this way? It's because fsetpos actually requires a 64-bit integer, but at the moment FilterMeister doesn't support them. So instead, we abuse the fact that the k0 and k1 variables are stored next to each other, and "pretend" they make a 64-bit integer. By giving the address of k0 (using &k0), the function will take the memory address and read enough of memory to make the 64-bit integer.*

See Also

fopen, **fread**, **fseek**, **fwrite**, **fclose**

fsin

Syntax

```
double fsin(double x)
```

Arguments

x
: An angle in radians, as a double-precision floating point value

Return

Sine of radian angle x, as a double-precision floating point value

Description

Calculates the sine of the given radian angle x.

Example

```
%fml

float result;
float PI = 3.14159275;
float degrees = 45.0;
float radians;

radians = degrees * PI / 180;
result = fsin(radians);
printf("Sine of %lf degrees (%lf radians) is %lf",
degrees, radians, result);
```

```
%%EOF
```

ftell

Syntax

```
int ftell(FILE * file)
```

Arguments

file
A pointer to the file that you are working with (obtained with **fopen**).

Return

Returns the current position in the given file, or -1 if an error occurred.

Description

Returns the current position in the given file stream. You can use this in conjunction with **fseek** to get the size of a file by first seeking to the end of the file, then retrieving the current position with ftell.

See Also

fopen, **fread**, **fseek**, **fwrite**, **fclose**

fwrite

Syntax

```
int fwrite(void* buffer, int size, int numitems, void* file)
```

Arguments

buffer
: A pointer to the data buffer where the data read from the file will be stored.

size
: The size of the data blocks you want to write out (eg if writing 32-bit integers, you might set this to 4 bytes).

numitems
: The number of blocks of data you want to write to the file (eg number of integers you want to write to the file in a row).

file
: The pointer/handle to the already opened file you want to write to.

Return

Returns the number of items that were successfully written to the file.

Description

Writes data to a previously opened file or stream.

Comments

The size and numitems parameters can be confusing - it's often easier to set the size parameter to the number of bytes you want to write to the file, and keep the numitems parameter set to 1.

See Also

fopen, **fread**, **fseek**, **fsetpos**, **fclose**

gamma

Syntax

```
int gamma(int i)
```

Arguments

i
: The intensity to be corrected. **i** must be higher or equal to 0 and lower or equal to 255.

Return

The gamma-corrected value of **i**.

Description

Allows the user to efficiently calculate gamma correction. To be used in conjunction with the setGamma function.

Example

```
int val;

setGamma(1.8);

for (x = x_start; x < x_end; x++) {
  for (y = y_start; y < y_end; y++) {
    for (z = 0; z < 3; z++) {

      val = gamma(src(x, y, z));
      pset(x, y, z, val);
```

```
        }
    }
}
```

See Also

setGamma

get

Syntax

`int get(int item)`

Arguments

item
: Numeric identifier of an item in the buffer.

Return

The value of the buffered item requested.

Description

FilterMeister has a small internal buffer of `N_CELLS` integer items which can be accessed by means of the `get` and `put` functions. They provide the simplest means for storing integer data since they require no variable to be declared.

Currently `N_CELLS` is 1024. It may increase in future releases of FilterMeister, but will never decrease. `N_CELLS` will always be a power of 2.

By default, the items in the buffer are initialized to zero at the end of the executing code block. Calling the `cell_preserve(1)` function changes this behavior so the buffer values are stored between separate handlers, making them ideal for transporting information between them.

The `get` function takes an integer argument in the range of 0 up to and including `(N_CELLS - 1)` and returns the value stored in that position in the buffer.

Example

```
%ffp

OnFilterStart:
{
    put(10, 0);
    Info("The value of buffer position 0 is %d", get(0));
}
```

See Also

put, **cell_preserve**

getAppTheme

Syntax

```
int getAppTheme(void)
```

Return

Returns true if the hosting application has enabled Windows XP / Vista style theming, false otherwise.

Description

Checks if the hosting application has enabled Windows themes.

Examples

```
%fml
OnFilterStart: {
  if (getAppTheme()) {
    printf("Host application is themed.");
  }
  else printf("Host application does not support
themes.");
  return false;
}
```

See Also

setCtlTheme

getArray

Syntax

```
int getArray (int nr, int x, int y, int z)
```

Arguments

nr
: Number of the array. Values from 0 to 99 are accepted.

x, y, z
: x, y, and z coordinates of a cell in the array. If you allocated a one-dimensional array, set y and z to zero. If you allocated a two-dimensional array, set z to zero.

Return

Returns the value that was stored at the specified coordinates in the array.

If the specified coordinates lie outside the array, getArray will follow the edge-mirroring behavior defined by **set__edge__mode** for the x and y co-ordinates. If you don't want this behaviour, you can use the unsafe **___getArray** function instead (which will crash or give undesired results if the co-ordinates are outside the array bounds).

By default (ie edgeMode == 0), x and y will be clamped to between 0 and X-1 or Y-1 as appropriate, while the z co-ordinate remains unchanged. If edgeMode == 1, getArray will return a zero value.

If the co-ordinates still lie outside the array, getArray will return zero.

Description

It lets you read a value from an array.

Example

See **allocArray**

See Also

allocArray, **freeArray**, **putArray**, **getArrayDim**, **copyArray**

getArrayAddress

Requires FM 1.0 Beta 8.4 (Sep 2007) or newer

Syntax

```
INT_PTR getArrayAddress(int arraynr)
```

Arguments

arraynr
The number of the built-in FM array that you want a pointer to.

Return

Returns a C-language memory address pointer to the given FM array.

Description

Returns the memory address of the requested built-in FM array. FilterMeister provides up to 100 in-built arrays numbered from 0-99. You might need to use this if working with the built-in arrays in conjunction with C language functions.

Example

```
%fml

Title:    "Test copyResToArray"
Category: "FM Example Code"
Embed:    Other="test.txt"
```

```
// Make a control to display text on the interface
ctl[10]: STATICTEXT, Size=(100, 40), Text="This text will
be replaced"

OnFilterStart: {

  bool loaded = false;

  // Load the test.txt file embedded in the plugin into
Array #0
  loaded = copyResToArray("FMDATA", "test.txt", 0);

  // If the resource loaded, copy the (max) first 255
characters
  // into an internal string variable, so it correctly
ends
  // with a zero byte to mark the end of string
  if (loaded) {
    strncpy(str1, getArrayAddress(0), 255);
  }
  else {
    strncpy(str1, "Could not find test.txt file", 255);
  }

  // Set control #10 to be labelled with the contents of
str1
  setCtlText(10, str1);

  return true;
}
```

See Also

allocArray, copyResToArray, getArray, freeArray

getArrayDim

Syntax

```
int getArrayDim (int nr, int dim)
```

Arguments

nr
: Number of the Array. Values from 0 to 99 are accepted.

dim
: Determines which dimensions will be queried. Set it to 0 for X, 1 for Y and 2 for Z. See allocArray. **New in FM 1.0 Beta 9d:** Set this argument to -1 to return the base address of the Array; -2 to return the optional Buffer ID; -3 to return the total Array size in bytes; -4 to return the number of bytes per Array element; or -5 to return the padding size of the Array.

Return

Returns the amount of cells of one of the three dimensions of an Array (or various other information about the Array), depending on the *dim* parameter.

Description

Usually you know the sizes of the three dimensions of your Array(s). But in some cases it is convenient to use this function to check it again. For other information about the Array, such as the base address or Buffer ID, this API may be the only way to obtain the desired information.

See Also

allocArray, **allocArrayPad**, **freeArray**, **getArray**, **putArray**, **copyArray**

getArrayString

Syntax

```
int getArrayString(int nr, int index)
```

Arguments

nr
: Number of the Array. Values from 0 to 99 are accepted.

index
: The index number of the item in the array you want to retrieve

Return

Returns a pointer to the string if it exists, or a pointer to the string "Not Available" otherwise.

Description

Retrieves a string stored in the built-in Arrays.

Comment

Note that you must allocate space for the array first using the allocArray function, otherwise these functions will fail.

Example

```
%ffp

ctl(0): combobox, "Harry\nJim\nSally", val=0,
action=preview, size=(*,200)
```

```
OnFilterStart:{

  // Allocate Array for storing 3 strings of max. 256
bytes length
  allocArray(0,3,256,0,1);

  // Store the Strings
  putArrayString (0,0, "Hello, Harry!");
  putArrayString (0,1, "Hello, Jimmy.");
  putArrayString (0,2, "Hello, Sally, old girl!");

  // Display
  Info ("%s", getArrayString(0,ctl(0)) );

  freeArray(0);

  return false;
}
```

See Also

allocArray, **putArrayString**, **freeArray**

getAsyncKeyState

Syntax

```
int fm_getAsyncKeyState (int vkey)
```

Arguments

vkey
Virtual Key Constant. For more information see the **[MSDN Virtual Key Codes page]**

Return

The return value specifies whether the key was pressed since the last call to GetAsyncKeyState, and whether the key is currently up or down. If the most significant bit is set, the key is down, and if the least significant bit is set, the key was pressed after the previous call to GetAsyncKeyState.

Description

Determines whether a key is up or down at the time the function is called, and whether the key was pressed after a previous call to getAsyncKeyState.

Example

```
%ffp

ctl[10]: PUSHBUTTON, Text="Hold down the CTRL key and
click me", pos=(250,50), size=(150,15)

OnCtl(n):
```

```
{
  int r;

  if (n==10 && e == FME_CLICKED)  {

    if (getAsyncKeyState(VK_CONTROL) < 0)
      Info ("The Control Key was just pressed.");
    else
      ErrorOk ("You didn't press the Control Key.");
  }

  return false;
}
```

See Also

getAsyncKeyStateF, VK codes

getAsyncKeyStateF

Syntax

```
int fm_getAsyncKeyStateF (int vkey)
```

Arguments

vkey
: Virtual Key Constant. For more information see the **[MSDN Virtual Key Codes page]**

Return

The return value specifies whether the key was pressed since the last call to a member of the getAsyncKeyState family of functions, and whether the key is currently up or down. If the most significant bit is set, the key is down, and if the least significant bit is set, the key was pressed after the previous call to GetAsyncKeyState.

getAsyncKeyStateF also checks if the window is currently active. If not, all keys will be seen as "up" (that is; not pressed).

Description

Determines whether a key is up or down at the time the function is called, and whether the key was pressed after a previous call to getAsyncKeyState.

Example

```
%ffp
```

```
ctl(10): PUSHBUTTON, Text="Hold down the CTRL key and
click me", Pos=(250, 50), Size=(150, 15)

OnCtl(n):
{
  int r;

  if (n == 10 && e == FME_CLICKED)  {

    if (getAsyncKeyStateF(VK_CONTROL) < 0)
      Info ("The Control Key was just pressed.");
    else
      ErrorOk ("You didn't press the Control Key.");

  }

  return false;
}
```

See Also

FME_KEYDOWN, **FME_KEYUP**, **getAsyncKeyState**, **VK codes**

getc

Syntax

```
int getc(int * fileptr)
```

Arguments

fileptr
Pointer to a file opened using **fopen**.

Return

The byte (unsigned) read from the file if successful, otherwise -1.

Description

Reads a single byte from the file referenced by **fileptr**.

This function is equivalent to **fgetc** but can be used in ways that can corrupt **fileptr**. It is advised to always use **fgetc** instead of this function.

See Also

fopen, **putc**, **fgetc**

getCpuReg

Syntax

`int getCpuReg(int nr)`

Arguments

nr
: The number of the specified CPU register to retrieve the value from, where 0 = edi, 1 = esi, 2 = ebp, 3 = esp, 4 = ebx, 5 = edx, 6 = ecx, 7 = eax, 8 = eip

Return

The value stored in that CPU register as an integer.

Description

Retrieves the value currently stored in a CPU register.

Example

```
%fml

ForEveryTile:
{
  Info("CPU Register edi = %d", getCpuReg(0));
  Info("CPU Register esi = %d", getCpuReg(1));
  Info("CPU Register ebp = %d", getCpuReg(2));
  Info("CPU Register esp = %d", getCpuReg(3));
  Info("CPU Register ebx = %d", getCpuReg(4));
  Info("CPU Register edx = %d", getCpuReg(5));
  Info("CPU Register ecx = %d", getCpuReg(6));
  Info("CPU Register eax = %d", getCpuReg(7));
  Info("CPU Register eip = %d", getCpuReg(8));
  return true;
}
```

See Also

finit, **fstsw**, **fstcw**, **fldcw**

getCtlClass

Requires FM 1.0 Beta 8.4 (Sep 2007) or newer

Syntax

```
int getCtlClass(int n)
```

Return

Returns the class of control number **n**. If the class returned is 0 (or constant CC_UNUSED) it does not exists (not created or deleted).

One of the following will be returned:

Constant name	*Value*
CC_ANIMATION	24
CC_BITMAP	20
CC_CHECKBOX	7
CC_COMBOBOX	12
CC_EDIT	15
CC_FRAME	17
CC_GROUPBOX	9
CC_ICON	21
CC_IMAGE	19
CC_LISTBOX	11
CC_OWNERDRAW	10
CC_PREVIEW	25
CC_PROGRESSBAR	14
CC_PUSHBUTTON	6

CC_RADIOBUTTON	8
CC_RECT	18
CC_SCROLLBAR	2
CC_SLIDER	13
CC_SPINNER	4
CC_STANDARD	1
CC_STATICTEXT	16
CC_TAB	27
CC_TOOLTIP	23
CC_TRACKBAR	3
CC_UNUSED	0
CC_UPDOWN	5
CC_ZOOM	26

Please note that not all of the above constants and control types are available yet.

getCtlColor

Syntax

```
int getCtlColor(int index)
```

Arguments

index
The index of the control that you want to get the color of.

Return

The current color of control c as an RGB triple.

Description

getCtlColor returns the current color of control c, as an RGB triple.

See Also

getCtlVal, **setCtlColor**

getCtlCoord

Requires FM 1.0 Beta 6 (May 2005) or newer

Syntax

```
int getCtlCoord(int index, int type)
```

Arguments

index
: The index of the control that you want to get the mouse coordinates for.

type
: **0** = Gets the x coordinate of the mouse pointer relative to this control (in pixels)
: **1** = Gets the y coordinate of the mouse pointer relative to this control (in pixels)

Return

Returns the x or y coordinate of the mouse cursor relative to the top left corner of the specified control, in pixels.

Returns -1 if the mouse pointer is outside the bounds of the control's window, or if there is any other error.

Description

Allows you to determine the coordinates (in pixels) of the current mouse cursor position over a control, for example when a button is clicked.

If **index** is equal to **CTL_PREVIEW**, the coordinates over the (primary) preview display are returned, adjusted relative to the top left corner of the proxy image (0,0). To get a coordinate relative to the full original image, multiply the proxy coordinate by **scaleFactor** (e.g., multiply by 4 at 25% zoom).

For other controls, the coordinates are relative to the top left corner of the control (e.g., for an **OWNERDRAW** control).

Example

```
// returns the y mouse coordinate
// relative to user control 10
y = getCtlCoord(10, 1);
```

Comments

- `getCtlCoord(CTL_PREVIEW, 0)` is equivalent to **getPreviewCoordX**().
- `getCtlCoord(CTL_PREVIEW, 1)` is equivalent to **getPreviewCoordY**().

See Also

getPreviewCoordX, **getPreviewCoordY**, **getCtlPos**

getCtlDivisor

Syntax

```
int getCtlDivisor(int n)
```

Arguments

n
: The index number of the user control whose divisor you want to retrieve

Return

The divisor value assigned to the user control.

Description

This function gets the divisor value for **STANDARD** and **SCROLLBAR** controls.

Example

```
int d;
d = getCtlDivisor(4);
d = getCtlDivisor(5);
```

See Also

setCtlDivisor

getCtlHandle

Syntax

`int* getCtlHandle(int n)`

Arguments

n
: The index of the user control you want the handle of.

Return

A handle to the requested control

Description

Gets the window handle (pointer) of the specific control. The handle can then be used with other Win32 API commands that require a window/control handle.

Example

```
// This code loads the user32.dll
// DLL included with Windows and
// uses it to hide and show the
// scrollbar control.

ctl[1]: STANDARD, Text="Example"

int* lib_user32;
int* functionPointer;
int* controlHandle;
int returnval;
```

```
// Load the DLL library
lib_user32 = loadLib("user32");
if (lib_user32) {

  // Get the function in the DLL
  functionPointer = getLibFn(lib_user32, "ShowWindow");
  if (functionPointer) {

    // Get the handle of the dialog
    controlHandle = getCtlHandle(1);

    // Call the function to hide the scrollbar
    msgBox(MB_OK, "Hiding...", "About to hide scrollbar");
    returnval = callLib(functionPointer, controlHandle,
SW_HIDE);

    // Call the function to show the dialog window
    msgBox(MB_OK, "Showing...", "About to show scrollbar
again");
    returnval = callLib(functionPointer, controlHandle,
SW_SHOW);

  }
  else msgBox(MB_OK, "Error", "Function wasn't loaded");

  // Free the library DLL
  freeLib(lib_user32);

}
else msgBox(MB_OK, "Error", "DLL was not loaded");
```

See Also

getDialogHandle

getCtlItemCount

Requires FM 1.0 Beta 9d (June 2008) or newer

Syntax

```
int getCtlItemCount (int n)
```

Arguments

n
: The index of the control for which you want to obtain the item count.

Return

Returns 0 if not a valid control index. Returns the number of list items in a **LISTBOX** or **COMBOBOX** control. Returns the number of tabs in a TAB control. Returns -1 for any other control.

Description

Usually you know the number of items or tabs in a control, but this function can be useful for helping you keep track of this information.

Example

```
// Get item count for list box LB1.
nItems = getCtlItemCount(nLB1);
```

See Also

getCtlItemText, **deleteCtlItem**, **deleteCtlItems**, **getCtlClass**

getCtlItemText

Syntax

```
int getCtlItemText(int n, int item, char* str)
```

Arguments

n
: The number of the control

item
: The number of the item to retrieve

str
: Pointer to a string where the result will be stored

Return

Returns false if the control number is out of range or not in use, or if LB_ERR was returned by the internal Win32 SendMessage calls. Returns true otherwise.

Description

Gets the text of a **COMBOBOX**, **LISTBOX** or **TAB** control item.

Comment

This function only works with LISTBOX, COMBOBOX and TAB controls. It will currently return true if used with other control types, even though the function didn't succeed.

Example

```
%fml

ctl[0]: COMBOBOX(VSCROLL),
        Text="Harry\nLarry\nBarry\n"
        "Gary\nCarrie\nSally",
        Val=0, Size=(*,50)
ctl[6]: STATICTEXT, Text=""

OnCtl(n): {

  // Display selected text value
  if (n == 0) {
    getCtlItemText(0, ctl(0), str0);
    setCtlText(6, str0);
  }

  return true;
}
```

See Also

deleteCtlItem, **deleteCtlItems**

getCtlPos

Requires FM 1.0 Beta 8.4 (Sep 2007) or newer

Syntax

```
int getCtlPos(int index, int type)
```

Arguments

index
: The index of the control that you want to get the position or size of.

type
: **0** = The horizontal co-ordinate of the user control's position (in DBUs)
1 = The vertical co-ordinate of the user control's position (in DBUs)
2 = The width of the user control (in DBUs)
3 = The height of the user control (in DBUs)

Return

The horizontal/vertical coordinates or the width/height of the control, depending on the value of the type parameter.

Description

Gets the position and width of a user control.

Don't forget that all measurements are set in DBUs (dialog box units).

Example

```
// returns the width of user control 10
w = getCtlPos(10, 2);
```

See Also

setCtlPos, getCtlColor, getCtlVal

getCtlRange

Requires FM 1.0 Beta 6 (May 2005) or newer

Syntax

```
int getCtlRange(int n, int type)
```

Arguments

n
  The number of the control whose range you wish to get.
type
  0 = The minimum value of the range of user control n.
  1 = The maximum value of the range of user control n.

Return

Either the minimum or maximum of the user control range as an integer.

Description

This function retrieves either the minimum or maximum value of the range of the user control with the index n.

Example

```
// Retrieve maximum range of
// user control number 0.
x = getCtlRange(0, 1);

// Retrieve minimum range of
// user control number 23.
x = getCtlRange(23, 0);
```

See Also

setCtlRange, setCtlDivisor

getCtlTab

Requires FM 1.0 Beta 9.1 (Feb 2009) or newer

Syntax

```
int getCtlTab(int n, int t)
```

Arguments

n
: The number of the user control you want to locate

t
: Set to **0** to retrieve the number of the TAB control the user control is assigned to, or to **1** to retrieve the tabsheet item number the control is assigned to.

Return

If **t** is 0, the function returns the tab sheet the control is on. If **t** is non-zero, it returns the tabsheet number of the tab the control has been assigned to. Returns -1 if the user control number is out of range.

Description

Returns the tab control or tab sheet that a user control has been assigned to.

Comment

Due to the design of this function, the tab control index number should be 1 or higher. If the tab is control #0, you won't be able to use getCtlTab to retrieve the sheet number the control is on.

Example

```
%fml
ctl[1]: TAB, Text="page0\npage1", Pos=(250, 5), Size=(200,
100)
ctl[2]: STANDARD, Text="Slider", Pos=(280,*)
ctl[8]: PUSHBUTTON, Text="Move slider to other tab sheet",
Pos=(250, 120), Size=(150,*)

OnCtl(n): {
  if (n==8 && e==FME_CLICKED) {
    // If not already on page1,
    // move slider to page1
    if (getCtlTab(2, 1) != 1) {
      setCtlTab(2, 1, 1);
    }
    else {
      // Move back to page0
      setCtlTab(2, 1, 0);
    }
  }
  return false;
}
```

See Also

setCtlTab, TAB

getCtlText

Requires FM 1.0 Beta 8.4 (Sep 2007) or newer

Syntax

```
int getCtlText(int ctlnum)
```

Arguments

ctlnum
: The number of the control whose text you would like to retrieve.

Return

Returns a pointer (actually a char*) to a string containing the text property of the given user control.

Description

Gets the text label / property of a user control with the index ctlnum. Note that not all user controls support text strings.

Example

```
strcpy(str1, getCtlText(CTL_OK));
msgBox(MB_OK, "The text label of CTL_OK is", str1);
```

See Also

getCtlVal, setCtlText

getCtlVal

Syntax

```
int getCtlVal(int n)
```

Arguments

n
: The index of the user control you want the value of.

Return

The value associated with the user control.

Description

This function retrieves the current value of the user control with the index n. Be aware of the range the user control can return. For example, a push button can only return the values 0 and 1.

Example

```
R,G,B: c+ctl(0) //adds the current user control value to
the current pixel color
```

See Also

ctl

getcwd

Syntax

```
string getcwd(string buffer, int maxlen)
```

Arguments

buffer
: A pointer to a string (character buffer) where the result should be stored.

maxlen
: The maximum length of the path that can be stored in the given character buffer.

Return

Returns a string (pointer to a character buffer) if successful, NULL if there was an error.

Description

Returns the current working directory.

See Also

mkdir, **chdir**, **rmdir**

getDialogHandle

Syntax

`int* getDialogHandle()`

Return

A handle to the main dialog window.

Description

Gets the window handle of the filter dialog.

Examples

```
// This code loads the user32.dll
// DLL included with Windows and
// uses it to hide and show the
// main filter window.

int* lib_user32;
int* functionPointer;
int* dialogHandle;
int returnval;

// Load the DLL library
lib_user32 = loadLib("user32");
if (lib_user32) {

  // Get the function in the DLL
  functionPointer = getLibFn(lib_user32, "ShowWindow");
  if (functionPointer) {
    // Get the handle of the dialog
    dialogHandle = getDialogHandle();

    // Call the function to hide the dialog window
    msgBox(MB_OK, "Hiding...", "About to hide main dialog
window");
    returnval = callLib(functionPointer, dialogHandle,
SW_HIDE);

    // Call the function to show the dialog window
    msgBox(MB_OK, "Showing...", "About to show dialog
window again");
    returnval = callLib(functionPointer, dialogHandle,
SW_SHOW);

  }
  else msgBox(MB_OK, "Error", "Function wasn't loaded");

  // Free the library DLL
  freeLib(lib_user32);

}
else msgBox(MB_OK, "Error", "DLL was not loaded");
```

See Also

getCtlHandle

getDialogHeight

Syntax

```
getDialogHeight()
```

Description

Gets the current height of the dialog in vertical DBUs (VDBU).

Example

```
Info("DialogSize: %d x %d", getDialogWidth(),
getDialogHeight());
```

See Also

getDialogWidth, **setDialogMinMax**, **setDialogPos**, **VDBUsToPixels**

getDialogPos

Syntax

```
int getDialogPos(int w, int t)
```

Arguments

w
: A switch to choose which value to return: **0** for the left / x co-ordinate, **1** for the top / y co-ordinate, **2** for the window width and **3** for the window height.

t
: A Boolean flag indicating whether x and y are absolute screen coordinates (t == true), or whether they are relative to the client area (t == false).

Return

The requested window dimension as an integer.

Description

Gets the position and size of the filter dialog window. If **t** is true, x and y are absolute screen coordinates; otherwise, x and y are relative to the upper-left corner of the client area in the host application's main window.

All measurements are in dialog box units (DBUs).

Examples

```
ctl[0]: PUSHBUTTON, "Where are we now?", Size=(90,*)
ctl[8]: STATICTEXT, "", Size=(140, *)
```

```
OnCtl(n): {
  if (n==0) {
    int leftx, topy, width, height;
    leftx = getDialogPos(0, 0);
    topy = getDialogPos(1, 0);
    width = getDialogPos(2, 0);
    height = getDialogPos(3, 0);
    sprintf(str1, "X: %d, Y: %d, Width: %d, Height: %d",
leftx, topy, width, height);
    setCtlText(8, str1);
  }
  return true;
}
```

See Also

getDialogWidth, getDialogHeight, setDialogPos

getDialogWidth

Syntax

`getDialogWidth()`

Description

Gets the current width of the dialog in horizontal DBUs (HDBU).

Example

```
Info("DialogSize: %d x %d", getDialogWidth(),
getDialogHeight());
```

See Also

getDialogHeight, **setDialogMinMax**, **setDialogPos**,
HDBUsToPixels

getDisplaySettings

Syntax

```
int getDisplaySettings(int s)
```

Arguments

s
: Set to 0 for the bit depth, to 1 for the horizontal resolution in pixels, to 2 for the vertical resolution in pixels and to 3 for the refresh rate in Hz of the screen.

Return

Returns a value according to the used *s* parameter.

Description

Lets you know the bit depth, resolution and refresh rate of the screen. Especially the refresh rate is useful if you want to display an animation in the preview with the help of **setTimerEvent** and **updatePreview**. Because if you draw more frames per second than the refresh rate is able to display, your animation will flicker.

Example

```
%ffp

ForEveryTile:
{

  Info (" %d bit \n %d x %d pixel \n %d Hz",
getDisplaySettings(0), getDisplaySettings(1),
getDisplaySettings(2), getDisplaySettings(3));

  return true;
}
```

See Also

setTimerEvent, **updatePreview**

getImageTitle

Syntax

```
bool getImageTitle(char text[256])
```

Arguments

text
A string to contain the image title text. Must be at least 256 chars long.

Return

Returns true if successful, else false.

Description

This function retrieves the image title text property.

Comment

Note that not all graphics program implement this feature reliably. Adobe Photoshop® is the only program where you should assume this feature is available.

Example

```
getImageTitle(str0);
```

getLibFn

Requires FM 1.0 Beta 9.0 (April 2008) or newer

Syntax

```
int getLibFn(void *dllHandle, char *functionName)
```

Arguments

dllHandle
: A handle (obtained with **loadLib**) to the DLL the function is in.

functionName
: The name of the function to get a pointer to.

Return

Returns a pointer to the DLL function.

Description

Locates a specific function in a DLL, allowing you to call the function from within FilterMeister by using **callLib**.

Behind the scenes, this function is just a wrapper around the **[GetProcAddress]** Win32 API function.

Example

```
// This code loads the user32.dll
// DLL included with Windows and
// uses it to display a YES/NO
// Message Box.
```

```

int* lib_user32;
int* functionPointer;
int returnval;

// Load the DLL library
lib_user32 = loadLib("user32");
if (lib_user32) {

  // Get the function in the DLL
  functionPointer = getLibFn(lib_user32, "MessageBoxA");
  if (functionPointer) {

    // Call the function
    strcpy(str0, "The window text is here");
    strcpy(str1, "Caption Text");
    returnval = callLib(functionPointer, NULL, str0, str1,
MB_YESNO);

    // Process return value
    if (returnval == IDYES)
      msgBox(MB_OK, "Yes!", "Yes was clicked");
    if (returnval == IDNO)
      msgBox(MB_OK, "No :(", "No was clicked");

  }
  else msgBox(MB_OK, "Error", "Function wasn't loaded");

  // Free the library DLL
  freeLib(lib_user32);

}
else msgBox(MB_OK, "Error", "DLL was not loaded");
```

See Also

loadLib, **callLib**, **freeLib**

getLocaleInfo

Syntax

```
int getLocaleInfo (int locale, int type, char * buffer, int buffer_size)
```

Arguments

locale
: The locale identifier **LCID** of the locale or one of the constants LOCALE_SYSTEM_DEFAULT or LOCALE_USER_DEFAULT to specify either the system or user's default locale.

type
: An identifier for the type of information required. See below for a list of constants for this argument.

buffer
: Pointer to a character array (string) used to store the requested information.

buffer_size
: The maximum size of the buffer, this value should be the size of the allocated string or 255 in case you're using one of the predeclared strings (str0 up to str9).

Return

A value indicating the success (6) or a possible error which occurred:

- 0 = Insufficient buffer
- 5 = Invalid type
- 6 = Success
- 7 = Invalid locale identifier

Description

Retrieves information about a locale specified by identifier.

Internally, this function is just a wrapper around the [GetLocaleInfo] Win32 API function.

Type constants

The following is a list constants for all existing types of locale info.

LOCALE_ILANGUAGE
: Language identifier indicating the language. The maximum number of characters allowed for this string is 5.

LOCALE_SLANGUAGE
: Full localized name of the language.

LOCALE_SENGLANGUAGE
: Full English name of the language from the International Organization for Standardization (ISO) Standard 639. This is always restricted to characters mappable into the ASCII 127-character subset.

LOCALE_SABBREVLANGNAME
: Abbreviated name of the language, created by taking the 2-letter language abbreviation from the ISO Standard 639 and adding a third letter, as appropriate, to indicate the sublanguage.

LOCALE_SNATIVELANGNAME
: Native name of the language.

LOCALE_ICOUNTRY
: Country code, based on international phone codes, also referred to as IBM country codes. The maximum number of characters allowed for this string is 6.

LOCALE_SCOUNTRY
: Full localized name of the country.

LOCALE_SENGCOUNTRY

Full English name of the country. This is always restricted to characters mappable into the ASCII 127-character subset.

LOCALE_SABBREVCTRYNAME

Abbreviated name of the country from the ISO Standard 3166.

LOCALE_SNATIVECTRYNAME

Native name of the country.

LOCALE_IDEFAULTLANGUAGE

Language identifier for the principal language spoken in this locale. This is provided so that partially specified locales can be completed with default values. The maximum number of characters allowed for this string is 5.

LOCALE_IDEFAULTCOUNTRY

Country code for the principal country in this locale. This is provided so that partially specified locales can be completed with default values. The maximum number of characters allowed for this string is 6.

LOCALE_IDEFAULTCODEPAGE

Original equipment manufacturer (OEM) code page associated with the country. The maximum number of characters allowed for this string is 6.

LOCALE_SLIST

Character(s) used to separate list items. For example, a comma is used in many locales.

LOCALE_IMEASURE

System of measurement. This value is 0 if the metric system (Syst©me International d'Unit©s, or S.I.) is used and 1 if the U.S. system is used. The maximum number of characters allowed for this string is 2.

LOCALE_SDECIMAL

Character(s) used as the decimal separator.

LOCALE_STHOUSAND

Character(s) used to separate groups of digits to the left of the decimal.

LOCALE_SGROUPING

Sizes for each group of digits to the left of the decimal. An explicit size is needed for each group; sizes are separated by semicolons. If the last value is zero, the preceding value is repeated. To group thousands, specify 3;0, for example.

LOCALE_IDIGITS

Number of fractional digits. The maximum number of characters allowed for this string is 3.

LOCALE_ILZERO

Specifier for leading zeros in decimal fields. The maximum number of characters allowed for this string is 2. The specifier can be one of the following values: 0 = No leading zeros, 1 = Leading zeros

LOCALE_SNATIVEDIGITS

Native equivalents to ASCII 0 through 9.

LOCALE_SCURRENCY

String used as the local monetary symbol.

LOCALE_SINTLSYMBOL

Three characters of the international monetary symbol specified in ISO 4217, "Codes for the Representation of Currencies and Funds," followed by the character separating this string from the amount.

LOCALE_SMONDECIMALSEP

Character(s) used as the monetary decimal separator.

LOCALE_SMONTHOUSANDSEP

Character(s) used as the monetary separator between groups of digits to the left of the decimal.

LOCALE_SMONGROUPING

Sizes for each group of monetary digits to the left of the decimal. An explicit size is needed for each group; sizes are separated by semicolons. If the last value is zero, the preceding value is repeated. To group thousands, specify 3;0, for example.

LOCALE_ICURRDIGITS

Number of fractional digits for the local monetary format. The maximum number of characters allowed for this string is 3.

LOCALE_IINTLCURRDIGITS
Number of fractional digits for the international monetary format. The maximum number of characters allowed for this string is 3.

LOCALE_ICURRENCY
Positive currency mode. The maximum number of characters allowed for this string is 2. The mode can be one of the following values: 0 = Prefix, no separation, 1 = Suffix, no separation, 2 = Prefix, 1-char. separation, 3 = Suffix, 1-char. separation

LOCALE_INEGCURR
Negative currency mode. The maximum number of characters allowed for this string is 3. The mode can be one of the following values (right side is an example): 0 = ($1.1), 1 = -$1.1, 2 = $-1.1, 3 = $1.1-, 4 = (1.1$), 5 = -1.1$, 6 = 1.1-$, 7 = 1.1$-, 8 = -1.1 $ (space before $), 9 = -$ 1.1 (space after $), 10 = 1.1 $- (space before $), 11 = $ 1.1- (space after $), 12 = $ -1.1 (space after $), 13 = 1.1- $ (space before $), 14 = ($ 1.1) (space after $), 15 = (1.1 $) (space before $)

LOCALE_SDATE
Character(s) for the date separator.

LOCALE_STIME
Character(s) for the time separator.

LOCALE_STIMEFORMAT
Time formatting strings for this locale.

LOCALE_SSHORTDATE
Short date formatting string for this locale.

LOCALE_SLONGDATE
Long date formatting string for this locale.

LOCALE_IDATE
Short date format-ordering specifier. The maximum number of characters allowed for this string is 2. The specifier can be one of the following values: 0 = Month-Day-Year, 1 = Day-Month-Year, 2 = Year-Month-Day

LOCALE_ILDATE

Long date format-ordering specifier. The maximum number of characters allowed for this string is 2. The specifier can be one of the following values: 0 = Month-Day-Year, 1 = Day-Month-Year, 2 = Year-Month-Day

LOCALE_ITIME

Time format specifier. The maximum number of characters allowed for this string is 2. The specifier can be one of the following values: 0 = AM / PM 12-hour format, 1 = 24-hour format

LOCALE_ICENTURY

Specifier for full 4-digit century. The maximum number of characters allowed for this string is 2. The specifier can be one of the following values: 0 = Abbreviated 2-digit century, 1 = Full 4-digit century

LOCALE_ITLZERO

Specifier for leading zeros in time fields. The maximum number of characters allowed for this string is 2. The specifier can be one of the following values: 0 = No leading zeros for hours, 1 = Leading zeros for hours

LOCALE_IDAYLZERO

Specifier for leading zeros in day fields. The maximum number of characters allowed for this string is 2. The specifier can be one of the following values: 0 = No leading zeros for days, 1 = Leading zeros for days

LOCALE_IMONLZERO

Specifier for leading zeros in month fields. The maximum number of characters allowed for this string is 2. The specifier can be one of the following values: 0 = No leading zeros for months, 1 = Leading zeros for months

LOCALE_S1159

String for the AM designator.

LOCALE_S2359

String for the PM designator.

LOCALE_SDAYNAME1

Native long name for Monday.

LOCALE_SDAYNAME2

Native long name for Tuesday.

LOCALE_SDAYNAME3
Native long name for Wednesday.

LOCALE_SDAYNAME4
Native long name for Thursday.

LOCALE_SDAYNAME5
Native long name for Friday.

LOCALE_SDAYNAME6
Native long name for Saturday.

LOCALE_SDAYNAME7
Native long name for Sunday.

LOCALE_SABBREVDAYNAME1
Native abbreviated name for Monday.

LOCALE_SABBREVDAYNAME2
Native abbreviated name for Tuesday.

LOCALE_SABBREVDAYNAME3
Native abbreviated name for Wednesday.

LOCALE_SABBREVDAYNAME4
Native abbreviated name for Thursday.

LOCALE_SABBREVDAYNAME5
Native abbreviated name for Friday.

LOCALE_SABBREVDAYNAME6
Native abbreviated name for Saturday.

LOCALE_SABBREVDAYNAME7
Native abbreviated name for Sunday.

LOCALE_SMONTHNAME1
Native long name for January.

LOCALE_SMONTHNAME2
Native long name for February.

LOCALE_SMONTHNAME3
Native long name for March.

LOCALE_SMONTHNAME4
Native long name for April.

LOCALE_SMONTHNAME5
Native long name for May.

LOCALE_SMONTHNAME6

Native long name for June.
LOCALE_SMONTHNAME7
Native long name for July.
LOCALE_SMONTHNAME8
Native long name for August.
LOCALE_SMONTHNAME9
Native long name for September.
LOCALE_SMONTHNAME10
Native long name for October.
LOCALE_SMONTHNAME11
Native long name for November.
LOCALE_SMONTHNAME12
Native long name for December.
LOCALE_SABBREVMONTHNAME1
Native abbreviated name for January.
LOCALE_SABBREVMONTHNAME2
Native abbreviated name for February.
LOCALE_SABBREVMONTHNAME3
Native abbreviated name for March.
LOCALE_SABBREVMONTHNAME4
Native abbreviated name for April.
LOCALE_SABBREVMONTHNAME5
Native abbreviated name for May.
LOCALE_SABBREVMONTHNAME6
Native abbreviated name for June.
LOCALE_SABBREVMONTHNAME7
Native abbreviated name for July.
LOCALE_SABBREVMONTHNAME8
Native abbreviated name for August.
LOCALE_SABBREVMONTHNAME9
Native abbreviated name for September.
LOCALE_SABBREVMONTHNAME10
Native abbreviated name for October.
LOCALE_SABBREVMONTHNAME11
Native abbreviated name for November.
LOCALE_SABBREVMONTHNAME12

Native abbreviated name for December.

LOCALE_SPOSITIVESIGN

String value for the positive sign.

LOCALE_SNEGATIVESIGN

String value for the negative sign.

LOCALE_IPOSSIGNPOSN

Formatting index for positive values. The maximum number of characters allowed for this string is 2. The index can be one of the following values: 0 = Parentheses surround the amount and the monetary symbol, 1 = The sign string precedes the amount and the monetary symbol, 2 = The sign string succeeds the amount and the monetary symbol, 3 = The sign string immediately precedes the monetary symbol, 4 = The sign string immediately succeeds the monetary symbol.

LOCALE_INEGSIGNPOSN

Formatting index for negative values. This index uses the same values as LOCALE_IPOSSIGNPOSN. The maximum number of characters allowed for this string is 2.

LOCALE_IPOSSYMPRECEDES

Position of monetary symbol in a positive monetary value. This value is 1 if the monetary symbol precedes the positive amount, 0 if it follows it. The maximum number of characters allowed for this string is 2.

LOCALE_IPOSSEPBYSPACE

Separation of monetary symbol in a positive monetary value. This value is 1 if the monetary symbol is separated by a space from a positive amount, 0 if it is not. The maximum number of characters allowed for this string is 2.

LOCALE_INEGSYMPRECEDES

Position of monetary symbol in a negative monetary value. This value is 1 if the monetary symbol precedes the negative amount, 0 if it follows it. The maximum number of characters allowed for this string is 2.

LOCALE_INEGSEPBYSPACE

Separation of monetary symbol in a negative monetary value. This value is 1 if the monetary symbol is separated by a space from the negative amount, 0 if it is not. The maximum number of characters allowed for this string is 2.

LOCALE_NOUSEROVERRIDE

This constant may be OR'ed with any other LCTYPE constant in a call to the GetLocaleInfo function. This always causes the function to bypass any user overrides, and return the system default value for the other LCTYPE specified in the function call, based on the given LCID.

getOpenFileName

Syntax

```
int getOpenFileName(int flags, string *file, int maxFile,
int *fileOffset, int *extensionOffset, string *fileTitle, int
nMaxFileTitle, string *filter, string *customFilter, int
maxCustFilter, int *filterIndex, string initialDir, string
dialogTitle, string defExt, int *outputflags)
```

Arguments

flags
: A set of bit flags that can be chosen to modify the behavior of the dialog. Available flags include OFN_HIDEREADONLY to hide the user-selectable "Read Only" option in the dialog, and OFN_FORCESHOWHIDDEN to force hidden files to be shown to the user. A full list of the available bit flags is in the **[OPENFILENAME structure]** Win32 documentation.

file
: A string containing the prefilled value for the File Name edit control. This string will be overwritten with the name of the file that the user has chosen on return of the function.

maxFile
: The maximum length of the string buffer *file* that contains the initial File Name

fileOffset
: The offset in bytes to the beginning of the actual filename (after the path) in the string *file*.

extensionOffset
: The offset in bytes to the beginning of the filename extension (after the dot) in the string *file*.

fileTitle

The filename and file extension (without path) of the file the user has chosen. Can be set to NULL to ignore.

maxFileTitle

The maximum length of the string buffer *fileTitle* that receives the filename & file extension the user chose.

filter

A string buffer containing pairs of null terminated strings, defining file extension filters the user can choose from to narrow their choices, together with a caption for each filter set. These typically show up next to the File Name edit field in the Open File dialog window.

customFilter

A string buffer to store a file extension filter the user might have typed in manually

maxCustFilter

The maximum length of the string buffer *customFilter*, storing the user's own custom entered extension filter.

filterIndex

The number of the file extension filter the user has chosen from the provided list in *filter*. Numbering starts from 1. If 0, the user has entered a custom filter of their own.

initialDir

A string containing the path of the initial directory to be displayed in the dialog. Can be set to NULL, in which case the current directory will likely be used. For the full rules and behavior of which folder will be chosen when the dialog is invoked, see the Win32 documentation for the **[OPENFILENAME structure].**

dialogTitle

A string containing the caption to be used for the Open File dialog window

defExt

A string containing the default file extension for the user to choose from (not containing periods or asterisks)

outputflags

An integer DWORD containing the flags as may have been modified by the Open File dialog after the user interacted with it.

Return

Returns 0 if the user specified a file name and clicked the OK button, otherwise it returns an error code as returned by the Win32 **[CommDlgExtendedError]** API function.

Description

This function displays the Windows "Open File" common dialog to the user, letting them choose a file to open.

Internally, this function is mostly a wrapper around the Win32 API function **[GetOpenFileName]**.

Example

```
%fml

// The following will ask the user to choose a
// TXT file that will be loaded and displayed
// on the filter interface

ctl[3]: STATICTEXT, Text="Test", Pos=(250,20), Size=
(200,130)

void *TEXT_FILE;
int status;

status = getOpenFileName(
         OFN_HIDEREADONLY,   //flags
         str0, 256,          //set, receive file name
         NULL, NULL,         //file/ext offsets
         NULL, 0,            //file title for display
                             //file filters...
         "Text Files (*.txt;*.md)\0*.txt;*.md\0"
         "All Files (*.*)\0*.*\0",
         NULL, 0,            //no custom filter
         NULL,               //filter index
         filterInstallDir,   //initial dir
         "Open Text File",   //dialog title
         "txt",              //def. extension
         NULL                //output flags
         );

if (status == 0) {
  // User has clicked on OK and chosen a file

  // Attempt to open file
  TEXT_FILE = fopen(str0, "rb");
  if (TEXT_FILE) {

    // Get the filesize
    unsigned int filesize;
    fseek(TEXT_FILE, 0L, SEEK_END);
    filesize = ftell(TEXT_FILE);
    fseek(TEXT_FILE, 0, SEEK_SET);

    // Allocate memory block as big as file
    void* textmemblock = calloc(filesize, 1);

    // Read entire text file into memory
    if (textmemblock) {
      fread(textmemblock, 1, filesize, TEXT_FILE);
      setCtlText(3, textmemblock);
    }

    // Close the file
    fclose(TEXT_FILE);
  }
  else msgBox(MB_OK | MB_ICONWARNING, "Error", "Can't find
that file.");
}
```

See Also

getSaveFileName

getPixel

Syntax

```
int getPixel(int x, int y)
```

Arguments

x
: X-coordinate (in pixels) of the relative position (originating top-left at 0, 0) in the control where you want to retrieve a pixel color from.

y
: Y-coordinate (in pixels) of the relative position (originating top-left at 0, 0) in the control where you want to retrieve a pixel color from.

Returns

The 8-bit **RGB** color value of the pixel at the given co-ordinates as a single 32-bit integer.

Description

Call getPixel to retrieve an individual pixel color from an **OWNERDRAW** control. First call **startSetPixel**, then use getPixel, setPixel and any of the other Control drawing functions to draw to the control. Finish with a call to **endSetPixel**. This function is not for getting pixels from the preview / final image; use **src** or **pget** for that instead.

Example

```
%fml
ctl[0]: OWNERDRAW, Color=RGB(0,0,0), Size=(100,100)

OnFilterStart: {

  // Get the size of the canvas
  int width, height;
  startSetPixel(0);
  width = getSetPixelWidth();
  height = getSetPixelHeight();

  // Draw purple line at y = 100
  for (int xp = 0; xp < width; xp++) {
    setPixel(xp, 100, RGB(192,0,255));
  }

  // Duplicate at y = 200 line at y = 100
  for (int xp = 0; xp < width; xp++) {
    setPixel(xp, 200, getPixel(xp, 100));
  }
  endSetPixel(0);

  return false;
}
```

See Also

startSetPixel, **setPixel**, **endSetPixel**

getPreviewCursor

Requires FM 1.0 Beta 8.4 (Sep 2007) or newer

Syntax

```
int getPreviewCursor(void)
```

Return

The integer resource number of the cursor being used.

Description

Retrieves the pointer icon resource being used when the mouse is over the preview image. Typical values are

```
103  : Default Windows pointer (hand)
IDC_ARROW    (32512) : Arrow pointer
IDC_IBEAM    (32513) : I beam (text cursor)
IDC_WAIT     (32514) : Busy pointer
IDC_CROSS    (32515) : Cross
IDC_UPARROW  (32516) : Up arrow
IDC_SIZE     (32640) : Size pointer
IDC_HELP     (32641) : pointer with question mark
IDC_SIZENWSE (32642) : NWSE diagonal arrow
IDC_SIZENESW (32643) : NESW diagonal arrow
IDC_SIZEWE   (32644) : WE arrow
IDC_SIZENS   (32645) : NS arrow
IDC_SIZEALL  (32646) : Size all crossed-arrows
IDC_NO       (32648) : NO pointer
IDC_APPSTARTING (32650) : Application Starting pointer
IDC_ICON     (32651) : ? custom cursor?
```

Comments

Bear in mind that when using these cursors, cursors may differ for different machines depending on the user's cursor preferences and any themes running.

Example

```
OnFilterStart: {
  // Displays preview cursor icon
  // resource number. This will
  // be 103 by default for the
  // hand icon
  Info("Preview icon resource number: %d",
getPreviewCursor());
  return false;
}
```

See Also

[setPreviewCursor]

getPreviewCoordX

Syntax

`int getPreviewCoordX (void)`

Return

Returns the x coordinate in the image above which the mouse pointer is placed.

Description

Together with **getPreviewCoordY** you can use it to get the coordinates of the mouse pointer above the image in the preview. The returned coordinates are already converted to image coordinates. But it is recommended to multiply the returned value with scaleFactor to get a coordinate relative to the full image and not just a coordinate relative to the e.g. 33% zoomed version of the image.

Example

```
%ffp

ctl(0): STATICTEXT, "Please right click on the preview to
set a cross.", Size=(100,20)

OnCtl(n):
{

  if (n == CTL_PREVIEW && e == FME_RIGHTCLICKED_DOWN) {
    j0 = getPreviewCoordX() * scaleFactor;
    j1 = getPreviewCoordY() * scaleFactor;
    doAction(CA_PREVIEW );
  }

  return false;
}


ForEveryTile:{

  int g,h,z, color;
  int PreviewX = j0/scaleFactor;
  int PreviewY = j1/scaleFactor;

  // Calculate color of the cross
  color = ( src(PreviewX, PreviewY, 0) + src(PreviewX,
PreviewY, 1)+ src(PreviewX, PreviewY, 2) ) / 3;
  if (color > 128 && color < 196) color=0;
  if (color > 64 && color < 128) color=255;
  else color = 255 - color;

  // Display Cross
  if (doingProxy){
    for (z = 0; z < Z; z++){
      for (g = -7; g < 8; g++)
        if (g < -1 || g > 1) pset(PreviewX + g, PreviewY,
z, color);
      for (h = -7; h < 8; h++)
        if (h < -1 || h > 1) pset(PreviewX, PreviewY + h,
z, color );
    }
  }

  return true;
}
```

See Also

getPreviewCoordY, scaleFactor

getPreviewCoordY

Syntax

```
int getPreviewCoordY (void)
```

Return

Returns the y coordinate in the image above which the mouse pointer is placed.

Description

Together with **getPreviewCoordX** you can use it to get the coordinates of the mouse pointer above the image in the preview. The returned coordinates are already converted to image coordinates. But it is recommended to multiply the returned value with scaleFactor to get a coordinate relative to the full image and not just a coordinate relative to the e.g. 33% zoomed version of the image.

Example

```
%ffp

ctl(0): STATICTEXT, "Please right click on the preview to
set a cross.", Size=(100,20)

OnCtl(n):
{

  if (n == CTL_PREVIEW && e == FME_RIGHTCLICKED_DOWN) {
    j0 = getPreviewCoordX() * scaleFactor;
    j1 = getPreviewCoordY() * scaleFactor;
    doAction(CA_PREVIEW);
  }

  return false;
}


ForEveryTile:{

  int g, h, z, color;
  int PreviewX = j0 / scaleFactor;
  int PreviewY = j1 / scaleFactor;

  // Calculate color of the cross
  color = ( src(PreviewX, PreviewY, 0) + src(PreviewX,
PreviewY, 1) + src(PreviewX, PreviewY, 2) )/3;
  if (color > 128 && color < 196) color = 0;
  if (color > 64 && color < 128) color = 255;
  else color = 255-color;

  // Display Cross
  if (doingProxy){
    for (z = 0; z < Z; z++){
      for (g = -7; g < 8; g++)
        if (g < -1 || g > 1) pset(PreviewX + g, PreviewY,
z, color);
      for (h = -7; h < 8; h++)
        if (h < -1 || h > 1) pset(PreviewX, PreviewY + h,
z, color );
    }
  }

  return true;
}
```

See Also

getPreviewCoordX, scaleFactor

getRegPath

Syntax

```
int getRegPath(char * path, int length)
```

Arguments

path
A pointer to the string which will contain the path.

length
Maximum length of the **path** string.

Return

Returns ERROR_SUCCESS if the operation was successful, otherwise it returns one of the following integer error codes:

ERROR_SUCCESS	(==0) no error
ERROR_FILE_NOT_FOUND	key or value name not found
ERROR_MORE_DATA	buffer wasn't big enough (e.g., getRegString, getRegData)
ERROR_NO_MORE_ITEMS	index >= # of values or subkeys (enumRegValue, enumRegSubKey)
ERROR_INVALID_FUNCTION	bad top-level key, etc
ERROR_INVALID_DATA	wrong data type or size (or size > 2048)
ERROR_BADDB	registry database is corrupt
ERROR_BADKEY	registry key is invalid
ERROR_CANTOPEN	registry key could not be opened

ERROR_CANTREAD	registry key could not be read
ERROR_CANTWRITE	registry key could not be written
ERROR_REGISTRY_CORRUPT	registry is corrupt
ERROR_REGISTRY_IO_FAILED	input/output to registry failed
ERROR_KEY_DELETED	Illegal operation attempted on a Registry key which has been marked for deletion.
ERROR_KEY_HAS_CHILDREN	cannot delete a key with subkeys (Windows NT)

Description

Gets the current registry path.

The **length** should be at least of size MAX_PATH + 1 in order to fit any possible registry path. The string **path** must be large enough to hold **length** number of characters.

Example

```
// Display the current registry path
getRegPath(&str0, 256);
Info("Current registry path: %s", str0);
```

See Also

setRegPath, **getRegRoot**

getRegInt

Syntax

```
int getRegInt(int *iValue, char *szValueName, ...)
```

Arguments

iValue
: A pointer to an integer which will contain the integer from the registry.

szValueName
: The key name of the integer to retrieve from the registry

Return

Returns ERROR_SUCCESS if the operation was successful, otherwise it returns one of the following integer error codes:

ERROR_SUCCESS	(==0) no error
ERROR_FILE_NOT_FOUND	key or value name not found
ERROR_MORE_DATA	buffer wasn't big enough (e.g., getRegString, getRegData)
ERROR_NO_MORE_ITEMS	index >= # of values or subkeys (enumRegValue, enumRegSubKey)
ERROR_INVALID_FUNCTION	bad top-level key, etc
ERROR_INVALID_DATA	wrong data type or size (or size > 2048)
ERROR_BADDB	registry database is corrupt
ERROR_BADKEY	registry key is invalid
ERROR_CANTOPEN	registry key could not be

	opened
ERROR_CANTREAD	registry key could not be read
ERROR_CANTWRITE	registry key could not be written
ERROR_REGISTRY_CORRUPT	registry is corrupt
ERROR_REGISTRY_IO_FAILED	input/output to registry failed
ERROR_KEY_DELETED	Illegal operation attempted on a Registry key which has been marked for deletion.
ERROR_KEY_HAS_CHILDREN	cannot delete a key with subkeys (Windows NT)

Description

Gets an integer value from the current registry path.

Example

```
// Display the current registry path
getRegPath(&str0, 256);
Info("Current registry path: %s", str0);
```

See Also

setRegPath, getRegRoot

getRegRoot

Syntax

```
int getRegRoot(int * hkey)
```

Arguments

hkey
Pointer to an integer value which will hold the root identification.

Return

Returns ERROR_SUCCESS if the operation was successful, otherwise it returns one of the following integer error codes:

ERROR_SUCCESS	(==0) no error
ERROR_FILE_NOT_FOUND	key or value name not found
ERROR_MORE_DATA	buffer wasn't big enough (e.g., getRegString, getRegData)
ERROR_NO_MORE_ITEMS	index >= # of values or subkeys (enumRegValue, enumRegSubKey)
ERROR_INVALID_FUNCTION	bad top-level key, etc
ERROR_INVALID_DATA	wrong data type or size (or size > 2048)
ERROR_BADDB	registry database is corrupt
ERROR_BADKEY	registry key is invalid
ERROR_CANTOPEN	registry key could not be opened

ERROR_CANTREAD	registry key could not be read
ERROR_CANTWRITE	registry key could not be written
ERROR_REGISTRY_CORRUPT	registry is corrupt
ERROR_REGISTRY_IO_FAILED	input/output to registry failed
ERROR_KEY_DELETED	Illegal operation attempted on a Registry key which has been marked for deletion.
ERROR_KEY_HAS_CHILDREN	cannot delete a key with subkeys (Windows NT)

Description

Gets the current registry root.

Currently, only two types of registry roots are possible, identified by the constants HKEY_LOCAL_MACHINE and HKEY_CURRENT_USER.

Example

```
// Check to see if we're working with HKEY_LOCAL_MACHINE
int * root;
getRegRoot(&root);
Info("Working with local machine: %s", (root ==
HKEY_LOCAL_MACHINE? "true" : "false"));
```

See Also

setRegRoot, getRegPath

getRegString

Syntax

```
int getRegString(int szString, int iMaxLen, int
szValueName[, varargs]...)
```

Arguments

szString
: is the address of the string to be stored or retrieved

iMaxLen
: is the size of the szString buffer in bytes, which specifies the longest character string that can be stored in szString, including the terminating null character

szValueName
: is the name of the string to be set or retrieved, and may contain printf-style formatting codes as well as FM !-codes.

varargs
: is a list of optional arguments used to perform printf-style formatting on the szValueName string.

Return

Returns ERROR_SUCCESS if the operation was successful, otherwise it returns one of the following integer error codes:

ERROR_SUCCESS	(==0) no error
ERROR_FILE_NOT_FOUND	key or value name not found
ERROR_MORE_DATA	buffer wasn't big enough (e.g., getRegString, getRegData)
ERROR_NO_MORE_ITEMS	index >= # of values or subkeys (enumRegValue,

	enumRegSubKey)
ERROR_INVALID_FUNCTION	bad top-level key, etc
ERROR_INVALID_DATA	wrong data type or size (or size > 2048)
ERROR_BADDB	registry database is corrupt
ERROR_BADKEY	registry key is invalid
ERROR_CANTOPEN	registry key could not be opened
ERROR_CANTREAD	registry key could not be read
ERROR_CANTWRITE	registry key could not be written
ERROR_REGISTRY_CORRUPT	registry is corrupt
ERROR_REGISTRY_IO_FAILED	input/output to registry failed
ERROR_KEY_DELETED	Illegal operation attempted on a Registry key which has been marked for deletion.
ERROR_KEY_HAS_CHILDREN	cannot delete a key with subkeys (Windows NT)

Description

Fetches a C-style character string from the Windows Registry.

Currently, only two types of registry roots are possible, identified by the constants HKEY_LOCAL_MACHINE and HKEY_CURRENT_USER.

Comment

Note that this function does not work for REG_EXPAND_SZ strings, which are a different data type to standard REG_SZ strings.

Example

```
// Saves a title for your filter dialog box, and retrieves
it on next invocation
putRegString("Filter #2", "Dialog title");  //save title
for next time
        . . .

getRegString(str0, 256, "Dialog title");    //retrieve
saved title
if (strcmp(str0, "")==0) {
    //title is null or missing...
    strcpy(str0, "Default title");  //set a default title
}
setDialogText(str0);   //set title for our filter dialog
```

See Also

getRegRoot, setRegPath

getResSize

Requires FM 1.0 Beta 8.5 (Nov 2007) or newer

Syntax

```
int getResSize(string restype, string resname)
```

Arguments

restype
The type of the resource. This is either a string (probably "FMDATA" in most cases), or an integer if the resource is a built-in resource type.

resname
The name of the resource

Return

Returns 0 if the resource can't be found or an error occurs, otherwise returns the size of the resource in bytes.

Description

Gets the size of a file that has been embedded as a resource in the plugin DLL.

Some useful resource type values:

1 - RT_CURSOR	Cursor resource
2 - RT_BITMAP	Bitmap resource
3 - RT_ICON	Hardware-dependent icon resource
4 - RT_MENU	Menu resource
5 - RT_DIALOG	Dialog resource

6 - RT_STRING	String-table entry
8 - RT_FONT	Font resource
16 - RT_VERSION	Version resource
23 - RT_HTML	HTML Resource

Example

```
%fml

OnFilterStart: {

  int ressize = 0;

  // Get the size of the built-in FM logo BMP
  // RT_BITMAP Bitmaps are resource type 2
  ressize = getResSize(2, "LOGO2.BMP");

  // Display the size of the resource file
  Info("Size of resource: %d bytes", ressize);

  return true;
}
```

See Also

getResAddress, copyResToArray

getResAddress

Requires FM 1.0 Beta 8.5 (Nov 2007) or newer

Syntax

```
char* getResAddress(string restype, string resname)
```

Arguments

restype
: The type of the resource. This is either a string (probably "FMDATA" in most cases), or an integer (see below) if the resource is a built-in resource type.

resname
: The string name or integer index of the resource.

Return

Returns 0 / false if the resource can't be found or an error occurs, otherwise returns a pointer to the first byte of the resource.

Description

Gets a pointer to data that has been embedded as a resource in the plugin DLL. You can get the size of the data buffer using getResSize.

Some useful resource type values:

1 - RT_CURSOR	Cursor resource
2 - RT_BITMAP	Bitmap resource
3 - RT_ICON	Hardware-dependent icon resource

4 - RT_MENU	Menu resource
5 - RT_DIALOG	Dialog resource
6 - RT_STRING	String-table entry
8 - RT_FONT	Font resource
16 - RT_VERSION	Version resource
23 - RT_HTML	HTML Resource
24 - RT_MANIFEST	Manifest Resource

Example

```
%fml

OnFilterStart: {

  int ressize = 0;
  void* respointer = NULL;

  // Get the size of the manifest resource
  // Type 24 is RT_MANIFEST
  // 2 is the index of the manifest file
  ressize    = getResSize(24, 2);

  // Get a pointer to the manifest data
  respointer = getResAddress(24, 2);

  // Display the size of the resource file
  Info("Size of resource: %d bytes", ressize);

  // Display the manifest string itself
  Info(respointer);

  return true;
}
```

See Also

copyResToArray, getArrayAddress, getResSize

getSaveFileName

Syntax

```
int getSaveFileName(int flags, string *file, int maxFile, int *fileOffset, int *extensionOffset, string *fileTitle, int nMaxFileTitle, string *filter, string *customFilter, int maxCustFilter, int *filterIndex, string initialDir, string dialogTitle, string defExt, int *outputflags)
```

Arguments

flags
: A set of bit flags that can be chosen to modify the behavior of the dialog. Available flags include OFN_HIDEREADONLY to hide the user-selectable "Read Only" option in the dialog, and OFN_FORCESHOWHIDDEN to force hidden files to be shown to the user. A full list of the available bit flags is in the **[OPENFILENAME structure]** Win32 documentation.

file
: A string containing the prefilled value for the File Name edit control. This string will be overwritten with the name of the file that the user has chosen on return of the function.

maxFile
: The maximum length of the string buffer *file* that contains the initial File Name

fileOffset
: The offset in bytes to the beginning of the actual filename (after the path) in the string *file*.

extensionOffset
: The offset in bytes to the beginning of the filename extension (after the dot) in the string *file*.

fileTitle

The filename and file extension (without path) of the file the user has chosen. Can be set to NULL to ignore.

maxFileTitle

The maximum length of the string buffer *fileTitle* that receives the filename & file extension the user chose.

filter

A string buffer containing pairs of null terminated strings, defining file extension filters the user can choose from to narrow their choices, together with a caption for each filter set. These typically show up next to the File Name edit field in the Save File dialog window.

customFilter

A string buffer to store a file extension filter the user might have typed in manually

maxCustFilter

The maximum length of the string buffer *customFilter*, storing the user's own custom entered extension filter.

filterIndex

The number of the file extension filter the user has chosen from the provided list in *filter*. Numbering starts from 1. If 0, the user has entered a custom filter of their own.

initialDir

A string containing the path of the initial directory to be displayed in the dialog. Can be set to NULL, in which case the current directory will likely be used. For the full rules and behavior of which folder will be chosen when the dialog is invoked, see the Win32 documentation for the **[OPENFILENAME structure].**

dialogTitle

A string containing the caption to be used for the Open File dialog window

defExt

A string containing the default file extension for the user to choose from (not containing periods or asterisks)

outputflags

An integer DWORD containing the flags as may have been modified by the Save File dialog after the user interacted with it.

Return

Returns 0 if the user specified a file name and clicked the OK button, otherwise it returns an error code as returned by the Win32 **[CommDlgExtendedError]** API function.

Description

This function displays the Windows "Save File" common dialog to the user, letting them choose a file to open.

Internally, this function is mostly a wrapper around the Win32 API function **[GetSaveFileName]**.

See Also

getOpenFileName

getSetPixelHeight

Syntax

```
int getSetPixelHeight(void);
```

Return

The height of the SetPixel canvas in pixels.

Description

Gets the height in pixels of the current SetPixel canvas.

Example

```
%fml
ctl[0]: OWNERDRAW, Color=RGB(0,0,0), Size=(100,100)
ctl[7]: STATICTEXT, Text="", Pos=(*, 120)

OnFilterStart: {

  // Get the size of the canvas
  int width, height;
  startSetPixel(0);
  width = getSetPixelWidth();
  height = getSetPixelHeight();
  endSetPixel(0);

  // Display the result
  sprintf(str0, "Pixel Dimensions: %dx%d", width, height);
  setCtlText(7, str0);
  return false;
}
```

See Also

endSetPixel, **getSetPixelWidth**, **OWNERDRAW**, **startSetPixel**

getSetPixelWidth

Syntax

`int getSetPixelWidth(void);`

Return

The width of the SetPixel canvas in pixels.

Description

Gets the width in pixels of the current SetPixel canvas.

Example

```
%fml
ctl[0]: OWNERDRAW, Color=RGB(0,0,0), Size=(100,100)
ctl[7]: STATICTEXT, Text="", Pos=(*, 120)

OnFilterStart: {

  // Get the size of the canvas
  int width, height;
  startSetPixel(0);
  width = getSetPixelWidth();
  height = getSetPixelHeight();
  endSetPixel(0);

  // Display the result
  sprintf(str0, "Pixel Dimensions: %dx%d", width, height);
  setCtlText(7, str0);
  return false;
}
```

See Also

endSetPixel, getSetPixelHeight, OWNERDRAW, startSetPixel

getSpecialFolder

Requires FM 1.0 Beta 8.4 (Sep 2007) or newer

Syntax

```
int getSpecialFolder(int CSIDL, string path)
```

Arguments

CSIDL
: A CSIDL (Constant Special Item ID List) value that identifies the folder of interest. A full list of CSIDL values can be found at **https://docs.microsoft.com/en-us/windows/win32/shell/csidl**

path
: The string where the folder path result will be stored

Return

Returns true if the operation was successful, false otherwise

Description

Retrieves the path of a Windows "Special Folder", such as the user's Desktop or My Documents folder. This function is useful for locating the folders where a program can save its data, particularly on Windows Vista systems.

Some useful CSIDL Values:

CSIDL_APPDATA
: Hidden folder for application data specific to that user. Use for things like INI files that the user shouldn't edit.

CSIDL_COMMON_APPDATA
A hidden application data folder shared with all users of a computer. Use this for settings/data that are the same for all users of the computer.

CSIDL_COMMON_DOCUMENTS
Like the My Documents folder, but shared with all users.

CSIDL_COMMON_PICTURES
A public folder of pictures shared by all users on the computer

CSIDL_DESKTOPDIRECTORY
The user's Desktop folder.

CSIDL_FONTS
A virtual folder that contains fonts. A typical path is C:\Windows\Fonts.

CSIDL_MYPICTURES
The user's My Pictures folder.

CSIDL_PERSONAL
The user's My Documents folder.

Example

```
OnFilterStart: {
  bool errOK;
  errOK = getSpecialFolder( CSIDL_COMMON_APPDATA , str8);
  if (errOK == true) msgBox(MB_OK, "Hello", str8);
  return true;
}
```

getSysColor

Syntax

```
int getSysColor(int colorIndex)
```

Arguments

colorIndex
  The color display element value to retrieve

Display Element Values

These values can also be found in the official **[MSDN GetSysColor]** docs.

COLOR_3DDKSHADOW	Dark shadow for three-dimensional display elements.
COLOR_3DFACE	Face color for three-dimensional display elements and for dialog box backgrounds.
COLOR_3DHIGHLIGHT	Highlight color for three-dimensional display elements (for edges facing the light source.)
COLOR_3DHILIGHT	Highlight color for three-dimensional display elements (for edges facing the light source.)
COLOR_3DLIGHT	Light color for three-dimensional display elements (for edges facing the light source.)
COLOR_3DSHADOW	Shadow color for three-dimensional display elements (for

	edges facing away from the light source).
COLOR_ACTIVEBORDER	Active window border.
COLOR_ACTIVECAPTION	Active window title bar.
COLOR_APPWORKSPACE	Background color of multiple document interface (MDI) applications.
COLOR_BACKGROUND	Desktop.
COLOR_BTNFACE	Face color for three-dimensional display elements and for dialog box backgrounds.
COLOR_BTNHIGHLIGHT	Highlight color for three-dimensional display elements (for edges facing the light source.)
COLOR_BTNHILIGHT	Highlight color for three-dimensional display elements (for edges facing the light source.)
COLOR_BTNSHADOW	Shadow color for three-dimensional display elements (for edges facing away from the light source).
COLOR_BTNTEXT	Text on push buttons.
COLOR_CAPTIONTEXT	Text in caption, size box, and scroll bar arrow box.
COLOR_DESKTOP	Desktop.
COLOR_GRADIENTACTIVECAPTION	Right side color in the color gradient of an active window's title bar.
COLOR_GRADIENTINACTIVECAPTION	Right side color in the color gradient of an inactive window's title bar.
COLOR_GRAYTEXT	Grayed (disabled) text.
COLOR_HIGHLIGHT	Items selected in a control.

COLOR_HIGHLIGHTTEXT	Text of items selected in a control.
COLOR_HOTLIGHT	Color for a hyperlink or hot-tracked item.
COLOR_INACTIVEBORDER	Inactive window border.
COLOR_INACTIVECAPTION	Inactive window caption.
COLOR_INACTIVECAPTIONTEXT	Color of text in an inactive caption.
COLOR_INFOBK	Background color for tooltip controls.
COLOR_INFOTEXT	Text color for tooltip controls.
COLOR_MENU	Menu background.
COLOR_MENUHILIGHT	The color used to highlight menu items when the menu appears as a flat menu. *(Not supported on Windows 2000.)*
COLOR_MENUBAR	The background color for the menu bar when menus appear as flat menus. *(Not supported on Windows 2000.)*
COLOR_MENUTEXT	Text in menus.
COLOR_SCROLLBAR	Scroll bar gray area.
COLOR_WINDOW	Window background.
COLOR_WINDOWFRAME	Window frame.
COLOR_WINDOWTEXT	Text in windows.

Return

Returns the color element as an integer. Use the **Rval**, **GVal** and **BVal** functions to retrieve the color component values.

Description

Retrieves the current color of the specified display element.

Example

```
%ffp

OnFilterStart: {
  // Make background light blue
  int color = getSysColor(COLOR_HIGHLIGHT);
  setDialogColor(color);
  return false;
}
```

See Also

setCtlColor, **setDialogColor**

getSysMem

Syntax

```
int getSysMem(int switch);
```

Return

Different integer values depending on the value of Switch - see description.

Description

Returns some values about the system memory.

Switch	Returned value
0	Percent of memory in use
1	Bytes of physical memory
2	Free physical memory bytes
3	Bytes of paging file
4	Free bytes of paging file
5	User bytes of address space
6	Free user bytes

Comment

Note that there is bug in this function that can cause memory access violations, resulting in a crash in the host application - *please do not use this function.*

getSystemDefaultLCID

Syntax

```
int getSystemDefaultLCID()
```

Return

A locale identifier **LCID**.

Description

Returns an identifier for the system's default locale.

See Also

getUserDefaultLCID, **LCID**

getThreadRetVal

Requires FM 1.0 Beta 9d (June 2008) or newer

Syntax

```
int getThreadRetVal(int hThread)
```

Arguments

hThread
: Specifies the handle of the thread whose return value we want.

Return

Returns the exit code of the specified thread. If the thread is still running, returns STILL_ACTIVE. Returns 0 if **hThread** is not a valid thread handle.

Description

Use this function to retrieve the exit code value for the specified worker thread. If the thread has completed, the exit code is the return value from the **OnCtl** handler. If the thread is still running, the code STILL_ACTIVE is returned. If the thread handle is not valid, or the function fails in some other way, 0 is returned.

Comments

The current implementation of this function is broken. **getThreadRetVal** will currently only return 0 or STILL_ACTIVE.

See Also

System Functions, countProcessors, triggerThread, waitForThread, isThreadActive, terminateThread

getUserDefaultLCID

Syntax

```
int getUserDefaultLCID()
```

Return

A locale identifier **LCID**.

Description

Returns an identifier for the user's default locale.

Example

```
%ffp

OnFilterStart: {
  //getLocaleInfo (LOCALE_USER_DEFAULT,
LOCALE_SENGLANGUAGE , str0, 255 );
  getLocaleInfo (getUserDefaultLCID(), LOCALE_SENGLANGUAGE
, str0, 255 );
  Info (str0);
  return false;
}
```

See Also

getSystemDefaultLCID, **LCID**

getWindowsVersion

Syntax

`int getWindowsVersion()`

Return

Returns an integer code representing the Windows version - see example below.

Windows 7 - 11 detection is only available in FilterMeister 1.0.9h (Dec 2021) and newer.

Description

Detects the Windows version the host graphics application thinks it is running on. This could differ from the version of Windows that the user actually has if the host application is running in a compatibility mode, or if the application does not have the appropriate manifest for the newer Windows version.

Example

```
%fml

ForEveryTile:{

  int n = getWindowsVersion();

  switch (n) {
    default:
    case 0:
      strcpy (str0, "Unknown");
      break;
    case 16:
      // Requires FM 1.0.9h
      strcpy (str0, "Windows 11");
      break;
    case 15:
      // Requires FM 1.0.9h
      strcpy (str0, "Windows 10");
      break;
    case 14:
      // Requires FM 1.0.9h
      strcpy (str0, "Windows 8.1");
      break;
    case 13:
      // Requires FM 1.0.9h
      strcpy (str0, "Windows 8");
      break;
    case 12:
      // Requires FM 1.0.9h
      strcpy (str0, "Windows 7");
      break;
    case 11:
      strcpy (str0, "Windows Vista");
      break;
    case 10:
      strcpy (str0, "Windows 2003");
      break;
    case 9:
      strcpy (str0, "Windows XP");
      break;
    case 8:
      strcpy (str0, "Windows 2000");
      break;
    case 7:
      strcpy (str0, "Windows NT");
      break;
    case 6:
      strcpy (str0, "Windows ME");
      break;
    case 5:
      strcpy (str0, "Windows 98 SE");
      break;
    case 4:
      strcpy (str0, "Windows 98");
      break;
    case 3:
      strcpy (str0, "Windows 95 OSR2");
      break;
    case 2:
      strcpy (str0, "Windows 95");
      break;
    case 1:
      strcpy (str0, "Win32s");
      break;
  }

  Info ("You are running %s", str0 );

  return true;
}
```

See Also

getDisplaySettings

grad2D

Syntax

```
int grad2D(int x, int y, int X, int Y, int grad, int dist,
int repeat)
```

Arguments

x
: Current x coordinate in the gradient field

y
: Current y coordinate in the gradient field

X
: Vertical size of the gradient field

Y
: Horizontal size of the gradient field

grad
: Gradient type (0 = Horizontal, 1 = Vertical, 2 = Diagonal, 3 = Diagonal2, 4 = Radial, 5 = Ellipsoid, 6 = Pyramid, 7 = Beam, 8 = Angular, 9 = Star, 10 = Quarter Pyramid , 11 = Quarter Pyramid 2, 12 = Quarter Radial)

dist
: 0 for linear output, 1 for sine distributed output

repeat
: number of gradient repetitions, 0 for no repetition

Return

Returns the gradient value at the coordinates (x,y)

Description

This is a gradient engine. It lets you choose between currently 12 gradient types and contains a parameter for switching between linear and sine distributed output. Additionally there's a parameter for setting the number of gradient repetitions. grad2D() returns a grayscaled gradient value. To produce color gradients, you have to scale the output for the r, g and b values yourself or use the function for each r, g and b value individually. You could also use grad2d() twice with different parameters and use blend() to merge the gradients. So there are a lot of different possibilities. Currently only returns 8-bit color values.

Example

```
%ffp

ctl(0): COMBOBOX(vscroll), Action=preview,
        Color=#FFFFFF, Fontcolor=#0000ff,
        Size=(60,200),
        Text="Horizontal\nVertical\n"
        "Diagonal\nDiagonal2\nRadial\n"
        "Ellipsoid\nPyramid\nBeam\n"
        "Angular\nStar\nQuarterPyramid1\n"
        "QuarterPyramid2\nQuarter Radial",
        Val=0
ctl(5): "Repeat", Pos=(270,30), Range=(0,128)
ctl(1): CHECKBOX, Text="Invert", Pos=(270,40)
ctl(2): CHECKBOX, Text="Sine Distribution", Pos=(270,50)
ctl(3): CHECKBOX, Text="Do Some Color Mixing", Pos=
(270,70)


ForEveryTile:
{
  int gradient, col;
  for (y=y_start; y < y_end; y++) {

    updateProgress(y,y_end);

    for (x=x_start; x < x_end; x++) {

      for (z=0; z < Z; z++) {

        if (ctl(3)) col=(z+1); else col=1;

        gradient = grad2D(x, y, X*col, Y*col, ctl(0), 
ctl(2), ctl(5));

        if (ctl(1)) gradient = 255 - gradient;

        pset(x, y, z, gradient);
      }
    }
  }

  return true;
}
```

See Also

blend

gray

Syntax

```
int gray(int r, int g, int b, int rweight, int gweight, int bweight)
```

Arguments

r
: The Red pixel color value

g
: The Green pixel color value

b
: The Blue pixel color value

rweight
: The weighting given to the red color in the grayscale algorithm

gweight
: The weighting given to the green color in the grayscale algorithm

bweight
: The weighting given to the blue color in the grayscale algorithm

Return

The new grayscale pixel value.

Description

Applies a simple grayscale effect to a given RGB pixel value, according to the weightings provided.

Example

```
ctl[0]: "Red Weight", Range=(1,500), Val=250
ctl[1]: "Green Weight", Range=(1,500), Val=250
ctl[2]: "Blue Weight", Range=(1,500), Val=250

ForEveryTile: {
  int r, g, b;
  for (y = y_start; y < y_end; y++) {
    for (x = x_start; x < x_end; x++) {
       r = src(x,y,0);
       g = src(x,y,1);
       b = src(x,y,2);
       for (z=0; z < 3; z++) {
          pset(x,y,z, gray(r, g, b, ctl(0), ctl(1), 
ctl(2)));
       }
    }
  }
  return true;
}
```

See Also

blend, **contrast**, **gamma**, **saturation**

GROUPBOX

Syntax

`ctl[`*`n`*`]: GROUPBOX(`*`Class Specific Properties`*`), `*`Other Properties`*

Description

The Groupbox is a rectangular outline with a text label that can be used to visually surround a number of related controls. It is also useful for defining groups of Radiobuttons. Note that the groupbox is not an actionable control.

Class Specific Properties

CENTER
: Aligns the text label to the center.

FLAT
: Gives the groupbox a flat appearance.

GROUP
: Defines the end of the radio button group.

LEFT
: Aligns the text label to the left. *(default)*

RIGHT
: Aligns the text label to the right.

Other Properties

Text
: Defines text to be put on the top line of the groupbox area *(default = no text)*

Val
: Assigns a value to the groupbox *(default = 0)*

Color
: Sets text background color *(default = transparent)*

FontColor
: Sets text color *(default = white)*

Example

```
ctl[1]: GROUPBOX(CENTER), Text="Groupie", Size=(70,50),
Color=CadetBlue, FontColor=Red
```

See Also

RADIOBUTTON.

Gval

Syntax

```
int Gval(int rgb)
```

Arguments

rgb
: Either a 32-bit RGB triple or a 32-bit RGBA quadruple; in either case, the red, green and blue channels are represented as eight bit values, as is the alpha channel in the RGBA form.

Return

A value in the range 0 to 255 inclusive.

Description

The return value represents the value of the green channel, extracted from the triple (or quadruple).

Example

```
// Gives the green channel value
// from current foreground color
green = Gval(fgColor);
```

See Also

Aval, **Bval**, **Rval**

haveMask

Syntax

```
bool haveMask
```

Description

Boolean variable that is true when a non-rectangular area has been selected.

See Also

msk, **isFloating**

HDBUsToPixels

Syntax

`HDBUsToPixels(int hdbu)`

Arguments

hdbu
: Number of HDBUs to convert to pixels

Description

Converts HDBU (horizontal dialog base units, the measurement by which FilterMeister dialogs are constructed) to real on-screen pixels measurement. Note that the result of this conversion depends on the users' Windows installation and may vary.

Example

```
Info("DialogSize in pixels: %d x %d",
HDBUsToPixels(getDialogWidth()),
VDBUsToPixels(getDialogHeight()));
```

See Also

PixelsToHDBUs, **PixelsToVDBUs**, **HDBUsToPixels**

hostSerialNumber

Syntax

```
int hostSerialNumber
```

Description

The host application's serial number. Most graphics programs (including Photoshop® on Windows) do not support this feature.

Comment

The Mac version of Adobe Photoshop® apparently sets this field to a valid serial number, but on Windows versions this field is always set to 0.

Example

```
sprintf(str0, "Your host application serial number is:
%d", hostSerialNumber);

if (hostSerialNumber == 0) {
  strcat(str0, "\n\n(This means your graphics program
probably\ndoesn't support serial numbers.)");
}

Info(str0);
```

hostSig

Syntax

`int hostSig`

Description

The host application provides its signature in this variable. Adobe Photoshop’s signature is '8BIM' (`0x3842494D`, or 943868237 as an integer). In theory, you can check for Photoshop as the host application with the following code; however, some other ill-behaved hosts also set hostSig to '8BIM'.

Example

```
%ffp

ForEveryTile:
{

  Info("%d", hostSig);

  if (hostSig == '8BIM')
    Info("Host is Adobe Photoshop.");

  return true;

}
```

hsl2rgb

Syntax

```
int hsl2rgb (int h, int s, int l, int z)
```

Arguments

h
: Hue value

s
: Saturation value

l
: Lightness value

z
: Determines which value is returned. z=0 for Red, z=1 for Green, z=2 for Blue

Return

Returns the red, green or blue value from 0 to 255 depending on the value of z

Description

Lets you convert HSL values to RGB values.

Example

```
%ffp

ctl(0): "Adjust H", Range=(-255,255), val=0
ctl(1): "Adjust S", Range=(-255,255), val=0
ctl(2): "Adjust L", Range=(-255,255), val=0
```

```
ForEveryTile: {

  int r,g,b,h,s,l;

  for (y= y_start; y < y_end; y++) {
    if (updateProgress(y,y_end)) abort();
    for (x = x_start; x < x_end; x++) {

      r = src(x,y,0);
      g = src(x,y,1);
      b = src(x,y,2);

      h = rgb2hsl (r,g,b,0);
      s = rgb2hsl (r,g,b,1);
      l = rgb2hsl (r,g,b,2);

      // Do the HSL adjustment
      h = h + ctl(0);
      s = s + ctl(1);
      l = l + ctl(2);

      pset( x, y, 0, hsl2rgb (h,s,l,0) );
      pset( x, y, 1, hsl2rgb (h,s,l,1) );
      pset( x, y, 2, hsl2rgb (h,s,l,2) );
    }
  }

  return true;
}
```

See Also

rgb2hsl

hypot

Syntax

```
double hypot(double x, double y)
```

Arguments

x
: First distance.

y
: Second distance.

Return

The euclidean distance of the two parameters.

Description

Calculates the hypotenuse (euclidean distance): The distance between the endpoint of two adjoining perpendicular distances. Solves Pythagoras formula; $\text{distance}^2 = x^2 + y^2$

iceil

Syntax

```
int iceil(double number)
```

Arguments

number
: Any double or float number.

Return

The rounded value as an integer.

Description

Returns the smallest integral value greater than or equal to **number**, as an integer.

Example

```
%ffp

OnFilterStart:
{
    Info("Rounding 2.345 to ceiling gives %d",
iceil(2.345));
    Info("Rounding -2.345 to ceiling gives %d",
iceil(-2.345));
}
```

See Also

ceil, **ichop**, **ifloor**, **iround**

ichop

Syntax

```
int chop(double number)
```

Arguments

number
: Any double or float number.

Return

The truncated value as an integer.

Description

Returns the value of **number** truncated, towards 0.0, to an integral value, as an integer.

Example

```
%ffp

OnFilterStart:
{
    Info("Chopping 2.543 towards 0.0 gives %d",
ichop(2.543));
    Info("Chopping -2.345 towards 0.0 gives %d",
ichop(-2.345));
}
```

See Also

chop, **iceil**, **ifloor**, **iround**, **ipow**

ICON

Syntax

```
ctl[n]: ICON(Class Specific Properties), Other Properties
```

Description

The class ICON allows the filter developer to place an icon in the dialog window. By default, this user control is not actionable.

Class Specific Properties

CENTERIMAGE
: Scales the icon to original size and centers the image within the control.

NOTIFY
: Makes the user control actionable and activates tooltip.

Other Properties

Val
: Assigns a value to the icon, but only when it is disabled. *(default = 0)*

Example

```
ctl[0]: ICON, Image="Rubberduck.ico"
```

Comment

Once the icon is actionable, its value definitions are lost. The reason is that an action returns a specific value and overwrites (once the mouse button is clicked over the icon) the icon's value.

When standalone filters are created, icons are not embedded by default. You can use the Embed: function to embed the icon into the standalone filter file.

The icon file should be present in the active directory or in any of the directories set in the PATH or FM_PATH variables (check your AUTOEXEC.BAT file).

See Also

BITMAP, **IMAGE**, **METAFILE**

ifloor

Syntax

`int ifloor(double number)`

Arguments

number
: Any double or float number.

Return

The rounded value, as an integer.

Description

Returns the largest integral value smaller than or equal to **number**, as an integer.

Example

```
%ffp

OnFilterStart:
{
    Info("Rounding 2.345 to floor gives %d",
ifloor(2.345));
    Info("Rounding -2.345 to floor gives %d",
ifloor(-2.345));
}
```

See Also

floor, **iceil**, **ichop**, **iround**

iget

Syntax

```
int iget(double x, double y, int z, int buffer, int mode)
```

Arguments

x, y
: The x and y coordinates in the image. They have to be float or double values, otherwise no interpolation is done.

z
: The z coordinates or color channel

buffer
: Set 0 for the input buffer, 1 for the first tile buffer, 2 for the second tile buffer and 3 for the output buffer.

mode
: Interpolation method: 0 for nearest neighbor (no interpolation), 1 for bisquare, 2 for bicosine, 3 for bilinear and 4 for bicubic.

Return

Returns an interpolated color value from the coordinates (x,y,z).

Considering the mathematical properties of the interpolation functions, the returned color value may sometimes be greater than 255 or smaller than 0. You may want to limit it between 0 and 255, if you're using the value as an argument to functions that don't automatically limit it (e.g., the **RGB** function).

Description

This function makes it easy to create real smooth effects. It lets you choose between 5 different interpolation methods. This function does some value caching, so it works quite fast. Currently it only works for 8 bit images.

Comments

It's worth remembering that this function returns `int` and not `double`. If you try using iget and find that your image is all black, it's probably because you got your ints and doubles mixed up somewhere.

Example

```
%ffp

// Demonstrates image resizing

ctl(0): "Resize", Range=(0,800), Val=200
ctl(10): COMBOBOX(vscroll), Action=preview,
         Color=#FFFFFF, Fontcolor=#0000ff,
         Pos=(325,40), Size=(70,200),
         Text="Nearest Neighbor\n"
         "Bisquare\nBicosine\nBilinear\n"
         "Bicubic", Val=0
ctl(100): STATICTEXT, Pos=(325,70), "",
         Fontcolor=black


ForEveryTile:
{
  float p1,q1,fracx,fracy,CalcD,dx,dy;
  int Xnew,Ynew,p,q,Calc;
  int m,n;
  const int startclock = clock();
  int endclock;

  // New image dimensions
  Xnew = X*ctl(0)/100;
  Ynew = Y*ctl(0)/100;

  // rows
  for (y = y_start; y < y_end; y++) {
    updateProgress(y, y_end);

    // columns
    for (x = x_start; x < x_end; x++) {

      p1=(float)x*X/Xnew;
      q1=(float)y*Y/Ynew;

      // channels
      for (z=0; z < zmax; z++) {

        if (y < Ynew && x < Xnew) {
          Calc = iget(p1,q1,z,0,ctl(10));
        } else {
          // Set the rest of the image to black
          Calc = 0;
        }
        pset(x, y, z, Calc);

      }
    }
  }

  // Display calculation time
  endclock = clock() - startclock;
  setCtlTextv(100, "Calculation Time: %d ms", endclock);

  return true;
}
```

See Also

src, tget, t2get, pget

IMAGE

Syntax

```
ctl[n]: IMAGE(Class Specific Properties), Other Properties
```

Description

The class IMAGE allows the filter designer to place a bitmap in the dialog window. The bitmap will be transparent according to the top-left pixel color. By default, this user control is not actionable.

Class Specific Properties

NOTIFY
Makes the user control actionable and activates tooltip.

SUNKEN
Draws a half-sunken border around the image.

Other Properties

Val
Assigns a value to the image, but only when it is disabled. *(default = 0)*

Example

```
ctl[0]: IMAGE, Image="Logo.bmp"
ctl[1]: IMAGE(MODALFRAME), Image="aa.bmp"
```

Notes

Once the image is actionable, its value definitions are lost. The reason is that an action returns a specific value and overwrites (once the mouse button is clicked over the image) the image's value.

Currently only BMP files are supported.

When standalone filters are created, images are not embedded by default. You can use the Embed: function to embed the icon into the standalone filter file.

The image file should be present in the active directory or in any of the directories set in the PATH or FM_PATH variables (check your AUTOEXEC.BAT file).

IMAGE also supports the Common Control Styles, such as BORDER, NOBORDER, TABSTOP, GROUP, VISIBLE, DISABLED, MODALFRAME, MOUSEOVER, MOUSEMOVE and more.

See Also

BITMAP, **ICON**, **METAFILE**

imageMode

Syntax

`int imageMode`

Description

The mode of the image being filtered, where Bitmap = 0, Gray Scale = 1, Indexed Color = 2, RGB Color = 3, CMYK Color = 4, HSL Color = 5, HSB Color = 6, Multichannel = 7, Duotone = 8, Lab Color = 9, 16-bit Gray Scale = 10, , 48-bit RGB Color = 11, 48-bit Lab Color = 12, 64-bit CMYK Color = 13, Deep Multichannel Color = 14 and 16-bit Duotone = 15.

Example

```
%ffp

OnFilterStart: {
  switch(imageMode) {
    case 0:
      // Same as BitmapMode
      Info("Bitmap (1-bit) Image");
      break;
    case 1:
      // Same as GrayScaleMode
      Info("Greyscale Image");
      break;
    case 2:
      // Same as IndexedColorMode
      Info("Indexed Color Image");
      break;
    case 3:
```

```
    // Same as RGBMode
    Info("RGB Image");
    break;
  case 4:
    // Same as CMYKMode
    Info("CMYK Image");
    break;
  case 5:
    // Same as HSLMode
    Info("HSL Image");
    break;
  case 6:
    // Same as HSBMode
    Info("HSB Image");
    break;
  case 7:
    // Same as MultichannelMode
    Info("Multichannel Image");
    break;
  case 8:
    // Same as DuotoneMode
    Info("Duotone Image");
    break;
  case 9:
    // Same as LabMode
    Info("Lab Color Image");
    break;
  case 10:
    // Same as Gray16Mode
    Info("16-bit Greyscale Image");
    break;
  case 11:
    // Same as RGB48Mode
    Info("48-bit RGB Color Image");
    break;
  case 12:
      // Same as Lab48Mode
      Info("48-bit Lab Color Image");
    case 13:
      // Same as CMYK64Mode
      Info("64-bit CMYK Color Image");
    case 14:
      // Same as DeepMultichannelMode
      Info("Deep Multichannel Color Image");
    case 15:
      // Same as Duotone16Mode
      Info("16-bit Duotone Image");
    default:
      Info ("Unknown image mode");
      break;
  }
  return false;
}
```

See Also

Filter Specifications

Info

Syntax

```
int Info(string promptString, ...)
```

Arguments

promptString
: Specifies the prompt string for the message window. This string may contain printf-style format descriptors, which will be expanded using the succeeding arguments. It may also contain FilterMeister-specific format descriptors.

...
: Variable number of arguments of varying types, should correspond to the format descriptors in promptString.

Return

IDOK once the user has clicked the Ok button.

Description

This function displays an information box containing a text string and an OK button.

Comment

You can also use FilterMeister format descriptors (such as the !M shown above) in the string passed to the Info function. For a full list of FilterMeister-specific extensions, please see the entry for **formatString**.

Example

```
Info ("Press OK to continue");
Info ("Your image measures: %d x %d", X, Y);
Info ("This is a\nmulti-line\nInfo-box.");
Info ("Your image mode is:\n!M");
```

See Also

msgBox, formatString

insertMenuItem

Syntax

```
int insertMenuItem (int hMenu, int uItem, LPSTR itemName,
int fState, int subMenu)
```

Arguments

hMenu
: Handle to the menu in which the new menu item is inserted.

uItem
: Unique identifier of the menu item. Must not be 0 or order may be changed.

itemName
: Menu item string. If itemName is "---", a horizontal line will be appear.

fState
: Menu item state, e.g. MFS_CHECKED, MFS_DEFAULT, MFS_DISABLED, MFS_ENABLED, MFS_GRAYED, MFS_HILITE, MFS_UNCHECKED, MFS_UNHILITE.

subMenu
: Handle to the submenu associated with the menu item. If the menu item is not an item that opens a submenu, please use 0.

Return

Returns true if succeeded.

Description

Inserts a new menu item into the given menu.

Comment

Menu items appear in the same order as the order of insertMenuItem() calls.

Example

```
%ffp

ctl[0]: PUSHBUTTON, "Click Me!"

OnCtl(n): {

  if (n==0 && e == FME_CLICKED){
    int menu=0;

    menu = createPopupMenu();

    insertMenuItem(menu, 1, "Do This",MFS_ENABLED , NULL);
    insertMenuItem(menu, 2, "Do That",MFS_ENABLED |
MFS_DEFAULT, NULL);
    insertMenuItem(menu, 3, "Do Nothing",MFS_ENABLED,
NULL);
    Info("Selection: %d", trackPopupMenu (menu, 1, 0,0,0)
);

    destroyMenu(menu);
  }

  return false;
}
```

See Also

createPopupMenu, **trackPopupMenu**, **destroyMenu**

ipow

Syntax

```
int ipow(int x, int y)
```

Arguments

x
: The value to be raised to a power.

y
: The power to which the value is raised.

Return

x raised to the power of y.

Description

Returns x raised to the power of y: x^y

Example

```
%fml

OnFilterStart:
{
  // This should give 2x2x2 = 8
  Info("ipow(2, 3) gives %d", ipow(2, 3));
}
```

See Also

pow, **powi**

iround

Syntax

```
int iround(double number)
```

Arguments

number
Any double or float number.

Return

The rounded value as an integer.

Description

Returns the value of **number** rounded to the nearest or even integral value, as an integer.

Example

```
%ffp

OnFilterStart:
{
    Info("Rounding 2.543 gives %d", iround(2.543));
    Info("Rounding -2.500 gives %d", iround(-2.500));
}
```

See Also

iceil, ichop, ifloor, round

isFloating

Syntax

```
bool isFloating
```

Description

Boolean variable that is true when the selection is floating.

Comments

A floating selection is a temporary layer, that can under Photoshop be created by the following actions:

- Repositioning a selection with the Move Tool.
- Duplicating a selection by holding down the Alt key while dragging it with the Move Tool.

See Also

msk, **haveMask**

isThreadActive

Requires FM 1.0 Beta 9d (June 2008) or newer

Syntax

```
bool isThreadActive(int hThread)
```

Arguments

hThread
Specifies the handle of the thread to be checked, or 0 to check whether any worker thread is currently active.

Return

Returns **true** if the specified thread (or any worker thread when **hThread** is 0) is running, or **false** if the specified thread (or no threads) are running.

Description

Use this function to test whether a particular worker thread, or any worker thread, is still running. Set **hThread** to the handle returned from the call to **triggerThread** to test a particular thread, or set **hThread** to 0 to test if any worker thread created by **triggerThread** is still running. This function checks the current exit code of a thread, and returns **true** if the exit code is STILL_ACTIVE.

See Also

System Functions, **countProcessors**, **triggerThread**, **waitForThread**, **getThreadRetVal**, **terminateThread**

iuv2rgb

Syntax

```
int iuv2rgb(int i, int u, int v, int z)
```

Arguments

i
: i value

u
: u value

v
: v value

z
: Determines which value is returned. z=0 for Red, z=1 for Green, z=2 for Blue

Return

Returns the red, green or blue value depending on the value of z

Description

Lets you convert YUV values to RGB values.

Example

```
%ffp

ctl(0): "Y Adjust", Range=(-255,255), Val=0
ctl(1): "U Adjust", Range=(-255,255), Val=0
ctl(2): "V Adjust", Range=(-255,255), Val=0
```

```
ForEveryTile:
{
  int r,g,b,i,u,v;

  for (y = y_start; y < y_end; y++) {

    updateProgress(y,y_end);

    for (x = x_start; x < x_end; x++) {

      r=src(x,y,0);
      g=src(x,y,1);
      b=src(x,y,2);

      i=rgb2iuv(r,g,b,0) + ctl(0);
      u=rgb2iuv(r,g,b,1) + ctl(1);
      v=rgb2iuv(r,g,b,2) + ctl(2);

      r=iuv2rgb(i,u,v,0);
      g=iuv2rgb(i,u,v,1);
      b=iuv2rgb(i,u,v,2);

      pset(x, y, 0,  r);
      pset(x, y, 1,  g);
      pset(x, y, 2,  b);
    }
  }

  return true;
}
```

See Also

rgb2iuv

lab2rgb

Syntax

`int lab2rgb(int l, int a, int b, int z)`

Arguments

l
: Red value

a
: Green value

b
: Blue value

z
: Determines which value is returned. z=0 for Red, z=1 for Green, z=2 for Blue

Return

Returns the red, green or blue value depending on the value of z

Description

Lets you convert Lab values to RGB values.

Example

```
%ffp

ctl(0): "L Adjust", Range=(-255,255), Val=0
ctl(1): "a Adjust", Range=(-255,255), Val=0
ctl(2): "b Adjust", Range=(-255,255), Val=0
```

```
ForEveryTile:{

  int r,g,b,l,a,b2;

  for (y = y_start; y < y_end; y++) {

    if (updateProgress(y, y_end)) abort();

    for (x = x_start; x < x_end; x++){

      r = src(x,y,0);
      g = src(x,y,1);
      b = src(x,y,2);

      l = rgb2lab (r,g,b,0);
      a = rgb2lab (r,g,b,1);
      b2 = rgb2lab (r,g,b,2);

      // Do the Lab adjustment
      l = l + ctl(0);
      a = a + ctl(1);
      b2 = b2 + ctl(2);

      r = lab2rgb (l,a,b2,0);
      g = lab2rgb (l,a,b2,1);
      b = lab2rgb (l,a,b2,2);

      pset( x, y, 0, r );
      pset( x, y, 1, g );
      pset( x, y, 2, b );

    }
  }

  return true;
}
```

See Also

rgb2lab

LCID

Description

An LCID is a **L**o**C**ale **ID**entifier, it is used to uniquely specify a specific set of culture, language and other locally determined settings. An LCID is an integer number.

Following is a list of all LCID's currently implemented in FilterMeister (in alphabetic order).

Locale	**LCID**
Afrikaans - South Africa	1078
Albanian - Albania	1052
Amharic - Ethiopia	1118
Arabic - Saudi Arabia	1025
Arabic - Algeria	5121
Arabic - Bahrain	15361
Arabic - Egypt	3073
Arabic - Iraq	2049
Arabic - Jordan	11265
Arabic - Kuwait	13313
Arabic - Lebanon	12289
Arabic - Libya	4097
Arabic - Morocco	6145
Arabic - Oman	8193
Arabic - Qatar	16385
Arabic - Syria	10241
Arabic - Tunisia	7169

Arabic - U.A.E.	14337
Arabic - Yemen	9217
Armenian - Armenia	1067
Assamese	1101
Azeri (Cyrillic)	2092
Azeri (Latin)	1068
Basque	1069
Belarusian	1059
Bengali	1093
Bengali (Bangladesh)	2117
Bosnian (Bosnia/Herzegovina)	5146
Bulgarian	1026
Burmese	1109
Catalan	1027
Cherokee - United States	1116
Chinese - People's Republic of China	2052
Chinese - Singapore	4100
Chinese - Taiwan	1028
Chinese - Hong Kong SAR	3076
Chinese - Macao SAR	5124
Croatian	1050
Croatian (Bosnia/Herzegovina)	4122
Czech	1029
Danish	1030
Divehi	1125
Dutch - Netherlands	1043
Dutch - Belgium	2067
Edo	1126

English - United States	1033
English - United Kingdom	2057
English - Australia	3081
English - Belize	10249
English - Canada	4105
English - Caribbean	9225
English - Hong Kong SAR	15369
English - India	16393
English - Indonesia	14345
English - Ireland	6153
English - Jamaica	8201
English - Malaysia	17417
English - New Zealand	5129
English - Philippines	13321
English - Singapore	18441
English - South Africa	7177
English - Trinidad	11273
English - Zimbabwe	12297
Estonian	1061
Faroese	1080
Farsi	1065
Filipino	1124
Finnish	1035
French - France	1036
French - Belgium	2060
French - Cameroon	11276
French - Canada	3084
French - Democratic Rep. of Congo	9228

French - Cote d'Ivoire	12300
French - Haiti	15372
French - Luxembourg	5132
French - Mali	13324
French - Monaco	6156
French - Morocco	14348
French - North Africa	58380
French - Reunion	8204
French - Senegal	10252
French - Switzerland	4108
French - West Indies	7180
Frisian - Netherlands	1122
Fulfulde - Nigeria	1127
Gaelic (Ireland)	2108
Gaelic (Scotland)	1084
Galician	1110
Georgian	1079
German - Germany	1031
German - Austria	3079
German - Liechtenstein	5127
German - Luxembourg	4103
German - Switzerland	2055
Greek	1032
Guarani - Paraguay	1140
Gujarati	1095
Hausa - Nigeria	1128
Hawaiian - United States	1141
Hebrew	1037

Hindi	1081
Hungarian	1038
Ibibio - Nigeria	1129
Icelandic	1039
Igbo - Nigeria	1136
Indonesian	1057
Inuktitut	1117
Italian - Italy	1040
Italian - Switzerland	2064
Japanese	1041
Kannada	1099
Kanuri - Nigeria	1137
Kashmiri	2144
Kashmiri (Arabic)	1120
Kazakh	1087
Khmer	1107
Konkani	1111
Korean	1042
Kyrgyz (Cyrillic)	1088
Lao	1108
Latin	1142
Latvian	1062
Lithuanian	1063
Macedonian (FYROM)	1071
Malay - Malaysia	1086
Malay - Brunei Darussalam	2110
Malayalam	1100
Maltese	1082

Manipuri	1112
Maori - New Zealand	1153
Marathi	1102
Mongolian (Cyrillic)	1104
Mongolian (Mongolian)	2128
Nepali	1121
Nepali - India	2145
Norwegian (Bokmål)	1044
Norwegian (Nynorsk)	2068
Oriya	1096
Oromo	1138
Papiamentu	1145
Pashto	1123
Polish	1045
Portuguese - Brazil	1046
Portuguese - Portugal	2070
Punjabi	1094
Punjabi (Pakistan)	2118
Quecha - Bolivia	1131
Quecha - Ecuador	2155
Quecha - Peru	3179
Rhaeto-Romanic	1047
Romanian	1048
Romanian - Moldava	2072
Russian	1049
Russian - Moldava	2073
Sami (Lappish)	1083
Sanskrit	1103

Sepedi	1132
Serbian (Cyrillic)	3098
Serbian (Latin)	2074
Sindhi - India	1113
Sindhi - Pakistan	2137
Sinhalese - Sri Lanka	1115
Slovak	1051
Slovenian	1060
Somali	1143
Sorbian	1070
Spanish - Spain (Modern Sort)	3082
Spanish - Spain (Traditional Sort)	1034
Spanish - Argentina	11274
Spanish - Bolivia	16394
Spanish - Chile	13322
Spanish - Colombia	9226
Spanish - Costa Rica	5130
Spanish - Dominican Republic	7178
Spanish - Ecuador	12298
Spanish - El Salvador	17418
Spanish - Guatemala	4106
Spanish - Honduras	18442
Spanish - Latin America	58378
Spanish - Mexico	2058
Spanish - Nicaragua	19466
Spanish - Panama	6154
Spanish - Paraguay	15370
Spanish - Peru	10250

Spanish - Puerto Rico	20490
Spanish - United States	21514
Spanish - Uruguay	14346
Spanish - Venezuela	8202
Sutu	1072
Swahili	1089
Swedish	1053
Swedish - Finland	2077
Syriac	1114
Tajik	1064
Tamazight (Arabic)	414
Tamazight (Latin)	1119
Tamil	1097
Tatar	1092
Telugu	1098
Thai	1054
Tibetan - Bhutan	2129
Tibetan - People's Republic of China	1105
Tigrigna - Eritrea	2163
Tigrigna - Ethiopia	1139
Tsonga	1073
Tswana	1074
Turkish	1055
Turkmen	1090
Uighur - China	1152
Ukrainian	1058
Urdu	1056
Urdu - India	2080

Uzbek (Cyrillic)	2115
Uzbek (Latin)	1091
Venda	1075
Vietnamese	1066
Welsh	1106
Xhosa	1076
Yi	1144
Yiddish	1085
Yoruba	1130
Zulu	1077
HID (Human Interface Device)	1279

See Also

getSystemDefaultLCID, getUserDefaultLCID

ldexp

Syntax

```
double ldexp(double x, int exp)
```

Arguments

x
Double-precision floating point value.

exp
Exponent.

Return

Floating-point value equal to x * 2^^exp^^.

Description

Calculates the floating point value corresponding to the given mantissa and exponent, such that:

```
x * 2^^exp^^
```

where x parameter represents mantissa and exp parameter the exponent.

leaveCriticalSection

Requires FM 1.0 Beta 9d (June 2008) or newer

Syntax

```
bool leaveCriticalSection(int hCS)
```

Arguments

hCS
: Specifies the handle of the Critical Section to be exited, as returned by a call to **createCriticalSection**.

Return

This function returns **true** after immediately relinquishing ownership of the Critical Section. It returns **false** if hCS is zero.

Description

This function exits a specified Critical Section, thereby relinquishing ownership of it and allowing another waiting thread to enter the Critical Section. Critical Sections must be exited in the reverse order in which they were entered. It is an error to exit a Critical Section that is not owned by the current thread. A thread may enter a specific Critical Section several times (i.e., in a nested or recursive fashion), in which case it must also exit the Critical Section the same number of times before another thread can gain access to it.

For more information about Critical Sections, see the MSDN documentation about **[Critical Section Objects]**.

Example

See the createCriticalSection example.

See Also

System Functions, createCriticalSection, enterCriticalSection, tryEnterCriticalSection, deleteCriticalSection

Comments

One need not normally test the return value of **leaveCriticalSection**. This is merely a check to make sure that hCS is non-zero so the **LeaveCriticalSection** Win32 API won't cause a memory access violation.

linearInterpolate

Syntax

```
int linearInterpolate(int v1, int v2, double x)
```

Arguments

v1
: The first value to interpolate between

v2
: The second value to interpolate between

x
: The point between the two values to interpolate at, a floating value between 0.0 and 1.0.

Return

The integer result of interpolating between the two values.

Description

Interpolates between two values according to a linear function. If you have values at two known points, you can estimate (interpolate) the value somewhere between those two points using this function. This is useful if you need to estimate a pixel value "between" the actual pixels, for example when zooming into an image. Linear interpolation assumes that the values continue like a straight line between those two points.

Example

This example performs a kind of bilinear zoom operation.

```
%fml
ctl[0]: STANDARD, Text="Zoom", Val=100

ForEveryTile: {
  for (y=0; y < Y; y++) {
    for (x=0; x < X; x++) {
      for (z=0; z < Z; z++) {

        double srcx = 100.0 * x / ctl(0);
        double srcy = 100.0 * y / ctl(0);

        int topleft  = src((int)floor(srcx),
(int)floor(srcy), z);
        int topright = src((int)ceil(srcx),
(int)floor(srcy), z);
        int lwrleft  = src((int)floor(srcx),
(int)ceil(srcy), z);
        int lwrright = src((int)ceil(srcx),
(int)ceil(srcy), z);

        int interpolatedtop = linearInterpolate(topleft,
topright, srcx - floor(srcx));
        int interpolatedlwr = linearInterpolate(lwrleft,
lwrright, srcx - floor(srcx));
        int interpolated =
linearInterpolate(interpolatedtop, interpolatedlwr, srcy -
floor(srcy));

        pset(x, y, z, interpolated);

      }
    }
  }
  return true;
}
```

Comment

To perform bilinear interpolation across an image, it is easier to use the **iget** function, which does the hard work for you.

See Also

iget, **cosineInterpolate**

LISTBOX

Syntax

```
ctl[n]: LISTBOX(Class Specific Properties), Other Properties
```

Description

Listboxes are good for scrollable lists. If you need the "pull-down menu"-style, use the **COMBOBOX** class. The items in the listbox are specified in the text string separated with the new line escape sequence (\n) and each item has its individual value.

Class Specific Properties

DISABLENOSCROLL
    Used in conjunction with HSCROLL or VSCROLL; if the item amount is less than needed to require scrolling, the scrollbar is disabled and not removed.

HSCROLL
    If necessary, a horizontal scrollbar is activated.

INTEGRALHEIGHT
    The height of the listbox is resized according to the items' height. *(default)*

MULTICOLUMN
    Items are arranged in columns (also depends on Size-property)

NOINTEGRALHEIGHT
    The height of the listbox is resized according to the Size property, even if items are partially displayed.

SORT
    Sorts the items in alphabetical order. The values of the items are recomputed; top item is always 0 and continues with 1, 2, etc.

VSCROLL
: If necessary, a vertical scrollbar is activated.

Other Properties

Text
: Defines the listbox's text contents *(default = no Text)*

Val
: Assigns a value to the listbox and activates the item *(default = -1)*

Example

```
ctl[0]: LISTBOX, "Multiply\nScreen\nLighten\nDarken"
ctl[1]: LISTBOX(NOINTEGRALHEIGHT),
"Uno\nDos\nTres\nCuatro\nCinco", Val=2, Size=(40,30)
```

See Also

COMBOBOX

loadLib

Requires FM 1.0 Beta 9.0 (April 2008) or newer

Syntax

```
int loadLib(char *dllName)
```

Arguments

dllName
: The name of the DLL to load.

Return

Returns a handle to the DLL, or 0 if the DLL could not be loaded.

Description

Loads a DLL into memory, allowing you to call the functions contained in that DLL.

Example

```
// This code loads the user32.dll
// DLL included with Windows and
// uses it to display a YES/NO
// Message Box.

int* lib_user32;
int* functionPointer;
int returnval;

// Load the DLL library
```

```
lib_user32 = loadLib("user32");
if (lib_user32) {

  // Get the function in the DLL
  functionPointer = getLibFn(lib_user32, "MessageBoxA");
  if (functionPointer) {

    // Call the function
    strcpy(str0, "The window text is here");
    strcpy(str1, "Caption Text");
    returnval = callLib(functionPointer, NULL, str0, str1,
MB_YESNO);

    // Process return value
    if (returnval == IDYES)
      msgBox(MB_OK, "Yes!", "Yes was clicked");
    if (returnval == IDNO)
      msgBox(MB_OK, "No :(", "No was clicked");

  }
  else msgBox(MB_OK, "Error", "Function wasn't loaded");

  // Free the library DLL
  freeLib(lib_user32);

}
else msgBox(MB_OK, "Error", "DLL was not loaded");
```

See Also

callLib, getLibFn, freeLib

lockHost

Requires FM 1.0 Beta 9.1 (February 2009) or newer

Syntax

```
void* lockHost(int bufferID)
```

Arguments

bufferID
The ID of the memory buffer to lock.

Return

A pointer to the memory block allocated by the host application.

Description

Retrieves a pointer to a memory block allocated by the host application with the **allocHost** function. Since some graphics programs like Photoshop manage memory themselves, you might want to use this in preference to the C-language system memory functions like **malloc**.

Example

```
int bufferID = allocHost(100);
if (bufferID == NULL) {
  Warn("Could not allocate memory");
}
else {
  char* memptr = lockHost(bufferID);
  sprintf(memptr, "Message goes here!");
  Info(memptr);
  freeHost(bufferID);
}
```

See Also

allocHost, freeHost

lockWindow

Syntax

```
int lockWindow(int c)
```

Arguments

c
: Set it to 1 to suppress updates of the filter dialog. Set it to 0 to unlock the window again.

Return

If the function succeeds, the return value is nonzero. If the function fails, the return value is zero.

Description

This function disables or enables drawing in the filter dialog. It is only recommended to use it if you want to do a lot of changes to the filter dialog, e.g. moving, removing or adding dozens of control. Under some conditions, e.g. STRETCHED dialog attribute, this can take a few seconds. By using lockWindow(1) infront of the code and lockWindow(0) afterwards, the changes will take only a few milliseconds.

Example

```
%ffp

ctl(1): PUSHBUTTON, Text="Lock Window", Size=(60,15)
ctl(3): PUSHBUTTON, Text="UnLock Window", Size=(60,15)
ctl(6): STATICTEXT, "Press 'Lock Window', try to use the
```

```
zoom controls and then press 'UnLock Window'.", Size=
(100,50)

OnCtl(n):
{
  int r;

  if (n==3 && e == FME_CLICKED)  {
    lockWindow(0);
  }
  else if (n==1 && e == FME_CLICKED)  {
    lockWindow(1);
  }

  return false;
}
```

See Also

refreshWindow, **refreshCtl**, **refreshRgn**

log

Syntax

```
double log(double x)
```

Arguments

x
: Value to be processed.

Return

Logarithm of x.

Description

Returns the natural logarithm of parameter x.

See Also

log10

log10

Syntax

```
double log10(double x)
```

Arguments

x

Value to be processed.

Return

Logarithm base 10 of x.

Description

Returns the logarithm base 10 of parameter x: log,,10,, x.

See Also

log

malloc

Syntax

```
void* malloc(int size)
```

Arguments

size
The size of the memory to be reserved, measured in bytes.

Return

A pointer to the allocated memory.

Description

Reserves **size** bytes memory. If the memory could be allocated, a pointer to the first element in the reserved memory block is returned. If the memory could not be allocated (e.g. due to memory shortage or a high degree of memory fragmentation), a NULL value is returned instead.

Any memory reserved by use of this function must be manually deallocated by means of the **free** function, failure to do so will result in memory leakage and will ultimately crash the system.

Example

```
%ffp

OnFilterStart:
{
    // Allocate a string for
    // 255 characters
    char* buffer_1 = malloc(255);

    free(buffer_1);
}
```

See Also

calloc, free, realloc

map

For Filter Factory compatibility only

Syntax

```
int map(int i, int n)
```

Arguments

i
: The index number, divided by two, of the first of two consecutive slider controls

n
: The value to map between the range.

Return

The pixel value mapped to the narrower range of values.

Description

The map function takes two consecutive **STANDARD** values and maps the given pixel value between them. Lowering the Map 1 slider value causes the image to brighten, while raising the Map 2 slider value causes it to darken, if applied to an entire image.

Internally, the map function works like this:

```
(n < MAP2) ? 0 : (n > MAP1) ? 255 : (MAP1 == MAP2) ? 255 :
(n-MAP2)*255/(MAP1-MAP2);
```

Example

```
%ffp

ctl[2]: "Map 1", Range=(0,255), Val=255
ctl[3]: "Map 2", Range=(0,255), Val=0

ForEveryTile: {
  for (y = y_start; y < y_end; y++) {
    for (x = x_start; x < x_end; x++) {
       for (z=0; z < 3; z++) {
          pset(x,y,z, map(1, src(x,y,z)) );
       }
    }
  }
  return true;
}
```

max

Syntax

```
int max(int a, int b)
```

Arguments

a
  Any integer.

b
  Any integer.

Return

The greater value of **a** and **b**.

Description

Returns the greatest of the two given values **a** and **b**. A common use for **max** is to truncate a variable to a certain lower boundary.

Example

```
// sets p to 2
int p = max(1, 2);
```

See Also

add, **min**, **sub**

memchr

Syntax

```
void *memchr(const void *membuffer, int val, int size)
```

Arguments

membuffer
: The string / memory buffer to search

val
: The byte value to search for

size
: The size (in bytes) of the memory block to search. Must be equal to or less than the buffer size, or buffer overrun errors can occur.

Return

Returns a memory location pointer to the first occurrence of the given value in the memory block, or NULL if the value is not found.

Description

memchr searches a block of memory for the first occurrence of a given value. It can be used to find the first appearance of a letter in a string. For PHP programmers, memchr is similar to strpos.

Example

```
// Allocate memory for a string
char *stringbuffer;
stringbuffer = malloc(1000);
```

```
sprintf(stringbuffer, "whatever");

// Search for first letter e
char* stringpointer;
stringpointer = memchr(stringbuffer, 'e',
strlen(stringbuffer));

// Print everything after first
// letter e if found, or show
// a message if not
if (stringpointer != NULL) {
  // This should print "ever"
  printf("%s", stringpointer);
}
else {
  printf("Character not found.");
}

// Release the memory again
free(stringbuffer);
```

See Also

sprintf, **strncmp**, **strcmp**

memcmp

Syntax

```
int memcmp(void *mem1, void *mem2, int n)
```

Arguments

mem1
The first memory block to compare

mem2
The second memory block, to be compared with mem1

n
The size (in bytes) of the memory blocks to compare

Return

Returns 0 if the two memory blocks are identical. Otherwise, it returns a negative or positive integer depending on whether the value of the data in **mem1** is found to be "less than" or "greater than" the value of the data in **mem2**.

Description

memcmp performs a byte-wise comparison of two memory blocks, to determine if they contain exactly the same data.

See Also

memicmp, **strncmp**, **strcmp**

memcpy

Syntax

```
int memcpy(void *dest, void *src, int n)
```

Arguments

dest
: The memory block to copy data to

src
: The memory block that data will be copied from

n
: The number of bytes to be copied

Return

Returns a pointer to the destination (**dest**) memory block.

Description

memcpy copies data from one block of memory to another, up to the number of bytes given.

See Also

strncpy, **strcpy**, **memmove**

memicmp

UNIX-specific C function - not part of ANSI C

Syntax

```
int memicmp(void *mem1, void *mem2, int n)
```

Arguments

mem1
    The first memory block to compare
mem2
    The second memory block, to be compared with mem1
n
    The size (in bytes) of the memory blocks to compare

Return

Returns 0 if the two memory blocks are identical. Otherwise, it returns a negative or positive integer depending on whether the value of the data in **mem1** is found to be "less than" or "greater than" the value of the data in **mem2**.

Description

memicmp performs a byte-wise and case insensitive comparison of two memory blocks, to determine if they contain the same string data.

Comment

On Microsoft systems, this function is usually called via `_memicmp` instead (note the leading underscore).

See Also

memcmp, **strcmp**, **strncmp**, **stricmp**

memmove

Syntax

```
void *memmove(void *dest, void *src, int n);
```

Arguments

dest
: The memory block to copy data to

src
: The memory block that data will be copied from

n
: The number of bytes to be copied

Return

Returns a pointer to the destination (**dest**) memory block.

Description

Despite the name, memmove copies data from one block of memory to another, up to the number of bytes given. If the memory blocks you are trying to copy overlap, it is better to use memmove instead of memcpy, as memmove uses a temporary buffer during the copy.

Comment

Be careful not to cause buffer overruns by copying more data into the destination memory block than it can hold.

Example

```
// Allocate memory for the strings
char *dest = calloc(10, 1);
char *src = calloc(10, 1);

// Store initial string values
strcpy(dest, "oldstring");
strcpy(src, "newstring");

// Show before and after memmove
Info("Before memmove: dest = %s, src = %s", dest, src);
memmove(dest, src, 3);
Info("After memmove: dest = %s, src = %s", dest, src);
```

See Also

memcpy, strcpy, strncpy

memset

Syntax

```
void* memset(void *dest, int val, int n)
```

Arguments

dest
: The memory block to copy data to

val
: The value that all the bytes in the memory block will be set to, up to the provided length

n
: The number of bytes to be set

Return

Returns a pointer to the destination (**dest**) memory block.

Description

memset sets each byte in the memory block to the given value/character, up to the number of bytes given.

Example

```
%fml

OnFilterStart: {

  // Allocate 1000 bytes of memory
  char* strbuffer;
  strbuffer = malloc(1000);
  // Fill the memory block with zeros
  memset(strbuffer, 0, 1000);

  // Set short lengths to a letter,
  // progressively overwriting them
  memset(strbuffer, 'e', 10);
  memset(strbuffer, 'D', 8);
  memset(strbuffer, 'c', 6);
  memset(strbuffer, 'B', 4);
  memset(strbuffer, 'a', 2);

  // Prints 'aaBBccDDee'
  printf("%s", strbuffer);

  // Release the memory again
  free(strbuffer);
  return true;
}
```

See Also

free, **strncpy**, **strcpy**, **malloc**

METAFILE

Syntax

```
ctl[n]: METAFILE(Class Specific Properties), Other
Properties
```

Description

The class METAFILE allows one to place a metafile in the dialog window. By default, this user control is not actionable.

Class Specific Properties

CENTERIMAGE
: Scales the image to original size and centers it within the control.

NOTIFY
: Makes the user control actionable and activates tooltip.

Other Properties

Action
: The action that the plug-in should take when the bitmap is clicked on. Possible values include APPLY and CANCEL.

Image
: The path to the image that should be used for the bitmap.

Val
: Assigns a value to the image, but only when it is disabled. *(default = 0)*

Comment

WARNING: METAFILE support is currently broken in 64-bit versions of FilterMeister, and will cause the compiler to hang the graphics program. METAFILE still works in 32-bit versions of FM.

Once the metafile is actionable, its value definitions are lost. The reason is that a user control returns a specific value and overwrites (once the mouse button is clicked over the user control) the user control's value.

Currently the windows metafile (.WMF) and enhanced metafile (.EMF) formats are supported.

When standalone filters are created, metafiles are not embedded by default. You can use the Embed: function to embed the icon into the standalone filter file.

The image file should be present in the active directory or in any of the directories set in the PATH or FM_PATH variables (check your AUTOEXEC.BAT file).

Example

```
ctl[0]: METAFILE, Image="Airplane.wmf"
ctl[1]: METAFILE(MODALFRAME, NOTIFY),
Image="D:\\Graphics\\Button2.emf", Action=CANCEL
```

See Also

BITMAP, **ICON**, **IMAGE**

min

Syntax

```
int min(int a, int b)
```

Arguments

a
: Any integer.

b
: Any integer.

Return

The lower value of **a** and **b**.

Description

Returns the least of the two given values **a** and **b**.

A common use for **min** is to truncate a variable to a certain upper boundary.

Example

```
int p = min(1, 2);  // sets p to 1
```

See Also

add, **fmin**, **max**, **sub**

mix

Syntax

```
int mix(int a, int b, int n, int d)
```

Arguments

a
: An integer value to be mixed in.

b
: An integer value for the base value.

n
: An integer value which controls the mixture.

d
: An integer value for the range.

Return

An integer which is a mix of **a** into **b**, based on the position of **n** in range [0,**d**].

Description

This function calculates how much of **a** will be mixed into **b**, proportional to the position of **n** in the range from 0 to **d**. If **n** is 0, the value of **b** is returned, if **n** is equal to **d**, the value of **a** is returned. For any values of **n** in between 0 and **d**, a proportional value is calculated.

This version of the mix function maintains compatibility with the algorithm used by Filter Factory plugins.

Example

```
%ffp

ctl(0): "Darken", range = (0, 100)

R = mix(0, r, ctl(0), 100)
G = mix(0, g, ctl(0), 100)
B = mix(0, b, ctl(0), 100)
```

See Also

scl, **mix1**, **mix2**

mix1

Syntax

```
int mix1(int a, int b, int n, int d)
```

Arguments

a
    An integer value to be mixed in.

b
    An integer value for the base value.

n
    An integer value which controls the mixture.

d
    An integer value for the range.

Return

An integer which is a mix of **a** into **b**, based on the position of **n** in range [0,**d**].

Description

This function calculates how much of **a** will be mixed into **b**, proportional to the position of **n** in the range from 0 to **d**. If **n** is 0, the value of **b** is returned, if **n** is equal to **d**, the value of **a** is returned. For any values of **n** in between 0 and **d**, a proportional value is calculated.

This version of the mix function uses a slightly different algorithm, of the form `(d != 0) ? b - (b - a)*n/d : 0;`

Example

```
%ffp

ctl[0]: "Darken", Range=(0,100)
ctl[1]: COMBOBOX, "mix\nmix1\nmix2", Val=0, Size=(*,60)

ForEveryTile: {
  for (y = y_start; y < y_end; y++) {
    for (x = x_start; x < x_end; x++) {
       for (z=0; z < 3; z++) {
          switch(ctl(1)) {
            case 0:
              pset(x,y,z, mix(0, src(x,y,z), ctl(0), 100)
);
              break;
            case 1:
              pset(x,y,z, mix1(0, src(x,y,z), ctl(0), 100)
);
              break;
            case 2:
              pset(x,y,z, mix2(0, src(x,y,z), ctl(0), 100)
);
              break;
            default:
              pset(x,y,z, mix(0, src(x,y,z), ctl(0), 100)
);
          }
       }
    }
  }
  return true;
}
```

See Also

scl, mix, mix2

mix2

Syntax

```
int mix2(int a, int b, int n, int d)
```

Arguments

a
: An integer value to be mixed in.

b
: An integer value for the base value.

n
: An integer value which controls the mixture.

d
: An integer value for the range.

Return

An integer which is a mix of **a** into **b**, based on the position of **n** in range [0,**d**].

Description

This function calculates how much of **a** will be mixed into **b**, proportional to the position of **n** in the range from 0 to **d**. If **n** is 0, the value of **b** is returned, if **n** is equal to **d**, the value of **a** is returned. For any values of **n** in between 0 and **d**, a proportional value is calculated.

This version of the mix function uses a slightly different algorithm, of the form: `(d == 0) ? 0 : (b - a)*n >= 0 ? b - ((b - a)*n*2 + d)/(2*d) : b + ((a - b)*n*2 + d)/(2*d) ;`

Example

```
%ffp

ctl[0]: "Darken", Range=(0,100)
ctl[1]: COMBOBOX, "mix\nmix1\nmix2", Val=0, Size=(*,60)

ForEveryTile: {
  for (y = y_start; y < y_end; y++) {
    for (x = x_start; x < x_end; x++) {
       for (z=0; z < 3; z++) {
          switch(ctl(1)) {
            case 0:
              pset(x,y,z, mix(0, src(x,y,z), ctl(0), 100)
);
              break;
            case 1:
              pset(x,y,z, mix1(0, src(x,y,z), ctl(0), 100)
);
              break;
            case 2:
              pset(x,y,z, mix2(0, src(x,y,z), ctl(0), 100)
);
              break;
            default:
              pset(x,y,z, mix(0, src(x,y,z), ctl(0), 100)
);
          }
       }
    }
  }
  return true;
}
```

See Also

scl, mix, mix1

mkdir

Syntax

```
int mkdir(string path)
```

Arguments

path
The path of the folder you want to make/create.

Return

Returns 0 if the new directory was created successfully, -1 otherwise.

Description

Makes a directory/folder on the user's filesystem. Remember to use double backslashes in the path.

Example

```
if (mkdir("c:\\abc\\mynewfolder") == 0) {
  msgBox(MB_OK, "Successful", "The folder was created
successfully.");
}
else msgBox(MB_OK | MB_ICONWARNING, "Error", "The folder
could not be created.");
```

See Also

getSpecialFolder, rmdir

modf

Syntax

```
double modf(double x, double * ipart)
```

Arguments

x
: The floating point value to be split.

ipart
: Pointer to a double which will store the integer part.

Return

Fractional part of x.

Description

Breaks x in two parts: the integer (stored in location pointed by ipart) and the fraction (return value).

MODIFY

Syntax

`ctl[`*`n`*`]: MODIFY(`*`Class Specific Properties`*`), `*`Other Properties`*

Description

The class MODIFY helps you change certain properties of a user control without touching the other properties. This is useful for modifying predefined user controls such as the OK, Cancel, Logo or Edit pushbuttons.

Example

```
ctl[CTL_OK]: MODIFY, "Apply"
//changes Text property from OK to Apply
ctl[CTL_EDIT]: MODIFY, Pos=(200,20), Size=(30,30)
//repositions and resizes Edit button
```

See Also

NONE

mouseOverWhenInvisible

Syntax

```
int mouseOverWhenInvisible (int t)
```

Arguments

t

Set it to 0 to disable event triggering for disabled or invisible controls. Set it to 1 to enable event triggering for disabled or invisible controls.

Return

Always returns 1.

Description

By default FME_MOUSEOVER and FME_MOUSEOUT events are triggered even for disabled and invisible controls. This behaviour can be quite undesirable in some cases. So using mouseOverWhenInvisible(0) suppresses this behaviour and doesn't trigger any events for invisible or disabled controls. mouseOverWhenInvisible(1) activates the original behaviour again.

Example

```
%ffp

ctl(0): STATICTEXT(mouseover), "Move the mouse pointer
over me",size=(150,*), disable//invisible
ctl(4): CHECKBOX, "mouseOverWhenInvisible", size=
(100,*),val=1

onCtl(n):{

  if (n==0)
    Info ("Event was triggered");

  if (n==4 && e==FME_CLICKED)
    mouseOverWhenInvisible(ctl(4));

  return false;
}
```

See Also

FME_MOUSEOVER, **FME_MOUSEOUT**

msgBox

Syntax

```
int msgBox(int level, string titleBarText, string
dialogText)
```

Arguments

level
The type of message box that you want to display.
titleBarText
The text that will appear in the title bar of the message box window.
dialogText
The text that will appear in the dialog section of the message box window.

Return

The ID of the button that the user clicked on (eg IDYES, IDNO, IDCANCEL etc.)

Description

This function calls a message box with a user-defined title bar text, a user-defined dialog text and at least one pushbutton, which depends on the dialog level set. The following levels can be set:

MB_ICONERROR	Places an error icon in the message box
MB_ICONQUESTION	Places a YESNO question icon in the message box

MB_ICONWARNING	Places a warning sign icon in the message box
MB_ICONINFORMATION	Places an Info icon in the message box
MB_OK	Places one pushbutton: OK
MB_OKCANCEL	Places two pushbuttons: OK and Cancel
MB_ABORTRETRYIGNORE	Places three pushbuttons: Abort, Retry and Ignore
MB_YESNOCANCEL	Places three pushbuttons: Yes, No and Cancel
MB_YESNO	Places two pushbuttons: Yes and No
MB_RETRYCANCEL	Places two pushbuttons: Retry and Cancel
MB_APPLMODAL	Default: OK button, title bar and dialog text
MB_SYSTEMMODAL	Places a windows logo in the title bar

These levels can be combined by adding the symbol | between them. Note that the title bar text does not support escape sequences nor substrings.

Example

```
msgBox ( MB_YESNO, "Title bar text", "Dialog text" );
msgBox ( MB_ICONQUESTION | MB_YESNOCANCEL, "Poppy's
Filters Question", "Apply the filter?" );
msgBox ( MB_ICONINFORMATION | MB_SYSTEMMODAL | MB_OK,
"Filtermania", "This will last for hours" );
```

See Also

Info, **Warn**, **Error**

msize

Microsoft-specific C function - not part of ANSI C

Syntax

```
int msize(int pointer)
```

Arguments

pointer
A pointer to a memory block

Return

The size of the given memory block in bytes.

Description

Gives the size of a dynamically allocated block of memory, given only a pointer to that block of memory.

See Also

calloc, **free**

msk

Syntax

```
int msk(int x, int y)
```

Arguments

x
: Horizontal coordinate of the requested mask pixel, starting from the left at 0.

y
: Vertical coordinate of the requested mask pixel, starting from the top at 0.

Return

The mask value for the requested pixel in the range [0, 255] (regardless of whether the color depth of the source image).

0 means the pixel is completely outside the selection mask, 255 means the pixel is completely inside the selection mask.

Description

Use this function to determine whether a pixel is inside, outside or somewhere on the edge of the host application's selection mask.

Comments

By default, FilterMeister automatically applies the selection mask to pixels for the final render; you only need to check the mask to render a correct preview inside FilterMeister. Nonetheless, if you use a msk(..) == 0 check, you can increase performance by not calculating pixels outside the selection mask.

Example

```
%ffp
SupportedModes:RGBMode

 ForEveryTile: {
  for (x = x_start; x < x_end; x++) {
   for (y = y_start; y < y_end; y++) {
    m = msk(x, y);
    if (m == 0)continue; //speed up

    for (z = 0; z < 3; z++) { //don't process alpha
     int s = src(x, y, z);
     if (doingProxy)
      pset(x, y, z, mix(255 - s, s, m, 255)); //if you
wish, apply mask on preview
     else
      pset(x, y, z, s); //the mask is applied after
pressing ok
    }//end for z
   }//end for y
  }//end for x
  return true;
 }//end ForEveryTile
```

See Also

haveMask

NONE

Syntax

`ctl[n]: NONE`

Description

The class NONE deletes a particular user control. This is useful for deleting predefined user controls such as the OK, Cancel, Logo or Edit pushbuttons.

Example

```
ctl[CTL_OK]: NONE      //deletes OK pushbutton
ctl[CTL_EDIT]: NONE    //deletes Edit pushbutton
ctl[CTL_CANCEL]: NONE  //deletes Cancel pushbutton
ctl[CTL_LOGO]: NONE    //deletes FM logo
```

See Also

MODIFY

OWNERDRAW

Syntax

`ctl[`*`n`*`]: OWNERDRAW(`*`Class Specific Properties`*`), `*`Other Properties`*

Description

This user control is a simple rectangle you can colorize. Since it is actionable, you can use it as a simple pushbutton. One possible usage is the definition of an image map consisting of a dialog background image and several OWNERDRAW user controls.

Other Properties

Color
: Defines the background color of the user control. *(default = transparent)*

Example

```
ctl[0]: OWNERDRAW, Color=Red, Size=(50,50)
ctl[1]: OWNERDRAW, Action=APPLY
ctl[2]: OWNERDRAW, Disabled, Val=5
```

Notes

If you wish to use an owner-drawn user control as a settings control (where you can change the user control's value), you have to disable it.

Comments

See the **Control drawing functions** for a list of all functions you can use to draw to an OWNERDRAW control.

pget

Syntax

```
int pget(int x, int y, int z)
```

Arguments

x
: The x coordinate of the pixel to be retrieved.

y
: The y coordinate of the pixel to be retrieved.

z
: The channel number to be retrieved for the pixel: 0->red, 1->green, 2->blue, 3->alpha.

Return

The integer value of the specified pixel for the specified channel, in the output buffer.

Description

This function retrieves the value of a specified channel 'z' from the image pixel at position (x,y) in the output buffer; in effect, it serves a similar purpose to the src() function, but retrieves from the output buffer rather than the source image. The coordinates should usually be within the image, i.e. 0<=x<X and 0<=y<Y, while the channel number z should be in the range 0 to 3 inclusive. Channels 0, 1 and 2 are the red, green and blue channels respectively for RGB image modes, while channel 3 is the alpha (transparency) channel of an RGBA image and is valid only on a Photoshop layer with transparency/opacity. Note that use of the

pget() function in the ForEveryTile handler forces FilterMeister to handle the image as a single (and possibly large) tile.

Comments

How is pget different from src? src gets the pixel from the original image while pget gets the pixel from the output buffer. At the start of the filter the output buffer is the same as the original image, which is why they seem identical. pget is useful for times when you modify the output, but still need data from the original image.

Example

```
// Shift the whole image one pixel left

%ffp

R,G,B: pget(x+1, y, z)

A: a
```

See Also

src, **pset**, **tget**, **tset**, **t2get**, **t2set**

pgetp

Syntax

```
int pgetp(int x, int y)
```

Arguments

x, y
Image coordinates

Return

Returns the pixel value at the specified image coordinates

Description

This function lets you read a whole pixel from the output buffer. Unlike **pget** the returned value includes the values of all color channels (including the transparency channel if one is available) of the pixel. Using pgetp instead of **pget** takes only approximately half as much time. To decode the individual color values from the returned pixel value you have to use the **Rval**, **Gval**, **Bval** and **Aval** functions. Currently only works with 8 bit images.

See Also

srcp, **psetp**, **tgetp**, **tsetp**, **t2getp**, **t2setp**, **Rval**, **Gval**, **Bval** and **Aval**

pgetr

Syntax

```
int pgetr(int d, int m, int z)
```

Arguments

d
: An integer value for the 'direction' of a pixel.

m
: An integer value for the 'magnitude' of a pixel.

z
: The image channel of the pixel to return (eg 0 for red, 1 for green, 2 for blue when in RGB mode)

Return

The value of the pixel channel z at polar coordinates [d,m] in the output buffer.

Description

Pixels are usually addressed by their cartesian coordinates [x,y], but FilterMeister also allows the use of polar coordinates. Polar coordinates are expressed as [d,m], where 'd' represents the 'direction' to the pixel from the image's center point, and 'm' is the 'magnitude' of the distance from the center. The pgetr() function takes a pair of polar coordinates as arguments, and returns the pixel value in channel z at those co-ordinates.

See Also

tgetr, **c2d**, **c2m**, **r2x**, **r2y**

phaseshift

Syntax

```
int phaseshift(int pixel, int intensity)
```

Arguments

pixel
    The pixel color value to modify
intensity
    The intensity of the phaseshift effect (range 0 to 512)

Return

The newly phase shifted pixel value.

Description

Applies a simple phase shift effect, adjusting the hue and colors in the image. The stronger the phaseshift effect, the more psychedelic the results.

Example

```
ctl[0]: "Phase Shift", Range=(0,512), Val=0

ForEveryTile: {
  for (y = y_start; y < y_end; y++) {
    for (x = x_start; x < x_end; x++) {
       for (z=0; z < 3; z++) {
          pset(x,y,z, phaseshift( src(x,y,z), ctl(0) ));
       }
    }
  }
  return true;
}
```

See Also

blend, contrast, gamma, saturation, solarize

PixelsToHDBUs

Syntax

```
PixelsToHDBUs(int pixels)
```

Arguments

pixels
Number of pixels to convert to HDBUs

Description

Converts on-screen pixel measurement to HDBU (horizontal dialog base units, the measurement by which FilterMeister dialogs are constructed). Note that the result of this conversion depends on the users' Windows installation and may vary.

Example

```
Info("Screensize in DBUs: %d x %d",
  PixelsToHDBUs(getDisplaySettings(1)),
  PixelsToVDBUs(getDisplaySettings(2)));
```

See Also

HDBUsToPixels, **PixelsToVDBUs**, **VDBUsToPixels**

PixelsToVDBUs

Syntax

`PixelsToVDBUs(int pixels)`

Arguments

pixels
Number of pixels to convert to VDBUs

Description

Converts on-screen pixel measurement to VDBU (vertical dialog base units, the measurement by which FilterMeister dialogs are constructed). Note that the result of this conversion depends on the users' Windows installation and may vary.

Example

```
Info("Screensize in DBUs: %d x %d",
  PixelsToHDBUs(getDisplaySettings(1)),
  PixelsToVDBUs(getDisplaySettings(2)));
```

See Also

HDBUsToPixels, **PixelsToHDBUs**, **VDBUsToPixels**

planes

Definition

The **planes** variable contains the number of channels in the image. This includes all color channels and any additional alpha channels available. It returns the same value as the Z variable.

Example

```
int totalPlanes = planes;
int colorPlanesOnly = planesWithoutAlpha;
int alphaPlanes = planes - planesWithoutAlpha;

printf("Total channels/planes: %d\nColor planes only:
%d\nAlpha channels: %d", totalPlanes, colorPlanesOnly,
alphaPlanes);
```

See Also

planesWithoutAlpha, Z

planesWithoutAlpha

Syntax

```
int planesWithoutAlpha
```

Description

planesWithoutAlpha contains the number of image channels without alpha channels. For example, an RGB image would return 3 planes, no matter how many alpha channels it also has, if any. You can now calculate the number of available alpha channels by subtracting planesWithoutAlpha from **planes** or **Z**.

Example

```
int totalPlanes = planes;
int colorPlanesOnly = planesWithoutAlpha;
int alphaPlanes = planes - planesWithoutAlpha;

printf("Total channels/planes: %d\nColor planes only:
%d\nAlpha channels: %d", totalPlanes, colorPlanesOnly,
alphaPlanes);
```

See Also

planes, **Z**

platformData

Syntax

```
int platformData
```

Description

Contains a value that is passed from the host application to the plugin. Usually it is the window handle of the host application. In most cases this value is the same for a certain application running under a certain operating system (but not e.g. in Plugin Commander Pro). So you can use it to detect under which application and operating system your plugin is currently running. This is often more reliable than using **hostSig**.

Example

```
%ffp

ForEveryTile:{

  switch (platformData){
    case 0x12ef68:
      strcpy (str0, "PhotoImpact 7 under Windows 2000");
      break;
    case 0x12ef3c:
      strcpy (str0, "PhotoImpact 8 under Windows 2000");
      break;
    case 0x97ac74:
      strcpy (str0, "Paint Shop Pro 7 under Windows 
2000");
      break;
    case 0x12e554:
      strcpy (str0, "Paint Shop Pro 8 under Windows
2000");
      break;
    case 0x4b104e8:
      strcpy (str0, "Photoshop 7 under Windows 2000");
      break;
    default:
      strcpy (str0, "Something Unknown");
      break;
  }

  Info ("This plugin runs in %s", str0 );
  return true;
}
```

See Also

[hostSig]

playSoundWave

Syntax

`bool playSoundWave(string filepath)`

Arguments

filepath
    The filename of the sound file to play.

Return

Returns true if the file was played, false otherwise.

Description

Plays a sound asynchronously (ie the plug-in will continue while the sound is playing). The function first tries finding the sound file in the embedded plug-in resources, then tries searching the usual system folders for the sound file.

Example

```
// Play a thank you sound when
// the user buys the plug-in
playSoundWave("snd\\thanks.wav");
```

See Also

playSoundWaveLoop, playSoundWaveSync

playSoundWaveLoop

Syntax

```
bool playSoundWaveLoop(string filepath)
```

Arguments

filepath
    The filename of the sound file to play.

Return

Returns true if the file was played, false otherwise.

Description

Plays a sound file repeatedly. The plug-in will continue while the sound is playing/looping. The function first tries finding the sound file in the embedded plug-in resources, then tries searching the usual system folders for the sound file.

Example

```
// Annoy the user by playing a
// song over and over again
playSoundWaveLoop("sound\\lambchop-
thisisthesongthatdoesntend.wav");
```

See Also

playSoundWave, playSoundWaveSync

playSoundWaveSync

Syntax

```
bool playSoundWaveSync(string filepath)
```

Arguments

filepath
: The filename of the sound file to play.

Return

Returns true if the file was played, false otherwise.

Description

Plays a sound synchronously (ie the plug-in will stop all operation until the sound has completed). The function first tries finding the sound file in the embedded plug-in resources, then tries searching the usual system folders for the sound file.

Example

```
// Play an alert sound that stops
// all operations temporarily
playSoundWaveSync("awooga.wav");
```

See Also

playSoundWave, **playSoundWaveLoop**

pointer_to_buffer

Syntax

```
int pointer_to_buffer(int buffer, int x, int y, int z)
```

Arguments

buffer
: Set buffer=0 for the input buffer, 1 for the first tile buffer, 2 for the second tile buffer, 3 for the output buffer, 4 for the third tile buffer, and 5 for the fourth tile buffer

x
: x-coordinate

y
: y-coordinate

z
: color channel

Return

Returns a pointer to the specified x-, y- and z-coordinates of the specified buffer.

Description

The returned pointer can be used in conjunction with **memcpy**() or **memmove**() to copy or move whole pixels or a whole image row very quickly between the buffers or inside one buffer. Later when FM will directly support pointers, this function can be used to read or write image data much quicker. Please notice that the input and output buffers aren't always allocated as one continuous memory block. A new memory block might be allocated for each row. So you have to use pointer_to_buffer() for every new row. Currently works only for 8-bit images.

Example

```
%ffp

// Drag the slider to scroll the image vertically in the
preview!
ctl(0): "Move", Range=(-100,100), Val=0, track

ForEveryTile:
{
  setCtlRange(0,-(y_end-y_start),(y_end-y_start));

  for(y = y_start; y < y_end; ++y)
    memcpy (pointer_to_buffer(3,x_start,y,0) ,
pointer_to_buffer(0,x_start, egw(y_start,y_end-1,
y+ctl(0)) ,0) , (x_end-x_start)*3);

  return true;
}
```

See Also

memcpy, **memmove**

posterize

Syntax

```
int posterize(int pixel, int intensity)
```

Arguments

pixel
    The pixel color value to modify
strength
    The intensity of the posterize effect (range 0 to 255)

Return

The new posterized pixel value.

Description

Applies a simple posterize effect to reduce the number of colors in an image. The stronger the posterize effect, the fewer colors in the resulting image.

Example

```
ctl[0]: "Posterize", Range=(0,255), Val=255

ForEveryTile: {
  for (y = y_start; y < y_end; y++) {
    for (x = x_start; x < x_end; x++) {
       for (z=0; z < 3; z++) {
          pset(x,y,z, posterize( src(x,y,z), ctl(0) ));
       }
    }
  }
  return true;
}
```

See Also

blend, contrast, gamma, saturation

pow

Syntax

```
double pow(double x, double y)
```

Arguments

x
: The value to be raised to a power.

y
: The power to which the value is raised.

Return

x raised to the power of y.

Description

Returns x raised to the power of y: x^^y^^

See Also

ipow, **powi**

powi

Syntax

```
double powi(double x, int y)
```

Arguments

x
: The (double) value to be raised to a power.

y
: The (integer) power to which the value is raised.

Return

x raised to the power of y.

Description

Returns x raised to the power of y: x^y

Example

```
%fml

OnFilterStart:
{
  // This should give 2x2x2 = 8
  Info("powi(2.0, 3) gives %f", powi(2.0, 3));
}
```

See Also

pow, **pow**

printf

Syntax

```
int printf(string text, ...)
```

Arguments

text
The text to be displayed in a Message Box alert window.

Return

The number of characters written, not including the terminating null character, or -1 if an error occurred.

Description

Writes a formatted string to a message box, using printf-style formatting common to the C language.

Comments

Internally, FilterMeister's printf function is a wrapper around the Visual Studio function **[__vsnprintf]** and the Win32 function **[MessageBox]**.

printf Format Specifiers

%%	% symbol
%c	A single character
%d	A signed integer in decimal format
%f	A double precision floating point number in decimal format

%i	A signed integer
%o	An unsigned integer in octal format
%s	A string
%u	An unsigned integer
%x	An unsigned integer in lowercase hexadecimal format
%X	An unsigned integer in uppercase hexadecimal format

Example

```
int version = 1;
strcpy(str0, "Plug-In Name");
strcpy(str1, "Company Name");
printf("You are running %s made by %s, version %d", str0,
str1, version);
```

See Also

msgBox, **sprintf**, **snprintf**

pset

Syntax

```
void pset(int x, int y, int z, int v)
```

Arguments

x
: An integer pixel x-coordinate in the output buffer.

y
: An integer pixel y-coordinate in the output buffer.

z
: An integer channel number in the range 0 to 3.

v
: An integer value to be set for channel 'z', in the range 0 to 255.

Description

This function sets the value of one channel for a pixel in the output buffer; the pixel is at coordinates [x,y], and the channel to be set is given by 'z' (0 = red, 1 = green, 2 = blue, and 3 = alpha).

Example

```
%ffp

ForEveryTile:
{
  for (y=y_start; y<y_end; ++y)
  {
    for (x=x_start; x<x_end; ++x)
    {
      for (z=0; z<Z; ++z)
      {
        // set all pixels white!
        pset(x, y, z, 255);
      }
    }
  }
  return true;
}
```

See Also

tset,t2set,pget

psetp

Syntax

`int psetp(int x, int y, int val)`

Arguments

x, y
: Image coordinates

val
: Pixel value that shall be stored

Return

Always returns a value of 1

Description

This function lets you write a whole pixel to the output buffer. Using psetp instead of **pset** takes only approximately half as much time. You have to use the **RGB** or **RGBA** function to create a pixel value from individual color values. Make sure that the individual color values lie between 0 and 255, otherwise you will get a strange image effect when passing the pixel value to this function. For details please have a look at the example below. Currently only works with 8 bit images.

Example

```
%ffp

ctl(0): "Brightness", Size=(*,6), Range=(-300,300),
Val=100
```

```
ctl(2): CHECKBOX, "Use the faster srcp() and psetp()",
Size=(150,*), Val=0
ctl(10): STATICTEXT, pos=(*,60), fontcolor=red, Size=
(150,*)


ForEveryTile:{

  int c,r,g,b;
  int a=255;

  const int startclock = clock();
  int endclock;


  for (y=y_start; y < y_end; y++) {

    if (updateProgress(y,y_end)) abort();
    for (x=x_start; x < x_end; x++) {

      if (ctl(2)) {

        // Read a whole pixel
        c = srcp (x,y);

        // Explode it into the color values
        r =  Rval(c); //c & 0xff;
        g =  Gval(c); //c >> 8 & 0xff;
        b =  Bval(c); //c >> 16 & 0xff;
        if (Z > 3) a =  Aval(c); //c >> 24 & 0xff;

        // Adjust brightness
        r += ctl(0);
        g += ctl(0);
        b += ctl(0);
        if (Z > 3) a += ctl(0);
        // Makes sure that the color
        // values are in the right range,
        // otherwise we might get a
        // strange image result
        if (r < 0) r=0; else if (r > 255) r=255;
        if (g < 0) g=0; else if (g > 255) g=255;
        if (b < 0) b=0; else if (b > 255) b=255;
        if (Z > 3) { if (a < 0) a=0; else if (a > 255)
a=255;}

        // Write back the color values
        psetp (x,y, RGBA(r,g,b,a) );


      } else {



        r =  src(x,y,0);
        g =  src(x,y,1);
        b =  src(x,y,2);
        if (Z > 3) a =  src(x,y,3);

        r += ctl(0);
        g += ctl(0);
        b += ctl(0);
        if (Z > 3) a += ctl(0);

        pset (x,y,0,r);
        pset (x,y,1,g);
        pset (x,y,2,b);
        if (Z > 3) pset (x,y,3,a);
      }
    }
  }
  endclock = clock() - startclock;
  setCtlTextv(10, "Render time needed: %d ms", endclock);

  // Display after applying effect to image
  // Should make speed difference more clear
  if (!doingProxy) Info ("Render time needed: %d ms",
endclock);

  return true;
}
```

See Also

set_psetp_mode, **srcp**, **pgetp**, **tgetp**, **tsetp**, **t2getp**, **t2setp**, **RGB**, **RGBA**

psetr

Syntax

```
void psetr(int d, int m, int z, int v)
```

Arguments

d
: An integer direction from the origin in the output buffer.

y
: An integer magnitude from the origin in the output buffer.

z
: An integer channel number in the range 0 to 3.

v
: An integer value to be set for channel 'z', in the range 0 to 255.

Description

This function sets the value of one channel for a pixel in the output buffer, using polar coordinates (rather than cartesian) to address the pixel; the polar coordinates are relative to the image center. The channel to be set is given by 'z' (0 = red, 1 = green, 2 = blue, and 3 = alpha). NOTE: There is no guarantee that this function is able to completely populate the plane - some pixels in the output buffer may be unreachable because of rounding errors.

Example

```
%ffp

ForEveryTile: {
  int d, m;

  for (d=0; d<1024; ++d) {
    for (m=0; m<256; m+=3) {
      for (z=0; z<Z; ++z) {
        psetr(d, m, z, 256);
      }
    }
  }

  return true;
}
```

See Also

tsetr, **t2setr**, **pgetr**

PUSHBUTTON

Syntax

```
ctl[n]: PUSHBUTTON(Class Specific Properties), Other
Properties
```

Description

Pushbuttons are usually involved in some dialog action. They can be used not just for general actions such as applying or cancelling a filter, but also for start/stop buttons when you animate the preview window. A pushbutton control will return the value 1 when it has been pressed, and 0 when it hasn't.

Class Specific Properties

BOTTOM
Aligns the text label with the bottom of the pushbutton.
CENTER
Centers the text label within the pushbutton's text area.
FLAT
Flattens the pushbutton.
LEFT
Left-aligns the text label within the pushbutton's text area.
MULTILINE
Allows word-wrapping within the pushbutton's text area.
RIGHT
Right-aligns the text label within the pushbutton's text area.
TOP
Aligns the text label with the top of the pushbutton.
VCENTER
Vertically centers the text label within the pushbutton's text area.

Other Properties

Text

Defines the text label on the button. *(default = no text)*

Comments

Tooltips may not always work with the PUSHBUTTON control, due to a bug in some FilterMeister versions. They work on Windows 98SE, but not on Windows 2000 or Windows XP.

Example

```
ctl[2]: PUSHBUTTON (MULTILINE), "Press this button if you
want to apply the filter", Size=(70,50), Action=APPLY
```

See Also

ctl

put

Syntax

`int put(int data, int item)`

Arguments

data
: The integer value which is to be stored in the buffer.

item
: Numeric identifier of an item in the buffer.

Return

The integer value which was stored in the buffer.

Description

FilterMeister has a small internal buffer of `N_CELLS` integer items which can be accessed by means of the `get` and `put` functions. They provide the simplest means for storaging integer data since they require no variable to be declared.

Currently `N_CELLS` is 1024. It may increase in future versions of FilterMeister, but it will never decrease. `N_CELLS` will always be a power of 2.

By default, the items in the buffer are initialized to zero at the end of the executing code block. Calling the `cell_preserve(1)` function changes this behavior so the buffer values are stored between separate handlers, making them ideal for transporting information between them.

The `put` function takes two integer arguments. The first is the value which will be stored in the buffer. The second is in the range of 0 up to and including `(N_CELLS - 1)` and denotes the position within the buffer in which to store the data.

The return value of the `put` function (rarely used) is the integer value of the first argument.

Example

```
%ffp

OnFilterStart:
{
    put(10, 0);
    Info("The value of buffer position 0 is %d", get(0));
}
```

See Also

get, **cell_preserve**

putArray

Syntax

```
int putArray (int nr, int x, int y, int z, int val)
```

Arguments

nr
: Number of the array. Values from 0 to 99 are accepted.

x, y, z
: x, y, and z coordinates of a cell in the array. If you allocated a one-dimensional array, set y and z to zero. If you allocated a two-dimensional array, set z to zero.

val
: Value that will be stored at the specified coordinates in the array.

Return

Returns 0 for failure (invalid index nr, or invalid byte-size of Array), and 1 for success.

Description

Lets you store a value in an array. When storing a value into an Array with byte-size 1, the value will be clamped to the range [0,255]. For an Array of byte-size 2, the value is clamped to [0,65535]. If the byte-size is not 1, 2, or 4, a value of 0 is returned to indicate failure.

Example

See **allocArray**

See Also

allocArray, freeArray, getArray, fgetArray, fputArray, fillArray, ffillArray, getArrayDim, copyArray

putArrayString

Syntax

```
int putArrayString(int nr, int index, string str)
```

Arguments

nr
: Number of the array to store the string in. Values from 0 to 99 are accepted.

index
: The index/position number in the array where you want to store the string

str
: The string value to be stored

Return

Returns true / 1 if the string was stored successfully. Returns false / 0 if the array is out of range, not allocated, or if the string is too large to be stored in the array.

Description

Stores a string into an element of one of the built-in arrays.

Comment

Note that you must allocate space for the array first using the allocArray function, otherwise these functions will fail.

Example

```
%ffp

ctl(0): combobox, "Harry\nJim\nSally", val=0,
action=preview, size=(*,200)

OnFilterStart:{

  // Allocate Array for storing 3 strings of max. 256
bytes length
  allocArray(0,3,256,0,1);

  // Store the Strings
  putArrayString (0,0, "Hello, Harry!");
  putArrayString (0,1, "Hello, Jimmy.");
  putArrayString (0,2, "Hello, Sally, old girl!");

  // Display
  Info ("%s", getArrayString(0,ctl(0)) );

  freeArray(0);

  return false;
}
```

See Also

allocArray, getArrayString, freeArray

putc

Syntax

```
int putc(int c, int * fileptr)
```

Arguments

c
: An integer containing a byte value.

fileptr
: Pointer to a file opened using **fopen**.

Return

The value of **c** if successful, otherwise -1.

Description

Write a single byte, contained in variable **c** to the file referenced by **fileptr**.

This function is equivalent to **fputc** but can be used in ways that can corrupt **fileptr**. It is advised to always use **fputc** instead of this function.

See Also

fopen, **getc**, **fputc**, **fputs**

putRegString

Syntax

```
int putRegString(int szString, int szValueName[,
varargs]...)
```

Arguments

szString
: is the address of the string to be stored

szValueName
: is the name of the string to be retrieved, and may contain printf-style formatting codes as well as FM !-codes.

varargs
: is a list of optional arguments used to perform printf-style formatting on the szValueName string.

Return

Returns ERROR_SUCCESS if the operation was successful, otherwise it returns one of the following integer error codes:

ERROR_SUCCESS (==0)	no error
ERROR_FILE_NOT_FOUND	key or value name not found
ERROR_MORE_DATA	buffer wasn't big enough (e.g., getRegString, getRegData)
ERROR_NO_MORE_ITEMS	index >= # of values or subkeys (enumRegValue, enumRegSubKey)
ERROR_INVALID_FUNCTION	bad top-level key, etc
ERROR_INVALID_DATA	wrong data type or size (or size > 2048)

ERROR_BADDB	registry database is corrupt
ERROR_BADKEY	registry key is invalid
ERROR_CANTOPEN	registry key could not be opened
ERROR_CANTREAD	registry key could not be read
ERROR_CANTWRITE	registry key could not be written
ERROR_REGISTRY_CORRUPT	registry is corrupt
ERROR_REGISTRY_IO_FAILED	input/output to registry failed
ERROR_KEY_DELETED	Illegal operation attempted on a Registry key which has been marked for deletion.
ERROR_KEY_HAS_CHILDREN	cannot delete a key with subkeys (Windows NT)

Description

Stores a C-style character string in the Windows Registry. Currently, only two types of registry roots are possible, identified by the constants HKEY_LOCAL_MACHINE and HKEY_CURRENT_USER.

Comment

Note that this function does not work for REG_EXPAND_SZ strings, which are a different data type to standard REG_SZ strings.

Example

```
// Saves a title for your filter
// dialog box, and retrieves it
// on next invocation

// Save title for next time
putRegString("Filter #2", "Dialog title");

// Retrieve saved title
getRegString(str0, 256, "Dialog title");
if (strcmp(str0,"") == 0) {
    // Title is null or missing
    // so set a default title
    strcpy(str0, "Default title");
}
// Set title for filter dialog
setDialogText(str0);
```

See Also

getRegRoot, setRegPath, getRegString

quickFill

Syntax

```
int quickFill(int x, int y, int z, int buffer, int radius,
int xstart, int ystart, int xend, int yend)
```

Arguments

x, y, z
: image coordinates and color channel.

buffer
: Which image buffer to read values from: 0 = input buffer, 1= tempbuffer1, 2 = tempbuffer2, 3 = output buffer, 10 = Array0, 11= Array1, 12 = Array2 and so on...

radius
: The pixel values that lie in a radius around x,y on channel z will be placed in the put/get cells. The maximum possible radius is 32, because there are only 1024 put/get cells (32*32 = 1024). If you use a higher radius value, quickFill() will not work.

xstart, ystart, xend, yend
: The borders of the image buffer. You should use x_start, y_start, x_end and y_end if you have set "isTileable = true". Otherwise 0,0,X,Y.

Return

Returns 0 if there aren't enough put cells for storage, otherwise returns 1.

Description

quickFill is meant for filling the put/get cells with values from one of the image buffer or from an array. It is meant to be used in connection with quickMedian to calculate the median value.

See Also

quickMedian

quickMedian

Syntax

```
int quickMedian(int low, int high)
```

Arguments

low
: lowest put/get cell that contains values, usually zero.

high
: highest put/get cell that contains values, usually calculated as (radius*2+1) * (radius*2+1)

Return

The return value is the median value.

Description

Finds the median value within the given range of put/get cells.

Comment

Please notice that quickMedian() works fast for radius values from 1 to 8, but above that it gets very slow. There are much faster algorithms that use tables or histogram for calculating median values with a high radius. There are also slightly faster algorithms for medians with a radius of 1 or 2.

Example

```
%ffp
ctl(7): "Radius", Range=(1,15), Val=1, Page=1
```

```
ForEveryTile:
{
  int val, dim;
  for (y=y_start; y<y_end; y++)
    for (x=x_start; x<x_end; x++)
      for (z=0; z<3; z++) {
        quickFill(x, y, z, 0, ctl(7), 0, 0, X, Y);
        dim = ctl(7)*2 + 1;
        val = quickMedian(0, dim*dim - 1);
        pset(x, y, z, val);
  }
  return true;
}
```

See Also

quickFill

r2x

Syntax

```
int r2x(int d, int m)
```

Arguments

d
: An integer value for the 'direction' of a pixel.

m
: An integer value for the 'magnitude' of a pixel.

Return

An integer giving the cartesian x coordinate for the pixel whose polar coordinates are [d,m].

Description

Pixels are usually addressed by their cartesian coordinates [x,y], but FilterMeister also allows the use of polar coordinates, and provides a set of functions for conversion between the two systems. Polar coordinates are expressed as [d,m], where 'd' represents the 'direction' to the pixel from the image's center point, and 'm' is the 'magnitude' of the distance from the center. The r2x() function takes a pair of polar coordinates as arguments, and returns the cartesian x coordinate of the corresponding pixel.

See Also

c2d, **c2m**, **r2y**

r2y

Syntax

```
int r2y(int d, int m)
```

Arguments

d
: An integer value for the 'direction' of a pixel.

m
: An integer value for the 'magnitude' of a pixel.

Return

An integer giving the cartesian y coordinate for the pixel whose polar coordinates are [d,m].

Description

Pixels are usually addressed by their cartesian coordinates [x,y], but FilterMeister also allows the use of polar coordinates, and provides a set of functions for conversion between the two systems. Polar coordinates are expressed as [d,m], where 'd' represents the 'direction' to the pixel from the image's center point, and 'm' is the 'magnitude' of the distance from the center. The r2y() function takes a pair of polar coordinates as arguments, and returns the cartesian y coordinate of the corresponding pixel.

See Also

c2d, **c2m**, **r2x**

RADIOBUTTON

Syntax

`ctl[n]: RADIOBUTTON(Class Specific Properties), Other Properties`

Description

Sometimes the programmer wants to give the user the ability to select one item from a small group of (usually more than two) options. This can be easily done with radio buttons, list boxes and combo boxes. When implementing radio buttons, the first radio button is defined as the group start, while a groupbox sets the group's end (see the example below). *Do not forget to activate one of the radio buttons with Val=1!*

Class Specific Properties

BOTTOM
: Aligns the radiobutton at the bottom of the checkbox's text area.

FLAT
: Creates a border around the radio button.

GROUP
: Defines the beginning of the radio button group.

LEFTTEXT
: Sets text to the left of the radio button.

MULTILINE
: Allows word-wrapping within the pushbutton's text area.

PUSHLIKE
: Causes the radiobutton to appear as a depressable pushbutton.

RIGHTBUTTON

Sets text to the left of the radio button.

TOP
: Aligns the radiobutton at the top of the checkbox's text area.

VCENTER
: Vertically centers the text label within the radiobutton's text area.

Other Properties

Text
: Defines the text label next to the radio button. *(default = no text)*

Val
: Assigns a value to the radio button. *(default = 0)*

Color
: Sets the text background color. *(default = transparent)*

FontColor
: Sets the color of the text. *(default = white)*

Action
: Sets the default action for the control. *(default = NONE)*

Comment

You do not need to visually "embrace" the radio buttons with the groupbox. You can create an invisible groupbox and the group items stay untouched.

Example

```
ctl[0]: RADIOBUTTON(GROUP), "Remove white", Val=1, Pos=
(210,20)
ctl[1]: RADIOBUTTON(MULTILINE, TOP), "Adjust contrast and
remove color", Pos=(210,30), Size=(70,20)
ctl[2]: GROUPBOX(GROUP, CENTER), "Test", Pos=(200,10),
Size=(90,40), Color=CadetBlue
```

Note that the first radio button is activated (value set to one) and declared as the first radio button of the group by way of the GROUP property. The second radio button is moved to the top text line. The last user control, the groupbox, defines the end of the group.

See Also

GROUPBOX

rand

Syntax

`int rand()`

Return

A random number between 0 and RAND_MAX

Description

Generates a pseudorandom number between 0 and RAND_MAX, which at the time of writing is 32767. You can use the modulo operator (%) to reduce the result to a defined range. You should also call the srand function before first calling the rand function, to ensure that the numbers do seem to be random and not predefined.

Example

```
%ffp

// Sets each of the control values to a random
// value between 0 and 200.

ctl(0): "Red",   range=( 0, 200 )
ctl(1): "Green", range=( 0, 200 )
ctl(2): "Blue",  range=( 0, 200 )

onFilterStart: {

  srand( clock() );
  setCtlVal( 0, rand() % 200 );
  setCtlVal( 1, rand() % 200 );
  setCtlVal( 2, rand() % 200 );

  return false;
}
```

See Also

srand, rst

realloc

Syntax

```
void* realloc(void* buffer, int size)
```

Arguments

buffer
A pointer to a previously allocated memory block.
size
The size of the memory to be reserved, measured in bytes.

Return

A pointer to the allocated memory.

Description

Resizes an amount of memory previously reserved by means of calloc or malloc to **size** number of bytes whilst retaining any information already in the buffer (so far as the new size specified permits).

If the memory could be allocated, a pointer to the first element in the reserved memory block is returned, memory may be moved around so the previously used pointer should be considered invalid after use in this function.

If the memory could not be allocated (i.e. due to memory shortage or a high degree of memory fragmentation), a NULL value is returned instead. Please note that should this happen, the original memory block is left untouched and must be freed manually.

When specifying a NULL pointer, this function works identical to **malloc**.

When specifying a new size of 0, memory is effectively freed and a NULL is returned since no memory is allocated.

Any memory reserved by use of this function must be manually deallocated by means of the **free** function, failure to do so will result in memory leakage and will ultimately crash the system.

Example

```
%ffp

OnFilterStart:
{
    // Allocate a string for 255 characters.
    char* buffer_1 = malloc(255);

    // We need some more memory, 512 characters will do.
    char* buffer_2 = realloc(buffer_1, 512);
    if(!buffer_2)
        free(buffer_1);

    free(buffer_2);
}
```

See Also

calloc, **free**, **malloc**

RECT

Syntax

```
ctl[n]: RECT(Class Specific Properties), Other Properties
```

Description

This user control class draws a rectangle in the dialog window. By default, this user control is not actionable.

Class Specific Properties

BLACK
: Defines the rectangle's background color as black. *(default)*

GRAY
: Defines the rectangle's background color as gray.

NOTIFY
: Makes the user control actionable and activates tooltip.

WHITE
: Defines the rectangle's background color as white.

Other Properties

Val
: Assigns a value to the frame, but only when it is disabled *(default = 0)*

Example

```
ctl[0]: RECT(NOTIFY), Action=ABOUT
ctl[1]: RECT(GRAY), Val=231, Disabled
```

Notes

Once the rectangle user control is actionable, its value definitions are lost. The reason is that an action returns a specific value and overwrites (once the mouse button is clicked over the user control) the user control's value.

refreshCtl

Syntax

```
int refreshCtl(int n)
```

Arguments

n
: The control number

Return

If the function succeeds, the return value is nonzero. If the function fails, the return value is zero.

Description

Redraws or updates a certain control.

See Also

lockWindow, **refreshWindow**, **refreshRgn**

refreshRgn

Syntax

```
int refreshRgn(region R)
```

Arguments

R

R is a region (see comments)

Return

If the function succeeds, the return value is nonzero. If the function fails, the return value is zero.

Description

Redraws or updates a certain region.

Comment

"region" correspond to a particular internal Windows type or object that might need to be better documented or defined. You can't directly cast it to an integer, despite it behaving like an integer.

Example

```
refreshRgn( createRectRgn(215, 30, 410, 250) );
```

See Also

setDialogRegion, **lockWindow**, **refreshWindow**, **refreshCtl**

refreshWindow

Syntax

```
int refreshWindow(void)
```

Return

If the function succeeds, the return value is nonzero. If the function fails, the return value is zero.

Description

Redraws or updates the filter dialog.

See Also

lockWindow, **refreshCtl**, **refreshRgn**

resetAllCtls

Requires FM 1.0 Beta 8.5 (15 Nov 2007) or newer

Syntax

`bool resetAllCtls(void)`

Return

Returns true if the function succeeded, or false if the internal RedrawWindow Win32 API call failed for some reason.

Description

Resets the entire plugin user interface back to the default state when FilterMeister is loaded without any code. This deletes any controls you may have created in your code previously, and even re-initializes the built-in strings like the Dialog Title, the Filter Author, Copyright & Version strings, and resets the background image and drag-mode of the dialog.

NB: This function might not be intended for use by the filter developer. It has never been publicly documented in any release notes. However, it has been accessible to the filter developer ever since FM 1.0 Beta 8.5 in November 2007.

Example

```
%ffp

ctl[2]: STANDARD, Text="This will vanish in about 2
seconds"

```

```
OnFilterStart: {
  sleep(2000);
  resetAllCtls();
  return true;
}
```

See Also

deleteCtl

remove

Syntax

```
int remove(const char *pathname)
```

Arguments

pathname
    The path to the filesystem entry to delete.

Return

0 if successful, otherwise -1.

Description

Deletes a file or directory from the filesystem by specifying the path to that entry through the **pathname** argument.

If successful, 0 is returned, otherwise -1 is returned and an error is set.

rename

Syntax

```
int rename(const char *old, const char *new)
```

Arguments

old
: The current name of the filesystem entry.

new
: The new name of the filesystem entry.

Return

0 if successful, otherwise -1.

Description

Renames a file or directory from the filesystem by specifying the name and/or path to that entry through the **old** and **new** arguments.

If successful, 0 is returned, otherwise -1 is returned and an error is set.

You can use this function to rename files and directories or to move files and subdirectories into other directories.

rewind

Syntax

```
void rewind(FILE * file)
```

Arguments

file
: A pointer to the file that you are working with (obtained with **fopen**).

Description

Rewinds the position indicator in the file back to the very beginning of the file.

See Also

fopen, **fread**, **fseek**, **fwrite**, **fclose**

RGB

Syntax

`int RGB(int r, int g, int b)`

Arguments

r
: Value for the red channel in the range 0 to 255 inclusive.

g
: Value for the green channel in the range 0 to 255 inclusive.

b
: Value for the blue channel in the range 0 to 255 inclusive.

Return

An integer representing the specified color as a 32-bit triple.

Description

This function can be used to construct color triples to be passed to, for example, the functions that set dialog control colors.

Example

```
// Set the color of control number 2
setCtlColor(2, RGB(255,192,255));
```

See Also

RGBA, Aval, Bval, Gval, Rval

rgb2cmyk

Syntax

```
int rgb2cmyk(int r, int g, int b, int z)
```

Arguments

r
: Red value

g
: Green value

b
: Blue value

z
: Determines which value is returned. z=0 for C (Cyan), z=1 for M (Magenta), z=2 for Y (Yellow) and z=3 for K (Black)

Return

Returns the C, M, Y, K value from 0 to 255 depending on the value of z

Description

Lets you convert RGB color values to CMYK color values.

Example

```
%ffp

ctl(0): "Adjust C", Range=(-255,255), val=0
ctl(1): "Adjust M", Range=(-255,255), val=0
ctl(2): "Adjust Y", Range=(-255,255), val=0
ctl(3): "Adjust K", Range=(-255,255), val=0

ForEveryTile:{

  int r,g,b,cyan,mag,yel,k;

  for (y= y_start; y < y_end; y++) {

    if (updateProgress(y, y_end)) abort();

    for (x = x_start; x < x_end; x++) {

      r = src(x,y,0);
      g = src(x,y,1);
      b = src(x,y,2);

      cyan = rgb2cmyk(r,g,b,0);
      mag = rgb2cmyk(r,g,b,1);
      yel = rgb2cmyk(r,g,b,2);
      k = rgb2cmyk(r,g,b,3);

      // Do the CMYK adjustment
      cyan = cyan + ctl(0);
      mag = mag + ctl(1);
      yel = yel + ctl(2);
      k = k + ctl(3);

      pset( x, y, 0, cmyk2rgb(cyan,mag,yel,k,0) );
      pset( x, y, 1, cmyk2rgb(cyan,mag,yel,k,1) );
      pset( x, y, 2, cmyk2rgb(cyan,mag,yel,k,2) );
    }
  }

  return true;
}
```

See Also

cmyk2rgb, hsl2rgb

rgb2hsl

Syntax

```
int rgb2hsl(int r, int g, int b, int z)
```

Arguments

r
: Red value

g
: Green value

b
: Blue value

z
: Determines which value is returned. z=0 for H (Hue), z=1 for S (Saturation), z=2 for L (Lightness)

Return

Returns the H, S or L value from 0 to 255 depending on the value of z

Description

Lets you convert RGB values to HSL values.

Example

```
%ffp

ctl(0): "Adjust H", Range=(-255,255), val=0
ctl(1): "Adjust S", Range=(-255,255), val=0
ctl(2): "Adjust L", Range=(-255,255), val=0
```

```
ForEveryTile:{

  int r,g,b,h,s,l;

  for (y= y_start; y < y_end; y++) {

    if (updateProgress(y, y_end)) abort();

    for (x = x_start; x < x_end; x++) {

      r = src(x,y,0);
      g = src(x,y,1);
      b = src(x,y,2);

      h = rgb2hsl (r,g,b,0);
      s = rgb2hsl (r,g,b,1);
      l = rgb2hsl (r,g,b,2);

      // Do the HSL adjustment
      h = h + ctl(0);
      s = s + ctl(1);
      l = l + ctl(2);

      pset( x, y, 0, hsl2rgb (h,s,l,0) );
      pset( x, y, 1, hsl2rgb (h,s,l,1) );
      pset( x, y, 2, hsl2rgb (h,s,l,2) );
    }
  }

  return true;
}
```

See Also

hsl2rgb

rgb2iuv

Syntax

`int rgb2iuv(int r, int g, int b, int z)`

Arguments

r
: Red value

g
: Green value

b
: Blue value

z
: Determines which value is returned. z=0 for i, z=1 for u, z=2 for v

Return

Returns the i, u or v value depending on the value of z

Description

Lets you convert RGB values to YUV values.

Example

```
%ffp

ctl(0): "Y Adjust", Range=(-255,255), Val=0
ctl(1): "U Adjust", Range=(-255,255), Val=0
ctl(2): "V Adjust", Range=(-255,255), Val=0

```

```
ForEveryTile:
{
  int r,g,b,i,u,v;

  for (y = y_start; y < y_end; y++) {

    updateProgress(y,y_end);

    for (x = x_start; x < x_end; x++) {

      r = src(x,y,0);
      g = src(x,y,1);
      b = src(x,y,2);

      i = rgb2iuv(r,g,b,0) + ctl(0);
      u = rgb2iuv(r,g,b,1) + ctl(1);
      v = rgb2iuv(r,g,b,2) + ctl(2);

      r = iuv2rgb(i,u,v,0);
      g = iuv2rgb(i,u,v,1);
      b = iuv2rgb(i,u,v,2);

      pset(x, y, 0, r);
      pset(x, y, 1, g);
      pset(x, y, 2, b);

    }
  }

  return true;
}
```

See Also

iuv2rgb

rgb2lab

Syntax

```
int rgb2lab(int r, int g, int b, int z)
```

Arguments

r
: Red value

g
: Green value

b
: Blue value

z
: Determines which value is returned. z=0 for L, z=1 for a, z=2 for b

Return

Returns the L, a or b value from 0 to 255 depending on the value of z

Description

Lets you convert RGB values to Lab values.

Example

```
%ffp

ctl(0): "L Adjust", Range=(-255,255), Val=0
ctl(1): "a Adjust", Range=(-255,255), Val=0
ctl(2): "b Adjust", Range=(-255,255), Val=0
```

```
ForEveryTile:{

  int r,g,b,l,a,b2;

  for (y= y_start; y < y_end; y++) {

    if (updateProgress(y, y_end)) abort();
    for (x = x_start; x < x_end; x++) {

      r = src(x,y,0);
      g = src(x,y,1);
      b = src(x,y,2);

      l = rgb2lab (r,g,b,0);
      a = rgb2lab (r,g,b,1);
      b2 = rgb2lab (r,g,b,2);

      // Do the Lab adjustment
      l = l + ctl(0);
      a = a + ctl(1);
      b2 = b2 + ctl(2);

      r = lab2rgb (l,a,b2,0);
      g = lab2rgb (l,a,b2,1);
      b = lab2rgb (l,a,b2,2);

      pset( x, y, 0, r );
      pset( x, y, 1, g );
      pset( x, y, 2, b );

    }
  }

  return true;
}
```

See Also

lab2rgb

rgb2ycbcr

Syntax

```
int rgb2ycbcr(int r, int g, int b, int z)
```

Arguments

r
: Red value

g
: Green value

b
: Blue value

z
: Determines which value is returned. z=0 for Y, z=1 for Cb, z=2 for Cr

Return

Returns the Y, Cb or Cr value from 0 to 255 depending on the value of z

Description

Lets you convert RGB values to YCbCr values.

Example

```
%ffp

ctl(0): "Y (Luminancy)", Range=(-255,255), Val=0
ctl(1): "Cb (Chroma Blue)", Range=(-255,255), Val=0
ctl(2): "Cr (Chroma Red)", Range=(-255,255), Val=0
```

```
ForEveryTile:{

  int r,g,b,i,cb,cr;

  for (y= y_start; y < y_end; y++) {

    if (updateProgress(y, y_end)) abort();

    for (x = x_start; x < x_end; x++) {

      r = src(x,y,0);
      g = src(x,y,1);
      b = src(x,y,2);

      i = rgb2ycbcr (r,g,b,0);
      cb = rgb2ycbcr (r,g,b,1);
      cr = rgb2ycbcr (r,g,b,2);

      // Do the adjustment
      i = i + ctl(0);
      cb = cb + ctl(1);
      cr = cr + ctl(2);

      pset( x, y, 0, ycbcr2rgb (i,cb,cr,0) );
      pset( x, y, 1, ycbcr2rgb (i,cb,cr,1) );
      pset( x, y, 2, ycbcr2rgb (i,cb,cr,2) );

    }
  }

  return true;
}
```

See Also

ycbcr2rgb

RGBA

Syntax

```
int RGBA(int r, int g, int b, int a)
```

Arguments

r
: Value for the red channel in the range 0 to 255 inclusive.

g
: Value for the green channel in the range 0 to 255 inclusive.

b
: Value for the blue channel in the range 0 to 255 inclusive.

a
: Value for the alpha channel in the range 0 to 255 inclusive.

Return

An integer representing the specified color as a 32-bit quadruple.

Description

This function can be used to construct 32-bit color quadruples from four 8-bit channel components.

See Also

RGB, **Aval**, **Bval**, **Gval**, **Rval**

rmdir

Syntax

```
int rmdir(string path)
```

Arguments

path
The path of the folder you want to remove.

Return

Returns 0 if the new directory was removed successfully, -1 otherwise.

Description

Removes a directory/folder from the user's filesystem. The folder must already be empty. Remember to use double backslashes in the path.

Example

```
if (rmdir("c:\\abc\\mynewfolder") == 0) {
  msgBox(MB_OK, "Successful", "The mynewfolder folder was
removed successfully.");
}
else msgBox(MB_OK | MB_ICONWARNING, "Error", "The folder
could not be removed.");
```

See Also

getSpecialFolder, **mkdir**, **chdir**

rnd

Syntax

`int rnd(int a, int b)`

Arguments

a
: An integer value for the lower limit.

b
: An integer value for the upper limit.

Return

A pseudorandom value between **a** and **b**.

Description

Returns a pseudorandom integer between **a** and **b**. If you need a more random number, you should first "seed" the random value with a call to srand first.

Example

```
%ffp

// This example generates random RGB pixel
// values each time that it is run.

OnFilterStart: {

  // We only call srand to seed the random
  // value once in order to make the
  // results more random.  If we kept
  // calling srand(clock()) through our
  // code, we would end up with similar
  // results for each rand call.

  srand( clock() );
  return true;
}

ForEveryPixel: {
  R = rnd( 0, 255 );
  G = rnd( 0, 255 );
  B = rnd( 0, 255 );
}
```

See Also

rand, **srand**

Comments

Since a call to srand should ideally be placed in the OnFilterStart handler, rnd is not well suited for use in the RGB handler (unless you're happy to get the same result each time you run your filter). Try using the ForEveryPixel or ForEveryTile handlers instead.

round

Syntax

`double round(double number)`

Arguments

number
: Any double or float number.

Return

The rounded value.

Description

Returns the value of **number** rounded to the nearest or even integral value.

Example

```
%ffp

OnFilterStart:
{
    Info("Rounding 2.543 gives %f", round(2.543));
    Info("Rounding -2.500 gives %f", round(-2.500));
}
```

See Also

ceil, **chop**, **floor**, **C Runtime Functions**

rst

Syntax

`int rst(int seed)`

Arguments

seed
An integer seed value for the random number generator.

Return

Returns the seed value, after being modified internally by a random value.

Description

Calling rst sets the pseudo-random number generator to a new state which is a function of both the value of 'seed' and the previous state of the pseudo-random number generator. The C run-time library routine srand(1) is implicitly called at the start of each filter run to set the pseudo-random number generator to a fixed known state, so the result of each run with otherwise identical parameters will generally produce exactly the same result. If you want the result to be different on each run, then call rst(seed) with some unpredictable value of 'seed' - e.g. call `rst(clock())` to seed the pseudo-random number generator from the elapsed time clock; or call `rst(ctl(k))` to seed the pseudo-random number generator with the current value of control k; etc.

See Also

rand, **srand**

Rval

Syntax

`int Rval(int rgb)`

Arguments

rgb
: Either a 32-bit RGB triple or a 32-bit RGBA quadruple; in either case, the red, green and blue channels are represented as eight bit values, as is the alpha channel in the RGBA form.

Return

A value in the range 0 to 255 inclusive.

Description

The return value represents the value of the red channel, extracted from the triple (or quadruple).

Example

```
red = Rval(fgColor);  // Gives the red channel value from
the current foreground color
```

See Also

Aval, **Bval**, **Gval**

samplingSupport

Syntax

```
int samplingSupport
```

Description

Indicates whether the host supports non-1:1 sampling for the proxy preview. 0 means no sampling support, 1 means integral sampling support, and 2 means fractional sampling support. Photoshop 3.0.1+ supports integral sampling steps; future versions may support non-integral sampling steps.

saturation

Syntax

```
int saturation(int r, int g, int b, int z, int sat)
```

Arguments

r
The Red pixel color value

g
The Green pixel color value

b
The Blue pixel color value

z
The pixel channel that will be returned by the function: **0** for red, **1** for green, **2** for blue.

sat
The amount of saturation to apply on a range of -500 to 500, with 0 meaning no saturation adjustment. Negative values desaturate the image.

Return

The new pixel color channel value after the saturation effect is applied.

Description

Applies a simple saturation effect to a given RGB pixel value.

Example

```
ctl[0]: "Saturation", Range=(-500,500), Val=0

ForEveryTile: {
  int r, g, b;
  for (y = y_start; y < y_end; y++) {
    for (x = x_start; x < x_end; x++) {
       r = src(x,y,0);
       g = src(x,y,1);
       b = src(x,y,2);
       for (z=0; z < 3; z++) {
          pset(x,y,z, saturation(r, g, b, z, ctl(0)));
       }
    }
  }
  return true;
}
```

See Also

blend, contrast, gamma, gray

scaleFactor

Syntax

```
int scaleFactor
```

Description

scaleFactor is an integer that defines the scale ratio (zoom factor) of the image in the preview proxy window. You can use this value to ensure your image looks correct at all zoom levels, or to help you calculate the original image co-ordinates when a user clicks on a pixel in the preview window. scaleFactor will be set to 1 at 100% zoom, 3 for 33% zoom, 5 for 20% and so on. Think of it as the denominator in the scale ratio - a 50% zoom image is 1/2 size, so the scaleFactor is 2. A 20% zoom image is 1/5 the original size, so the scaleFactor is 5 (and so on).

Example

```
%ffp

ctl(0): STATICTEXT, "Please right click on the preview to
set a cross.", size=(100,20)


OnCtl(n): {

  if (n == CTL_PREVIEW && e == FME_RIGHTCLICKED_DOWN) {
    j0 = getPreviewCoordX() * scaleFactor;
    j1 = getPreviewCoordY() * scaleFactor;
    doAction(CA_PREVIEW);
  }
  return false;
}


ForEveryTile: {

  int g, h, z, color;
  int PreviewX = j0/scaleFactor;
  int PreviewY = j1/scaleFactor;

  // Calculate color of the cross
  color = ( src(PreviewX, PreviewY, 0) + src(PreviewX,
PreviewY, 1) + src(PreviewX, PreviewY, 2) ) /3;
  if (color > 128 && color < 196) color=0;
  if (color > 64 && color < 128) color=255;
  else color = 255-color;

  // Display Cross
  if (doingProxy){
    for (z=0; z < Z; z++) {
      for (g=-7; g < 8; g++)
        if (g < -1 || g > 1) pset(PreviewX + g, PreviewY,
z, color);
      for (h=-7; h < 8; h++)
        if (h < -1 || h > 1) pset(PreviewX, PreviewY + h,
z, color );
    }
  }

  return true;
}
```

See Also

Constants, setZoom, getPreviewCoordX, getPreviewCoordY

scl

Syntax

```
int scl(int a, int il, int ih, int ol, int oh)
```

Arguments

a
: An integer value for the 'A' value.

il
: An integer value for the lower limit of the input range.

ih
: An integer value for the higher limit of the input range.

ol
: An integer value for the lower limit of the output range.

oh
: An integer value for the higher limit of the output range.

Return

An integer in the range of [**ol**,**oh**].

Description

Scales value **a** from range [**il**,**ih**] to range [**ol**,**oh**].

Comments

If **il** and **oh** are 0, the following simple formula performs better:

```
result = a * oh / ih;
```

Example

```
%ffp

ctl(0): "Stretch", range = (0, 255)

R = scl(r, 0, 255, -ctl(0), 255 + ctl(0));
G = scl(g, 0, 255, -ctl(0), 255 + ctl(0));
B = scl(b, 0, 255, -ctl(0), 255 + ctl(0));
```

See Also

mix

SCROLLBAR

Syntax

```
ctl[n]: SCROLLBAR(Class Specific Properties), Other Properties
```

Description

In contrast to the **STANDARD** control, using the SCROLLBAR control will invoke a scrollbar without text and edit control.

Class Specific Properties

HORZ
: Changes scrollbar orientation to horizontal. *(default)*

VERT
: Changes scrollbar orientation to vertical.

Other Properties

Divisor
: Sets the divisor for the text box. *(default = 1)*

Gamma
: Sets the gamma for what position in the slider corresponds to a given value. *(default = 150?)*

Line
: Sets the left/right button jump unit. *(default = 1)*

NoTrack
: Prevents the preview window updating when dragging the scrollbar's handle. *(default)*

Page
: Sets the scrollbar paging jump unit. *(default = 10)*

Range

Sets the numerical range the scrollbar can return. *(default = (0, 255))*

Track

Updates the preview window when dragging the scrollbar's handle.

Val

Initializes the scrollbar's value. *(default = 0)*

Example

```
ctl[0]: SCROLLBAR(MODALFRAME, HORZ), Val=10, Disable
ctl[2]: SCROLLBAR(VERT), Range=(-10,10), Val=0, Track
```

See Also

STANDARD

scrollPreview

Syntax

```
static int scrollPreview(int mode, int ox, int oy)
```

Arguments

mode
Set to 0 for relative mode and 1 for absolute mode

ox, oy
For relative mode, horizontal and vertical offset in pixel units. For absolute mode, the coordinates of the center of the preview.

Returns

Always returns 1.

Description

Scrolls the preview image (the part of the image displayed in the preview window) into the new position. Scrolling can be relative to current position or absolute.

This function only work if the image is larger than the preview. If the image is fully displayed in the preview, this function has no effect.

Warning: Only use scrollPreview() in the **OnCtl** and **OnFilterStart** handlers, otherwise you will get some unwanted effects in the preview! Calling it several times in a loop also causes redraw problems. But if works fine if only used for moving the image in the preview to a certain position, if followed by `doAction(CA_PREVIEW)`.

Example

`scrollPreview(0, 100, 100)` - scroll the preview image 100 pixels down and to the right

`scrollPreview(0, -100, -100)` - scroll the preview image 100 pixels up and to the left

`scrollPreview(1, 0, 0)` - display the top left corner of the image in the preview

`scrollPreview(1, 200, 200)` - center the preview image at the coordinate (200,200)

`scrollPreview(1, X/2, Y/2)` - display the center of the image in the preview

See Also

setZoom, scaleFactor

set_array_mode

Syntax

```
bool set_array_mode (int mode)
```

Arguments

mode
: Determines the default Array allocation mode. Set it to 1 to allocate Arrays using the host application's buffer allocation API, if any (this is the default). Set it to 0 to allocate Arrays from the C runtime heap (this is the fallback case).

Return

Returns **true** if the requested allocation mode was set, or **false** if the request could not be honored (e.g., if the host application does not support the buffer allocation API). In the latter case, FM falls back to case 0 (using **malloc** and **free**).

Description

If the host application provides a buffer allocation API, it is usually more effective to use this API instead of the basic C runtime heap allocation API, since the host application is able to coordinate FM's buffer allocation requests with its own memory allocation needs.

If FM allocates Arrays from the C runtime heap, either because the host application provides no buffer allocation API or because the filter calls set_array_mode(0), then be aware that FM may fail to allocate an Array, even if the host application has allocated more memory to itself than is necessary.

Example

```
if (set_array_mode(1)) {
  Info("Using host application buffer allocation API.");
}
else {
  Warn("Unable to use host buffer allocation. Using
malloc/free instead.");
}
```

See Also

allocArray, **allocArrayPad**, **freeArray**, **getArray**, **putArray**, **copyArray**, **malloc**, **free**

set_bitdepth_mode

Syntax

```
void set_bitdepth_mode(int mode)
```

Arguments

mode
Integer which can take only two values: 8 or 16

Description

Setting mode to 16 will make the **rgb2hsl** and **hsl2rgb** functions treat the passed color values as 16-bit color values.

Setting mode to 8 will activate the default behaviour of these functions.

Comments

16-bit mode means that values are in 0 to 32768 range. (Photoshop does not use the full 0 to 65535 range of 16-bit values.)

set_edge_mode

Syntax

`int set_edge_mode(int mode)`

Arguments

mode
: Set to **0** for a repeating edge, **1** for a black border around the image, **2** to wrap around the edges of the image to the other side, **3** to mirror the edges of the image.

Return

False / zero if the mode value is out of range (less than 0 or greater than 4). True otherwise.

Description

Sets how the plug-in should behave when accessing pixels beyond the edges of the image (ie whether the plug-in should show those pixels as black, mirror the edge, or wrap around to the other side of the image).

Example

```
%ffp

ctl(0): "Edge Mode", Range=(0,3), val=0
ctl(1): "Image Offset", Range=(0,100), val=30

ForEveryTile:{

  int r, g, b, offset;

  set_edge_mode( ctl(0) );
  offset = ctl(1);

  for (y= y_start; y < y_end; y++) {

    if (updateProgress(y, y_end)) abort();

    for (x = x_start; x < x_end; x++) {

      r = src(x-offset, y-offset, 0);
      g = src(x-offset, y-offset, 1);
      b = src(x-offset, y-offset, 2);

      pset( x, y, 0, r );
      pset( x, y, 1, g );
      pset( x, y, 2, b );
    }
  }

  return true;
}
```

See Also

src, **pget**

set_psetp_mode

Syntax

```
void set_psetp_mode(n)
```

Arguments

n
: Set **n** to 1, if you want to be able to set alpha to 0 with **psetp**. Set **n** to 0 to restore default behavior again.

Description

By default, you can't set the alpha channel to zero with **psetp**. This is an intended behavior, that allows to use **psetp** function in conjuction with **RGB** function without messing up alpha channel (i.e psetp(x,y,RGB(r,g,b))). To cancel this behavior, just do the instruction set_psetp_mode(1).

Example

```
%ffp
SupportedModes:RGBMode

// The default color is orange,
// but you can change it with
// sliders

ctl[0]: "R value", Range=(0,255), Val=255
ctl[1]: "G value", Range=(0,255), Val=128
ctl[2]: "B value", Range=(0,255), Val=0

ForEveryTile: {
  int color;

  // Check if alpha channel available
  if ( Z<4 ) {
    Info("This plug-in needs to "
      "be applied to a layer to "
      "achieve the effect");
    return true;
  }

  //------- the important line
  set_psetp_mode(1);

  for( y=y_start; y<y_end; y++) {
    for(x=x_start; x<x_end; x++) {

      // Chess board pattern
      if( ((x%100)<50) ^ ((y%100)<50) ) {
        a=255;
      } else {
        a=0;
      }
      color = RGBA(ctl(0), ctl(1), ctl(2), a);
      psetp(x,y,color);

    } // end for x
  } // end for y

  return true;

} // end ForEveryTile
```

See Also

psetp, **srcp**, **pgetp**, **RGB**, **RGBA**

setAngleArc

Syntax

```
bool setAngleArc(int x, int y, int radius, int a, int b, int
color)
```

Arguments

x
: Horizontal x-coordinate of the center of the pie / circle, in pixels.

y
: Vertical y-coordinate of the center of the pie / circle, in pixels.

radius
: Radius of the arc, in pixels.

a
: Start angle of the arc

b
: Ending angle of the arc

color
: Color value for the outlined arc-piece

Return

True if the function succeeded, false otherwise

Description

Render the outline of an arc-shape to an **OWNERDRAW** control.

Example

```
%fml
Title: "Wakka Wakka Outline"

ctl(1): OWNERDRAW(drawitem), Pos=(300,50), Size=(100, 100)

OnFilterStart: {

  // Maximum X/Y pixel co-ords of ownerdraw
  int maxxpixel = HDBUsToPixels(100)-1;
  int maxYpixel = VDBUsToPixels(100)-1;

  // Start using setPixel drawing on
  // OWNERDRAW control 1
  startSetPixel(1);

  // Draw a black / blue background gradient
  setRectGradient(0, 0, maxxpixel, maxYpixel, RGB(0, 0,
0), RGB(0, 0, 255), false);

  // Draw a round pill-colored circle
  setEllipseFill(180, 115, 230, 165, RGB(255, 255, 192));

  // Draw an outline of a yellow arc-shaped
  // character who says wakka wakka wakka
  setAngleArc(140, 140, 100, 3.1415 * 2.0 * 37.5 / 100.0,
3.1415 * 2.0 * 12.5 / 100.0, RGB(255,255,0));

  // Stop using setPixel drawing
  endSetPixel(1);

  return true;
}
```

See Also

OWNERDRAW, startSetPixel, setAngleArcFill, setEllipseFill, setFill, setRectGradient, endSetPixel

setAngleArcFill

Syntax

```
bool setAngleArcFill(int x, int y, int radius, int a, int b,
int color)
```

Arguments

x
: Horizontal x-coordinate of the center of the pie / circle, in pixels.

y
: Vertical y-coordinate of the center of the pie / circle, in pixels.

radius
: Radius of the arc, in pixels.

a
: Start angle of the arc

b
: Ending angle of the arc

color
: Color value for the filled arc piece.

Return

True if the function succeeded, false otherwise

Description

Render a filled arc-shape to an **OWNERDRAW** control.

Example

```
%fml
Title: "Wakka Wakka"

ctl(1): OWNERDRAW(drawitem), Pos=(300,50), Size=(100, 100)

OnFilterStart: {

  // Maximum X/Y pixel co-ords of ownerdraw
  int maxxpixel = HDBUsToPixels(100)-1;
  int maxYpixel = VDBUsToPixels(100)-1;

  // Start using setPixel drawing on OWNERDRAW control 1
  startSetPixel(1);

  // Draw a black / blue background gradient
  setRectGradient(0, 0, maxxpixel, maxYpixel, RGB(0, 0,
0), RGB(0, 0, 255), false);

  // Draw a round pill-colored circle
  setEllipseFill(180, 115, 230, 165, RGB(255, 255, 192));

  // Draw a yellow arc-shaped character
  // who says wakka wakka wakka
  setAngleArcFill(140, 140, 100, 3.1415 * 2.0 * 37.5 /
100.0, 3.1415 * 2.0 * 12.5 / 100.0, RGB(255,255,0));

  // Stop using setPixel drawing
  endSetPixel(1);

  return true;
}
```

See Also

OWNERDRAW, **startSetPixel**, **setAngleArc**, **setEllipseFill**, **setFill**, **setRectGradient**, **endSetPixel**

setBitmap

Requires FM 1.0 Beta 9.1 (February 2009) or newer

Syntax

```
int setBitmap(int x, int y, char * name)
```

Arguments

x
: The horizontal coordinate at which to draw the bitmap.

y
: The vertical coordinate at which to draw the bitmap.

name
: Null-terminated string with the name of the bitmap to use. Can be either the name of an embedded resource or a filename. Filenames may include directory information.

Return

Integer containing value TRUE if operation completed successful, otherwise FALSE.

Description

Draw a specified bitmap on the current ownerdraw of buffer canvas at the specified coordinates.

If coordinates are negative or the bitmap will not fit the canvas, it is clipped at the edges of the canvas.

See Also

startSetPixel

setBitmapStretch

Requires FM 1.0 Beta 9.1 (February 2009) or newer

Syntax

```
int setBitmapStretch(int x, int y, int iName, int width, int
height)
```

Arguments

x
: The horizontal coordinate at which to draw the bitmap.

y
: The vertical coordinate at which to draw the bitmap.

name
: Null-terminated string with the name of the bitmap to use. Can be either the name of an embedded resource or a filename. Filenames may include directory information.

width
: The horizontal size to which the bitmap will be stretched or shrunk.

height
: The vertical size to which the bitmap will be stretched or shrunk.

Return

Integer containing value TRUE if operation completed successful, otherwise FALSE.

Description

Draw a stretched or shrunk (according to specified dimensions) bitmap on the current ownerdraw of buffer canvas at the specified coordinates.

If coordinates are negative or the bitmap will not fit the canvas, it is clipped at the edges of the canvas.

See Also

setBitmap

setBitmapStretchTransparent

Requires FM 1.0 Beta 9.1 (February 2009) or newer

Syntax

```
int setBitmapStretchTransparent(int x, int y, int iName, int
width, int height, UINT color)
```

Arguments

x
: The horizontal coordinate at which to draw the bitmap.

y
: The vertical coordinate at which to draw the bitmap.

name
: Null-terminated string with the name of the bitmap to use. Can be either the name of an embedded resource or a filename. Filenames may include directory information.

width
: The horizontal size to which the bitmap will be stretched or shrunk.

height
: The vertical size to which the bitmap will be stretched or shrunk.

color
: The color (RGB) value of the transparent parts. Pixels in the bitmap with this color will not be drawn.

Return

Integer containing value TRUE if operation completed successful, otherwise FALSE.

Description

Draw a streched or shrunk (according to specified dimensions) bitmap on the current ownerdraw of buffer canvas at the specified coordinates, without drawing pixels of the specified color.

If coordinates are negative or the bitmap will not fit the canvas, it is clipped at the edges of the canvas.

See Also

RGB, **RGBA**, **setBitmap**

setBitmapTile

Requires FM 1.0 Beta 9.1 (February 2009) or newer

Syntax

```
int setBitmapTile(int x, int y, int iName, int tileWidth,
int tileHeight, int tileIndex)
```

Arguments

x
: The horizontal coordinate at which to draw the bitmap.

y
: The vertical coordinate at which to draw the bitmap.

name
: Null-terminated string with the name of the bitmap to use. Can be either the name of an embedded resource or a filename. Filenames may include directory information.

tileWidth
: Width of each tile in the bitmap.

tileHeight
: Height of each tile in the bitmap.

tileIndex
: Index number of the tile to be drawn, ordered along rows then columns similar to this text.

Return

Integer containing value TRUE if operation completed successful, otherwise FALSE.

Description

Draw part of the specified bitmap on the current ownerdraw of buffer canvas at the specified coordinates.

The bitmap is divided in tiles of the specified dimensions which are accessed using the tileIndex starting from 0. If you have 3 columns and 2 rows, a tileIndex of 3 will be the first column on the second row. If the specified dimensions are such that they do not fit the dimensions of the bitmap, no partial tiles are created.

If coordinates are negative or the bitmap will not fit the canvas, it is clipped at the edges of the canvas.

tileWidth and tileHeight must be 1 or more, otherwise FALSE will be returned.

If tileIndex is negative or beyond the number of available tiles, nothing will be drawn.

See Also

startSetPixel

setBitmapTransparent

Requires FM 1.0 Beta 9.1 (February 2009) or newer

Syntax

```
int setBitmapTransparent(int x, int y, int iName, UINT
color)
```

Arguments

x
: The horizontal coordinate at which to draw the bitmap.

y
: The vertical coordinate at which to draw the bitmap.

name
: Null-terminated string with the name of the bitmap to use. Can be either the name of an embedded resource or a filename. Filenames may include directory information.

color
: The color (RGB) value of the transparent parts. Pixels in the bitmap with this color will not be drawn.

Return

Integer containing value TRUE if operation completed successful, otherwise FALSE.

Description

Draw a specified bitmap on the current ownerdraw of buffer canvas at the specified coordinates, without drawing pixels of the specified color.

If coordinates are negative or the bitmap will not fit the canvas, it is clipped at the edges of the canvas.

See Also

RGB, **RGBA**, **setBitmap**

setClickDrag

Syntax

```
int setClickDrag(int b)
```

Arguments

b

- **0** for the default state. Only the FME_RIGHTCLICKED_DOWN and FME_RIGHTCLICKED_UP events will be triggered for the preview.
- **1** for moving the image in the preview with the right mouse button. Only the FME_LEFTCLICKED_DOWN and FME_LEFTCLICKED_UP events will be triggered for the preview.
- **2** for making it impossible to move the image in the preview. All four mentioned events will be triggered for the preview.

Return

Always true.

Description

This function lets you influence the way that the image is moved in the preview. By default the user has to left click on the preview and drag to move the image. But with this function you can also specify that the user has to use the right mouse button or that the image isn't movable at all. This feature is helpful if you want to program a feature that e.g. lets the user draw something on the preview with the left mouse button.

See Also

setZoom

setCtlAction

Syntax

```
setCtlAction(int index, int action)
```

Arguments

index
The index of the control that should perform this action.
action
The action you want the control to perform.

Description

Sets the default action for a control. You can set the action to any of the following:

Symbolic Constant	**Control Action**
CA_PREVIEW	Updates the proxy preview window.
CA_APPLY	Applies the filter to the original source image and exits the plug-in.
CA_CANCEL	Exits the plug-in filter without modifying the original source image.
CA_EDIT	Enters or exits source code editing mode (ignored in standalone filters).
CA_ABOUT	Displays the ABOUT dialog box.
CA_RESET	Resets all controls to their initial values.
CA_NONE	Performs no action.
CA_SIZE	Triggers a FME_SIZE (resize) event. That way you don't need to duplicate your sizing code

	anywhere else.

Example

```
// Set control 5 to apply the filter effect
setCtlAction( 5, CA_APPLY );
```

See Also

doAction

setCtlAnchor

Syntax

```
int setCtlAnchor(int n, int flags)
```

Arguments

n
: The number of the control to modify

flags
: A combination of the anchoring constants ANCHOR_LEFT, ANCHOR_RIGHT, ANCHOR_TOP and ANCHOR_BOTTOM.

Return

Returns false if the control number is out of range or not in use, true otherwise.

Description

Changes how a control is repositioned when the dialog is resized.

Example

```
%fml
ctl[0]: STATICTEXT, Anchor=ANCHOR_TOP|ANCHOR_RIGHT,
Text="Resize window to see anchor effect", Size=(120,*)
ctl[4]: PUSHBUTTON, Text="Top", Size=(60,*)
ctl[5]: PUSHBUTTON, Text="Left", Size=(60,*)
ctl[6]: PUSHBUTTON, Text="Right", Size=(60,*)
ctl[7]: PUSHBUTTON, Text="Bottom", Size=(60,*)
ctl[8]: PUSHBUTTON, Text="Bottom Right", Size=(60,*)
```

```
setCtlAnchor(4, ANCHOR_TOP);
setCtlAnchor(5, ANCHOR_LEFT);
setCtlAnchor(6, ANCHOR_RIGHT);
setCtlAnchor(7, ANCHOR_BOTTOM);
setCtlAnchor(8, ANCHOR_RIGHT | ANCHOR_BOTTOM);
```

See Also

updateAnchors

setCtlBuddyStyle

Syntax

```
int setCtlBuddyStyle(int n, int buddy, int flags)
```

Arguments

n
: The number of the **STANDARD** or **SLIDER** control to modify

buddy
: Set to **1** to modify the edit box, or to **0** to modify the text label

flags
: The style flags to assign to this control.

Style Flags

ES_CENTER	Centers the text in the edit box when editing
ES_LEFT	Left-aligns the text in the edit box when editing
ES_LOWERCASE	Forces all letters to be lowercase in the editbox
ES_NUMBER	Only allow numbers to be typed into the editbox
ES_RIGHT	Right-aligns the text in the edit box when editing
ES_UPPERCASE	Forces all letters to be uppercase in the editbox
SS_BLACKFRAME	Adds a black rectangle around the text label
SS_BLACKRECT	Replaces the label with a blackbox

SS_ETCHEDFRAME	Adds an etched frame to the text label
SS_ETCHEDHORZ	Adds an etched frame to the text label
SS_GRAYFRAME	Adds a gray rectangle around the text label
SS_GRAYRECT	Replaces the label with a gray box
SS_WHITEFRAME	Adds a white rectangle around the text label
SS_WHITERECT	Replaces the label with a white box
WS_BORDER	Enables the single border outline of the editbox

Return

Returns false is control number n is out of range or unused. Otherwise, returns the result of the internal SetWindowPos function (true if succeeded, false otherwise).

Description

Changes the window style of the labels and edit boxes of a STANDARD or SLIDER control.

Comment

This functions must be followed by **refreshCtl** or **refreshWindow**, or the changes may not take effect.

Due to a bug, **refreshCtl** and **refreshWindow** may effect the text label rendering differently.

Note that you cannot remove window styles with this function. To remove a style, use **clearCtlBuddyStyle**.

Example

```
%ffp
```

```
ctl(0): PUSHBUTTON, "Make Changes", Size=(60, *)
ctl(1): PUSHBUTTON, "Clear Changes", Size=(60, *), Pos=
(*,20)
ctl[4]: STANDARD, "Example"

OnCtl(n):{

  if (n==0 && e==FME_CLICKED) {

    // Make editbox Uppercase & Right-aligned
    setCtlBuddyStyle(4, 1, WS_BORDER | ES_UPPERCASE |
ES_RIGHT);
    setCtlBuddyStyle(4, 2, SS_GRAYFRAME);
    refreshWindow();
  }

  if (n==1 && e==FME_CLICKED) {

    // Clear previously set styles
    clearCtlBuddyStyle(4, 1, WS_BORDER | ES_UPPERCASE |
ES_RIGHT);
    clearCtlBuddyStyle(4, 2, SS_GRAYFRAME);
    refreshWindow();
  }

  return false;
}
```

See Also

clearCtlBuddyStyle, **refreshCtl**, **STANDARD**, **SLIDER**

setCtlColor

Syntax

```
setCtlColor(int n, int color)
```

Arguments

n
: The index of the user control whose color you wish to change.

color
: The color you wish to change the background of the control to.

Description

This function changes the background color of the user control with the index n. The color value can be an RGB triplet (as in RGB(r,g,b)) or a COLOR function (as in COLOR(MediumOrchid)). The following user controls support background coloring: standard scrollbars, trackbars, checkboxes, radio buttons, group boxes, owner draw controls, static text and icons. To set the background color to transparent, set the value color to -1.

Example

```
setCtlColor(0, RGB(255,0,255) );      //set background
color to magenta
setCtlColor(6, COLOR(X11.indianred)); //set background
color to indian red
setCtlColor(23, -1);                  //set background
color to transparent
```

See Also

chooseColor, getCtlColor, RGB, setCtlFontColor

setCtlDefVal

Requires FM 1.0 Beta 9.1 (February 2009) or newer

Syntax

```
int setCtlDefVal(int n, int val)
```

Arguments

n
: The number of the control to set a default value for

val
: The default value of the control

Return

Returns false if the control number is out of range or not in use, true otherwise.

Description

Sets the default value for the control when the user double clicks its text label.

Example

```
// DefVal sets the double-click default value
ctl[0]: STANDARD, "Control 0", DefVal=192

// Val sets the initial control value
ctl[1]: STANDARD, "Control 1", Val=0

OnFilterStart: {
  // Double click on the text of Control
  // 1 to set to the default value of 128
  setCtlDefVal(1, 128);
  return true;
}
```

See Also

setCtlVal

setCtlDivisor

Requires FM 0.4.21 Beta 2 (June 3, 2003) or newer

Syntax

```
setCtlDivisor(int n, int v)
```

Arguments

n
: The index of the user control whose value you want to change

v
: The value you want to assign to the user control

Description

This function changes the divisor of standard and scrollbar controls

Example

```
setCtlDivisor(4, 10);
setCtlDivisor(5, 1);
```

See Also

setCtlGamma

setCtlEditSize

Syntax

```
int setCtlEditSize(int n, int w, int h)
```

Arguments

n
: The number of the user control to resize

w
: The new width of the edit box

h
: The new height of the edit box

Return

Returns true (non-zero) if the given control **n** is a **STANDARD** or **SLIDER** control, false (zero) otherwise.

Description

Resizes the text edit box to the right of a standard slider control.

Example

```
ctl[0]: STANDARD(HORZ), Text="Width", Val=20

OnCtl(n): {
  if (n == 0) {
    // Resize width of slider edit box
    setCtlEditSize(0, ctl(0), 10);
  }
  return false;
}
```

See Also

setCtlPos

setCtlFocus

Requires FM 1.0 Beta 6 (May 2005) or newer

Syntax

```
int setCtlFocus(int n)
```

Arguments

n
: The number of the user control that will receive focus

Return

Returns false if the control number is out of range, a preview control or not in use. Returns NULL if the parameter is otherwise invalid. Otherwise, returns a handle to the user control.

Description

Sets the keyboard focus to the specified user control.

Example

```
%fml
ctl[0]: STANDARD(MOUSEOVER), Text="0", Val=0
ctl[1]: STANDARD(MOUSEOVER), Text="1", Val=75
ctl[2]: STANDARD(MOUSEOVER), Text="2", Val=150
ctl[3]: STANDARD(MOUSEOVER), Text="3", Val=225

OnCtl(n): {

  // When the mouse is moved over
  // one of the slider controls
  // it will light up with focus.

  if (e == FME_MOUSEOVER) {
    setCtlFocus(n);
  }
  return true;
}
```

setCtlFont

Requires FM 1.0 Beta 9.1 (Feb 2009) or newer

Syntax

```
int setCtlFont(int n, int i)
```

Arguments

n
: The number of the control to set the font for

i
: The index number of the font (from 0 - 31) to use, or -1 for the default GUI font.

Return

Returns true if setting the font succeeded, false otherwise.

Description

Sets the font that the control will use when drawing text. The font must first have been created/assigned using the **createFont** function.

Comment

Behind the scenes, FilterMeister sends the **[WM__SETFONT]** message to the control via the **[SendMessage]** Win32 function.

Example

```
// Make control #10 use the default Windows UI font
setCtlFont(10, -1);
```

See Also

createFontdeleteFont

setCtlFontColor

Syntax

```
setCtlFontColor(int index, int color)
```

Arguments

index
: The index of the control whose font color you wish to change.

color
: The color to change the font text to.

Description

This function changes the text color of the user control with the given index. The color value can be an RGB triplet (as in RGB(r,g,b)) or a COLOR function (as in COLOR(DarkGreenCopper)). The following user controls support text coloring: standard scrollbars, checkboxes, radio buttons, group boxes and static text.

Example

```
//set text color to yellow
setCtlFontColor( 0, RGB(255,255,0) );

//set text color to "Money Green"
setCtlFontColor( 0, COLOR(MoneyGreen) );

//set text color to default
setCtlFontColor( 0, -1 );
```

See Also

setCtlColor

setCtlFontSysColor

Syntax

```
int setCtlFontSysColor(int n, int colorIndex)
```

Arguments

n
: The number of the control that will have its text color changed

colorIndex
: The display element color code that will be applied to the text of the control

Display Element Values

These values can also be found in the official **[MSDN GetSysColor]** docs.

COLOR_3DDKSHADOW	Dark shadow for three-dimensional display elements.
COLOR_3DFACE	Face color for three-dimensional display elements and for dialog box backgrounds.
COLOR_3DHIGHLIGHT	Highlight color for three-dimensional display elements (for edges facing the light source.)
COLOR_3DHILIGHT	Highlight color for three-dimensional display elements (for edges facing the light source.)
COLOR_3DLIGHT	Light color for three-dimensional display elements (for edges facing

	the light source.)
COLOR_3DSHADOW	Shadow color for three-dimensional display elements (for edges facing away from the light source).
COLOR_ACTIVEBORDER	Active window border.
COLOR_ACTIVECAPTION	Active window title bar.
COLOR_APPWORKSPACE	Background color of multiple document interface (MDI) applications.
COLOR_BACKGROUND	Desktop.
COLOR_BTNFACE	Face color for three-dimensional display elements and for dialog box backgrounds.
COLOR_BTNHIGHLIGHT	Highlight color for three-dimensional display elements (for edges facing the light source.)
COLOR_BTNHILIGHT	Highlight color for three-dimensional display elements (for edges facing the light source.)
COLOR_BTNSHADOW	Shadow color for three-dimensional display elements (for edges facing away from the light source).
COLOR_BTNTEXT	Text on push buttons.
COLOR_CAPTIONTEXT	Text in caption, size box, and scroll bar arrow box.
COLOR_DESKTOP	Desktop.
COLOR_GRADIENTACTIVECAPTION	Right side color in the color gradient of an active window's title bar.
COLOR_GRADIENTINACTIVECAPTION	Right side color in the color gradient of an inactive window's

	title bar.
COLOR_GRAYTEXT	Grayed (disabled) text.
COLOR_HIGHLIGHT	Items selected in a control.
COLOR_HIGHLIGHTTEXT	Text of items selected in a control.
COLOR_HOTLIGHT	Color for a hyperlink or hot-tracked item.
COLOR_INACTIVEBORDER	Inactive window border.
COLOR_INACTIVECAPTION	Inactive window caption.
COLOR_INACTIVECAPTION TEXT	Color of text in an inactive caption.
COLOR_INFOBK	Background color for tooltip controls.
COLOR_INFOTEXT	Text color for tooltip controls.
COLOR_MENU	Menu background.
COLOR_MENUHILIGHT	The color used to highlight menu items when the menu appears as a flat menu. *(Not supported on Windows 2000.)*
COLOR_MENUBAR	The background color for the menu bar when menus appear as flat menus. *(Not supported on Windows 2000.)*
COLOR_MENUTEXT	Text in menus.
COLOR_SCROLLBAR	Scroll bar gray area.
COLOR_WINDOW	Window background.
COLOR_WINDOWFRAME	Window frame.
COLOR_WINDOWTEXT	Text in windows.

Return

Returns TRUE if the function succeeded.

Description

Changes the text color of a control to a specific Windows UI color, dependent on the user's UI theme and color scheme settings.

Example

```
%fml

ctl[0]: STANDARD, "My Light Blue Text"

OnFilterStart: {
  // Make scrollbar light blue
  setCtlFontSysColor(0, COLOR_HIGHLIGHT);
  return false;
}
```

See Also

getSysColor, **setCtlColor**, **setCtlSysColor**, **setDialogColor**

setCtlGamma

Syntax

```
int setCtlGamma(int num, int gamma)
```

Arguments

num
: The index number of the user control you want to change

gamma
: A value between 1 and 1000. *Default = 100*

Return

1 if the function completed correctly, **0** if the control is not a **STANDARD** or **SCROLLBAR** control, **-1** if the given control number is not a valid control.

Description

Changes how the minimum and maximum values of a **STANDARD** or **SCROLLBAR** control are spread out across the slider control. Changing the gamma value is useful if you want the user to have more precision at one end of the slider values - for example, a blur where the user needs precision over small blur radius values between 1 - 2 pixels, but the difference between very large radius values doesn't matter so much. The default gamma value is 100, where the control value changes linearly across the entire slider.

Example

```
%fml
ctl[3]: CHECKBOX, Text="Higher Gamma"
```

```
ctl[4]: STANDARD, Text="Control", gamma=50, divisor=10

OnCtl(n): {

  // If Checkbox value changed
  if (n == 3) {
    if (ctl(3) == true) {
      // Change slowly at start
      // quickly at end
      setCtlGamma(4, 500);
    }
    else {
      // Change quickly at start
      // slowly at end
      setCtlGamma(4, 50);
    }
    return true;
  }
  return false;
}
```

See Also

setCtlDivisor

setCtlImage

Syntax

```
setCtlImage(int ctl, string filename, string type)
```

Arguments

ctl
: The number of the control whose image you want to change.

filename
: The filename of the image you want to display.

type
: The type of the image file you want to display (see below).

Description

setCtlImage changes the image displayed by an **IMAGE**, **METAFILE**, **BITMAP**, or **ICON** control at run time. The type is a single char constant that specifies the type of image file, as follows:

```
'B' - bitmap file (.bmp)
'W' - Windows (old-style) metafile (.wmf)
'E' - enhanced metafile (.emf)
'I' - icon file (.ico)
'C' - cursor file (.cur)
'J' - JPEG file (.jpg)
'G' - GIF file (.gif)
'M' - MIG (mouse-ivo graphics) file (.mig)
'0' - unspecified file type
```

Example

```
// Sets control number 8 to display the progress5.bmp
// bitmap image, stored in the bitmaps subdirectory.
setCtlImage(8, "bitmaps\\progress5.bmp", 'B');
```

Comments

- Only the 'B', 'W', 'E', and 'I' file types are recognized in the current release. The remaining types are reserved for future use.
- If you have problems embedding an image in your plug-in, check that you have used double backslashes (ie the \ character) in the pathname of the image file, everywhere you use it. Although using forward slashes will work for loading external image files, it will not work for embedded images.
- Also check that the filename you use in setCtlImage is the same one that you embedded. If you use a full pathname when embedding, you must also use the full pathname when calling setCtlImage.

See Also

BITMAP, **IMAGE**, **METAFILE**, **ICON**

setCtlLineSize

Syntax

```
setCtlLineSize(int index, int size)
```

Arguments

index
The index of the control whose line jump unit you wish to change.

size
The new size of the line jump unit.

Description

This function changes the line jump unit of a user control (e.g. the amount the control value increases by when clicking on the arrow buttons of a scrollbar). setCtlLineSize only applies to the user controls **STANDARD** and **SCROLLBAR**.

Example

```
setCtlLineSize(0, 12);
setCtlLineSize(5, 2);
```

See Also

setCtlPageSize

setCtlOrder

Requires FM 1.0 Beta 9.1 (February 2009) or newer

Syntax

```
setCtlOrder(int n, int order)
```

Arguments

n
: The index of the user control whose order you wish to change.

order
: The z-order you wish to set the control to.

Description

Sets the z-order of control n, ordered with 1 on top. Controls with higher orders are moved down accordingly. (This is unlike the usual definition of z-order, ordered with 1 on the bottom.)

Example

```
%ffp

Dialog: Size=(590,300)

ctl(1): Rect(Modalframe, White), pos=(380,50), size=(100,
100)
ctl(2): Rect(Modalframe, Gray), pos=(430,70), size=(100,
100)
ctl(3): Rect(Modalframe, Black), pos=(405,100), size=(100,
100)
```

```
ctl(4): Pushbutton, text="White", Pos=(386,283),
Anchor=ANCHOR_RIGHT|ANCHOR_BOTTOM
ctl(5): Pushbutton, text="Gray", Pos=(423,283),
Anchor=ANCHOR_RIGHT|ANCHOR_BOTTOM
ctl(6): Pushbutton, text="Black", Pos=(460,283),
Anchor=ANCHOR_RIGHT|ANCHOR_BOTTOM

OnCtl(n):
{

  if (n==4)
  {
    setCtlOrder(1, 1); // white on top
  }

  else if (n==5)
  {
    setCtlOrder(2, 1); // gray on top
  }

  else if (n==6)
  {
    setCtlOrder(3, 1); // black on top
  }

  return false;
}


ForEveryTile:
{

  return true;
}
```

setCtlPageSize

Syntax

```
setCtlPageSize(int index, int size)
```

Arguments

index
: The index of the control whose line jump unit you wish to change.

size
: The new size of the page jump unit.

Description

This function changes the page jump unit of a user control (e.g. the amount the control value changes by when clicking between the arrow buttons and handle of a scrollbar). Applies only to the user controls **STANDARD**, **SCROLLBAR** and **TRACKBAR**.

Example

```
setCtlPageSize(0, 12);
setCtlPageSize(5, 2);
```

See Also

setCtlLineSize

setCtlPixelPos

Requires FM 1.0 Beta 9.1 (Feb 2009) or newer

Syntax

```
setCtlPixelPos(int n, int x, int y, int w, int h)
```

Arguments

n
: The number of the user control to resize/reposition

x
: The horizontal co-ordinate of the user control's new position (in pixels)

y
: The vertical co-ordinate of the user control's new position (in pixels)

w
: The new width of the user control (in pixels)

h
: The new height of the user control (in pixels)

Description

This function repositions and resizes the user control with the index n. The values x, y represent the horizontal and vertical position and w, h the width and height of the user control. Set the value to -1 to use the default or last defined measurement. With this function, all measurements are set in pixels - use **setCtlPos** if you want to use Dialog Box Units / DBUs instead.

Example

```
setCtlPixelPos(4, 230, 20, 40, 30);
setCtlPixelPos(5, -1, -1, 50, -1);   //change only the
width of user control 5
```

See Also

setCtlPos

setCtlPos

Syntax

```
setCtlPos(int n, int x, int y, int w, int h)
```

Arguments

n
: The number of the user control to resize/reposition

x
: The horizontal co-ordinate of the user control's new position (in DBUs)

y
: The vertical co-ordinate of the user control's new position (in DBUs)

w
: The new width of the user control (in DBUs)

h
: The new height of the user control (in DBUs)

Description

This function repositions and resizes the user control with the index n. The values x, y represent the horizontal and vertical position and w, h the width and height of the user control. Set the value to -1 to use the default or last defined measurement. Don't forget that all measurements are set in DBUs (dialog box units).

Example

```
setCtlPos(4, 230, 20, 40, 30);
setCtlPos(5, -1, -1, 50, -1);   //change only the width of
```

```
user control 5
```

setCtlProperties

Syntax

```
int setCtlProperties(int n, int properties);
```

Arguments

n
The index of the control whose properties you wish to change.

properties
The properties you wish to set in the control.

Return

Returns an integer with the previous properties before they were changed by the function, or -1 if the control number is out of range or if the control doesn't exist.

Description

This function sets the properties of a control. The properties to be set should be specified in the second argument. To remove a property from a control, use **clearCtlProperties**.

The following properties are available:

- CTP_PREVIEWDRAG
- CTP_MOUSEMOVE
- CTP_DRAWITEM
- CTP_CONTEXTMENU
- CTP_MOUSEOVER
- CTP_EXTENDEDUI

- CTP_SIDE_MASK
- CTP_RIGHT
- CTP_LEFT
- CTP_BOTTOM
- CTP_TOP
- CTP_ORIENT_MASK
- CTP_VERT
- CTP_HORZ
- CTP_READONLY
- CTP_TRACK

Example

```
// Enable tracking for this control
setCtlProperties(6, CTP_TRACK);
```

See Also

clearCtlProperties

setCtlRange

Syntax

```
setCtlRange(int n, int lo, int hi)
```

Arguments

n
: The number of the control whose range you wish to change.

lo
: The minimum range value

hi
: The maximum range value

Description

This function changes the range of the user control with the index n. You cannot change the range of all controls however - for example, you cannot change the range of a checkbox, since it would make no sense to do so.

Example

```
// Changes user control 0 to take values from -128 to 128.
setCtlRange(0, -128, 128);

// Changes user control 23 to take values from 5 to -5.
setCtlRange(23, 5, -5);
```

See Also

setCtlVal, **setCtlDivisor**

setCtlScripting

Requires FM 1.0 Beta 9.1 (February 2009) or newer

Syntax

```
int setCtlScripting(int n, int state)
```

Arguments

n
: The control number that will have scripting enabled/disabled

flags
: Set to **true / 1** to enable scripting, or **false / 0** to disable scripting.

Return

Returns false if the control number is out of range or not in use, true otherwise.

Description

Enables and disables scripting for a user control.

Example

```
setCtlScripting(0, true);
setCtlScripting(1, false);
```

setCtlSysColor

Syntax

```
int setCtlSysColor(int n, int colorIndex)
```

Arguments

n
The number of the control that will have its color changed

colorIndex
The display element color code that will be applied to the control

Display Element Values

These values can also be found in the official [MSDN GetSysColor] docs.

COLOR_3DDKSHADOW	Dark shadow for three-dimensional display elements.
COLOR_3DFACE	Face color for three-dimensional display elements and for dialog box backgrounds.
COLOR_3DHIGHLIGHT	Highlight color for three-dimensional display elements (for edges facing the light source.)
COLOR_3DHILIGHT	Highlight color for three-dimensional display elements (for edges facing the light source.)
COLOR_3DLIGHT	Light color for three-dimensional display elements (for edges facing the light source.)

COLOR_3DSHADOW	Shadow color for three-dimensional display elements (for edges facing away from the light source).
COLOR_ACTIVEBORDER	Active window border.
COLOR_ACTIVECAPTION	Active window title bar.
COLOR_APPWORKSPACE	Background color of multiple document interface (MDI) applications.
COLOR_BACKGROUND	Desktop.
COLOR_BTNFACE	Face color for three-dimensional display elements and for dialog box backgrounds.
COLOR_BTNHIGHLIGHT	Highlight color for three-dimensional display elements (for edges facing the light source.)
COLOR_BTNHILIGHT	Highlight color for three-dimensional display elements (for edges facing the light source.)
COLOR_BTNSHADOW	Shadow color for three-dimensional display elements (for edges facing away from the light source).
COLOR_BTNTEXT	Text on push buttons.
COLOR_CAPTIONTEXT	Text in caption, size box, and scroll bar arrow box.
COLOR_DESKTOP	Desktop.
COLOR_GRADIENTACTIVECAPTION	Right side color in the color gradient of an active window's title bar.
COLOR_GRADIENTINACTIVECAPTION	Right side color in the color gradient of an inactive window's title bar.

COLOR_GRAYTEXT	Grayed (disabled) text.
COLOR_HIGHLIGHT	Items selected in a control.
COLOR_HIGHLIGHTTEXT	Text of items selected in a control.
COLOR_HOTLIGHT	Color for a hyperlink or hot-tracked item.
COLOR_INACTIVEBORDER	Inactive window border.
COLOR_INACTIVECAPTION	Inactive window caption.
COLOR_INACTIVECAPTIONTEXT	Color of text in an inactive caption.
COLOR_INFOBK	Background color for tooltip controls.
COLOR_INFOTEXT	Text color for tooltip controls.
COLOR_MENU	Menu background.
COLOR_MENUHILIGHT	The color used to highlight menu items when the menu appears as a flat menu. *(Not supported on Windows 2000.)*
COLOR_MENUBAR	The background color for the menu bar when menus appear as flat menus. *(Not supported on Windows 2000.)*
COLOR_MENUTEXT	Text in menus.
COLOR_SCROLLBAR	Scroll bar gray area.
COLOR_WINDOW	Window background.
COLOR_WINDOWFRAME	Window frame.
COLOR_WINDOWTEXT	Text in windows.

Return

Returns TRUE if the function succeeded.

Description

Changes the color of a control to a specific Windows UI color, dependent on the user's UI theme and color scheme settings.

Example

```
%fml

ctl[0]: SCROLLBAR

OnFilterStart: {
  // Make scrollbar light blue
  setCtlSysColor(0, COLOR_HIGHLIGHT);
  return false;
}
```

See Also

getSysColor, **setCtlColor**, **setDialogColor**

setCtlTab

Syntax

```
int setCtlTab(int n, int t, int s)
```

Arguments

n
: The number of the control to add to a tab sheet

t
: The number of the tab to add the control to

s
: The number of the tabsheet to add the control to

Return

Returns false if control number **n** is out of range or not in use, or if **t** is not a valid tab control. Returns true otherwise.

Description

Assigns a user control to a specific tab control & tab sheet on that control.

Comment

Due to the design of the **getCtlTab** function, the tab control index number should be 1 or higher. If the tab is control #0, you won't be able to use **getCtlTab** to retrieve the sheet number the control is on.

Example

```
%fml
ctl[1]: TAB, Text="page0\npage1", Pos=(250, 5), Size=(200, 
100)
ctl[2]: STANDARD, Text="Slider", Pos=(280,*)
ctl[8]: PUSHBUTTON, Text="Move slider to other tab sheet", 
Pos=(250, 120), Size=(150,*)

OnCtl(n): {
  if (n==8 && e==FME_CLICKED) {
    // If not already on page1,
    // move slider to page1
    if (getCtlTab(2, 1) != 1) {
      setCtlTab(2, 1, 1);
    }
    else {
      // Move slider to page0
      setCtlTab(2, 1, 0);
    }
  }
  return false;
}
```

See Also

getCtlTab, TAB

setCtlText

Syntax

```
setCtlText(int ctlnum, string text)
```

Arguments

ctlnum
: The number of the control whose text you would like to change.

text
: A string containing the text you'd like the label text changed to.

Description

This function changes the text property of the user control with the index ctlnum. Note that not all user controls support text strings. Also keep in mind that the user control has to be resized if your text string exceeds the current size. Substrings and escape sequences are allowed.

Example

```
setCtlText(1, "Rewrite me");
setCtlText(CTL_OK, "Apply");  //rename OK pushbutton text
to Apply
```

See Also

setCtlTextv

setCtlTextv

Syntax

```
bool setCtlTextv(int ctlnum, string textv, ...)
```

Arguments

ctlnum
: The number of the control whose text you want to change.

textv
: A string containing the formatted text you want the label text changed to.

...
: Additional variables to substitute for the formatting symbols in **textv**

Return

Returns false/zero if the text couldn't be set, true/non-zero otherwise.

Description

Sets the text label for user control (with index number **ctlnum**) to the expanded printf-style formatted string. Note that not all user controls support text strings. Also keep in mind that the user control has to be resized if your text string exceeds the current size. Substrings and escape sequences are allowed.

Example

```
%ffp
```

```
ctl(10): STATICTEXT, pos=(*,60),fontcolor=red,size=(150,*)

ForEveryTile:{

  const int startclock = clock();
  int endclock;

  for (y = y_start; y < y_end; y++) {
  for (x = x_start; x < x_end; x++) {

    // Effect code goes here

  }}

  endclock = clock() - startclock;
  setCtlTextv(10, "Render time needed: %d %s", endclock,
"ms");

  return true;
}
```

See Also

setCtlText

setCtlTheme

Requires FM 1.0 Beta 8.4 (Sep 2007) or newer

Syntax

```
int setCtlTheme(int n, int state)
```

Arguments

n
: The index number of the control to change

state
: Theme state of the control. Set to **-1** for the default theme, **0** to disable themes, or **1** to enable Windows Themes on this control.

Return

Returns -1 if the control number is out of range or not in use, otherwise it returns S_OK if the function succeeded, or an HRESULT error code as returned by the **SetWindowTheme Win32 API function**.

Description

Enables or disables Windows XP/Vista themes for a specific control.

Comment

Note that this function may not work correctly at the moment.

Example

```
%ffp

Category:   "FilterMeister"
Author:     "Harald Heim"
Title:      "Theme Demo"
Filename:   "ThemeDemo.8bf"
Copyright:  "No Copyright"
Description:"Theme Demo"
Version:    "1.0"
URL:        "http://thepluginsite.com"
About:      "!T \n!D\n"
            "!c\n!U"

Dialog: Theme=on

ctl(0): "Slider"//,theme=off
ctl(1): CHECKBOX, "Checkbox"//, color=COLOR_BTNFACE
ctl(2): STATICTEXT, "statictext"
ctl(3): PUSHBUTTON, "Button"//,theme=on
ctl(5): COMBOBOX, "Combobox", val=0

ctl(8): CHECKBOX, "Activate Theme"


OnCtl(n):{

  if (n==8 && e==FME_CLICKED){

    setDialogTheme(ctl(8));

    setCtlTheme(CTL_PROGRESS,ctl(8));
    setCtlTheme(CTL_ZOOM,ctl(8));

    setCtlTheme(CTL_FRAME,ctl(8));
    refreshCtl(CTL_FRAME);//Avoid Redraw problems
    setCtlTheme(0,ctl(8));
    setCtlTheme(1,ctl(8));
    setCtlTheme(2,ctl(8));
    setCtlTheme(3,ctl(8));
    setCtlTheme(5,ctl(8));

    //Info ("%d",getCtlColor(1));

  }

  return false;
}
```

setCtlThumbSize

Syntax

```
setCtlThumbSize(int n, int v)
```

Arguments

n
: The index of the user control whose value you want to change

v
: The value you want to assign to the user control

Description

This function changes the setting of the thumb size of standard and scrollbar controls.

Example

```
setCtlThumbSize(4, 10);
setCtlThumbSize(5, 1);
```

setCtlTicFreq

Syntax

```
int setCtlTicFreq(int n, int m)
```

Arguments

n
: The index of the trackbar control to change.

m
: The interval between tick marks.

Return

Returns true / 1 if the function succeed. Returns -1 if the control number is out of range or not in use. Returns false if the control is not a **TRACKBAR** or **SLIDER** control.

Description

Sets the frequency with which tick marks are displayed for slider control n. For example, if the frequency is set to 2, a tick mark is displayed for every other increment in the slider's range. The default setting for the frequency is 1 (that is, every increment in the range is associated with a tick mark).

Comments

The **SLIDER** or **TRACKBAR** must have the AUTOTICKS style set for this function to work.

Example

```
%fml
ctl[0]: TRACKBAR(AUTOTICKS), Range=(0,100)

OnFilterStart: {
  // Set a tick mark at 20, 40, 60 etc.
  setCtlTicFreq(0, 20);
  return true;
}
```

See Also

TRACKBAR, **setCtlVal**

setCtlToolTip

Syntax

```
setCtlToolTip(int index, string text, int style)
```

Arguments

index
The index of the control to display this ToolTip.

text
The text that the ToolTip should display. If this argument is NULL or "", the ToolTip for this control is deleted.

style
Determines the style of the ToolTip.

Description

This function changes or deletes the tool tip text and style for the user control with the given index.

The following styles can be set (and may be OR-ed together):

symbolic value	numeric	style
(default value)	0	Shows ToolTip under cursor.
TTF_CENTERTIP	0x0002	Shows ToolTip under user control (centered).
TTF_RTLREADING	0x0004	Displays right-to-left text.
TTF_TRACK	0x0020	ToolTip tracks the cursor.
TTF_ABSOLUTE	0x0080	Causes the ToolTip window to be displayed at specific coordinates. *(not yet implemented)*

TTF_TRANSPARENT	0x0100	Causes the ToolTip control to forward mouse event messages to the parent window.

Comments

Tooltips may not always work, due to a bug in some FilterMeister versions. They work on Windows 98SE, but not on Windows 2000 or Windows XP. This bug was fixed as of FilterMeister 1.0 Beta 8.7.

Example

```
setCtlToolTip(0, "Follow me!!", 0);
setCtlToolTip(5, "Center me!!", TTF_CENTERTIP);
```

See Also

setCtlColor

setCtlVal

Syntax

```
int setCtlVal(int n, int v)
```

Arguments

n
: The index of the user control whose value you want to change

v
: The value you want to assign to the user control

Return

Returns the previous value of control **n**, or -1 if **n** is not a valid control index.

Description

This function changes the current value of the user control with the index **n**. Note that the value set has to be one within the user control's range. Also note that not all user controls support value settings (for example, you can only save values in a **FRAME** user control when it is disabled).

Comment

setCtlVal() sets COMBOBOX control to 0 after 'OK'

- A user has reported this problem. What happens is that when you call setCtlVal() for a **COMBOBOX** or a **LISTBOX**, FM not only attempts to set the control to the desired value, but it also immediately reads back the value of the control and stores the read-back value as the new value for the control -- just in case, for example, you specify a value greater than the maximum value of the control, in which case the value is set to the maximum value instead.

- Now, if you call setCtlVal() after 'Ok' has been pressed, the actual control no longer exists (because the filter dialog has already been destroyed). In this case, the read-back value is 0, and that is what is cached as the new current value of the control, and what will be returned by a call to **ctl** or **getCtlVal**.
- Note that this control read-back occurs only for certain controls (e.g., **COMBOBOX** and **LISTBOX**), and not for others (such as sliders), so your results of calling setCtlVal() in this situation will vary, but in general it is a good idea to avoid calling setCtlVal() when the proxy preview dialog is not being displayed.
- Since this behavior, although "correct", is obviously not what the Filter Designer might be expecting, we will change the behavior of setCtlVal() in a future release of FM to *always* cache the requested value of a control in this situation.
- In the meantime, a suggested workaround is to use the **doingProxy** flag to bypass the "bug" in such a situation, as follows:

```
//avoid calling setCtlVal when no dialog is displayed
if (doingProxy) setCtlVal(n, v);
```

Example

```
setCtlVal(4, 30);
int iPrev = setCtlVal(5, 0);
```

See Also

ctl, **getCtlVal**

setDialogColor

Syntax

`setDialogColor(int color)`

Arguments

color
: The color to set the background dialog window to.

Description

Changes the background color of the main filter window.

Example

```
// Set the background window color
// to deep blue
setDialogColor(RGB(0,0,96));
```

See Also

setDialogImage, **setDialogGradient**

setDialogDragMode

Syntax

```
int setDialogDragMode(int mode)
```

Arguments

mode
: Integer setting the dialog drag mode

Return

Always returns true

Description

Sets how the filter dialog box can be dragged around the screen. Valid values are 0 (title bar only), 1 (background and title) and 2 (dragging disabled). The default mode is title bar only (0).

Example

```
// Make the dialog draggable when
// clicking & holding on the
// background
setDialogDragMode(1);
```

See Also

setDialogImage, **setDialogColor**

setDialogEvent

Syntax

```
int setDialogEvent(int state)
```

Arguments

state
An integer of bitwise flags of events to enable. Set to **1** to enable Init events, **2** to enable Cancel events, **4** to enable Keypress events.

Return

Always returns true

Description

Enables certain dialog events to be processed / triggered in your own FilterMeister code. You can use this function to handle when keys are pressed while your filter dialog has user focus, to run some code when the filter is first initialized, or to run some code before the Cancel button/event is processed.

Comment

Note that setDialogEvent can only enable events, it cannot remove / deregister / clear them. To clear an event hook, you must use **clearDialogEvent**. (Behind the scenes, setDialogEvent does a bitwise-OR operation against the internal event state variable, to ensure that previously set bits enabling events remain enabled.)

There appears to be a bug in Beta 9g and possibly newer FilterMeister versions that prevents **FME_CANCEL** events from being processed in code, even if you use this function to enabled Cancel events.

Example

```
ctl[0]: CHECKBOX, "Enable Init Events"
ctl[1]: CHECKBOX, "Enable Cancel Events"
ctl[2]: CHECKBOX, "Enable Keypress Events"
ctl[4]: STATICTEXT, ""
ctl[5]: STATICTEXT, ""


OnCtl(n): {

  if (e == FME_INIT) {
    Info("The Init event was intercepted.");
    return true;
  }

  if (e == FME_CANCEL) {
    Info("Cancel event was intercepted!");
    return true;
  }

  if (e == FME_KEYDOWN) {
    printf("Key down!");
    return true;
  }


  if (n >= 0 && n <= 2 && e == FME_CLICKED) {

    int statevalue = ctl(0)*1 + ctl(1)*2 + ctl(2)*4;

    // Enable events we turned on
    setDialogEvent( statevalue );
    sprintf(str1, "setDialogEvent(%d) called",
statevalue);
    setCtlText(4, str1);

    // Clear events that we turned off
    clearDialogEvent( statevalue ^ 7); // Use bitwise XOR
    sprintf(str2, "clearDialogEvent(%d) called",
statevalue ^ 7);
    setCtlText(5, str2);
  }

  return false;
}
```

See Also

clearDialogEvent, **FME_INIT**, **FME_CANCEL**, **FME_KEYDOWN**, **FME_KEYUP**

setDialogGradient

Syntax

```
setDialogGradient(int color1, int color2, int direction)
```

Arguments

color1
    The color the background dialog color gradient starts at.
color2
    The color the background color gradient fades to.
direction
    The direction the gradient fades in: 0 for vertical gradient, 1 for horizontal

Description

Changes the background window color to a color gradient between the first & second color, in either vertical or horizontal manner.

Example

```
// Set the background window from
// bright blue to black
setDialogGradient( RGB(0,0,255), RGB(0,0,0), 0);
```

See Also

setDialogImage, setDialogColor

setDialogImage

Syntax

```
setDialogImage (string filename)
```

Arguments

filename
The full pathname of the image to be applied to the dialog background.

Return

Returns non-zero if the function succeeds, zero if the function fails.

Description

This function changes the image used in the filter dialog background. Under some conditions, e.g. STRETCHED dialog attribute, this could take some time to redraw - by using lockWindow(1) in front of the code and lockWindow(0) afterwards, the changes should only take a few milliseconds.

Example

```
setDialogImage( "bitmaps\\myDialogBackground.bmp" );
```

See Also

setDialogImageMode

setDialogImageMode

Syntax

```
setDialogImageMode(int mode, int stretchMode)
```

Arguments

mode
: How the background image will be displayed. 0 for exact size, 1 for tiled, 2 for stretched-to-fit.

stretchmode
: The algorithm used to stretch the bitmap to fit.

Description

Changes how the background image is displayed / resized in the filter dialog background. Under some conditions, e.g. STRETCHED dialog attribute, this could take some time to redraw - by using **lockWindow**(1) in front of the code and **lockWindow**(0) afterwards, the changes should only take a few milliseconds.

The stretchMode parameter takes values from the Windows SetStretchBltMode function (see **the MSDN reference**). There is almost no good reason to use anything except the COLORONCOLOR algorithm, which is the default, so it is best to leave stretchMode set to 0 at all times.

Example

```
// Make the background image tiled / repeating
setDialogImageMode(1, 0);
```

```
// Make the background image stretch
// to fit the window
setDialogImageMode(2, 0);
```

See Also

setDialogImage

setDialogMinMax

Syntax

```
setDialogMinMax(int minX, int minY, int maxX, int maxY)
```

Arguments

minX
: The minimum width (in DBUs) the filter dialog can be resized to

minY
: The minimum height (in DBUs) the filter dialog can be resized to

maxX
: The maximum width (in DBUs) the filter dialog can be resized to

maxY
: The maximum height (in DBUs) the filter dialog can be resized to

Description

Sets the minimum and maximum width/height that the filter dialog can be resized to.

Example

```
setDialogMinMax( getDialogWidth(), getDialogHeight(),
getDialogWidth(), getDialogHeight() );
```

See Also

getDialogWidth, **getDialogHeight**

setDialogPos

Syntax

```
setDialogPos(bool fAbs, int x, int y, int w, int h)
```

Arguments

fAbs
: A Boolean flag indicating whether x and y are absolute screen coordinates (fAbs == true), or whether they are relative to the client area (fAbs == false). If x == -1 and y == -1, then fAbs indicates whether the dialog is to be centered within the host client area (fAbs == false) or the working area of the entire screen (fAbs == true).

x
: The x coordinate of the upper-left corner of the filter dialog (in DBUs).

y
: The y coordinate of the upper-left corner of the filter dialog (in DBUs).

w
: The width of the filter dialog (in DBUs).

h
: The height of the filter dialog (in DBUs).

Description

Sets the position and size of the filter dialog window. If fAbs is true, x and y are absolute screen coordinates; otherwise, x and y are relative to the upper-left corner of the client area in the host application's main window. If x and y are set to -1, the dialog window will be centered within the host client area or the working area of the entire screen, depending on whether fAbs is false or true, respectively. Otherwise, if either x or y is negative, the position of the dialog will not be changed. If w or h is negative, the size of the dialog window will not be changed.

All measurements are in dialog box units (DBUs).

Comments

The use of negative coordinates as flag values conflicts with window coordinates in a multi-monitor environment. These flag values may change in a future release.

Examples

```
/* Set dialog to absolute position (70, 80), width 120,
height 60... */
setDialogPos( true, 70, 80, 120, 60 );

/* Change dialog width to 110, height to 50, without
moving upper-left corner... */
setDialogPos( false, -1, 0, 110, 50 );

/* Center dialog within the host client area, without
changing the size... */
setDialogPos( false, -1, -1, -1, 0 );
```

See Also

getDialogWidth, getDialogHeight, getDialogPos

setDialogRegion

Syntax

```
int setDialogRegion(region R)
```

Arguments

R
A visual region R of any shapes (see comments).

Return

Returns non-zero if the function succeeds, zero if the function fails.

Description

Allows to reshape the dialog, by using a region. In the example code below, the dialog becomes circular.

Comment

"region" correspond to a particular Windows type object that might need to be better documented or defined. You can't directly cast it to an integer, despite it behaving like an integer.

Example

```
%ffp
Dialog: Size=(300,300)
Dialog: Drag = Background
ctl[CTL_CANCEL]: PUSHBUTTON, Text="Cancel", Pos=(55,150),
size=(40,14)
```

```
ctl[CTL_OK]: PUSHBUTTON, Text="OK", Pos=(100,150), size=
(40,14)
ctl[CTL_PREVIEW]: Modify, Pos=(65,25), Size=(100,100)

OnFilterStart:{
  setDialogRegion( createCircularRgn(20, 20, 200) );
  return false;
}

//------ Any effect ....
R:255-R
G:255-G
B:255-B
```

See Also

refreshRgn, **createRectRgn**, **createRoundRectRgn**, **createCircularRgn**, **createEllipticRgn**, **createPolyRgn**

setDialogShowState

Syntax

`bool setDialogShowState(int state)`

Arguments

state
Defines how the dialog window should be shown.

Return

Returns false/zero if the dialog window was previously hidden, true/non-zero otherwise.

Description

Can be used to show, hide, minimize and maximize the filter dialog window. The values for the state parameter are equal to those used in the **[ShowWindow function documented at MSDN]**.

Comment

This function is a lightweight wrapper around the Windows Win32 ShowWindow function, **[documented here on MSDN]**.

Example

```
// Hide the dialog window
setDialogShowState(0);
// SW_HIDE = 0
// Show the dialog window again
setDialogShowState(5);
// SW_SHOW = 5
```

See Also

setDialogStyle

setDialogSizeGrip

Syntax

```
int setDialogSizeGrip(int state)
```

Arguments

state
: Set to **0** to hide the size grip, **1** to enable and show the size grip control.

Return

Always returns true.

Description

Enables or disables the resizing size grip control in the bottom right-hand corner of the window.

Example

```
%fml

ctl(0): CHECKBOX, "Enable Size Grip"

OnCtl(n):{
  setDialogSizeGrip(ctl(0));
  return false;
}
```

setDialogStyle

Syntax

```
int setDialogStyle(int flags)
```

Arguments

flags
    An integer of bitwise flags enabling different window styles

Return

Returns the previous settings of the flags as an integer if the function succeeds. Returns zero if it failed.

Description

Sets (or enables) a particular style on a dialog. For example, you can change the window border to a thin style using WS_BORDER, or to a thick resizeable border using WS_THICKFRAME. The **[window style flags are defined at MSDN]**. You can clear these settings later using clearDialogStyle.

Comments

Internally, SetDialogStyle is a wrapper around the **[SetWindowLong Win32 API function]** with GWL_STYLE as the second parameter.

Example

```
// Change the border to a thin style
setDialogStyle(WS_BORDER);
```

setDialogStyleEx

Syntax

```
int setDialogStyleEx(int flags)
```

Arguments

flags
An integer of bitwise flags enabling different window styles

Return

Returns the previous settings of the flags as an integer if the function succeeds. Returns zero if it failed.

Description

Sets (or enables) a particular extended style on a dialog. For example, you can make a window respond to drag & drop files using WS_EX_ACCEPTFILES. The **[extended window style flags are defined at MSDN]**. You can clear these settings later using **clearDialogStyleEx**.

Comments

Internally, SetDialogStyleEx is a wrapper around the **[SetWindowLong Win32 API function]** with GWL_EXSTYLE as the second parameter.

Example

```
// Make the dialog accept files that are
// dropped onto it via drag and drop
```

```
setDialogStyleEx(WS_EX_ACCEPTFILES);
```

See Also

setDialogStyle, **clearDialogStyle**, **clearDialogStyleEx**

setDialogSysColor

Syntax

```
int setDialogSysColor(int colorIndex)
```

Arguments

colorIndex
The code number of the display element color that will be applied to the dialog background

Display Element Values

These values can also be found in the official [MSDN GetSysColor] docs.

COLOR_3DDKSHADOW	Dark shadow for three-dimensional display elements.
COLOR_3DFACE	Face color for three-dimensional display elements and for dialog box backgrounds.
COLOR_3DHIGHLIGHT	Highlight color for three-dimensional display elements (for edges facing the light source.)
COLOR_3DHILIGHT	Highlight color for three-dimensional display elements (for edges facing the light source.)
COLOR_3DLIGHT	Light color for three-dimensional display elements (for edges facing the light source.)
COLOR_3DSHADOW	Shadow color for three-dimensional display elements (for

	edges facing away from the light source).
COLOR_ACTIVEBORDER	Active window border.
COLOR_ACTIVECAPTION	Active window title bar.
COLOR_APPWORKSPACE	Background color of multiple document interface (MDI) applications.
COLOR_BACKGROUND	Desktop.
COLOR_BTNFACE	Face color for three-dimensional display elements and for dialog box backgrounds.
COLOR_BTNHIGHLIGHT	Highlight color for three-dimensional display elements (for edges facing the light source.)
COLOR_BTNHILIGHT	Highlight color for three-dimensional display elements (for edges facing the light source.)
COLOR_BTNSHADOW	Shadow color for three-dimensional display elements (for edges facing away from the light source).
COLOR_BTNTEXT	Text on push buttons.
COLOR_CAPTIONTEXT	Text in caption, size box, and scroll bar arrow box.
COLOR_DESKTOP	Desktop.
COLOR_GRADIENTACTIVECAPTION	Right side color in the color gradient of an active window's title bar.
COLOR_GRADIENTINACTIVECAPTION	Right side color in the color gradient of an inactive window's title bar.
COLOR_GRAYTEXT	Grayed (disabled) text.
COLOR_HIGHLIGHT	Items selected in a control.

COLOR_HIGHLIGHTTEXT	Text of items selected in a control.
COLOR_HOTLIGHT	Color for a hyperlink or hot-tracked item.
COLOR_INACTIVEBORDER	Inactive window border.
COLOR_INACTIVECAPTION	Inactive window caption.
COLOR_INACTIVECAPTIONTEXT	Color of text in an inactive caption.
COLOR_INFOBK	Background color for tooltip controls.
COLOR_INFOTEXT	Text color for tooltip controls.
COLOR_MENU	Menu background.
COLOR_MENUHILIGHT	The color used to highlight menu items when the menu appears as a flat menu. *(Not supported on Windows 2000.)*
COLOR_MENUBAR	The background color for the menu bar when menus appear as flat menus. *(Not supported on Windows 2000.)*
COLOR_MENUTEXT	Text in menus.
COLOR_SCROLLBAR	Scroll bar gray area.
COLOR_WINDOW	Window background.
COLOR_WINDOWFRAME	Window frame.
COLOR_WINDOWTEXT	Text in windows.

Return

Returns TRUE if the function succeeded.

Description

Changes the dialog background color to that of a specific Windows UI color, dependent on the user's UI theme and color scheme settings.

Example

```
%fml

OnFilterStart: {
  // Make dialog background light blue
  setDialogSysColor(COLOR_HIGHLIGHT);
  return false;
}
```

See Also

getSysColor, **setCtlColor**, **setCtlSysColor**, **setDialogColor**

setDialogText

Syntax

```
bool setDialogText(string title)
```

Arguments

title
The text to be set as the title bar caption.

Return

Returns false/zero if the text couldn't be set, true/non-zero otherwise.

Description

Sets the caption in the title bar of the filter dialog. You can also use **formatString** compatible codes in the text parameter here.

Example

```
setDialogTitle("Your title bar title goes here.");
```

See Also

setDialogTextv, **formatString**, **setCtlText**

setDialogTextv

Syntax

```
bool setDialogTextv(string title, ...)
```

Arguments

title
: The text to be set as the title bar caption.

Return

Returns false/zero if the text couldn't be set, true/non-zero otherwise.

Description

Sets the caption in the title bar of the filter dialog. You can also use printf-style formatting codes in the text parameter, which are filled in by the additional parameters you provide.

Example

```
int version = 1;
strcpy(str0, "Plug-In Name");
strcpy(str1, "Company Name");
setDialogTitlev("%s made by %s, version %d", str0, str1,
version);
```

See Also

setDialogText, formatString, setCtlText

setDialogTheme

Requires FM 1.0 Beta 8.4 (Sep 2007) or newer

Syntax

```
int setDialogTheme(int state)
```

Arguments

state
: Theme state of the control. Set to **0** to disable themes, or **1** to enable Windows Themes on this control.

Return

Returns -1 if the dialog is out of range or not in use, otherwise it returns S_OK if the function succeeded, or an HRESULT error code as returned by the **SetWindowTheme Win32 API function**.

Description

Enables or disables the modern Windows UI theme (from XP/Vista onwards) on the filter dialog window. Note that setting the dialog theme does not apply the theme to the controls on the interface, use setCtlTheme on each control to do this. Note that the dialog might not adopt the modern theme if your host graphics program is running in a compatibility mode emulating an older version of Windows.

Example

```
%ffp
```

```
Category:   "FilterMeister"
Author:     "Harald Heim"
Title:      "Theme Demo"
Filename:   "ThemeDemo.8bf"
Copyright:  "No Copyright"
Description:"Theme Demo"
Version:    "1.0"
URL:        "http://thepluginsite.com"
About:      "!T \n!D\n"
            "!c\n!U"

Dialog: Theme=on

ctl(0): "Slider"//,theme=off
ctl(1): CHECKBOX, "Checkbox"//, color=COLOR_BTNFACE
ctl(2): STATICTEXT, "statictext"
ctl(3): PUSHBUTTON, "Button"//,theme=on
ctl(5): COMBOBOX, "Combobox", val=0

ctl(8): CHECKBOX, "Activate Theme"


OnCtl(n):{

  if (n==8 && e==FME_CLICKED){

    setDialogTheme(ctl(8));

    setCtlTheme(CTL_PROGRESS,ctl(8));
    setCtlTheme(CTL_ZOOM,ctl(8));

    setCtlTheme(CTL_FRAME,ctl(8));
    refreshCtl(CTL_FRAME);//Avoid Redraw problems

    setCtlTheme(0,ctl(8));
    setCtlTheme(1,ctl(8));
    setCtlTheme(2,ctl(8));
    setCtlTheme(3,ctl(8));
    setCtlTheme(5,ctl(8));

    //Info ("%d",getCtlColor(1));

  }

  return false;
}
```

See Also

setCtlTheme

setEllipse

Syntax

```
bool setEllipse(int left, int top, int right, int bottom,
int color)
```

Arguments

left
: Left-most bound of the ellipse, in pixels.

top
: Top-most bound of the ellipse, in pixels.

right
: Right-most bound of the ellipse, in pixels.

bottom
: Bottom-most bound of the ellipse, in pixels.

color
: Color value for the ellipse.

Returns

True if the function succeeded, false otherwise

Description

Render an ellipse or circle to an **OWNERDRAW** control.

Example

```
%fml

ctl(1): OWNERDRAW(drawitem), Pos=(300,50), Size=(100, 100)

```

```
OnFilterStart: {

  // Maximum X/Y pixel co-ords of ownerdraw
  int maxxpixel = HDBUsToPixels(100)-1;
  int maxYpixel = VDBUsToPixels(100)-1;

  // Start using setPixel drawing on OWNERDRAW control 1
  startSetPixel(1);

  // Draw rectangle around OWNERDRAW area
  setRectFrame(0, 0, maxxpixel, maxYpixel, RGB(0,0,0));

  // Draw a filled red ellipse
  setEllipseFill(5, 5, 200, 150, RGB(255, 0, 0));

  // Draw a blue circle
  setEllipse(25, 50, 150, 175, RGB(0, 0, 255));

  // Stop using setPixel drawing
  endSetPixel(1);

  return true;
}
```

See Also

OWNERDRAW, startSetPixel, setEllipseFill, setRectFrame, endSetPixel

setEllipseFill

Syntax

```
bool setEllipseFill(int left, int top, int right, int
bottom, int color)
```

Arguments

left
: Left-most bound of the ellipse, in pixels.

top
: Top-most bound of the ellipse, in pixels.

right
: Right-most bound of the ellipse, in pixels.

bottom
: Bottom-most bound of the ellipse, in pixels.

color
: Color value for the ellipse.

Returns

True if the function succeeded, false otherwise

Description

Render an ellipse or circle filled with the given color to an **OWNERDRAW** control.

Example

```
%fml

ctl(1): OWNERDRAW(drawitem), Pos=(300,50), Size=(100, 100)
```

```
OnFilterStart: {

  // Maximum X/Y pixel co-ords of ownerdraw
  int maxxpixel = HDBUsToPixels(100)-1;
  int maxYpixel = VDBUsToPixels(100)-1;

  // Start using setPixel drawing on OWNERDRAW control 1
  startSetPixel(1);

  // Draw rectangle around OWNERDRAW area
  setRectFrame(0, 0, maxxpixel, maxYpixel, RGB(0,0,0));

  // Draw a filled red ellipse
  setEllipseFill(5, 5, 200, 150, RGB(255, 0, 0));

  // Draw a blue circle
  setEllipse(25, 50, 150, 175, RGB(0, 0, 255));

  // Stop using setPixel drawing
  endSetPixel(1);

  return true;
}
```

See Also

OWNERDRAW, startSetPixel, setEllipse, setRectFrame, endSetPixel

setFill

Syntax

```
int setFill(int color)
```

Arguments

color
An **RGB** a color value to fill the **OWNERDRAW** canvas

Return

Returns true if the function succeeded, false otherwise.

Description

Fills an **OWNERDRAW** control canvas with a solid color. setFill must be surrounded with **startSetPixel** and **endSetPixel** controls, which sets which control the fill command will apply to.

Examples

```
%fml
ctl[0]: STANDARD, "Red", Val=200
ctl[1]: STANDARD, "Green", Val=0
ctl[2]: STANDARD, "Blue", Val=0
ctl[4]: OWNERDRAW, Size=(100,100), Color=RGB(200,0,0)

OnCtl(n): {
  if (n >= 0 && n <= 2) {
    startSetPixel(4);
    setFill( RGB( ctl(0), ctl(1), ctl(2)));
    endSetPixel(0);
  }
  return false;
}
```

See Also

OWNERDRAW, startSetPixel, endSetPixel, RGB

setFont

Syntax

```
void setFont(int size, float rotation, bool bold, bool italic, char* font)
```

Arguments

size
  Horizontal offset from left edge in pixels.
rotation
  Rotation angle in radians.
bold
  Use a bold style.
italic
  Use a italic style.
font
  The name of the font. e.g. "Arial".

Description

Set the font to use for **setText** and **setTextV**.

Comment

This function is only for use when drawing on an **OWNERDRAW** using the **startSetPixel** and **endSetPixel** functions. To change the font associated with a control, use **createFont** and **setCtlFont** instead.

Example

```
%ffp
ctl[0]: OWNERDRAW, Color=RGB(0,0,0), Size=(100,100)

OnFilterStart: {
  startSetPixel(0);
  setFont(17, 0, true, false, "Arial");
  setText(5, 0, RGB(255,255,255), TA_LEFT | TA_TOP,
"Player 1");
  setText(145, 0, RGB(255,255,255), TA_RIGHT | TA_TOP,
"Player 2");
  endSetPixel(0);

  return false;
}
```

See Also

setText, **setTextV**, **startSetPixel**, **endSetPixel**

setGamma

Syntax

```
void setGamma(double gamma)
```

Arguments

gamma
: The gamma correction value.

Description

Precomputes a gamma correction table for gamma value **gamma**, where **gamma** is a float or double value.

If this function is not called, calls to the **gamma** function will use a gamma correction of 1.0, effectively no correction at all.

To be used in conjunction with the **gamma** function.

Example

```
setGamma(1.8);
for (x = x_start; x < x_end; x++) {
  for (y = y_start; y < y_end; y++) {
    for (z = 0; z < 3; z++) {
      pset(x, y, z, gamma(src(x, y, z)));
    }
  }
}
```

See Also

gamma

setLine

Syntax

```
void setLine(int x1, int y1, int x2, int y2, int color)
```

Arguments

x1
: Horizontal offset from the left edge of the start of the line, in pixels.

y1
: Vertical offset from the top edge of the start of the line, in pixels.

x2
: Horizontal offset from the left edge of the end of the line, in pixels.

y2
: Vertical offset from the top edge of the end of the line, in pixels.

color
: Color value for the text.

Description

Render a line to an **OWNERDRAW** control.

Example

```
%fml

ctl[3]: OWNERDRAW(drawitem), Pos=(300,50), Size=(100, 100)

OnFilterStart: {
  const int x = HDBUsToPixels(getCtlPos(n, 3));
  const int y = VDBUsToPixels(getCtlPos(n, 3));

  // Start drawing on the OWNERDRAW
  startSetPixel(3);

  // Fill the OWNERDRAW with black pixels
  setFill(RGB(0,0,0));

  // Draw a diagonal red line
  setLine(0, 0, x, y, COLOR(RED));

  // Stop drawing to the OWNERDRAW
  endSetPixel(3);

  return true;
}
```

See Also

OWNERDRAW, **startSetPixel**, **HDBUsToPixels**, **VDBUsToPixels**, **setFill**, **setLine**, **endSetPixel**

setThinLineAA

Requires FM 1.0 Beta 8.5 (Nov 2007) or newer

Syntax

```
bool setThinLineAA(int x1, int y1, int x2, int y2, int color)
```

Arguments

x1
: Horizontal offset from the left edge of the start of the line, in pixels.

y1
: Vertical offset from the top edge of the start of the line, in pixels.

x2
: Horizontal offset from the left edge of the end of the line, in pixels.

y2
: Vertical offset from the top edge of the end of the line, in pixels.

color
: Color value for the text.

Returns

Always returns true.

Description

Render an anti-aliased line to an **OWNERDRAW** control.

Example

```
%fml

ctl[3]: OWNERDRAW(drawitem), Pos=(300,50), Size=(100, 100)

OnFilterStart: {

  const int x = HDBUsToPixels(getCtlPos(n, 3));
  const int y = VDBUsToPixels(getCtlPos(n, 3));

  // Start drawing on the OWNERDRAW
  startSetPixel(3);

  // Fill the OWNERDRAW with black pixels
  setFill(RGB(0,0,0));

  // Draw a diagonal red line
  setPenWidth(1);
  setLine(x, 0, 0, y, COLOR(RED));

  // Draw a diagonal yellow antialiased line
  setThinLineAA(0, 0, x, y, COLOR(YELLOW));

  // Stop drawing to the OWNERDRAW
  endSetPixel(3);

  return true;
}
```

See Also

OWNERDRAW, startSetPixel, HDBUsToPixels, VDBUsToPixels, setFill, setLine, endSetPixel

setPenWidth

Requires FM 1.0 Beta 8.5 (Nov 2007) or newer

Syntax

```
bool setPenWidth(int width)
```

Arguments

width
: Width of the graphics pen, in device units

Returns

Always returns true.

Description

Sets the width of the graphics pen, used when drawing on an **OWNERDRAW** control using the **startSetPixel** graphics methods.

Example

```
%fml

ctl[3]: OWNERDRAW(drawitem), Pos=(300,50), Size=(100, 100)

OnFilterStart: {

  const int x = HDBUsToPixels(getCtlPos(n, 3));
  const int y = VDBUsToPixels(getCtlPos(n, 3));

  // Start drawing on the OWNERDRAW
  startSetPixel(3);

  // Fill the OWNERDRAW with black pixels
  setFill(RGB(0,0,0));

  // Draw a diagonal line with a width
  // of 4 device units
  setPenWidth(4);
  setLine(0, 0, x, y, COLOR(RED));

  // Stop drawing to the OWNERDRAW
  endSetPixel(3);

  return true;

}
```

See Also

OWNERDRAW, startSetPixel, setAngleArc, setFill, setLine, setRectFrame, endSetPixel

setPixel

Syntax

```
void setPixel(int x, int y, int color)
```

Arguments

x
X-coordinate (in pixels) of the relative position (originating top-left at 0, 0) in the control where you want to draw a pixel.

y
Y-coordinate (in pixels) of the relative position (originating top-left at 0, 0) in the control where you want to draw a pixel.

color
The **RGB** color value of the pixel to be drawn.

Description

Call setPixel to draw an individual pixel to an **OWNERDRAW** control. First call **startSetPixel**, then use setPixel or any of the Control drawing functions to draw to the control and finish with a call to **endSetPixel**. This function is not for setting pixels in the preview / final image; use **pset** for that instead.

See Also

startSetPixel, **endSetPixel**

setPreviewCursor

Syntax

```
setPreviewCursor(int n)
```

Arguments

n

An integer constant describing the cursor image to use.

Description

This function defines which standard Windows pointer to use when the mouse is over the preview image. Suitable values for 'int n' are:

```
103  : Default Windows pointer (hand)
IDC_ARROW    (32512) : Arrow pointer
IDC_IBEAM    (32513) : I beam (text cursor)
IDC_WAIT     (32514) : Busy pointer
IDC_CROSS    (32515) : Cross
IDC_UPARROW  (32516) : Up arrow
IDC_SIZE     (32640) : Size pointer
IDC_HELP     (32641) : pointer with question mark
IDC_SIZENWSE (32642) : NWSE diagonal arrow
IDC_SIZENESW (32643) : NESW diagonal arrow
IDC_SIZEWE   (32644) : WE arrow
IDC_SIZENS   (32645) : NS arrow
IDC_SIZEALL  (32646) : Size all crossed-arrows
IDC_NO       (32648) : NO pointer
IDC_APPSTARTING (32650) : Application Starting pointer
IDC_ICON     (32651) : ? custom cursor?
```

Comments

Bear in mind that when using these cursors, cursors may differ for different machines depending on the user's cursor preferences and any themes running.

Example

```
onFilterStart:{
  // Sets preview cursor to
  // accurate cross-hair
  setPreviewCursor(32515);
  return false;
}
```

See Also

getPreviewCursor

setRectangle

Requires FM 1.0 Beta 9 (Apr 2008) or newer

Syntax

`bool setRectangle(int left, int top, int right, int bottom, int color)`

Arguments

left
: top-left x coordinate of the rectangle.

top
: top-left y coordinate of the rectangle.

right
: bottom-right x coordinate of the rectangle.

bottom
: bottom-right y coordinate of the rectangle.

color
: Color value for the rectangle.

Return

True if the function succeeded, false otherwise

Description

Draw a rectangle with the given color on an **OWNERDRAW** control.

Example

```
%fml

ctl(1): OWNERDRAW(drawitem), Pos=(300,50), Size=(100, 100)

OnFilterStart: {

  // Maximum X/Y pixel co-ords of ownerdraw
  int maxxpixel = HDBUsToPixels(100)-1;
  int maxYpixel = VDBUsToPixels(100)-1;

  // Start using setPixel drawing on OWNERDRAW control 1
  startSetPixel(1);
  setFill(COLOR(BLACK));

  // Draw a yellow rectangle in OWNERDRAW area
  setRectangle(20, 20, maxxpixel-20, maxYpixel-20,
RGB(255,255,0));

  // Stop using setPixel drawing
  endSetPixel(1);

  return true;
}
```

See Also

OWNERDRAW, **startSetPixel**, **setEllipse**, **setEllipseFill**, **setRectFill**, **setRectGradient**, **endSetPixel**

setRectFill

Syntax

```
int setRectFill(int left, int top, int right, int bottom,
UINT color)
```

Arguments

left
: The left co-ordinate of the rectangle to fill in the Ownerdraw control

top
: The top co-ordinate of the rectangle to fill in the Ownerdraw control

right
: The right co-ordinate of the rectangle to fill in the Ownerdraw control

bottom
: The bottom co-ordinate of the rectangle to fill in the Ownerdraw control

color
: An **RGB** value of the color you want to fill the defined rectangle with

Return

to be completed

Description

setRectFill fills a rectangular section of an OWNERDRAW control with a single color.

Example

```
ctl(1): OWNERDRAW(drawitem), pos=(300,50), size=(100, 100)
startSetPixel(1);
setRectFill(10, 20, 30, 40, RGB(255, 0, 0));
endSetPixel(1);
```

See Also

startSetPixel, **setRectGradient**, **endSetPixel**

setRectFrame

Syntax

```
bool setRectFrame(int left, int top, int right, int bottom,
int color)
```

Arguments

left
: top-left x coordinate of the rectangle.

top
: top-left y coordinate of the rectangle.

right
: bottom-right x coordinate of the rectangle.

bottom
: bottom-right y coordinate of the rectangle.

color
: Color value for the rectangle.

Return

True if the function succeeded, false otherwise

Description

Draw a rectangle with the given color on an **OWNERDRAW** control.

Example

```
%fml

ctl(1): OWNERDRAW(drawitem), Pos=(300,50), Size=(100, 100)
```

```
OnFilterStart: {

  // Maximum X/Y pixel co-ords of ownerdraw
  int maxxpixel = HDBUsToPixels(100)-1;
  int maxYpixel = VDBUsToPixels(100)-1;

  // Start using setPixel drawing on OWNERDRAW control 1
  startSetPixel(1);

  // Draw rectangle around OWNERDRAW area
  setRectFrame(0, 0, maxxpixel, maxYpixel, RGB(0,0,0));

  // Draw a filled red ellipse
  setEllipseFill(5, 5, 200, 150, RGB(255, 0, 0));

  // Draw a blue circle
  setEllipse(25, 50, 150, 175, RGB(0, 0, 255));

  // Stop using setPixel drawing
  endSetPixel(1);

  return true;
}
```

See Also

OWNERDRAW, **startSetPixel**, **setEllipse**, **setEllipseFill**, **setRectFill**, **setRectGradient**, **endSetPixel**

setRectGradient

Requires FM 1.0 Beta 9.1 (February 2009) or newer

Syntax

```
int setRectGradient(int left, int top, int right, int
bottom, UINT color_TL, UINT color_BR, int horizontal)
```

Arguments

left
: Left coordinate of the gradient box.

top
: Top coordinate of the gradient box.

right
: Right coordinate of the gradient box.

bottom
: Bottom coordinate of the gradient box.

color_TL
: Color to use for the top or left edge of the gradient.

color_BR
: Color to use for the bottom or right edge of the gradient.

horizontal
: Boolean value determining whether to draw a horizontal (TRUE) or vertical (FALSE) gradient.

Return

Integer containing value TRUE if operation completed successful, otherwise FALSE.

Description

Draw a smooth gradient, either horizontal or vertical on the current ownerdraw or buffer canvas.

See Also

startSetPixel

setRegPath

Syntax

```
int setRegPath(char * path[, args...])
```

Arguments

path
A string specifying the path.

args
printf style arguments with FM extensions.

Return

Returns ERROR_SUCCESS if the operation was successful, otherwise it returns one of the following integer error codes:

ERROR_SUCCESS	(==0) no error
ERROR_FILE_NOT_FOUND	key or value name not found
ERROR_MORE_DATA	buffer wasn't big enough (e.g., getRegString, getRegData)
ERROR_NO_MORE_ITEMS	index >= # of values or subkeys (enumRegValue, enumRegSubKey)
ERROR_INVALID_FUNCTION	bad top-level key, etc
ERROR_INVALID_DATA	wrong data type or size (or size > 2048)
ERROR_BADDB	registry database is corrupt
ERROR_BADKEY	registry key is invalid
ERROR_CANTOPEN	registry key could not be opened

ERROR_CANTREAD	registry key could not be read
ERROR_CANTWRITE	registry key could not be written
ERROR_REGISTRY_CORRUPT	registry is corrupt
ERROR_REGISTRY_IO_FAILED	input/output to registry failed
ERROR_KEY_DELETED	Illegal operation attempted on a Registry key which has been marked for deletion.
ERROR_KEY_HAS_CHILDREN	cannot delete a key with subkeys (Windows NT)

Description

Sets the the current registry path.

By default, FM sets the registry path to "Software\\!O\\!C\\!t", which will expand to "Software\\<Organization>\\<Filter-category>\\<Filter-title>", as recommended by Microsoft.

Example

```
// Sets the registry key and path back to the FM default
values.
setRegRoot(HKEY_CURRENT_USER);
setRegPath("Software\\!O\\!C\\!t");
```

See Also

getRegPath, setRegRoot

setRegRoot

Syntax

```
int setRegRoot(int hKey)
```

Arguments

hKey

The section of the registry you want to access. hKey can be set to either HKEY_LOCAL_MACHINE or HKEY_CURRENT_USER, depending which section you need to access.

Return

Returns ERROR_SUCCESS if the operation was successful, otherwise it returns one of the following integer error codes:

ERROR_SUCCESS	(==0) no error
ERROR_FILE_NOT_FOUND	key or value name not found
ERROR_MORE_DATA	buffer wasn't big enough (e.g., getRegString, getRegData)
ERROR_NO_MORE_ITEMS	index >= # of values or subkeys (enumRegValue, enumRegSubKey)
ERROR_INVALID_FUNCTION	bad top-level key, etc
ERROR_INVALID_DATA	wrong data type or size (or size > 2048)
ERROR_BADDB	registry database is corrupt
ERROR_BADKEY	registry key is invalid
ERROR_CANTOPEN	registry key could not be

	opened
ERROR_CANTREAD	registry key could not be read
ERROR_CANTWRITE	registry key could not be written
ERROR_REGISTRY_CORRUPT	registry is corrupt
ERROR_REGISTRY_IO_FAILED	input/output to registry failed
ERROR_KEY_DELETED	Illegal operation attempted on a Registry key which has been marked for deletion.
ERROR_KEY_HAS_CHILDREN	cannot delete a key with subkeys (Windows NT)

Description

Sets the the current registry root key.

Currently, only two types of registry roots are possible, identified by the constants HKEY_LOCAL_MACHINE and HKEY_CURRENT_USER.

Example

```
// Sets the registry key and path back to the FM default
values.
setRegRoot(HKEY_CURRENT_USER);
setRegPath("Software\\!O\\!C\\!t");
```

See Also

getRegRoot, setRegPath

setText

Syntax

```
void setText(int x, int y, int color, int flags, char* text)
```

Arguments

x
: Horizontal offset from left edge in pixels.

y
: Vertical offset from top edge in pixels.

color
: Color value for the text.

flags
: Any combination of TA_* and VTA_* flags. Available alignment constants are TA_BASELINE, TA_BOTTOM, TA_TOP, TA_CENTER, TA_LEFT, TA_RIGHT, TA_NOUPDATECP, TA_RTLREADING, TA_UPDATECP, VTA_BASELINE, VTA_CENTER

text
: The text to render.

Description

Render text to an ownerdraw control.

Example

```
const int x = HDBUsToPixels(getCtlPos(n, 2));
const int y = VDBUsToPixels(getCtlPos(n, 1));
startSetPixel(3);
setText(x / 2, 0, COLOR(WHITE), VTA_CENTER | TA_BASELINE,
getCtlText(n));
endSetPixel(3);
```

See Also

setFont, **setTextv**

setTextv

Syntax

```
void setTextv(int x, int y, int color, int flags, char* text, ...)
```

Arguments

x
: Horizontal offset from left edge in pixels.

y
: Vertical offset from top edge in pixels.

color
: Color value for the text.

flags
: Any combination of TA_* and VTA_* flags. Available alignment constants are TA_BASELINE, TA_BOTTOM, TA_TOP, TA_CENTER, TA_LEFT, TA_RIGHT, TA_NOUPDATECP, TA_RTLREADING, TA_UPDATECP, VTA_BASELINE, VTA_CENTER

text
: The text to render.

...
: Additional variables to substitute for the formatting symbols in **text**

Description

Renders formatted text to an ownerdraw control.

Example

```
%fml
ctl[0]: OWNERDRAW, color=RGBA(0,0,0,255), Size=(100,100)
ctl[7]: STANDARD, Text="Value", Pos=(*, 120)

ForEveryTile: {
  startSetPixel(0);
  setFill(RGB(0,0,88));
  setTextv(60, 25, RGB(0,0,0), VTA_CENTER | TA_BASELINE,
"Value is: %d", ctl(7));
  endSetPixel(0);
  return true;
}
```

See Also

setFont, **setTextv**

setThinEllipseAA

Syntax

```
bool setThinEllipseAA(int left, int top, int right, int
bottom, int color)
```

Arguments

left
: Left-most bound of the ellipse, in pixels.

top
: Top-most bound of the ellipse, in pixels.

right
: Right-most bound of the ellipse, in pixels.

bottom
: Bottom-most bound of the ellipse, in pixels.

color
: Color value for the ellipse.

Returns

True if the function succeeded, false otherwise

Description

Render an anti-aliased ellipse or circle to an **OWNERDRAW** control.

Example

```
%fml

ctl(1): OWNERDRAW(drawitem), Pos=(300,50), Size=(100, 100)
```

```

OnFilterStart: {

  // Maximum X/Y pixel co-ords of ownerdraw
  int maxxpixel = HDBUsToPixels(100)-1;
  int maxYpixel = VDBUsToPixels(100)-1;

  // Start using setPixel drawing on OWNERDRAW control 1
  startSetPixel(1);

  // Draw rectangle around OWNERDRAW area
  setRectFrame(0, 0, maxxpixel, maxYpixel, RGB(0,0,0));

  // Draw an outline of a red ellipse
  setThinEllipseAA(5, 5, 200, 150, RGB(255, 0, 0));

  // Draw an outline of a blue circle
  setThinEllipseAA(25, 50, 150, 175, RGB(0, 0, 255));

  // Stop using setPixel drawing
  endSetPixel(1);

  return true;
}
```

See Also

OWNERDRAW, **startSetPixel**, **setEllipseFill**, **setRectFrame**, **endSetPixel**

setTimerEvent

Syntax

```
int setTimerEvent(int nr, int t, int state)
```

Arguments

nr
: Timer number. Values from 0 to 9 are allowed.

t
: Time in milliseconds. Determines how often a timer is triggered. For a value of 1000, the timer is triggered every second.

state
: Set to 1 for activated and to 0 for deactivated.

Return

If the function fails to create a timer, the return value is zero, otherwise it is nonzero.

Description

Lets you activate or deactivate one of 10 available timers. When activated, the timer will trigger an FME_TIMER event every *t* milliseconds until the timer is deactivated again. The possibilities are unlimited with this function. You can use it to e.g. program a stopwatch (see Example below), display a flicker-free animation, perform asynchronous and multi-threaded-like tasks, stop a running calculation after a certain time period.

Example

```
%ffp

ctl(0): STATICTEXT, Size=(140,10)
ctl(1): STATICTEXT, Size=(140,10)
ctl(3): PUSHBUTTON, Text="Start", Size=(50,15)
ctl(5): PUSHBUTTON, Text="Stop", Size=(50,15)

OnCtl(n):
{
  int r;
  static int count0,count1;

  if (n==0 && e == FME_TIMER)  {

    count0++;
    setCtlTextv (1,"%d ms",count0*10);

  } else if (n==1 && e == FME_TIMER)  {

    count1++;
    setCtlTextv (0,"%d seconds",count1);

  } else if (n==3 && e == FME_CLICKED){

    setCtlText (0,"");
    setTimerEvent(0,10,1);
    setTimerEvent(1,1000,1);

  } else if (n==5 && e == FME_CLICKED){

    count0=count1=0;
    setTimerEvent(0,0,0);
    setTimerEvent(1,0,0);
  }
  return true;
}
```

setZoom

Syntax

`static int setZoom (int n)`

Arguments

n
: Lets you specify a zoom factor from 1 to 16. A value of -888 sets the default zoom that shows the whole image.

Return

Returns true if it succeeded and false if the current zoom factor is identical with the *n* parameter.

Description

Lets you set the zoom factor of the preview directly from code. But be warned! Only use setZoom in the **OnCtl** and **OnFilterStart** handlers, otherwise you will get some unwanted effects in the preview!

Example 1

```
%ffp

ctl(1): "Preview Zoom", Range=(1,16)

OnFilterStart:{

  setZoom(ctl(1));

  return false;
}
```

Example 2

```
%ffp

// Puts a 100% button next to the
// default zoom control
ctl(100): PUSHBUTTON, "100%", Pos=(150,164), Size=(30,10)

OnCtl(n):
{
  // Clicking the button sets
  // the preview window to 100%
  if (n == 100) setZoom(1);
  return false;
}
```

See Also

scaleFactor

shellExec

Syntax

```
int shellExec(string verb, string filename, string params, string defaultdir)
```

Arguments

verb
: is the shell operation to be performed (e.g., "open", "print", or "explore").

filename
: the name of the file to be operated upon; it may be an executable file, a document file, or a URL.

params
: specifies any parameters to be passed to the application.

defaultdir
: specifies the default directory for the execution of the command.

Return

-1 if the operation is successful, or an integer error code otherwise.

Description

Lets your filter execute a Windows shell command. shellExec is useful if you want to run an external program, or to load a webpage.

Comments

Any unneeded parameter may be set to NULL. If NULL is specified for "Verb", the function opens the file or URL specified by "Filename".

Example

```
// The following call will open file help.html from
// the filter's installation directory using the default
web browser:
shellExec("open", "help.html", NULL, filterInstallDir);
```

See Also

getWindowsVersion

sinbell

Syntax

`int sinbell(int a)`

Arguments

a
: Input value from 0 to 1024

Return

Sine-bell-shaped output value from 0 to 1024

Description

This is a replacement for a gaussian function. It calculates very quickly. Of course it is only an approximation of a gaussian function.

Example

```
%ffp

R=G=B= sinbell(x*1024/X ) > 1024 - y*1024/Y ? 0:255
```

sizeof

Syntax

```
int sizeof( type )
```

Arguments

type
The name of a built-in C language data type

Return

Returns the number of bytes required to store a variable of the given data type.

Description

Gives the number of bytes needed to store a variable of the given data type. This is useful when the size of a data type might vary on different computers (eg a 64-bit int requires more bytes than a 32-bit int).

Example

```
printf("Size of an int: %d bytes\nSize of a float: %d
bytes\nSize of a char: %d bytes", sizeof(int),
sizeof(float), sizeof(char));
```

sleep

Syntax

```
sleep(int milliseconds)
```

Arguments

milliseconds
    The time in milliseconds for which the filter's execution should be suspended.

Description

The sleep function delays execution of the filter for a specific period of time. It is not advisable to use this function in the RGBA or ForEveryPixel handlers, as this will cause the filter to pause for every pixel in the image and slow program execution dramatically.

Example

```
sleep(10);  // Pauses execution for 10 milliseconds
```

SLIDER

Syntax

```
ctl[n]: SLIDER(Class Specific Properties), Other Properties
```

Description

The Slider user control is a trackbar that includes both a text label and a numeric edit control.

Class Specific Properties

AUTOTICKS
  Automatically sets tick marks according to range amount.
BOTH
  Makes the thumb square and displays ticks on both sides of the handle
BOTTOM
  Shows tick marks along the bottom of the slider & changes the thumb to point downwards *(default)*
HORZ
  Changes scrollbar orientation to horizontal. *(default)*
LEFT
  Changes orientation of handle and ticks.
NOTHUMB
  Hides the thumb of the control (ie the trackbar/slider itself)
NOTICKS
  Hides the tick marks on the slider control.
RIGHT
  Changes orientation of handle and ticks *(default with VERT active)*
TOP

Shows tick marks along the top of the slider & changes the thumb to point upwards

TOPTIP

Displays the trackbar value as a tooltip beside the handle when dragging the handle.

VERT

Changes scrollbar orientation to vertical.

Other Properties

Color

Sets the background color of the trackbar.

Divisor

Sets the divisor for the number displayed text box, eg 10 for one decimal place. *(default = 1)*

EditStyleEx

Modifies the appearance of the control. Valid values include 'clientedge' and 'staticedge'.

FontColor

Sets the font color. *(default = White)*

NoEditBorder

Hides the border around the text edit portion of the control

Page

Sets the trackbar paging jump unit. *(default = 10)*

Range

Sets the numerical range the scrollbar can return. *(default = (0, 255))*

Text

Defines the text label next to the scrollbar. *(default = no text)*

Val

Initializes the scrollbar's value. *(default = 0)*

Example

```
ctl[0]: SLIDER, Text="bottom, ticks", Val=rnd(0,255),
FontColor=Blue
ctl[1]: SLIDER(noticks), Text="bottom, noticks",
Divisor=10, Val=rnd(0,255)
ctl[2]: SLIDER(top, noticks), Text="top, noticks",
Val=rnd(0,255), NoEditBorder, EditStyleEx=staticedge
ctl[3]: SLIDER(both, noticks), Text="both, noticks",
Val=rnd(0,255)
ctl[4]: SLIDER(enableselrange, noticks),
Text="enableselrange, noticks", Size=(90,12),
Val=rnd(0,255), NoEditBorder, EditStyleEx=clientedge
ctl[5]: SLIDER(enableselrange, both, noticks),
Text="enableselrange, both, noticks", Size=(90,12),
Val=rnd(0,255)
ctl[6]: SLIDER(both, noticks), Text="larger, both,
noticks", Size=(90,12), Val=rnd(0,255)
ctl[7]: SLIDER(top, noticks), Text="larger, top, noticks",
Size=(90,12), Val=rnd(0,255)
```

Notes

It is best to set the Range before setting the Val. If the Val is set first, it might be clipped by the default Range (0,255) before your custom Range is set. If you set the Range first, you can set Val to any value within that range.

See Also

STANDARD, SCROLLBAR, TRACKBAR

snprintf

Part of the ANSI C99 language standard

Syntax

```
int snprintf(char * buffer, int n, const char * format, ...
)
```

Arguments

buffer
: A pointer to the string buffer where the result will be stored

n
: Maximum number of bytes to be written to the string buffer

format
: Specifies the format of the resulting string, including types of the variables to be included in the string.

...
: Additional variables to substitute for the formatting symbols in **format**.

Return

The number of characters written, not including the terminating null character, or -1 if an error occurred.

Description

sprintf "prints" a formatted string (up to a defined length), except instead of printing it to a terminal console, it stores the result in a string variable. The formatting uses the printf-style formatting common to the C language.

Comment

To prevent security problems and memory errors caused by buffer overruns, you should always use this function (snprintf) instead of sprintf.

printf Format Specifiers

%%	% symbol
%c	A single character
%d	A signed integer in decimal format
%f	A double precision floating point number in decimal format
%i	A signed integer
%o	An unsigned integer in octal format
%s	A string
%u	An unsigned integer
%x	An unsigned integer in lowercase hexadecimal format
%X	An unsigned integer in uppercase hexadecimal format

Example

```
int version = 1;
strcpy(str0, "Plug-In Name");
strcpy(str1, "Company Name");

// Note: size of str2 is 255 bytes
snprintf(str2, 255, "You are running %s made by %s,
version %d", str0, str1, version);

printf(str2);
```

See Also

formatString, **printf**, **snprintf**

solarize

Syntax

```
int solarize(int pixel, int s)
```

Arguments

pixel
  The pixel color value to modify
s
  Solarize intensity, range 0 to 255

Return

The newly solarized pixel value.

Description

Applies a simple solarize effect to the image, which morphs an image with its inverse.

Example

```
ctl[0]: "Solarize", Range=(0,255), Val=0

ForEveryTile: {
  for (y = y_start; y < y_end; y++) {
    for (x = x_start; x < x_end; x++) {
       for (z=0; z < 3; z++) {
          pset(x,y,z, solarize( src(x,y,z), ctl(0) ));
       }
    }
  }
  return true;
}
```

See Also

blend, **contrast**, **gamma**, **saturation**

sprintf

Syntax

```
int sprintf (char * buffer, const char * format, ... )
```

Arguments

buffer
A pointer to the string buffer where the result will be stored

format
Specifies the format of the resulting string, including types of the variables to be included in the string.

Return

The number of characters written, not including the terminating null character, or -1 if an error occurred.

Description

sprintf "prints" a formatted string, except instead of printing it to a terminal console, it stores the result in a string variable. The formatting uses the printf-style formatting common to the C language.

Comment

For prevent security problems and memory errors caused by buffer overruns, you should use the **snprintf** version of this function instead. sprintf is included only for backwards compatibility.

printf Format Specifiers

%%	% symbol
%c	A single character
%d	A signed integer in decimal format
%f	A double precision floating point number in decimal format
%i	A signed integer
%o	An unsigned integer in octal format
%s	A string
%u	An unsigned integer
%x	An unsigned integer in lowercase hexadecimal format
%X	An unsigned integer in uppercase hexadecimal format

Example

```
int version = 1;
strcpy(str0, "Plug-In Name");
strcpy(str1, "Company Name");
sprintf(str2, "You are running %s made by %s, version %d",
str0, str1, version);
printf(str2);
```

See Also

formatString, **printf**, **snprintf**

sqr

Syntax

```
int sqr(int x)
```

Arguments

x

Non-negative integer value.

Return

Square root of x.

Description

Returns the square root of parameter x.

sqrt

Syntax

```
double sqrt(double x)
```

Arguments

x
: Non-negative floating point value.

Return

Square root of x.

Description

Returns the square root of parameter x.

srand

Syntax

```
void srand(int seed);
```

Arguments

seed
: An integer value that seeds (initializes) the random number generator.

Description

srand seeds the pseudo-random number generator functions. If you seed the generator with a known value, the same sequence of 'random' numbers will occur each time after being seeded with that value. This is useful if you need reproducible but still random-seeming values (eg if you want a graphics effect preset that seems to behave randomly, but can be shared with others to give the same result).

Example

```
%fml
ctl[0]: STANDARD, Text="Seed Value"
ctl[5]: STATICTEXT, Text="", Size=(160,*)

// Changing slider zero gives
// different random numbers, but
// the same numbers each time for
// each specific setting

OnFilterStart: {
  srand(ctl(0));
  sprintf(str1, "'Random' values: %d, %d, %d", rand(),
rand(), rand());
  setCtlText(5, str1);
  return true;
}
```

See Also

rnd, **rand**

src

Syntax

```
int src(int x, int y, int z)
```

Arguments

x
: The x coordinate of the pixel to be retrieved.

y
: The y coordinate of the pixel to be retrieved.

z
: The channel number to be retrieved for the pixel: 0->red, 1->green, 2->blue, 3->alpha.

Return

The integer value of the specified pixel for the specified channel.

Description

This function retrieves the value of a specified channel 'z' from the image pixel at position (x,y). The coordinates should usually be within the image, ie 0 ≤ x ≤ X and 0 ≤ y ≤ Y, while the channel number z should be in the range 0 to 3 inclusive. Channels 0, 1 and 2 are the red, green and blue channels respectively, while channel 3 is the alpha (transparency) channel and is valid only on a layer. Note that use of the src() function in the ForEveryTile handler forces FilterMeister to handle the image as a single (and possibly large) tile.

If the specified coordinates lie outside the image dimensions, src will follow the edge-mirroring behavior defined by set_edge_mode)). By default (ie edgeMode == 0), x and y will be clamped into the ranges 0 ≤ x < X and 0 ≤ y < Y. If edgeMode == 1, getArray will return a zero value, while edgeMode == 2 will wrap negative co-ordinate values to the other side of the image.

Comments

How is pget different to src? src gets the pixel from the original image while pget gets the pixel from the output buffer. At the start of the filter the output buffer is the same as the original image, which is why they seem identical. pget is useful for times when you modify the output, but still need data from the original image.

Example

```
// A very simple slight blur!

%ffp

R,G,B: (src(x - 1, y, z) + src(x + 1, y, z)) / 2

A: a
```

See Also

pget, tget, t2get, pset, tset, t2set, set_edge_mode

srcp

Syntax

`int srcp(int x, int y)`

Arguments

x, y
: Image coordinates

Return

Returns the pixel value at the specified image coordinates

Description

This function lets you read a whole pixel from the source buffer. Unlike **src** the returned value includes the values of all color channels (including the transparency channel if one is available) of the pixel. Using scrp instead of **src** takes only approximately half as much time. To decode the individual color values from the returned pixel value you have to use the **Rval**, **Gval**, **Bval** and **Aval** functions. Currently only works with 8 bit images.

Example

```
%ffp

ctl(0): "Brightness", Size=(*,6), Range=(-300,300),
Val=100
ctl(2): CHECKBOX, "Use the faster srcp() and psetp()",
size=(150,*), Val=0
ctl(10): STATICTEXT, Pos=(*,60), Fontcolor = red, Size=
(150,*)


ForEveryTile: {

  int c,r,g,b;
  int a=255;

  const int startclock = clock();
  int endclock;


  for (y=y_start; y<y_end; y++) {

    if (updateProgress(y,y_end)) abort();

    for (x=x_start; x<x_end; x++) {

    if (ctl(2)) {

      // Read a whole pixel
      c = srcp (x,y);

      // Explode it into the
      // color values
      r =  Rval(c); //c & 0xff;
      g =  Gval(c); //c >> 8 & 0xff;
      b =  Bval(c); //c >> 16 & 0xff;
      if (Z>3) a =  Aval(c); //c >> 24 & 0xff;

      // Adjust brightness
      r += ctl(0);
      g += ctl(0);
      b += ctl(0);
      if (Z>3) a += ctl(0);
      // Makes sure that the
      // color values are in the
      // right range, otherwise
      // we might get a strange
      // image result
      if (r<0) r=0; else if (r>255) r=255;
      if (g<0) g=0; else if (g>255) g=255;
      if (b<0) b=0; else if (b>255) b=255;
      if (Z>3) {if (a<0) a=0; else if (a>255) a=255;}

      // Write back the color values
      psetp (x,y, RGBA(r,g,b,a) );


    } else {

      r =  src(x,y,0);
      g =  src(x,y,1);
      b =  src(x,y,2);
      if (Z>3) a =  src(x,y,3);

      r += ctl(0);
      g += ctl(0);
      b += ctl(0);
      if (Z>3) a += ctl(0);

      pset (x,y,0,r);
      pset (x,y,1,g);
      pset (x,y,2,b);
      if (Z>3) pset (x,y,3,a);

    }


  }}
```

```
  endclock = clock() - startclock;
  setCtlTextv(10, "Render time needed: %d ms", endclock);

  // Display after applying the
  // effect to the image. Makes
  // the speed difference clearer
  if (!doingProxy) Info ("Render time needed: %d ms",
endclock);

  return true;
}
```

See Also

pgetp, **psetp**, **tgetp**, **tsetp**, **t2getp**, **t2setp**, **Rval**, **Gval**, **Bval** and **Aval**

STANDARD

Syntax

```
ctl[n]: STANDARD(Class Specific Properties), Other
Properties
```

Description

The Standard user control is a simple scrollbar including a text label and a numeric edit control. Standard is the default user control type, so you don't need to specify the class property STANDARD when defining the control.

Class Specific Properties

HORZ
: Changes scrollbar orientation to horizontal. *(default)*

VERT
: Changes scrollbar orientation to vertical.

Other Properties

Color
: Sets the text background color. *(default = CadetBlue)*

FontColor
: Sets the font color. *(default = White)*

Line
: Sets the left/right button jump unit. *(default = 1)*

NoTrack
: Prevents the preview window updating when dragging the scrollbar's handle. *(default)*

Page
: Sets the scrollbar paging jump unit. *(default = 10)*

Range
: Sets the numerical range the scrollbar can return. *(default = (0, 255))*

Text
: Defines the text label next to the scrollbar. *(default = no text)*

Track
: Updates the preview window when dragging the scrollbar's handle.

Val
: Initializes the scrollbar's value. *(default = 0)*

Example

```
ctl[0]: STANDARD(MODALFRAME, HORZ), Text="Plasma effect",
Val=10, Disable
ctl[2]: STANDARD(VERT), "Pressure", Range=(-10,10), Val=0,
Track
ctl[3]: "Threshold", Range=(-128,128), Val=0
```

Notes

It is best to set the Range before setting the Val. If the Val is set first, it might be clipped by the default Range (0,255) before your custom Range is set. If you set the Range first, you can set Val to any value within that range.

See Also

SCROLLBAR

startSetPixel

Syntax

```
void startSetPixel(int ctl)
```

Arguments

ctl
: The control number of the Ownerdraw control on which you want to draw.

Description

Call startSetPixel before drawing on an **OWNERDRAW** control to select which control you will be drawing on. Use **setPixel** to change individual pixels or any of the Control drawing functions to draw to the control. After finished drawing, you must call **endSetPixel** to display the drawn control to the screen.

Example

```
%fml

ctl(1): OWNERDRAW(drawitem), Pos=(300,50), Size=(100, 100)

OnFilterStart: {

  // Maximum X/Y pixel co-ords of ownerdraw
  int maxxpixel = HDBUsToPixels(100)-1;
  int maxYpixel = VDBUsToPixels(100)-1;

  // Start setPixel on OWNERDRAW control 1
  startSetPixel(1);

  // Draw rectangle around OWNERDRAW area
  setLine(0, 0, maxxpixel, 0, RGB(0,0,0));
  setLine(0, 0, 0, maxYpixel, RGB(0,0,0));
  setLine(maxxpixel, 0, maxxpixel, maxYpixel, RGB(0,0,0));
  setLine(0, maxYpixel, maxxpixel, maxYpixel, RGB(0,0,0));

  // Fill a Rectangle in the OWNERDRAW
  setRectFill(10, 20, 30, 40, RGB(255, 0, 0));

  // Stop using setPixel drawing
  endSetPixel(1);

  return true;
}
```

See Also

endSetPixel, **setPixel**, **setRectFill**

STATICTEXT

Syntax

`ctl[`*`n`*`]: STATICTEXT(`*`Class Specific Properties`*`), `*`Other Properties`*

Description

This user control places a text in the dialog window. By default, this user control is not actionable.

Class Specific Properties

CENTER
: Center-aligns the text.

LEFT
: Aligns the text to the left. *(default)*

LEFTNOWORDWRAP
: Aligns the text to the left and deactivates word wrapping.

NOTIFY
: Makes the user control actionable and activates tooltip.

RIGHT
: Aligns the text to the right.

Other Properties

Text
: Defines the text contents. *(default = no text)*

Val
: Assigns a value to the static text. *(default = 0)*

Color
: Defines the background color. *(default = transparent)*

FontColor

Defines the text color. *(default = white)*

Example

```
ctl[0]: STATICTEXT(MODALFRAME, NOTIFY), "OK", Color=Red,
FontColor=Yellow, Action=APPLY, Size=(60,30)
ctl[3]: STATICTEXT(CENTER), "Edit me"
```

Notes

Once the static text user control is actionable, its value definitions are lost. The reason is that an action returns a specific value and overwrites (once the mouse button is clicked over the user control) the user control's value. For example, the action APPLY returns a value of 2.

strcat

Syntax

```
char* strcat(char *dest, const char* src)
```

Arguments

dest
: The destination string to which the source string is to be appended.

src
: The source string to be appended on the destination.

Return

A pointer to the resulting string, this is identical to the pointer given as **dest**.

Description

Appends string **src** to the end of string **dest**, overwriting the NULL character at the end of **dest** with the first character of **src** up to and including its terminating NULL character.

String **dest** must have enough space reserved for the resulting string, otherwise results may be undefined.

See Also

strncat

strchr

Syntax

```
char* strchr(const char* s, int c)
```

Arguments

s
: A pointer to the string to be scanned.

c
: The character to find.

Return

A pointer to the character found or NULL if the character was not found.

Description

Find the first occurrence of the given character **c** in the string **s** and return its location.

See Also

strrchr

strcoll

Syntax

```
int strcoll(const char* s1, const char* s2)
```

Arguments

s1
: Pointer to the string to compare.

s2
: Pointer to the string to which the first string is compared to.

Return

An integer value indicating the result of the comparison.

Description

Compares two strings, based on the rules of the current system locale.

If zero is returned, both strings are identical. If a negative value is returned, string **s1** is found to be less then **s2**. If a value positive and greater then zero is returned, string **s1** is greater than **s2**.

Characters are compared by their locale collation order - that is, the dictionary order for the locale. Whereas in extended ASCII accented characters come after all a-z characters, in a French locale (for example) the accented characters are in their dictionary order for the string comparison.

See Also

strcmp, **stricoll**, **strncoll**, **strnicoll**

strcmp

Included in ANSI C, C89, C99

Syntax

```
int strcmp(const char* s1, const char* s2)
```

Arguments

s1
    Pointer to the string to compare.
s2
    Pointer to the string to which the first string is compared to.

Return

An integer value indicating the result of the comparison.

Description

Compares two strings.

If a negative value is returned, string **s1** is found to be less then **s2**. If zero is returned, both strings are identical. If a value positive and greater then zero is returned, string **s1** is greater than **s2**.

Characters are compared by their order in the ASCII character map, this means numbers will be considered less then alphabetical characters, likewise capitals are considered greater than non-capitals.

See Also

C Runtime Functions, **strncmp**

strcpy

Syntax

```
char* strcpy(char* dest, const char* src)
```

Arguments

dest
: The string to which the source string is to be copied.

src
: The string which is to be copied to the destination string.

Return

A pointer to the destination string, identical to **dest**.

Description

Copies the entire string **src** to the string **dest** including the trailing NULL character.

The string **dest** must be large enough to hold the string **src**, otherwise results are undefined.

See Also

strncpy

strcspn

Syntax

```
int strcspn(const char* s, const char* reject)
```

Arguments

s
    The string to scan.
reject
    A string containing the characters to scan for.

Return

The length of the initial matching segment.

Description

Returns the length from the start of string **s** which does not contain any of the characters specified in string **reject**.

See Also

strpbrk, **strspn**, **strstr**

strdate

Syntax

```
strdate( string text )
```

Arguments

text
String buffer that the current date will be stored in.

Description

This function retrieves the current date and converts it to the string form 05/22/09.

Example

```
strdate( str0 );
```

See Also

strtime

strdup

Syntax

```
char* strdup(const char* s)
```

Arguments

s
The string to be duplicated.

Return

Pointer to the new string.

Description

Duplicates the string **s** into a newly allocated memory location.

Since this function allocates memory, the newly created string must at some point be destroyed by means of the **free** function.

strerror

Syntax

```
char* strerror(int errnum)
```

Arguments

errnum
: The requested error number.

Return

A string describing the error.

Description

Return a string describing the specified error number **errnum**. If the error number is not known, an unknown error string is returned instead.

The string returned remains valid only until the next call to the strerror function.

stricoll

Syntax

```
int stricoll(const char* s1, const char* s2)
```

Arguments

s1
: Pointer to the string to compare.

s2
: Pointer to the string to which the first string is compared to.

Return

An integer value indicating the result of the comparison.

Description

Compares two strings, based on the rules of the current system locale, in a case-insensitive manner.

If zero is returned, both strings are identical. If a negative value is returned, string **s1** is found to be less then **s2**. If a value positive and greater then zero is returned, string **s1** is greater than **s2**.

Characters are compared by their locale collation order - that is, the dictionary order for the locale. Whereas in extended ASCII accented characters come after all a-z characters, in a French locale (for example) the accented characters are in their dictionary order for the string comparison.

The FilterMeister function stricoll is just a wrapper around the Microsoft-specific **[__stricoll]** function.

See Also

strcmp, strcoll, strncoll, strnicoll

stricmp

Microsoft-specific C function - not part of ANSI C

Syntax

```
int stricmp(const char* s1, const char* s2)
```

Arguments

s1
  Pointer to the string to compare.
s2
  Pointer to the string to which the first string is compared to.

Return

An integer value indicating the result of the comparison - less than zero if **s1** comes before **s2**, zero if the two strings are identical, and greater than zero if **s2** comes after **s1**.

Description

Compares two strings in a case insensitive manner. That is, both strings are converted to lowercase before comparison.

If a negative value is returned, string **s1** is found to be less then **s2**. If zero is returned, both strings are identical. If a value positive and greater then zero is returned, string **s1** is greater than **s2**.

Characters are compared by their order in the ASCII character map. This means numbers will be considered less then alphabetical characters.

Comment

If you are writing cross-platform compatible code, note that this function is a Microsoft-specific extension in the Windows C Runtime, and not a part of the Standard C language.

See Also

C Runtime Functions, **strcmp**, **strncmp**

stripEllipsis

Syntax

`char* stripEllipsis(char* s)`

Arguments

s
: The text that will have an ellipsis stripped.

Description

Returns string s with an ellipsis ("...") removed.

Example

```
%fml
ctl[2]: STATICTEXT, Text=""
ctl[4]: STATICTEXT, Text=""

OnFilterStart: {

  // Ellipsis is stripped
  strcpy(str0, "Waiting...");
  strcpy(str1, stripEllipsis(str0));
  setCtlText(2, str1);

  // Only last ellipsis is stripped
  strcpy(str3, "Still waiting..........");
  strcpy(str4, stripEllipsis(str3));
  setCtlText(4, str4);
```

```
   return true;
}
```

See Also

appendEllipsis, formatString

strlen

Syntax

```
int strlen(const char* s)
```

Arguments

s
The string of which to determine the length.

Return

An integer value indicating the length of s.

Description

Returns the length of string **s** excluding the trailing NULL character.

strlwr

Microsoft specific C function - not part of ANSI C

Syntax

```
char* strlwr(char* str)
```

Arguments

str
: The string to change to lower case.

Return

A pointer to the modified string.

Description

Converts a string to lower case. Note that this function modifies the original string.

See Also

strupr

strncat

Syntax

```
char* strncat(char *dest, const char* src, int n)
```

Arguments

dest
: The destination string to which the source string is to be appended.

src
: The source string to be appended on the destination.

n
: The number of characters from **src** to append.

Return

A pointer to the resulting string, this is identical to the pointer given as **dest**.

Description

Appends the first **n** characters of string **src** to the end of string **dest**, overwriting the NULL character at the end of **dest** with the first character of **src** up to **n** characters or its terminating NULL character is reached. If string **src** is longer than **n** characters, a NULL character will also be added.

String **dest** must have enough space reserved for the resulting string, otherwise results may be undefined.

See Also

strcat

strncmp

Syntax

```
int strncmp(const char* s1, const char* s2, int n)
```

Arguments

s1
: Pointer to the string to compare.

s2
: Pointer to the string to which the first string is compared to.

n
: The maximum number of characters to compare.

Return

An integer value indicating the result of the comparison.

Description

Compares up to the first **n** characters of two strings.

If a negative value is returned, string **s1** is found to be less then **s2**. If zero is returned, both strings are identical. If a value positive and greater then zero is returned, string **s1** is greater than **s2**.

Characters are compared by their order in the ASCII character map, this means number will be considered less then alphabetical characters, likewise capitals are considered greater than non-capitals.

See Also

strcmp

strncoll

Microsoft-specific C function - not part of ANSI C

Syntax

```
int strncoll(const char* s1, const char* s2, int n)
```

Arguments

s1
: Pointer to the string to compare.

s2
: Pointer to the string to which the first string is compared to.

n
: The maximum number of characters to compare.

Return

An integer value indicating the result of the comparison.

Description

Compares two strings, up to the n maximum number of characters, based on the rules of the current system locale.

If zero is returned, both strings are identical. If a negative value is returned, string **s1** is found to be less then **s2**. If a value positive and greater then zero is returned, string **s1** is greater than **s2**.

Characters are compared by their locale collation order - that is, the dictionary order for the locale. Whereas in extended ASCII accented characters come after all a-z characters, in a French locale (for example) the accented characters are in their dictionary order for the string comparison.

strncoll is regarded as safer than strcoll, because it can help prevent buffer overflows (that could occur if a string is not null-terminated) if the n maximum number of characters is correctly to set to the size of the string memory buffers or less.

The FM function strncoll is just a wrapper around the Microsoft-specific **[_strncoll]** function.

See Also

strcmp, **stricoll**, **strncoll**, **strnicoll**

strncpy

Syntax

```
char* strncpy(char* dest, const char* src, int n)
```

Arguments

dest
    The string to which the source string is to be copied.
src
    The string which is to be copied to the destination string.
n
    The maximum number of characters to copy.

Return

A pointer to the destination string, identical to **dest**.

Description

Copies the first **n** characters of string **src** to the string **dest** including the trailing NULL character.

If the copied characters do not include a trailing NULL character, one will be added to the end.

The string **dest** must be large enough to hold the string **src**, otherwise results are undefined.

See Also

strcpy

strnicmp

Syntax

```
int strnicmp(const char* s1, const char* s2, int n)
```

Arguments

s1
: Pointer to the string to compare.

s2
: Pointer to the string to which the first string is compared to.

n
: The maximum number of characters to compare.

Return

An integer value indicating the result of the comparison.

Description

Compares up to the first **n** characters of two strings, without regard for capitalization/case.

If a negative value is returned, string **s1** is found to be less than **s2**. If zero is returned, both strings are identical (ignoring case). If a value positive and greater than zero is returned, string **s1** is greater than **s2**.

Characters are compared by their order in the ASCII character map. This means digits will be considered less than alphabetical characters; likewise, capitals are considered less than non-capitals.

See Also

strcmp, strncmp

strnicoll

Microsoft-specific C function - not part of ANSI C

Syntax

```
int strnicoll(const char* s1, const char* s2, int n)
```

Arguments

s1
: Pointer to the string to compare.

s2
: Pointer to the string to which the first string is compared to.

n
: The maximum number of characters to compare.

Return

An integer value indicating the result of the comparison.

Description

Compares two strings, up to the n maximum number of characters, based on the rules of the current system locale, in a case-insensitive manner.

If zero is returned, both strings are identical. If a negative value is returned, string **s1** is found to be less then **s2**. If a value positive and greater then zero is returned, string **s1** is greater than **s2**.

Characters are compared by their locale collation order - that is, the dictionary order for the locale. Whereas in extended ASCII accented characters come after all a-z characters, in a French locale (for example) the accented characters are in their dictionary order for the string comparison.

strnicoll is regarded as safer than strcoll, because it can help prevent buffer overflows (that could occur if a string is not null-terminated) if the n maximum number of characters is correctly to set to the size of the string memory buffers or less.

The FM function strnicoll is just a wrapper around the Microsoft-specific **[__strnicoll]** function.

See Also

strcmp, **stricoll**, **strncoll**, **strnicoll**

strnset

Syntax

```
string strnset(string str, int c, int n)
```

Arguments

str
    The string to set
c
    The character which the string is to be set to
n
    The maximum number of characters in **str** to set

Return

A pointer to the new string.

Description

strnset sets at most **n** characters of string **str** to character **c**.

Example

```
int s = strcpy(&str0, "This is the message!");
s = strnset(s, 'X', 4);
Info("%s", s);
```

See Also

strset

strpbrk

Syntax

```
char* strpbrk(const char* s, const char* accept)
```

Arguments

s
: The string to scan.

accept
: A string containing characters to scan for in string **s**.

Return

Pointer to the first match found.

Description

Scans string **s** for the first occurrence of any of the characters specified in string **accept** and returns NULL if none of the characters were found or a pointer to the first character in **s** found.

See Also

strcspn, **strspn**, **strstr**

strrchr

Syntax

```
char* strrchr(const char* s, int c)
```

Arguments

s
: A pointer to the string to be scanned.

c
: The character to find.

Return

A pointer to the character found or NULL if the character was not found.

Description

Find the last occurrence of the given character **c** in the string **s** and return its location.

See Also

strchr

strrev

Syntax

```
string strrev(string str)
```

Arguments

str
The string to reverse

Return

A pointer to the reversed string.

Description

strrev reverses the given string. Note that the original string is modified/reversed - so the returned pointer is just a pointer to the original string.

Example

```
strcpy(str3, "This is the message");
strrev(str3);
msgBox(MB_OK | MB_ICONWARNING, "This is a test...", str3);
```

See Also

strcpy, sprintf

strset

Syntax

`string strset(string str, int c)`

Arguments

str
: The string to set

Return

A pointer to the new string.

Description

strnset sets all characters of string str to c (converted to a char).

Example

```
int s = strcpy(&str0, "This is the message!");
s = strset(s + strlen(str3) - 1, NULL);
Info("%s", s);
```

See Also

strnset

strspn

Syntax

```
int strspn(const char* s, const char* accept)
```

Arguments

s
: The string to scan.

accept
: A string containing the characters to scan for.

Return

The length of the initial matching segment.

Description

Returns the length from the start of string **s** which consists only of characters specified in string **accept**.

See Also

strcspn, **strpbrk**, **strstr**

strstr

Syntax

```
char* strstr(const char* haystack, const char* needle)
```

Arguments

haystack
The string to scan.
needle
The string to scan the haystack for.

Return

Pointer to the first match.

Description

Scans the string **haystack** for the first occurrence of the entire substring **needle**.

If found, a pointer to the first occurrence is returned; if not found, a NULL is returned instead.

See Also

strcspn, **strpbrk**, **strspn**

strtime

Syntax

```
strtime( string text )
```

Arguments

text
Pointer to a string buffer where the current time will be stored.

Description

This function retrieves the current time and converts it to a string in the form 15:54:35.

Example

```
strtime( str0 );
```

See Also

strdate

strtod

Syntax

```
double strtod(const char* nptr, char** endptr)
```

Arguments

nptr
: Pointer to the string containing the value to convert.

endptr
: Optional storage space for output by this function.

Return

The converted value from the string.

Description

This function will convert a string to a double value.

The string is expected to have optional leading white space followed by an optional plus (**+**) or minus (**-**) character, any number of digits including a single decimal point and optionally an exponent which consists of either an **E** or **e**, an optional plus or minus sign and one or more digits.

If **endptr** is not NULL, a pointer to the next character after the parsed portion of the string is stored in the memory location reference by **endptr**.

If the resultant value would overflow the boundaries of the double datatype, the maximum possible value is returned. If an underflow would occur, zero will be returned. In both cases an errorcode will be set which can be queried.

See Also

strtol, **strtoul**

strtok

Syntax

```
char* strtok(void *str, const char *delimiters)
```

Arguments

str
: The string to tokenize. On the second and repeated calls, this should be set to NULL to continue using the internal tokenizing buffer to search for the next token.

delimiters
: A string of characters to use as delimiters (eg space, comma, tab symbol) that will split the string into tokens.

Return

Returns a pointer to the internal string buffer pointing to the current token, or NULL if the end of the string has been reached with no more tokens found.

Description

Splits a string into string "tokens", split by the given delimiters. Each call to strtok returns a single token.

You can use this function to split a string into individual words by using the space character as a delimiter. You might also use it to split a line of a CSV (comma-separated-value) file by using the comma symbol as a delimiter.

Example

```
%fml

OnFilterStart: {
  char* strbuffer;
  char* token;
  strbuffer = malloc(1000);
  sprintf(strbuffer, "FM is awesome!");
  printf("%s", strbuffer);
  token = strtok(strbuffer, " ");
  while(token != NULL) {
    printf("%s", token);
    // Use NULL here, not strbuffer
    // otherwise infinite loop here!
    token = strtok(NULL, " ");
  }
  free(strbuffer);
  return true;
}
```

See Also

strncpy, strcpy

strtol

Syntax

```
long strtol(const char* nptr, char** endptr, int base)
```

Arguments

nptr
: Pointer to the string containing the value to convert.

endptr
: Optional storage space for output by this function.

base
: The numerical base of the number, anything from 2 up to and including 36 or zero.

Return

The converted value from the string.

Description

This function will convert a string to a long value.

The string is expected to have optional leading white space followed by an optional plus (**+**) or minus (**-**) character and any number of digits.

If **base** is set to 16 (hexadecimal), the string may begin with an optional '0x' prefix.

If **base** is set to 0, the string will be parsed as having base 10 (decimal) unless it starts with '0x', in which case is is parsed as having base 16 (hexadecimal), or '0', in which case it is parsed as having base 8 (octal). This allows this function to parse C-style numbers.

If **endptr** is not NULL, a pointer to the next character after the parsed portion of the string is stored in the memory location reference by **endptr**.

If the resultant value would overflow the boundaries of the long datatype, the maximum possible value is returned. If an underflow would occur, zero will be returned. In both cases an errorcode will be set which can be queried.

See Also

strtod, **strtoul**

strtoul

Syntax

```
unsigned long strtoul(const char* nptr, char** endptr, int
base)
```

Arguments

nptr
: Pointer to the string containing the value to convert.

endptr
: Optional storage space for output by this function.

base
: The numerical base of the number, anything from 2 up to and including 36 or zero.

Return

The converted value from the string.

Description

This function will convert a string to an unsigned long value.

If the string would produce a negative value, it's result is negated so this function always returns the absolute value.

The string is expected to have optional leading white space followed by an optional plus (**+**) or minus (**-**) character and any number of digits.

If **base** is set to 16 (hexadecimal), the string may begin with an optional '0x' prefix.

If **base** is set to 0, the string will be parsed as having base 10 (decimal) unless it starts with '0x', in which case is is parsed as having base 16 (hexadecimal), or '0', in which case it is parsed as having base 8 (octal). This allows this function to parse C-style numbers.

If **endptr** is not NULL, a pointer to the next character after the parsed portion of the string is stored in the memory location reference by **endptr**.

If the resultant value would overflow the boundaries of the long datatype, the maximum possible value is returned. If an underflow would occur, zero will be returned. In both cases an errorcode will be set which can be queried.

See Also

strtod, **strtol**

strupr

Microsoft specific C function - not part of ANSI C

Syntax

```
char* strupr(char* str)
```

Arguments

str
The string to change to upper case.

Return

A pointer to the modified string.

Description

Converts a string to upper case. Note that this function modifies the original string.

See Also

strlwr

strxfrm

Syntax

```
int strxfrm(char * dest, const char * src, int n)
```

Arguments

dest
: The string to which the source string is to be copied.

src
: The string which is to be copied to the destination string.

n
: The maximum number of characters to copy to the destination string.

Return

The length of the transformed string, even if higher than **n**.

Description

Transforms the string **src** according to the current locale and copies up to the first **n** characters to **dest**. The string is transformed in such a way that the **strcmp** function may be used on the transformed strings to determine the correct collating sequence according to the locale-specific collating conventions set by setlocale.

Returns the number of characters needed to store the transformed string, even if greater than **n**.

If **n** is zero, no part of the string is copied to **dest** (which now may also be NULL). Use this to find out the size required for **dest**.

See Also

strcmp

sub

Syntax

```
int sub(int a, int b, int c)
```

Arguments

a
: Any integer.

b
: Any integer.

c
: Any integer.

Return

The higher value of (a-b) and c.

Description

This function subtracts 'b' from 'a', then compares the result with 'c' and returns the higher of the two values. This function has been retained for compatibility with Filter Factory; the function max(a-b,c) will give the same result and computes faster.

Example

```
// sets 'p' to 4, because 5-1=4, and 4>2 !
int p = sub(5,1,2);
```

See Also

add, **max**, **min**

t2get

Syntax

```
int t2get(int x, int y, int z)
```

Arguments

x
: The x coordinate of the pixel to be retrieved.

y
: The y coordinate of the pixel to be retrieved.

z
: The channel number to be retrieved for the pixel: 0->red, 1->green, 2->blue, 3->alpha.

Return

The integer value of the specified pixel for the specified channel.

Description

This function retrieves the value of a specified channel 'z' from the pixel at position (x,y) in the second of FilterMeister's tile buffers. The coordinates should usually be within the image, ie 0<=x<=X and 0<=y<=Y, while the channel number z should be in the range 0 to 3 inclusive. Channels 0, 1 and 2 are the red, green and blue channels respectively, while channel 3 is the alpha (transparency) channel and is valid only on a layer.

Example

```
// Copy tile buffer 2 to tile buffer 1
```

```
for (y=y_start; y<Y; ++y) {
    for (x=x_start; x<X; ++x) {
        for (z=0; z<Z; ++z) {
            tset(x, y, z, t2get(x, y, z));
        }
    }
}
```

See Also

src, **pget**, **tget**, **pset**, **tset**, **t2set**

t2getp

Syntax

```
int t2getp(int x, int y)
```

Arguments

x, y
Image coordinates

Return

Returns the pixel value at the specified image coordinates

Description

This function lets you read a whole pixel from the second tile buffer. Unlike **t2get** the returned value includes the values of all color channels (including the transparency channel if one is available) of the pixel. Using t2getp instead of **t2get** takes only approximately half as much time. To decode the individual color values from the returned pixel value you have to use the **Rval**, **Gval**, **Bval** and **Aval** functions. Currently only works with 8 bit images.

See Also

srcp, **pgetp**, **psetp**, **tgetp**, **tsetp**, **t2setp**, **Rval**, **Gval**, **Bval**, **Aval**

t2getr

Syntax

```
int t2getr(int d, int m, int z)
```

Arguments

d
: An integer value for the 'direction' of a pixel.

m
: An integer value for the 'magnitude' of a pixel.

z
: The image channel of the pixel to return (eg 0 for red, 1 for green, 2 for blue when in RGB mode)

Return

The value of the pixel channel z at polar coordinates [d,m] in the second t-buffer.

Description

Pixels are usually addressed by their cartesian coordinates [x,y], but FilterMeister also allows the use of polar coordinates. Polar coordinates are expressed as [d,m], where 'd' represents the 'direction' to the pixel from the image's center point, and 'm' is the 'magnitude' of the distance from the center. The t2getr() function takes a pair of polar coordinates as arguments, and returns the pixel value in channel z at those co-ordinates in the second t buffer.

See Also

t2setr, pgetr, c2d, c2m, r2x, r2y

t2set

Syntax

```
void t2set(int x, int y, int z, int v)
```

Arguments

x
: An integer pixel x-coordinate in the second tile buffer.

y
: An integer pixel y-coordinate in the second tile buffer.

z
: An integer channel number in the range 0 to 3.

v
: An integer value to be set for channel 'z', in the range 0 to 255.

Description

This function sets the value of one channel for a pixel in the second tile buffer; the pixel is at coordinates [x,y], and the channel to be set is given by 'z' (0 = red, 1 = green, 2 = blue, and 3 = alpha).

Example

```
%ffp

ForEveryTile:
{
  for (y=y_start; y<y_end; ++y)
  {
    for (x=x_start; x<x_end; ++x)
    {
      for (z=0; z<Z; ++z)
      {
        // set pixels to white
        t2set(x, y, z, 255);
      }
    }
  }
  return true;
}
```

See Also

pset, tset, t2get

t2setp

Syntax

```
int t2setp(int x, int y, int val)
```

Arguments

x, y
Image coordinates

val
Pixel value that will be stored

Return

Always returns a value of 1

Description

This function lets you write a whole pixel to the second tile buffer. Using t2setp instead of **t2set** takes only approximately half as much time. You have to use the **RGB** or **RGBA** function to create a pixel value from individual color values. Make sure that the individual color values lie between 0 and 255, otherwise they will be mixed up when passing them to this function.

See Also

srcp, **pgetp**, **psetp**, **tgetp**, **tsetp**, **t2getp**, **RGB**, **RGBA**

t2setr

Syntax

```
void t2setr(int d, int m, int z, int v)
```

Arguments

d
: An integer direction from the origin in the second tile buffer.

m
: An integer magnitude from the origin in the second tile buffer.

z
: An integer channel number in the range 0 to 3.

v
: An integer value to be set for channel 'z', in the range 0 to 255.

Description

This function sets the value of one channel for a pixel in the second tile buffer, using polar coordinates (rather than cartesian) to address the pixel; the polar coordinates are relative to the image center. The channel to be set is given by 'z' (0 = red, 1 = green, 2 = blue, and 3 = alpha). NOTE: There is no guarantee that this function is able to completely populate the plane - some pixels in the output buffer may be unreachable because of rounding errors.

Example

```
%ffp
```

```
ForEveryTile: {

  int d, m;

  for (d=0; d < 1024; ++d) {
    for (m=0; m < 256; m+=3) {
      for (z=0; z < Z; ++z) {
        // create black circles
        t2setr(d, m, z, 0);
      }
    }
  }

  return true;
}
```

See Also

psetr,tsetr, t2getr

t3get

Requires FM 1.0 Beta 8.4 (Sep 2007) or newer

Syntax

```
int t3get(int x, int y, int z)
```

Arguments

x
: The x coordinate of the pixel to be retrieved.

y
: The y coordinate of the pixel to be retrieved.

z
: The channel number to be retrieved for the pixel: 0->red, 1->green, 2->blue, 3->alpha.

Return

The integer value of the specified pixel for the specified channel.

Description

This function retrieves the value of a specified channel 'z' from the pixel at position (x,y) in the third of FilterMeister's tile buffers. The coordinates should usually be within the image, ie 0<=x<=X and 0<=y<=Y, while the channel number z should be in the range 0 to 3 inclusive. Channels 0, 1 and 2 are the red, green and blue channels respectively, while channel 3 is the alpha (transparency) channel and is valid only on a layer.

Example

```
// Copy tile buffer 3 to the output buffer

for (y=y_start; y<Y; ++y) {
  for (x=x_start; x<X; ++x) {
    for (z=0; z<Z; ++z) {
      pset(x, y, z, t3get(x, y, z));
    }
  }
}
```

See Also

src, **pget**, **pset**, **tget**, **tset**, **t2get**, **t2set**, **t3set**, **t4get**, **t4set**

t3getp

Requires FM 1.0 Beta 8.4 (Sep 2007) or newer

Syntax

```
int t3getp(int x, int y)
```

Arguments

x
: The x coordinate of the pixel to be retrieved.

y
: The y coordinate of the pixel to be retrieved.

Return

Returns the pixel value at the specified image coordinates

Description

This function lets you read a whole pixel from the third tile buffer. Unlike **t3get** the returned value includes the values of all color channels (including the transparency channel if one is available) of the pixel. Using t3getp instead of **t3get** takes only approximately half as much time. To decode the individual color values from the returned pixel value you have to use the **Rval**, **Gval**, **Bval** and **Aval** functions. Currently only works with 8 bit images.

See Also

srcp, **pgetp**, **psetp**, **tgetp**, **tsetp**, **t2setp**, **t3setp**, **Rval**, **GVal**, **BVal**, **AVal**

t3getr

Requires FM 1.0 Beta 8.4 (Sep 2007) or newer

Syntax

```
int t3getr(int d, int m, int z)
```

Arguments

d
: An integer value for the 'direction' of a pixel.

m
: An integer value for the 'magnitude' of a pixel.

z
: The image channel of the pixel to return (eg 0 for red, 1 for green, 2 for blue when in RGB mode)

Return

The value of the pixel channel z at polar coordinates [d,m] in the third t-buffer.

Description

Pixels are usually addressed by their cartesian coordinates [x,y], but FilterMeister also allows the use of polar coordinates. Polar coordinates are expressed as [d,m], where 'd' represents the 'direction' to the pixel from the image's center point, and 'm' is the 'magnitude' of the distance from the center. The t3getr() function takes a pair of polar coordinates as arguments, and returns the pixel value in channel z at those co-ordinates in the third t buffer.

See Also

t3setr, **pgetr**, **c2d**, **c2m**, **r2x**, **r2y**

t3set

Requires FM 1.0 Beta 8.4 (Sep 2007) or newer

Syntax

```
void t3set(int x, int y, int z, int v)
```

Arguments

x
: An integer pixel x-coordinate in the third tile buffer.

y
: An integer pixel y-coordinate in the third tile buffer.

z
: An integer color channel number in the range 0 to 3.

v
: An integer pixel color value to be set for color channel 'z', in the range 0 to 255.

Description

This function sets the value of one channel for a pixel in the third tile buffer; the pixel is at coordinates [x,y], and the channel to be set is given by 'z' (0 = red, 1 = green, 2 = blue, and 3 = alpha).

Example

```
%ffp

ForEveryTile:
{
  for (y=y_start; y<y_end; ++y)
  {
    for (x=x_start; x<x_end; ++x)
    {
      for (z=0; z<Z; ++z)
      {
        // set all pixels white
        t3set(x, y, z, 255);
      }
    }
  }
  return true;
}
```

See Also

pset, tget, tset, t3get

t3setp

Requires FM 1.0 Beta 8.4 (Sep 2007) or newer

Syntax

```
int t3setp(int x, int y, int val)
```

Arguments

x
: The x coordinate of the pixel to be retrieved.

y
: The y coordinate of the pixel to be retrieved.

val
: Pixel value that shall be stored

Return

Always returns a value of 1

Description

This function lets you write a whole pixel to the third tile buffer. Using t3setp instead of **t3set** takes only approximately half as much time. You have to use the **RGB** or **RGBA** function to create a pixel value from individual color values. Make sure that the individual color values lie between 0 and 255, otherwise they will be mixed up when passing them to this function.

See Also

srcp, **pgetp**, **psetp**, **tgetp**, **tsetp**, **t2setp**, **t3setp**, **RGB**, **RGBA**

t3setr

Requires FM 1.0 Beta 8.4 (Sep 2007) or newer

Syntax

```
void t3setr(int d, int m, int z, int v)
```

Arguments

d
: An integer direction from the origin in the third tile buffer.

m
: An integer magnitude from the origin in the third tile buffer.

z
: An integer channel number in the range 0 to 3.

v
: An integer value to be set for channel 'z', in the range 0 to 255.

Description

This function sets the value of one channel for a pixel in the third tile buffer, using polar coordinates (rather than cartesian) to address the pixel; the polar coordinates are relative to the image center. The channel to be set is given by 'z' (0 = red, 1 = green, 2 = blue, and 3 = alpha). NOTE: There is no guarantee that this function is able to completely populate the plane - some pixels in the output buffer may be unreachable because of rounding errors.

Example

```
%ffp

ForEveryTile: {

  int d, m;

  for (d=0; d < 1024; ++d) {
    for (m=0; m < 256; m+=3) {
      for (z=0; z < Z; ++z) {
        // create black circles
        t3setr(d, m, z, 0);
      }
    }
  }

  return true;
}
```

See Also

psetr,**tsetr**, **t3getr**

t4get

Requires FM 1.0 Beta 8.5 (Nov 2007) or newer

Syntax

```
int t4get(int x, int y, int z)
```

Arguments

x
: The x coordinate of the pixel to be retrieved.

y
: The y coordinate of the pixel to be retrieved.

z
: The channel number to be retrieved for the pixel: 0->red, 1->green, 2->blue, 3->alpha.

Return

The integer value of the specified pixel for the specified channel.

Description

This function retrieves the value of a specified channel 'z' from the pixel at position (x,y) in the fourth of FilterMeister's tile buffers. The coordinates should usually be within the image, ie 0<=x<=X and 0<=y<=Y, while the channel number z should be in the range 0 to 3 inclusive. Channels 0, 1 and 2 are the red, green and blue channels respectively, while channel 3 is the alpha (transparency) channel and is valid only on a layer.

Example

```
// Copy tile buffer 4 to the output buffer

for (y=y_start; y<Y; ++y) {
  for (x=x_start; x<X; ++x) {
    for (z=0; z<Z; ++z) {
      pset(x, y, z, t4get(x, y, z));
    }
  }
}
```

See Also

src, pget, pset, tget, tset, t2get, t2set, t3get, t3set, t4set

t4getp

Requires FM 1.0 Beta 8.5 (Nov 2007) or newer

Syntax

```
int t4getp(int x, int y)
```

Arguments

x
: The x coordinate of the pixel to be retrieved.

y
: The y coordinate of the pixel to be retrieved.

Return

Returns the pixel value in the fourth tile buffer at the specified image coordinates

Description

This function lets you read a whole pixel from the fourth tile buffer. Unlike **t4get** the returned value includes the values of all color channels (including the transparency channel if one is available) of the pixel. Using t4getp instead of **t4get** takes only approximately half as much time. To decode the individual color values from the returned pixel value you have to use the **Rval**, **Gval**, **Bval** and **Aval** functions. Currently only works with 8 bit images.

See Also

srcp, **pgetp**, **psetp**, **tgetp**, **tsetp**, **t2getp**, **t3getp**, **t4setp**, **Rval**, **GVal**, **BVal**, **AVal**

t4getr

Requires FM 1.0 Beta 8.5 (Nov 2007) or newer

Syntax

```
int t4getr(int d, int m, int z)
```

Arguments

d
: An integer value for the 'direction' of a pixel.

m
: An integer value for the 'magnitude' of a pixel.

z
: The image channel of the pixel to return (eg 0 for red, 1 for green, 2 for blue when in RGB mode)

Return

The value of the pixel channel z at polar coordinates [d,m] in the third t-buffer.

Description

Pixels are usually addressed by their cartesian coordinates [x,y], but FilterMeister also allows the use of polar coordinates. Polar coordinates are expressed as [d,m], where 'd' represents the 'direction' to the pixel from the image's center point, and 'm' is the 'magnitude' of the distance from the center. The t4getr() function takes a pair of polar coordinates as arguments, and returns the pixel value in channel z at those co-ordinates in the fourth t buffer.

See Also

t4setr, **pgetr**, **c2d**, **c2m**, **r2x**, **r2y**

t4set

Requires FM 1.0 Beta 8.5 (Nov 2007) or newer

Syntax

```
void t4set(int x, int y, int z, int v)
```

Arguments

x
: An integer pixel x-coordinate in the fourth tile buffer.

y
: An integer pixel y-coordinate in the fourth tile buffer.

z
: An integer color channel number in the range 0 to 3.

v
: An integer pixel color value to be set for color channel 'z', in the range 0 to 255.

Description

This function sets the value of one channel for a pixel in the fourth tile buffer; the pixel is at coordinates [x,y], and the channel to be set is given by 'z' (0 = red, 1 = green, 2 = blue, and 3 = alpha).

Example

```
%ffp

ForEveryTile:
{
  for (y=y_start; y<y_end; ++y) {
    for (x=x_start; x<x_end; ++x) {
      for (z=0; z<Z; ++z) {
        // set all pixels white
        t4set(x, y, z, 255);
      }
    }
  }
  return true;
}
```

See Also

pset, **tset**, **tget**, **t4get**

t4setp

Requires FM 1.0 Beta 8.5 (Nov 2007) or newer

Syntax

```
int t4setp(int x, int y, int val)
```

Arguments

x
: The x coordinate of the pixel to be retrieved.

y
: The y coordinate of the pixel to be retrieved.

val
: Pixel value that shall be stored

Return

Always returns a value of 1

Description

This function lets you write a whole pixel to the fourth tile buffer. Using t4setp instead of **t4set** takes only approximately half as much time. You have to use the **RGB** or **RGBA** function to create a pixel value from individual color values. Make sure that the individual color values lie between 0 and 255, otherwise they will be mixed up when passing them to this function.

See Also

srcp, **pgetp**, **psetp**, **tgetp**, **tsetp**, **t4getp**, **RGB**, **RGBA**

t4setr

Requires FM 1.0 Beta 8.5 (Nov 2007) or newer

Syntax

```
void t4setr(int d, int m, int z, int v)
```

Arguments

d
: An integer direction from the origin in the third tile buffer.

m
: An integer magnitude from the origin in the third tile buffer.

z
: An integer channel number in the range 0 to 3.

v
: An integer value to be set for channel 'z', in the range 0 to 255.

Description

This function sets the value of one channel for a pixel in the fourth tile buffer, using polar coordinates (rather than cartesian) to address the pixel; the polar coordinates are relative to the image center. The channel to be set is given by 'z' (0 = red, 1 = green, 2 = blue, and 3 = alpha). NOTE: There is no guarantee that this function is able to completely populate the plane - some pixels in the output buffer may be unreachable because of rounding errors.

Example

```
%ffp

ForEveryTile: {

  int d, m;

  for (d=0; d < 1024; ++d) {
    for (m=0; m < 256; m+=3) {
      for (z=0; z < Z; ++z) {
        // create black circles
        t4setr(d, m, z, 0);
      }
    }
  }

  return true;
}
```

See Also

psetr,**tsetr**, **t3getr**

TAB

Syntax

```
ctl[n]: TAB(Class Specific Properties), Other Properties
```

Description

Tabs are useful for grouping related controls that don't all need to be onscreen at the same time. In graphics programs, this is often used to group separate features such as brightness/contrast, gamma, and hue/saturation/luminance into separate sets of controls the user can manipulate one at a time.

Class Specific Properties

BOTTOM
: Orients the tabs on the bottom instead of the top (or right if VERTICAL is enabled)

BUTTONS
: Uses pushbuttons instead of tabs to switch between sheets.

FLATBUTTONS
: When used with BUTTONS, makes the buttons look sunken into the dialog instead of raised.

HOTTRACK
: Enables hover/mouseover events (ie illuminates the tab labels when hovered over)

RIGHT
: Orients the tabs on the right side if VERTICAL is enabled, bottom otherwise

TABS
: Uses tabs to switch between sheets. *(default)*

VERTICAL

Orients the tabs vertically / side on

Other Properties

Pos
: The position of the tab control on the screen, in DBUs

Size
: The size of the tab control

Text
: Sets the initial labels for all tab sheets & therefore how many sheets it has *(default = no Text)*

Tooltip
: Sets a tooltip displayed when the mouse hovers over the tab labels.

Val
: Selects which tab sheet is visible and activates all controls within it *(default = 0)*

Example

```
%ffp

ctl[10]: TAB, Text="RGB\nHSL", Pos=(320,5), Size=
(160,100), Val=0

// Add to tab sheet 0, RGB
ctl[2]: "R", Tab=(10,0), Range=(-100, 100), Pos=(*,25)
ctl[3]: "G", Tab=(10,0), Range=(-100, 100), Pos=(*,35)
ctl[4]: "B", Tab=(10,0), Range=(-100, 100), Pos=(*,45)

// Add to tab sheet 1, HSL
ctl[5]: "H", Tab=(10,1), Range=(-100, 100), Pos=(*,25)
ctl[6]: "S", Tab=(10,1), Range=(-100, 100), Pos=(*,35)
ctl[7]: "L", Tab=(10,1), Range=(-100, 100), Pos=(*,45)
```

```

ForEveryPixel: {

   int hue = rgb2hsl(R, G, B, 0) + ctl(5);
   int sat = rgb2hsl(R, G, B, 1) + ctl(6);
   int lum = rgb2hsl(R, G, B, 2) + ctl(7);

   int red = hsl2rgb(hue, sat, lum, 0);
   int grn = hsl2rgb(hue, sat, lum, 1);
   int blu = hsl2rgb(hue, sat, lum, 2);

   R = red + ctl(2);
   G = grn + ctl(3);
   B = blu + ctl(4);
}
```

See Also

COMBOBOX

terminateThread

Requires FM 1.0 Beta 9d (June 2008) or newer

Syntax

```
bool terminateThread(int hThread)
```

Arguments

hThread
: Specifies the handle of the thread which we wish to terminate, or 0 to terminate all current worker threads.

Return

Returns **true** if all specified threads have terminated, or **false** if an error occurred.

Description

Use this function to terminate one or all worker threads that were created by calling **triggerThread**. To terminate one particular thread, set the **hThread** parameter to the value of the thread handle that was returned by the call to **triggerThread**. To terminate all current worker threads, set the **hThread** parameter to 0.

Note: This API can be dangerous to use, since forcibly terminating a thread can result in failure to release resources owned by the thread, and DLLs attached to the thread are not notified that the thread is terminating. Use this API only when absolutely necessary. For more details, see the documentation of the Windows **[TerminateThread function]**.

Comments

The current implementation of **terminateThread** recognizes only a parameter of 0 to terminate all threads; you **cannot** specify a thread handle to terminate a specific thread.

Example

See the example in **triggerThread**.

See Also

System Functions, **countProcessors**, **triggerThread**, **waitForThread**, **isThreadActive**, **getThreadRetVal**

testAbort

Syntax

`bool testAbort()`

Return

Returns true if the user has pressed the Esc key, false otherwise.

Description

The testAbort() function tests for a user-requested abort.

Since we often want to call testAbort() and **updateProgress**() at the same place in our filter program, FM provides a shortcut by calling testAbort() implicitly within updateProgress(), and returning the result of testAbort() as the result of updateProgress(). So we can kill two birds with one stone and replace the separate calls to testAbort() and updateProgress() with the following single call:

```
// Update the progress indicator
// and check for user abort on
// each new row...

if (updateProgress(y - y_start, y_end - y_start)) break;
```

Also note that the following lines are equivalent:

```
if (testAbort()) break;
```

```
if (getAsyncKeyState(VK_ESCAPE)<0) break;
```

Example

```
%ffp

// If processing takes too much
// time, user can abort at any
// time. Instead of using break,
// I chose to integrate testAbort
// directly in the test of the
// first "for loop", which is a
// cleaner solution.

ForEveryTile:
{
  // row by row, included testAbort here
  for ( y = 0; y < Y && !testAbort(); y++ )
  {

    // column by column
    for ( x = 0; x < X; x++ )
    {

      // channel by channel
      for ( z = 0; z < Z; z++ )
      {

        // draw something
        if((x%8)==(y%8)) pset(x,y,z,0);

        // update the preview
        // (only in active filter dialog)
        updatePreview( 0 );

      }
    }
  }
  return true;
}
```

See Also

updateProgress, **getAsyncKeyState**

tget

Syntax

```
int tget(int x, int y, int z)
```

Arguments

x
: The x coordinate of the pixel to be retrieved.

y
: The y coordinate of the pixel to be retrieved.

z
: The channel number to be retrieved for the pixel: 0->red, 1->green, 2->blue, 3->alpha.

Return

The integer value of the specified pixel for the specified channel.

Description

This function retrieves the value of a specified channel 'z' from the pixel at position (x,y) in the first of FilterMeister's tile buffers. The coordinates should usually be within the image, ie 0<=x<=X and 0<=y<=Y, while the channel number z should be in the range 0 to 3 inclusive. Channels 0, 1 and 2 are the red, green and blue channels respectively, while channel 3 is the alpha (transparency) channel and is valid only on a layer.

Example

```
// Copy tile buffer 1 to the output buffer
```

```
for (y=y_start; y<Y; ++y) {
    for (x=x_start; x<X; ++x) {
        for (z=0; z<Z; ++z) {
            pset(x, y, z, tget(x, y, z));
        }
    }
}
```

See Also

src, **pget**, **t2get**, **pset**, **tset**, **t2set**

tgetp

Syntax

```
int tgetp(int x, int y)
```

Arguments

x, y
Image coordinates

Return

Returns the pixel value at the specified image coordinates

Description

This function lets you read a whole pixel from the first tile buffer. Unlike **tget** the returned value includes the values of all color channels (including the transparency channel if one is available) of the pixel. Using tgetp instead of **tget** takes only approximately half as much time. To decode the individual color values from the returned pixel value you have to use the **Rval**, **Gval**, **Bval** and **Aval** functions. Currently only works with 8 bit images.

See Also

srcp, **pgetp**, **psetp**, **tsetp**, **t2getp**, **t2setp**, **Rval**, **Gval**, **Bval**, **Aval**

tgetr

Syntax

```
int tgetr(int d, int m, int z)
```

Arguments

d
: An integer value for the 'direction' of a pixel.

m
: An integer value for the 'magnitude' of a pixel.

z
: The image channel of the pixel to return (eg 0 for red, 1 for green, 2 for blue when in RGB mode)

Return

The value of the pixel channel z at polar coordinates [d,m] in the first t-buffer.

Description

Pixels are usually addressed by their cartesian coordinates [x,y], but FilterMeister also allows the use of polar coordinates. Polar coordinates are expressed as [d,m], where 'd' represents the 'direction' to the pixel from the image's center point, and 'm' is the 'magnitude' of the distance from the center. The tgetr() function takes a pair of polar coordinates as arguments, and returns the pixel value in channel z at those co-ordinates in the first t buffer.

See Also

tsetr, **pgetr**, **c2d**, **c2m**, **r2x**, **r2y**

time

Syntax

`int time(time_t *timeptr)`

Arguments

timeptr
: A pointer to an integer where the result is optionally stored. Usually you would set this to a NULL pointer and use the return value instead.

Return

The number of seconds since January 1, 1970 12:00:00 GMT (also known as the **[Unix Epoch]**).

Description

Returns the current time as expressed in seconds since the Unix Epoch.

Comments

You can find the number of days since the Unix Epoch by dividing the answer by 86400. (60 seconds * 60 minutes * 24 hours = 86400 seconds in a day.)

It's possible that the first parameter of the time function can take a pointer to a data structure - the time function in C allows passing a pointer to a time_t structure/object. However, since FilterMeister does not yet support the time_t type, the author of this documentation has assumed that the first parameter given to time must always be NULL.

See Also

clock

tmpfile

Syntax

`FILE* tmpfile()`

Return

A file pointer to a newly opened temporary file.

Description

Creates and opens a temporary file that is deleted when the file is closed.

Example

```
%fml

OnFilterStart: {
  void* newfile;
  newfile = tmpfile();
  if (!newfile) {
    printf("Unable to create file.");
  }
  else {
    // Do things with file here
    fclose(newfile);
  }
  return false;
}
```

See Also

fopen, **fclose**, **tmpnam**

tmpnam

Syntax

`char *tmpnam(char *s)`

Argument

s
: A pointer to a string where the filename will be stored. If s is `NULL`, the parameter is ignored.

Return

A pointer to a temporary/internal string buffer containing the temporary filename. Note that this string will be modified by further calls to tmpnam, so it should be regarded as a single use return value.

Description

Generates a guaranteed unique filename that can be used to create a temporary file.

Comment

Note that the generated filename will contain directory path characters, ie it may be preceded by a backslash on Windows machines.

Example

```
%fml
```

```
OnFilterStart: {
  printf("A possible temporary filename: %s",
tmpnam(NULL));
  return false;
}
```

See Also

fopen, **fclose**, **tmpfile**

tone

Syntax

```
int tone(int pixel, int h, int m, int d)
```

Arguments

pixel
: The pixel color value to modify

h
: Highlight adjustment, range -127 to 128

m
: Midtone adjustment, range -127 to 128

d
: Darkness adjustment, range -127 to 128

Return

The newly toned pixel value.

Description

Applies a simple highlight / midtone / darkness tone effect to the image.

Example

```
ctl[0]: "Highlights", Range=(-127,128), Val=0
ctl[1]: "Midtone", Range=(-127,128), Val=0
ctl[2]: "Darkness", Range=(-127,128), Val=0

ForEveryTile: {
  for (y = y_start; y < y_end; y++) {
    for (x = x_start; x < x_end; x++) {
       for (z=0; z < 3; z++) {
          pset(x,y,z, tone( src(x,y,z), ctl(0), ctl(1),
ctl(2) ));
       }
    }
  }
  return true;
}
```

See Also

blend, **contrast**, **gamma**, **saturation**

TRACKBAR

Syntax

```
ctl[n]: TRACKBAR(Class Specific Properties), Other
Properties
```

Description

This user control is a scrollbar with a different design. The filter designer can insert so-called ticks on one or two sides of the handle.

Class Specific Properties

AUTOTICKS
: Automatically sets tick marks according to range amount.

BOTH
: Sets ticks on adjacent sides of the handle according to orientation (HORZ or VERT).

BOTTOM
: Changes orientation of handle and ticks *(default)*

HORZ
: Horizontal scrollbar orientation *(default)*

LEFT
: Changes orientation of handle and ticks.

NOTHUMB
: Hides the thumb handle.

NOTICKS
: Hides the tick marks of the trackbar.

RIGHT
: Changes orientation of handle and ticks *(default with VERT active)*

TOP

Changes orientation of handle and ticks.

TOPTIP

Displays the trackbar value beside the handle when dragging the handle.

VERT

Vertical scrollbar orientation.

Other Properties

Color

Sets the text background color. *(default = COLOR_SCROLLBAR)*

NoTrack

Prevents the preview window updating when dragging the scrollbar's handle. *(default)*

Page

Sets the scrollbar paging jump unit. *(default = 10)*

Range

Sets the numerical range the scrollbar can return. *(default = (0, 255))*

Track

Updates the preview window when dragging the scrollbar's handle.

Val

Initializes the scrollbar's value. *(default = 0)*

Example

```
ctl[0]: TRACKBAR, Size=(50, 20), Color=Red
ctl[5]: TRACKBAR (VERT, BOTH, TOPTIP, AUTOTICKS), Range=
(0, 10), Size=(30, 90), Color=Yellow
```

Notes

Currently, trackbar values cannot be correctly read if the first item i1 of the Range=(i1,i2) property is greater than the second item i2. These problems will be resolved in future FM versions.

There is also a known issue with trackbars not responding to keyboard events. If the user modifies the trackbar value by using the keyboard, it will not trigger the events that would call the OnCtl handler, (even though this does work for scrollbar controls).

trackPopupMenu

Syntax

```
int fm_trackPopupMenu (int hMenu, int type, int x, int y,
int style)
```

Arguments

hMenu
: Handle to the menu to be displayed.

type
: If set to 1, the popup menu is displayed at the cursor coordinates. Otherwise it is displayed at the x,y coordinates.

x, y
: Coordinates of the popup menu in DBU. Supply only if type != 1.

style
: A few alignment and other options, e.g. TPM_HORIZONTAL, TPM_VERTICAL, TPM_CENTERALIGN, TPM_LEFTALIGN, TPM_RIGHTALIGN, TPM_BOTTOMALIGN, TPM_TOPALIGN, TPM_VCENTERALIGN, TPM_NONOTIFY, TPM_RETURNCMD, TPM_LEFTBUTTON, TPM_RIGHTBUTTON. Set it to zero for default behaviour.

Return

Returns the number of the menu item (uItem) that was selected by the user (but only if you use 0 or TPM_RETURNCMD or the style parameter). If no menu item was selected, zero is returned.

Description

Queries a menu for the last selected item by the user.

Example

```
%ffp

ctl[0]: PUSHBUTTON, "Click Me!"

OnCtl(n): {

  if (n==0 && e == FME_CLICKED) {

    int menu=0;

    menu = createPopupMenu();

    insertMenuItem(menu, 1, "Do This", MFS_ENABLED ,
NULL);
    insertMenuItem(menu, 2, "Do That", MFS_ENABLED |
MFS_DEFAULT, NULL);
    insertMenuItem(menu, 3, "Do Nothing", MFS_ENABLED,
NULL);
    Info("Selection: %d", trackPopupMenu (menu, 1, 0, 0,
0) );

    destroyMenu(menu);
  }

  return false;
}
```

See Also

createPopupMenu, **insertMenuItem**, **destroyMenu**

tri

Syntax

```
int tri (int a)
```

Arguments

a
: Normally a value between 0 and 1024, but it can also be higher

Return

A value between -511 and 512 is returned.

Description

tri works like sin(), but returns a triangular/linear value. Has the same effect as tricos(a-256).

See Also

tricos

tricos

Syntax

`int tricos(int a)`

Arguments

a
: Normally a value between 0 and 1024, but it can also be higher

Return

A value between -511 and 512 is returned.

Description

tricos works like cos(), but returns a triangular/linear value.

Example

```
%ffp

ctl(0): checkbox, Text="Use tricos() instead of cos()",
Size=(100,*)

ForEveryTile:
{

  int calc;

  for (y=y_start; y<y_end; y++) {

    updateProgress(y,y_end);

    for (x=x_start; x<x_end; x++) {

      if (ctl(0))
        calc = scl( tricos(x*1024/X) , -511, 512, 0, 255);
      else
        calc = scl( cos(x*1024/X) , -511, 512, 0, 255);

      for (z=0; z<Z; z++) {

        pset(x, y, z, calc);

      }
    }
  }

  return true;
}
```

See Also

tri

triggerEvent

Syntax

```
int triggerEvent(int n, int event, int previous)
```

Arguments

n
: the control number of the event (eg CTL_OK or a programmer defined value)

event
: the type of event to raise (eg FME_CLICKED, FME_CUSTOMEVENT)

previous
: a value to pass to the event handler (usually the previous value of the control)

Return

A value of 0 or 1, depending on whether the event handler code returned true or false.

Description

triggerEvent is used to artificially launch events, such as a dialog button click, the mouse moving over a dialog control, or to trigger timer events without resetting the actual timer. You can also use triggerEvent to write your own user-defined functions. By using FME_CUSTOMEVENT and creating custom values for the 'n' parameter, you can call specific sections of your OnCtl handler. You can even use the previous variable as a simple function parameter - though if you need more parameters, you are better off using the built-in global variables to pass and return values.

Example

```
OnCtl( n ):
{
  if ( e == FME_CUSTOMEVENT )
  {
    switch ( n )
    {
      case 0:
        // Execute this code
        if ( previous == 1 ) ...
        else if ( previous == 2 ) ...
        return true;
      case 1:
        // Execute that code
        ...
        return false;
    }
  }
  return false;
}
```

triggerThread

Requires FM 1.0 Beta 9d (June 2008) or newer

Syntax

```
int triggerThread(int n, int event, int previous)
```

Arguments

n
: An integer index which will be passed to the thread as parameter **n**. This value may be used however you please, but is typically used to select which routine will be executed within the **OnCtl** handler. If **event** is **FME_CUSTOMEVENT**, then **n** may be any integer value; otherwise, **n** must be a value between 0 and N_CTLS-1 (i.e., between 0 and 255, inclusive, in current FM releases).

event
: An integer value which is typically set to the code for one of the FilterMeister **Events**. This value will be passed to the thread as parameter **e**, and is used to detect that a thread trigger event has occurred within the **OnCtl** handler.

previous
: An integer value which will be passed to the thread as parameter **previous**. This value may be used to pass any value you want to the thread. For control events that are triggered by FM itself (using **triggerEvent**), the **previous** value usually contains the previous value of a control. With triggerThread() you can use **previous** to pass an additional parameter, such as a thread ID, to your thread.

Return

Returns the handle of the created thread if successful. Returns 0 if it failed (e.g., **n** is out of range when **event** is **FME_CUSTOMEVENT**; or memory could not be allocated for a new thread context record; or the Windows CreateThread API failed).

Description

This function creates a worker thread and schedules it to execute the **OnCtl** handler, passing it three parameters which may be accessed in OnCtl() as **n**, **e**, and **previous**, respectively. **n** is typically a function code specifying what section of OnCtl contains the code to be executed. **e** is typically **FME_CUSTOMEVENT**. The **previous** parameter may be used for any desired purpose.

triggerThread() is almost identical to **triggerEvent**(). The only difference is that **triggerEvent** executes the **OnCtl** handler directly in the current thread, while **triggerThread** causes the **OnCtl** handler to be executed in a new worker thread.

Example

```
%fml

int nTile;

OnFilterStart:
{
  nTile = 0;
  isTileable = true;
  return false;
}

ForEveryTile:
```

```
{
  int ncpus = countProcessors();

  nTile++;

  for (int i=0; i < ncpus; i++) {
    // create 'ncpus' worker threads
    // to execute function 99
    int hThread = triggerThread(99, FME_CUSTOMEVENT, i);
    if (hThread==0) {
      ErrorOk("Failed to create thread %d in tile %d", i,
nTile);
      return false;
    }
  }

  waitForThread(0, INFINITE, 0);
  Info("Tile %d is done!", nTile);
  return true;  // finished processing
}

OnCtl(n,e):
{
  switch (e) {
  case FME_CUSTOMEVENT:
    switch (n) {
    case 99: // function 99
      // 'previous' has our thread ID
      Info("Thread %d is executing", previous);
      // do some useful work here...
      return true;
    default:
      Warn("Unknown thread function %d", n);
      return false;
    } // switch n
    break;
  default:
    // probably a control event
    return false; // process it
  } // switch e

  return false;
}
```

See Also

System Functions, **Events**, **FME_CUSTOMEVENT**, **triggerEvent**, **countProcessors**, **waitForThread**, **isThreadActive**, **getThreadRetVal**, **terminateThread**

true

Description

`true` is a Boolean constant representing a logical state of "truth". In FilterMeister, any non-zero numeric value, any non-NUL character constant, or any non-null pointer represents a state of "truth", but the numeric value 1 is reserved as the canonical value for "truth". Thus, the Boolean constant `true` has a numeric value of 1 when evaluated in a numeric context.

Note, however, that *any* non-zero value represents "truth", so the numeric constants 2, 0.003, 255, -1.0, and -88888 (and the string constant "Hello world" or the character constant 'Z') will also evaluate as true in a Boolean or logical context.

Caution

A common cause of mistakes in C or FM programming is to assume that any true value has the value 1 (or `true`). It is bad programming style, and a frequent cause of subtle errors, to code for example:

```
if (flag==true) break;
```

since this code will *not* break if the value of flag is 77, which also represents truth. Unless the programmer is specifically testing for the value 1, the preferred idiom would be:

```
if (flag) break;
```

which will break when `flag` contains *any* true value (i.e., any value but 0, ±0.0, '\0', or `NULL`), and not just in the case that `flag` has the value 1.

Note that this caution does not apply to the Boolean constant `false`, since the canonical numeric value of `false` is 0. The only other representation of falsity is a NUL character or a null pointer, and the numeric value of these is always 0 in FM. Thus the following code fragment, while poor style, will always break correctly when `flag` has the value 0, ±0.0, `false`, '\0', or `NULL`.

```
if (flag==false) break;
```

Nonetheless, the preferred coding style is:

```
if (!flag) break;
```

Example

```
%ffp

OnFilterStart:{

  bool flag = true;

  Info("The integer value of true is %d", flag);

  if (true)
    Info("The gostak distims the doshes.");

  return false;
}
```

This snippet will display two message boxes with the messages:

```
The integer value of true is 1
The gostak distims the doshes.
```

See Also

false, **Constants**

tryEnterCriticalSection

Requires FM 1.0 Beta 9d (June 2008) or newer

Syntax

```
bool tryEnterCriticalSection(int hCS)
```

Arguments

hCS
: Specifies the handle of the Critical Section to be entered, as obtained by a call to **createCriticalSection**.

Return

Returns **true** if the Critical Section was successfully entered, **false** if another thread already owns the Critical Section or if hCS is zero.

Description

This function is similar to **enterCriticalSection**, but does not wait if another thread already owns the designated Critical Section--in which case it returns immediately with a return value of **false**. This allows the calling thread to perform other useful work instead of blocking while the Critical Section is busy.

For more information about Critical Sections, see the MSDN documentation about **[Critical Section Objects]**.

Example

In the following somewhat contrived example, **myThreadFunction** uses a Critical Section (**myCS**) to serialize access to **myGlobalFlag**. However, if the current thread cannot gain immediate access to the Critical Section via **tryEnterCriticalSection**, it goes off for a while (*no pun intended*) to perform some other useful work (perhaps updating the GUI) before retrying entry to the Critical Section. Note that we call **sleep(0)** at the end of the while-loop to voluntarily yield the rest of our timeslice so another ready thread may be able to run, rather than hogging an entire timeslice unnecessarily.

```
// global variables
int myGlobalFlag = 0;
int myCS = createCriticalSection();

int myThreadFunction(void) {

    while (!tryEnterCriticalSection(myCS)) {
    // do some other useful work here...
    sleep(0); //give up timeslice
  }

  // we have now entered the Critical Section
  // guarding access to myGlobalFlag...
  if (myGlobalFlag == 0) {
    myGlobalFlag = 1;
    // do something...
    myGlobalFlag = 0;
  }

  leaveCriticalSection(myCS);

  return 0; // exit code
}
```

See Also

System Functions, createCriticalSection, enterCriticalSection, leaveCriticalSection, deleteCriticalSection

tset

Syntax

```
void tset(int x, int y, int z, int v)
```

Arguments

x
: An integer pixel x-coordinate in the first tile buffer.

y
: An integer pixel y-coordinate in the first tile buffer.

z
: An integer channel number in the range 0 to 3.

v
: An integer value to be set for channel 'z', in the range 0 to 255.

Description

This function sets the value of one channel for a pixel in the first tile buffer; the pixel is at coordinates [x,y], and the channel to be set is given by 'z' (0 = red, 1 = green, 2 = blue, and 3 = alpha).

Example

```
%ffp

ForEveryTile:
{
  for (y=y_start; y < y_end; ++y)
  {
    for (x=x_start; x < x_end; ++x)
    {
      for (z=0; z < Z; ++z)
      {
        // set all pixels white
        tset(x, y, z, 255);
      }
    }
  }
  return true;
}
```

See Also

pset, **t2set**, **tget**

tsetp

Syntax

```
int tsetp(int x, int y, int val)
```

Arguments

x, y
: Image coordinates

val
: Pixel value that shall be stored

Return

Always returns a value of 1

Description

This function lets you write a whole pixel to the first tile buffer. Using tsetp instead of **tset** takes only approximately half as much time. You have to use the **RGB** or **RGBA** function to create a pixel value from individual color values. Make sure that the individual color values lie between 0 and 255, otherwise they will be mixed up when passing them to this function.

See Also

srcp, **pgetp**, **psetp**, **tgetp**, **t2getp**, **t2setp**, **RGB**, **RGBA**

tsetr

Syntax

```
void tsetr(int d, int m, int z, int v)
```

Arguments

d
: An integer direction from the origin in the first tile buffer.

y
: An integer magnitude from the origin in the first tile buffer.

z
: An integer channel number in the range 0 to 3.

v
: An integer value to be set for channel 'z', in the range 0 to 255.

Description

This function sets the value of one channel for a pixel in the first tile buffer, using polar coordinates (rather than cartesian) to address the pixel; the polar coordinates are relative to the image center. The channel to be set is given by 'z' (0 = red, 1 = green, 2 = blue, and 3 = alpha). NOTE: There is no guarantee that this function is able to completely populate the plane - some pixels in the output buffer may be unreachable because of rounding errors.

Example

```
%ffp

ForEveryTile:
```

```
{
  int d, m;

  for (d=0; d<1024; ++d)
  {
    for (m=0; m<256; m+=3)
    {
      for (z=0; z<Z; ++z)
      {
        // create white circles
        tsetr(d, m, z, 256);
      }
    }
  }

  return true;

}
```

See Also

psetr, t2setr, tgetr

updateAnchors

Syntax

```
int updateAnchors(int dialogWidth, int dialogHeight)
```

Arguments

dialogWidth
: The width of the filter dialog window. Set to -1 to have the function retrieve this itself.

dialogHeight
: The height of the filter dialog window. Set to -1 to have the function retrieve this itself.

Return

Always returns true.

Description

Updates the positioning anchors for all user controls in the dialog, as appropriate for the given dialog width & height.

Example

```
updateAnchors(-1, -1);
```

See Also

setCtlAnchor

updatePreview

Syntax

```
int updatePreview(int id)
```

Arguments

id
The ID of the proxy preview control to be updated.

Return

Returns 1 if doingProxy (and the preview was updated), 0 otherwise.

Description

updatePreview causes the proxy preview to be immediately updated with the current contents of the output image. This is useful for animating iterative algorithms in the preview window. At present, FM supports only one proxy preview, so the 'id' argument is ignored.

Example

```
%ffp

ForEveryTile:
{
  // row by row
  for( y = 0; y < Y; y++ )
  {
    // column by column
    for( x = 0; x < X; x++ )
    {
      // channel by channel
      for( z = 0; z < Z; z++ )
      {

        // apply an effect
        pset( x, y, z,
          src( x % 100, y % 100, z )
          + rnd( -50, 50 ) );
      }
    }

    // update the progress bar
    updateProgress( y, Y );

    // update the preview
    // (only in filter dialog)
    updatePreview( a );

  }
  return true;
}
```

See Also

updateProgress

updateProgress

Syntax

```
int updateProgress(int p, int max)
```

Arguments

p
: Number in the range 0 to max representing current filter progress

max
: The maximum value p can take

Return

Returns 1 if the user has pressed the Esc key, 0 otherwise.

Description

The function updateProgress(current,max) updates the progress bar on the plug-in or in the information bar at the bottom of the screen in your image editing program. The variable p describes the current progress in the range [0,max]. If the user presses the ESC key, a non-zero value is returned by updateProgress().

Example

```
%ffp

// This example updates the
// progress bar for every pixel
// in the image.  In more
// complicated filters this can
// slow down the plug-in. It may
// be better to update the
// progress bar for every Y row.

ForEveryTile:
{
  // row by row
  for ( y = 0; y < Y; y++ )
  {

    // column by column
    for ( x = 0; x < X; x++ )
    {

      // channel by channel
      for ( z = 0; z < Z; z++ )
      {

        // apply an effect
        pset( x, y, z,
          src( x % 100, y % 100, z )
          + rnd( -50, 50 ) );

        // update progress bar
        updateProgress( y, Y );

        // update the preview
        // (only in active
        // filter dialog)
        updatePreview( a );

      }
    }
  }
  return true;
}
```

See Also

updatePreview, testAbort

VDBUsToPixels

Syntax

`VDBUsToPixels(int vdbu)`

Arguments

vdbu
Number of VDBUs to convert to pixels

Description

Converts VDBU (vertical dialog base units, the measurement by which FilterMeister dialogs are constructed) to real on-screen pixels measurement. Note that the result of this conversion depends on the users' Windows installation and may vary.

Example

```
Info("DialogSize in pixels: %d x %d",
HDBUsToPixels(getDialogWidth()),
VDBUsToPixels(getDialogHeight()));
```

See Also

HDBUsToPixels, **PixelsToHDBUs**, **PixelsToVDBUs**

waitForThread

Requires FM 1.0 Beta 9d (June 2008) or newer

Syntax

```
bool waitForThread(int hThread, int ms, int userinput)
```

Arguments

hThread
: Specifies the handle of the thread for which we wish to await completion, or 0 to wait for the completion of all current worker threads.

ms
: Specifies the maximum time in milliseconds that we are willing to wait, or 0 to return immediately after checking for completion of the thread(s), or INFINITE (`0xFFFFFFFF`) if we are willing to wait forever.

userinput
: Set this value to 0 to discard all mouse and keyboard messages while waiting for the thread(s) to complete, or 1 to continue processing all messages while waiting.

Return

Returns **true** if all specified threads have completed, or **false** if we timed out while waiting.

Description

Use this function to wait for the completion of one or all worker threads that were created by calling **triggerThread**. To wait for one particular thread, set the first parameter to the value of the thread handle that was returned by the call to **triggerThread**. To wait for all current worker threads to complete, set the **hThread** parameter to 0.

The second parameter **ms** specifies how long to wait for the thread(s) to complete, in millisecond units. Specify 0 to return immediately after checking for completion of the thread(s), or INFINITE to wait indefinitely.

While waiting for the thread(s) to complete, **waitForThread** runs a message pump loop to process other messages in the meantime. If the **userinput** parameter is 1, the message loop will process all messages, including user input messages from the keyboard and mouse. If processing such user input messages interferes in an undesired way while the worker thread(s) are running, set this parameter to 0 to ignore and discard all such user input messages. For example, if the worker threads are processing the current image, then it would be undesirable to allow mouse or keyboard input to trigger the preview's zoom control, or to drag and resize the dialog window. In general, it is recommended to set this parameter to 0 unless you have specific requirements to do otherwise.

Comments

The current implementation of **waitForThread** recognizes only a parameter of 0 to wait for all threads; you cannot specify a thread handle to wait for a specific thread.

Example

See the **triggerThread example**.

See Also

System Functions, countProcessors, triggerThread, isThreadActive, getThreadRetVal, terminateThread

Warn

Syntax

```
Warn(string promptString, ...)
```

Arguments

promptString
: Specifies the prompt string for the warning message window. This string may contain printf-style format descriptors, which will be expanded using the succeeding arguments.

...
: Variable number of arguments of varying types, should correspond to the format descriptors in promptString.

Return

IDOK or IDCANCEL depending on which button the user clicks.

Description

This function displays a warning box containing a text string, an OK button and a Cancel button.

Example

```
Warn( "This might take a while..." );
Warn( "Your image is not a square" );
if ( Warn( "This might take a while..." ) == IDOK )
     //apply the following filter code...
```

See Also

msgBox, **Info**, **Error**

xyzcnv

Syntax

```
int xyzcnv(int x, int y, int z, int m11, int m12, int m13,
int m21, int m22, int m23, int m31, int m32, int m33,int d)
```

Arguments

x, y, z
: x, y and z coordinates of the image

m11 - m33
: Weights of the 3x3 pixel matrix

d
: Division value which usually should be m11 + m12 + m13 + m21 + m22 + m23 + m31 + m32 + m33. If you set d to zero this sum will be used automatically.

Return

Returns the convoluted value from the 3x3 matrix at the coordinates (x,y,z)

Description

Calculates a 3x3 convolution kernal. This is a substitute for the cnv() function which can't be used in the ForEveryTile handler. Currently works only with 8-bit images.

Example

```
%ffp

// Produces a Blur effect
```

```
ForEveryTile: {

  int calc;

  for (y=y_start; y<y_end; y++) {

    updateProgress(y,y_end);

    for (x=x_start; x<x_end; x++) {

      for (z=0; z<Z; z++) {
        calc = xyzcnv (x,y,z, 4,4,4, 4,16,4, 4,4,4, 0);
        pset(x, y, z, calc);
      }
    }
  }

  return true;
}
```

See Also

cnv

ycbcr2rgb

Syntax

```
int ycbcr2rgb(int y, int cb, int cr, int z)
```

Arguments

y
: Red value

cb
: Green value

cr
: Blue value

z
: Determines which value is returned. z=0 for red, z=1 for green, z=2 for blue

Return

Returns the red, green or blue value from 0 to 255 depending on the value of z

Description

Lets you convert YCbCr values to RGB values.

Example

```
%ffp

ctl(0): "Y (Luminancy)",
          Range=(-255,255), Val=0
ctl(1): "Cb (Chroma Blue)",
          Range=(-255,255), Val=0
ctl(2): "Cr (Chroma Red)",
          Range=(-255,255), Val=0

ForEveryTile: {

  int r,g,b,i,cb,cr;

  for (y= y_start; y < y_end; y++) {

    if (updateProgress(y,y_end)) abort();

    for (x = x_start; x < x_end; x++) {

      r = src(x,y,0);
      g = src(x,y,1);
      b = src(x,y,2);

      cb = rgb2ycbcr(r,g,b,1);
      cr = rgb2ycbcr(r,g,b,2);

      // Do the adjustment
      i = i + ctl(0);
      cb = cb + ctl(1);
      cr = cr + ctl(2);

      pset(x, y, 0, ycbcr2rgb(i,cb,cr,0));
      pset(x, y, 1, ycbcr2rgb(i,cb,cr,1));
      pset(x, y, 2, ycbcr2rgb(i,cb,cr,2));

    }
  }
  return true;
}
```

See Also

rgb2ycbcr

YesNo

Syntax

```
int YesNo(string promptString, ...)
```

Arguments

promptString
: Specifies the prompt string for the Yes/No message window. This string may contain printf-style format descriptors, which will be expanded using the succeeding arguments.

...
: Variable number of arguments of varying types, should correspond to the format descriptors in promptString.

Return

IDYES or IDNO depending on which button the user clicks.

Description

This function displays a message box containing a text string, a YES and a NO button.

Example

```
YesNo( "Will you marry me?" );
if ( YesNo( "Will you marry me?" ) == IDYES)
    Info( "Congratulations!");
else
    Info( "Maybe later..." );
```

See Also

Info, msgBox, Warn, YesNoCancel

YesNoCancel

Syntax

```
int YesNoCancel(string promptString, ...)
```

Arguments

promptString
: Specifies the prompt string for the Yes/No/Cancel message window. This string may contain printf-style format descriptors, which will be expanded using the succeeding arguments.

...
: Variable number of arguments of varying types, should correspond to the format descriptors in promptString.

Return

IDYES, IDNO or IDCANCEL depending on which button the user clicks.

Description

This function displays a message box containing a text string, a YES, a NO and a CANCEL button.

Example

```
YesNoCancel ( "Do you want to buy %d stocks?" , M + X );
if ( YesNoCancel( "Will you marry me?" ) == IDYES)
    Info( "Congratulations!" );
else
    Info( "Go to..." );
```

See Also

msgBox

Z

Syntax

```
int Z
```

Definition

The **Z** variable contains the number of channels (a.k.a. planes) in the image. This includes all color channels and any additional alpha channels available.

See Also

planes, **planesWithoutAlpha**

zoomFactor

Syntax

`int zoomFactor`

Description

An integer between 1 and 16 to indicate the current zoom factor of the proxy preview window, or 0 if the proxy zoom factor has not yet been set. (In the current implementation, the zoomFactor variable is essentially the same as the built-in scaleFactor variable.)

Example

```
%ffp

ctl(0): STATICTEXT, "Please right click on the preview to
set a cross.", size=(100,20)


OnCtl(n): {

  if (n == CTL_PREVIEW && e == FME_RIGHTCLICKED_DOWN) {
    j0 = getPreviewCoordX() * zoomFactor;
    j1 = getPreviewCoordY() * zoomFactor;
    doAction(CA_PREVIEW);
  }
  return false;
}



ForEveryTile: {
  int g, h, z, color;
  int PreviewX = j0/zoomFactor;
  int PreviewY = j1/zoomFactor;

  // Calculate color of the cross
  color = ( src(PreviewX, PreviewY, 0) + src(PreviewX,
PreviewY, 1) + src(PreviewX, PreviewY, 2) ) /3;
  if (color > 128 && color < 196) color=0;
  if (color > 64 && color < 128) color=255;
  else color =255-color;

  // Display Cross
  if (doingProxy){
    for (z=0; z < Z; z++) {
      for (g=-7; g < 8; g++)
        if (g < -1 || g > 1) pset(PreviewX + g, PreviewY,
z, color);
      for (h=-7; h < 8; h++)
        if (h < -1 || h > 1) pset(PreviewX, PreviewY + h,
z, color );
    }
  }

  return true;
}
```

See Also

Constants, scaleFactor, setZoom

Appendix

Frequently Asked Questions

Getting Started

Where can I find example source code?

If you look inside your FilterMeister installation directory, you will find lots of examples in the Democodes folder. Depending on where you have installed FilterMeister, the folder might be something like: C:\Program Files\FilterMeister\sourcecode\democodes. There are also coding examples on many pages of this Wiki.

Dialog Questions

How do I hide the FilterMeister logo and default controls?

You can use the **NONE** keyword with each control in the following manner:

```
ctl[CTL_LOGO]: NONE
```

The example above removes the gnome image from the bottom right corner of the plug-in interface. You can use similar code to hide the other built-in controls, such as the CTL_OK and CTL_CANCEL buttons. As of FilterMeister 1.0 Beta 9d, the only control that you can't hide is the CTL_PREVIEW. In some older versions of FilterMeister, there are some other controls you can't hide, and you may need to use a program like Resource Hacker (from **http://www.users.on.net/johnson/resourcehacker/**) to edit the plug-in interface directly.

How do I resize the preview control?

As of FilterMeister 1.0 Beta 9d, you can do this using FilterMeister code, or using a control definition. To create a preview that is 100 DBUs wide, 200 DBUs high, and positioned 10 DBUs from the top left corner, you would use the following control definition:

```
ctl[CTL_PREVIEW]: MODIFY, Pos=(10,10), Size=(100,200)
```

or the following in your code:

```
setCtlPos(CTL_PREVIEW, 10, 10, 100, 200);
```

If you are still using an older version of FilterMeister, you will need to use Resource Hacker instead to resize the window. This is how you would do it:

- Run Resource Hacker.
- Go to File -> Open and find your compiled .8bf file. You may need to select All Files (*.*) in the Files Of Type combobox in order to do this.
- Select Dialog -> FILTERPARAM -> 1033 from the tree menu on the left of screen.
- The default filter dialog appears. Click with the mouse to choose a control, and use the tab key to cycle through all the available controls. Notice that the preview control and the frame around it are two separate controls.
- Clicking on the preview control should display little blue dots around its edges. Resize the window by clicking near one of these blue dots and moving the mouse until the window is the desired size.
- For super-precise adjustments, you can edit the resource code directly in the text window. Not sure which control is which in the text? Just click on it in the dialog window, and a little red dot will appear next to the appropriate line in the Resource Hacker text window.

- When you're done, click the Compile Script button to update settings, and Hide Dialog to, uhh, hide the dialog.
- Now choose File -> Save As... and save the .8bf file under a new name. All done!

Is there a way of changing the about image screen?

Yes, but not by using FilterMeister code. You can design the about screen as a Windows Bitmap (BMP) file - 256 color images sized to 342 x 195 pixels seems to work best. You must first compile your plug-in into an 8bf file, then load the 8bf file using Resource Hacker, which is a free download.

- Run Resource Hacker.
- Go to File -> Open and find your .8bf file. You may need to select All Files (*.*) in the Files Of Type combobox in order to do this.
- Select Action -> Replace Bitmap from the menu. A new dialog should appear.
- Select the bitmap named 134 in the listbox on the right.
- Click on "Open file with new bitmap..." in the top left and locate the BMP file on your computer with the new About screen you have designed.
- If it is the correct bitmap and you're happy with it, click the Replace button on the right.
- Now choose File -> Save As... and save the .8bf file under a new name. All done!

Be sure to play around with Resource Hacker, there's a lot of nifty things you can do with it.

How do I prevent the user from resizing the filter dialog window?

Using Resource Hacker, you need to change WS_THICKFRAME to DS_MODALFRAME in the main dialog resource (FILTERPARAM

-> 1033). This works because Modal dialog windows cannot be resized. You have to do this in your compiled .8BF files - although you can modify FilterMeister itself, the changes won't apply to your compiled plug-ins.

How do I create a filter without a dialog?

Take a look at the FilterWithoutDialog.ffp source code in the sourcecode/codelibrary directory. The trick is to put doAction(CA_APPLY); at the very start of your OnFilterStart handler, so the effect is applied immediately.

Common Coding Errors

Why aren't X & Y the same as my image's width and height?

X and Y are the width & height of the "currently processed image" in the preview window. So if your preview window is zoomed out to 33%, you'll be working with an image 1/3rd the size. You can check the current zoom level with the scaleFactor constant, and you should take the scaleFactor into account in your algorithm if you want your image to look correct at all zoom levels. When you press the OK button, X & Y should always be the same as the image's original width & height. You can learn more about built-in constant values on the Constants page.

My plug-in works fine in FilterMeister, but the compiled plug-in hangs

Are you dividing by the **scaleFactor** or **zoomFactor** variables anywhere in your code? It's possible for these to take the value 0 in the compiled plug-in , so if you're dividing by them you may trigger a divide by zero error... or worse, you might get an infinitely large number. If you've got a for loop somewhere going from 0 to infinity, well, your plug-in is going to hang. The solution is to check if scaleFactor == 0 before you divide by scaleFactor. If scaleFactor is 0, use an alternate equation with 1 in place of scaleFactor.

My plug-in behaves strangely from time to time and I can't seem to find the cause.

Please check if you initialized important variables to zero. This can be done, e.g., by using instead of

```
int MyVariable;
```

the following line:

```
int MyVariable = 0;
```

If a variable wasn't initialized, it doesn't necessarily have a value of zero. It can have a random value and that may not be what your code expects.

Does FilterMeister support 16-bit bitdepth images?

The most needed functions like **src**, **pget**, **pset**, **getArray** and **putArray** support 16-bit values.

Why does the filter affect only half of my 16-bit image?

You are writing 8-bit values to the 16-bit image buffer. Make sure that you use values ranging from 0 to 32768 and not just 0 to 255. (Photoshop and the Photoshop SDK doesn't use the full range of a 16-bit integer, so the maximum value has to be 32768 instead.)

General Coding Questions

Can I use arrays in my code?

FilterMeister does not support standard C-language arrays. Instead, FilterMeister provides alternative functions like **allocArray**, **putArray** and **getArray**. Or you can dynamically allocate blocks of memory with the **malloc** function, just like you can in regular C.

Can I write my own user defined functions?

No, FilterMeister doesn't support functions. But you can create a block of code that you trigger as a custom event throughout your code using **triggerEvent**. You can also call functions in a Windows DLL (Dynamic Link Library) using **loadLib**, **getLibFn**, **callLib** and **freeLib**.

How do I make some code run only when the plug-in is first launched, not every time the filter runs?

As of recent FilterMeister versions (1.0 Beta 9d), you can use the **FME_INIT** event to run code the when the plug-in is launched. You need to add the 'initevent' Dialog parameter to your code for this to work. Here us some example code to make work:

```
%ffp
Dialog: initevent
OnCtl(n): {
  if (e==FME_INIT) {
    // Your code would go here
  }
  return false;
}
```

Known Bugs

Floating point overflow

Using a large number of floating-point (datatypes float and/or double) in a single statement can cause problems wherein the result will be unpredictable. Code will continue to run, but all floating-point calculations after the first occurrence of the bug will be unreliable.

Generally this will not occur but when some floating-point intensive functions are used (such as **iget**, **rgb2lab**, **lab2rgb**, **rgb2hsl** and **hsl2rgb**), problems may appear more quickly. The order in which these functions appear in the statement heavily influences this bug.

As a rule of thumb, always put the functions before the other parts. As such...

```
double value = iget(100., 100., 0, 0, 3) * 0.5; // This
works
```

Instead of the following...

```
double value = 0.5 * iget(100., 100., 0, 0, 3); // This
DOES NOT work
```

Image width of 1 pixel

If the image width is only 1 pixel an error occurs:

```
unexpected NULL outData
```

Signed / Unsigned operands

No difference between signed and unsigned operands of /,%,>>

```
A: x + y ? a : Info("0x8000000u/2u %x", 0x8000000u/2u) //
unsigned: 0x40000000
A:x + y ? a: Info("0x8000000u%5u = %i", 0x8000000u%5u) //
unsigned: 3
A:x + y ? a: Info("-2>>1 = %i", -2>>1) // signed: -1
```

Click Drag

setClickDrag(1): (right mouse key)

Preview flickering when pressing right mouse key and writing text.

Pressing Alt-Key once slows down update frequency.

```
%ffp
ctl(0): STATICTEXT
ctl(CTL_PREVIEW): preview(mousemove)

OnFilterStart:
{
  setClickDrag(1);
  return false;
}

OnCtl(n):
{
  if (n==CTL_PREVIEW && e==FME_MOUSEMOVE)
    setCtlTextv(0, "%i", getPreviewCoordX());
  return false;
}
```

requestRect()

Does not work with selections or zooms other than 1 (100%)

Variables not updated

In the OnFilterStart: handler system variables like scaleFactor, X, Y, x_start, ... are not yet updated after changes and have still old invalid values.

General Improvements in FilterMeister Version 1.0

Alpha mt5c07 (1.0 Alpha mt5c07) - 12 November 2015

- Alpha "As-Is" release. If you do not require the following specific changes, it is recommended that you continue your present development with the current mt5c06 "stable" release until such time that a fully regression-tested version is released.
- Contains a fix for the **fillArray** and **ffillArray** crashes with Arrays larger than 2GB.
- Contains a string literal pool twice as large as the previous release (now 128KB).
- Full release notes are here: **Release Notes for FM 1.0 mt5c07**
- You can download this version from the FilterMeister Website: **http://www.filtermeister.com/fmbeta/**

Alpha mt5c06 (1.0 Alpha mt5c06) - 4 May 2015

Beta 9g (1.0 Beta 9g) - 21 February 2014

- *Please do not use this version to produce release-able filters until it has undergone considerably more testing!*
- An **alpha-quality release**, currently available on 64-bit only.
- The main difference from 1.0.9f is the ability to create 64-bit plug-ins (as well as 32-bit plug-ins).
- Filters created in FM64 can load images with a maximum vertical or horizontal resolution of 300,000 pixels (up from 30,000).
- Exception handling isn't yet supported in FM64, so a Memory Access Violation could (will?) crash the host program.

- DLL loading, multithreading, scripting and floating point math may all encounter problems in this alpha version.

Beta 9f (1.0 Beta 9f) - June 2011

- Could probably be regarded as the latest stable release. Many commercial plug-ins have been released with Beta 9f.
- Fixes some issues with registering FilterMeister on Windows Vista systems & newer.

Beta 9e (1.0 Beta 9e) - 18 April 2010

- *Please do not use this version to produce release-able filters until it has undergone considerably more testing!*
- This is an **experimental release** in that it was built on a new platform from sources that have only recently been updated, and it has received almost no testing.
- The main difference from the previous release (1.0.9d) is that the problem with iget when accessing tile buffers in 16-bit mode has been fixed. (Note that this is also a *non-scripting* version; there are still problems to be resolved in the scripting version.)

Beta 9.1 (1.0 Beta 9.1) - February 2009

- Scripting
    - Scripting Support was added. FM plug-ins can now be scripted in Photoshop, PSP and Debabelizer and can be used as smart filters in Photoshop CS3 and CS4. Scripting does not work in Fireworks yet, although FM plug-ins can be used as live effects, but this is not recommended, because they display the dialog all of the time. By default all user-defined control are scripted unless you add "scripting=off" to the control definition or use **setCtlScripting**(n, false). FM

automatically sets controls to the values that were received by the host through scripting. This is done before the OnFilterStart handler. Make sure that your filter checks the new doingScripting variable, which indicates if script values were received, and that it does not overwrite control values from a file or the registry. But you can also manually get and set script values. **checkScriptVal**(n) lets you check if a script value was received or not and **getScriptVal**(n) lets you retrieve the script value. In the OnCtl handler if n==CTL_OK && e==FME_CLICKED, you can also manually set the script values, which are returned to the host, with **enableScriptVal**(n,-1) and **setScriptVal**(n,val). Otherwise FM will automatically pass the values of all scripting-enabled controls to the host.
    - There is also a non-scripting version of FM included in the file AfhFM10Beta91_NoScripting.zip for users who want to create non-scriptable plug-ins.
    - New UniqueID property which is necessary for scripting. FM generates a new random UniqueID at every execution and compilation, but better set it yourself in your filter code.
    - Added displayDialog constant. 1 when a dialog is displayed, 0 when not. Use this to differentiate between normal invocation or invocation through "last filter" or the scripting system.
- Multithreading
    - **waitForThread**() now offers -1 as the third parameter for deactivating the message pump. This will block the interface as long as waitForThread is running, but also help to avoid some problems.
    - Added **createSync**(), **waitForSync**() and **deleteSync**() for barrier synchronization of threads
- Other Improvements

    - Functions for allocating memory through the host: **allocHost**(size) allocates the memory and returns the buffer ID. **lockHost**(bufferID) returns the memory pointer. **freeHost**(bufferID) deallocates the memory. These functions have the advantage that the host (at least Photoshop) will provide additional free memory if necessary. Alternatively use the array functions if you additionally want to have automatic deallocation.
    - **copyResToArray**() now tries to open a file using the resource name in case it does not find a resource.
    - **___getArray**(), **___putArray**(), **___fgetArray**() and **___fputArray**() are fast versions of these array functions. They do not do any border checking, so they may produce error messages or even crashes if not used properly.
- Dialog Stuff
    - New **createFont**(), **deleteFont**() and **setCtlFont**() functions. You can use up to 32 different fonts in FM now. The fonts are deleted automatically when FM exits.
    - New **updateAnchors**(), **lockCtlScaling**() and **scaleCtls**() functions. The new NoCtlScaling and CtlScaling control properties can be used to temporarily stop and reactivate control repositioning and scaling.
    - Set default tooltip style to normal (should have been normal style already, but balloon was wrongly set as default style). Added **enableToolTipBalloon**() function, taking a boolean to set normal (false) or balloon (true) style tooltips for the entire plug-in. Returns boolean indicating old style.
    - Improved tab controls; You can now use tabs on tab sheets; recursive tabs.
    - Improved tab controls; if tab control is disabled, children controls are disabled as well (if visible).
    - Fixed visibility issues of controls on tab.

    - **setCtlTab**() now allows to remove controls from tab controls. To do that use -1 as the second and third parameter.
    - New **getCtlTab**() function
    - Added **ctlEnabledAs**(n). Returns the "enabled" state of control "n" as it is rendered based on parents (e.g. a tab control), as opposed to just it's internal state.
    - The Mousemove, mouseover and mouseout events are now only triggered for the button areas of a tab control and not for the whole tab control
    - The previous value is now correctly set for the FME_CLICKED event of tab controls
    - Added **setCtlOrder**() function for changing the z-order of controls
    - New **setCtlPixelPos**() function
    - New FME_CLICKED and FME_DBLCLK events when the user clicks or double clicks on a slider label
    - New **setCtlDefVal**() and defval control property for setting a default value. This default value is set when the user double clicks on the slider label.
    - New **getDialogHandle**() and **getCtlHandle**(n) functions for getting the window handle of the dialog or a control. This can be useful if you want to access the dialog or a control from a DLL.
    - New tool tip constants: TTF_CENTERTIP, TTF_RTLREADING, TTF_TRACK, TTF_ABSOLUTE, TTF_TRANSPARENT for **setCtlToolTip**()
    - New edit control constants: ES_LEFT, ES_CENTER, ES_RIGHT, ES_MULTILINE, ES_UPPERCASE, ES_LOWERCASE, ES_PASSWORD, ES_AUTOVSCROLL, ES_AUTOHSCROLL, ES_NOHIDESEL, ES_OEMCONVERT, ES_READONLY, ES_WANTRETURN, ES_NUMBER for **setCtlStyle**()
- setPixel Functions
    - Added **setBitmapTile**(int x, int y, int iName, int tileWidth, int tileHeight, int tileIndex)

    - Added **setBitmapStretch**(int x, int y, int iName, int width, int height)
    - Added **setBitmapTransparent**(int x, int y, int iName, UINT color)
    - Added **setBitmapStretchTransparent**(int x, int y, int iName, int width, int height, UINT color)
    - Fixed issue with **startSetPixel**(). When recompiling using autoload, **endSetPixel**() may not be called for drawing in progress, thus leaking resources. Now startSetPixel() destroys old resources before starting new.
    - Added **setBitmap**(int x, int y, char * name) to draw an embedded bitmap or file to ownerdraw. Clips properly, negative x/y is allowed (clips left/top). Does not switch GDI/DIB mode.
    - Added **setRectGradient**(int left, int top, int right, int bottom, unsigned int color_topleft, unsigned int color_bottomright, bool horizontal). Draws gradient rectangle. Set boolean false to draw vertical gradient. Does not switch GDI/DIB mode.

Beta 9d (1.0 Beta 9.0d) - 12 June 2008

- Main Improvements
    - Added **getCtlItemCount**(), including support for the CC_TAB class.
    - Added extended return values to **getArrayDim**.
    - Added Harry's multithreading API: **countProcessors**, **triggerThread**, **waitForThread**, **isThreadActive**, **getThreadRetVal**, **terminateThread**.
    - Added a set of Critical Section APIs: **createCriticalSection**, **enterCriticalSection**, **tryEnterCriticalSection**, **leaveCriticalSection**, **deleteCriticalSection**.

    - Added more manifest constants: INFINITE, VTA_LEFT, VTA_RIGHT, VTA_BOTTOM, VTA_TOP, WAIT_TIMEOUT, WAIT_FAILED, WAIT_OBJECT_0, WAIT_ABANDONED, WAIT_ABANDONED_0, WAIT_IO_COMPLETION.
- Bug Fixes
    - A crash in **setCtlThumbSize** was fixed.
    - A bug which caused the flashing focus highlight on an anchored STANDARD or SCROLLBAR control to fail to track the actual thumb position while resizing the dialog has been corrected.
    - A bug in which certain Array resources were never released has been fixed.
- Known Problems
    - In general, this is a preliminary implementation of multithreading. Some problems at present are:
        - There is no direct support yet for Thread Local Storage (TLS); instead, the entire FM context record is treated as TLS. This means that global and static variables will actually behave as though they are TLS. The only workaround to provide true shared global or static storage is to cache a variable in, say, the low or high range value of a user-defined control, using the **setCtlRange** and **getCtlRange** APIs. (Do not try to use the actual value of a control for global storage, since current control values are cached in the FM context record, and therefore act as TLS.)
        - Many of the basic FM API functions (such as **updatePreview**, **setCtlTextv**, etc.) are not yet thread-safe. If you want to call such a function in a threaded environment, you should explicitly serialize access to the function by surrounding the call to it with an **enterCriticalSection**/**leaveCriticalSection** pair

    of calls. In future releases, FM APIs will be made thread-safe whenever possible.
  - At present, the **getThreadRetVal** API is unable to return the correct exit code value of a thread.
  - The timeout value in the **waitForThread** API is very inaccurate (e.g., off by 3x or more).
  - There may still be some hidden resource leaks.

Beta 9c (1.0 Beta 9.0c) - 28 May 2008

- Main Improvements
  - Implemented the '!Y' descriptor (i.e., the current year formatted as 4 decimal digits) in **formatString**.
  - Implemented **ffillArray**, which is the floating-point analog of **fillArray**, and allows you to quickly fill an Array with a specified 16-, 32-, or 64-bit floating-point value.
  - Implemented unscaled pointer subtraction.
  - Warn that all pointer arithmetic is (currently) unscaled.
  - Warn that pointer comparisons are not yet implemented.
  - A floating-point value (float or double) can now be cast directly to an int with the (int) cast operator.
  - The compiler output pane in the Advanced (>>>) mode of the Editor display was widened by 40 DBUs to allow for more legible displays. The Font button was also repositioned.
- Bug Fixes
  - Fixed a regression bug in Beta 9a and Beta 9b which causes a crash when using the **msk**() API.
  - In the Advanced (>>>) mode Editor display, fixed a bug in the SDL dump of escape codes in strings when a non-ASCII char was treated as signed.

Beta 9b (1.0 Beta 9.0b) - 8 May 2008

- Bug Fixes
    - A problem with parsing floating-point numbers in the filter source code at Design time, when the default locale is not "English", has been fixed. (This bug was accidentally introduced in release 9a.)

Beta 9a (1.0 Beta 9.0a) - 5 May 2008

- Main Improvements
    - The maximum size of all global and static variables you can allocate has been increased from 400 bytes to 40 KB (i.e. 10,000 int variables or 5,000 double variables, or some combination thereof).
- Bug Fixes
    - A minor bug in **getAppTheme**() was (hopefully -- HH, please check!) fixed.
- Other Notes
    - This release was generated from the merger of Harry's and Alex's most recent source databases. No extensive regression testing has yet been performed against the previous release (1.0 Beta 9.0), so there is a possibility that some regression errors may be found; if so, please report them ASAP to the FMML group.

Beta 9 (1.0 Beta 9.0) - April 2008

- Main Improvements
    - FM can now use up to 3 GB of RAM under 32bit Windows (2000, XP, Vista). Under 64bit Windows it can now use up to 4 GB of RAM.
    - The FilterMeister Language (FML) is finally here. It is an alternative to the FilterFactory Plus (FFP) language that was used so far. To write code in FML you need to start your code with %fml instead of %ffp. In FML you can declare global variables outside handlers and use

control definitions like ctl(0): "Slider 1" inside handlers. You can also place variable declarations anywhere and do no need to put them at the top.
- You can now access DLLs with the following functions: **loadLib**(), **getLibFn**(), **callLib**(), **fcallLib**(), **callLibFmc**(), **fcallLibFmc**(), **freeLib**(). For third party DLLs you need to use **callLib**() and **fcallLib**(). The callLibFmc functions pass the FilterMeister context record as the first parameter to the called DLL function. This can be useful if you want to create your own DLLs for FilterMeister. FMLibTest_SourceCode.zip in the sourcecode\democodes subfolder contains a simple example for creating your own DLL with VC++.
- With the help of a resource editor you can now create multi-filter plug-ins with FM. FM now contains 32 entry points named ENTRYPOINT00 to ENTRYPOINT32. To have more than one filter in a .8bf file you need to use a resource editor to duplicate the pipl resource, change the 00 in ENTRYPOINT00 to 00, 01, 02 ... 29, 30 or 31 and edit the filter name in the pipl resource. If you copy a pipl resource from another FM .8bf file, you only need to change ENTRYPOINT00. In your filter code you can use the new ENTRYPOINT variable, which contains a value between 0 and 31, to check which menu item was selected. Future versions of FM will not require the use of a resource editor to create multi-filter plug-ins.
- All FM controls now reposition or scale automatically when the dialog is resized. All controls are anchored to the right side of the dialog by default. Only the preview, progress bar, zoom control and the OK and Cancel buttons have a different anchoring. If you want to change the anchoring of a control, you can use the Anchor control attribute or **setCtlAnchor**(). You can use a combination of the following constants: ANCHOR_LEFT, ANCHOR_RIGHT, ANCHOR_TOP and

    ANCHOR_BOTTOM. By default controls have ANCHOR_RIGHT set.
  - The .8bf plug-ins created with FM now have a resizable dialog, so you do not need to use a resource editor to enable it anymore.
- Dialog Stuff
  - The position and size of the preview, progressbar and zoom controls are now kept between sessions.
  - Scrollbar, standard, trackbar and slider controls now correctly work for value ranges below -32768 and above 32767.
  - You can now use more than 1024 characters for the text of edit, statictext, checkbox and radiobutton controls. But you can only read the first 1024 bytes with **getCtlText**() from them.
  - New **setCtlThumbSize**() function and thumbsize property for setting the thumb size of scrollbar, standard, trackbar and slider controls. For trackbar and slider controls you also need to use the FIXEDLENGTH attribute to make it work.
  - You can now change the background color of the preview with **setCtlColor**(CTL_PREVIEW, ...) or the color attribute. To remove the color again, simply use -1 as the color.
  - New **getDialogPos**() function for retrieving the position and size of the full window or the client area of the window.
  - If you remove the progressbar, zoom control or preview frame with "ctl(...):none", they do not appear again at the second invocation.
  - **setCtlFocus**() and **checkCtlFocus**() work again for CTL_PREVIEW.
  - **refreshCtl**() now also affects the labels and edit boxes of standard and slider controls.
  - **insertMenuItem**() does not accept values below 1 for the second parameter anymore.

    - Closing down FilterMeister with the x title bar button and no source code loaded caused a memory error message. This was fixed.
- XP/Vista Theme & Tab Control
    - **getAppTheme**() returns zero if the application that runs FM has visual styles disabled or 1 if it has visual styles enabled.
    - **setCtlTab**(n,t,s) or the new "Tab=(t,s)" attribute lets you assign controls to the sheets of a tab control. By doing so the controls automatically appear or vanish if a tab sheet is selected. You do not have to write some FM code to do that. Additionally if the tab control uses an XP/Vista theme, the color of checkboxes, radiobuttons and ownerdraws is automatically set to the color of the tab control.
    - **getCtlColor**() now returns the background color of tab controls. If the XP/Vista theme is activated, it will return an approximation of the tab background color, because themed tab controls have a gradient background.
    - If you use **enableCtl**() on a tab control, the controls that are assigned to the tab control also become visible or invisible.
    - If you use **refreshCtl**() on a tab control, it now also refreshes the controls that are assigned to the tab control.
    - Bug Fix: **setCtlVal**() now works correctly for tab controls.
    - Bug Fix: When using **setCtlText**() on a tab control, it removes the old tab sheets and does not add additional tab sheets anymore.
    - Bug Fix: **setCtlItemText**() does not add additional tab sheets to tab controls anymore if the specified tab sheet already exists.
- Other Improvements

    - Bug fix: Images that are only a few pixels wide or high produced an error message. This was fixed by setting the initial preview zoom to an appropriate value.
    - Arrays now support 16bit float values. To store them use **fputArray**() and to retrieve them use **fgetArray**(). For 16bit float values please use 2 as the last parameter in **allocArray**().
    - **getSysMem**() returns some values about the system memory.
    - **sizeof**() returns the size of a few basic types (e.g. sizeof(int)), integer and real constants, string literals, and a few basic scalar variable types. Other uses of sizeof will still result in compiler error or bug messages, or even incorrect results.
    - **formatString**() (which is also used by other functions like **setCtlTextv**() and **setDialogTextv**()) supports HTML entities and new special codes
    - **fmax**() and **fmin**() now take a variable number of arguments.
    - **fc2d**() now correctly asks for two instead of four parameters.
    - **triggerEvent**() now only accepts a control number from 0 to 256 for normal events and an unlimited control number for FME_CUSTOMEVENT without displaying a memory error message.
    - For **floor**(), **ceil**(), **chop**(), and **round**() inline code is produced, which makes them much faster.
- setPixel Functions
    - Bug fixed in **setThinLineAA**()
    - New **setRectangle**() function
    - **setText**() and **setTextv**() were updated

Beta 8.7 (1.0 Beta 8.7) - January 2008

- Bug Fixes

    - The black borders around the edit boxes of standard and slider controls vanished on the second invocation in Beta 8.6. This was fixed.
    - The labels of the checkboxes in Advanced (>>>) mode are visible again under XP/Vista.
    - The folder of the .ffp file is now set as the working folder on Compile. If no .ffp file has been loaded or saved, the filterInstallDir is set as the working folder on Compile. This avoids problems with embedded files not being found on Compile e.g. if a preset has been opened from another folder.
    - The dialog events are now cleared on Compile to avoid that they are triggered for other filter code in which they are not set.
- Improvements
    - Arrays now support float and double values. To store them use **fputArray**() and to retrieve them use **fgetArray**(). For float values use 4 and for double values use 8 as the last parameter in **allocArray**().
    - If the user cancels the filter (e.g. by pressing the ESC key or x button in the title bar or pressing Enter when the Cancel button has the input focus or choosing "Close" from the menu of the title bar icon), an event with n=CTL_CANCEL and e=FME_CLICKED will now be triggered. So you do not need to use "Dialog:CancelEvent" or setDialogEvent(2) anymore for these cases.
    - The number of global strings was increased from 10 to 20. You can access them by using str0 up to str19.
    - New **DESIGNTIME** variable which lets you check if the filter is running in FilterMeister (DESIGNTIME==true) or as a standalone plug-in (DESIGNTIME==false).
    - **getBufferAddress** and **pointer_to_buffer** now support the T3 and T4 buffers. The parameters are 4 and 5.
- setPixel Functions

    - New **setAngleArcFill** and **setAngleArc** functions
    - New **setTextv**() function that supports formatted text
- Installation
    - New multi-lingual installation
    - The HTML Manual now offers more information including an updated command reference.

Beta 8.6 (1.0 Beta 8.6) - November 2007

- Improvements
    - FM does not crash anymore on Make when embedding bitmaps in a plug-in
    - In some host applications (e.g. SignLab, Image Analyzer, Pixopedia, NiGulp, GIMP) memory error messages appeared and/or horizontal preview dragging was not possible. This is fixed now.
    - Some host applications (e.g. GIMP, PhotoBrush, PluginMaster) do not display a color dialog. **chooseColor** now displays the normal Windows color dialog for these applications.
- Dialog Stuff
    - Bug fix: Check boxes do not have a black label anymore when the dialog theme is switched on.
    - The styles of the label and edit box of standard and slider controls are now saved between filter invocations
    - **setCtlBuddyStyle**, **setCtlBuddyStyleEx**, **clearCtlBuddyStyle**, **clearCtlBuddyStyleEx** now save the styles between filter invocations
    - To make sure that your plug-in always uses the appropriate system colors please use the new SysColor option instead of the Color option when defining controls, e.g. "ctl(0): syscolor=COLOR_BTNFACE" or "Dialog: SysColor=COLOR_BTNFACE". There is also a FontSysColor option. These options correspond to

    new **setDialogSysColor**, **setCtlSysColor** and **setCtlFontSysColor** functions.
  - The new scrollFactor variable lets you increase the scroll speed in the preview. scrollFactor=1 is the default.
- SetPixel Stuff
  - A bug in **setThinLineAA** and **setThinEllipseAA** has been fixed.
  - **setEndCaps** sets the method used to cap the ends of lines with the constants PS_ENDCAP_ROUND, PS_ENDCAP_SQUARE and PS_ENDCAP_FLAT.
  - **setJoin** sets the method used to join thick lines at corners of shapes with the constants PS_JOIN_ROUND, PS_JOIN_BEVEL and PS_JOIN_MITER.
  - **setInsideFrame** determines whether the pen lines of shapes are drawn on the edge or inside the edge.
- Others
  - Floating point versions of **r2x**, **r2y**, **c2m** and **c2d** are now available. They are called **fr2x**, **fr2y**, **fc2m** and **fc2d**.
  - **FME_INIT** is now triggered after hitting the Compile button. This will make it easier to debug the filter code.
  - A **cell_preserve** alias was added for **cell_initialize**(). cell_preserve(1) keeps the values of put/get cells between preview updates and filter invocations.
  - New directory/folder functions: **chdir** lets you change the working folder, **getcwd** returns the current working folder, **mkdir** creates a new folder and **rmdir** deletes a folder.

Beta 8.5 (1.0 Beta 8.5) - November 2007

- Main

    - The **ForEveryTile** code buffer size was increased from 256K to 512K.
    - The literal pool was doubled, so that you can use more floating-point and string constants in your code.
- Dialog Theme
    - The FM title bar now has no theme on first invocation. If you activate the dialog theme in your filter code with **setDialogTheme**(1) or "Dialog: Theme=on", the titlebar will not lose its theme anymore like in previous FM versions.
    - Activating the dialog theme will now also activate the theme for all controls unless you specify that a control should not have a theme.
    - You can now use **setCtlTheme**(n,-1) or the control attribute "theme=default" to make a control use the same theme as the dialog. This is the default setting now.
- Events
    - Three new event constants replace old event constants: FME_CHANGED replaces FME_VALUECHANGED, FME_SETFOCUS replaces FME_SETEDITFOCUS and FME_KILLFOCUS replaces FME_KILLEDITFOCUS. For backward compatibility the old event constants are still valid and work.
- SetPixel Stuff
    - The **setPixel** functions do not produce a crash anymore if you apply them thousands of times.
    - The new **setPenWidth** function lets you change the broadness of drawn lines.
    - **setThinLineAA** and **setThinEllipseAA** draw anti-aliased lines and ellipses.
    - **startSetPixelSS**(n, factor) and **endSetPixelSS**(n) let you do super sampling for smoother results. The second parameter in startSetPixelSS is the factor by which the canvas will be larger.

    - **startSetPixelBuffer**(width, height) lets you draw on an internal buffer. So you do not need any controls to be present, so you can even use it after the user pressed OK on the plug-in dialog. Use the **getPixel**(x,y) function to read out the buffer. **endSetPixelBuffer** destroys the buffer.
    - **getSetPixelWidth** and **getSetPixelHeight** return the size of the canvas or buffer. They only work if used after **startSetPixel**, **startSetPixelSS** or **startSetPixelBuffer**.
- New Controls
    - New TAB control.
        - Control Value: The selected tab sheet.
        - Events: FME_CLICKED.
        - Styles: scrollopposite, bottom, right, multiselect, flatbuttons, forceiconleft, forcelabelleft, hottrack, vertical, tabs, buttons, singleline,multiline, rightjustify, fixedwidth, raggedright, focusonbuttondown, ownerdrawfixed, tooltips, focusnever (see **http://msdn2.microsoft.com/en-us/library/bb760549.aspx**).
    - New **EDIT** control.
        - Control Value: Not used.
        - Events: FME_CHANGED, FME_SETFOCUS, FME_KILLFOCUS.
        - Styles: left, center, right, multiline, uppercase, lowercase, password, autovscroll, autohscroll, nohidesel, oemconvert, readonly, wantreturn, number (see **http://msdn2.microsoft.com/en-us/library/bb775464.aspx**).
    - New **SLIDER** control (consisting of a label, trackbar and edit box)
        - Events: like **STANDARD** control: FME_CHANGED, FME_SETFOCUS (with mouseover style)

        - Styles: like **TRACKBAR** control
- More Dialog Stuff
    - The top border of the edit box of standard controls (and sliders) is not cut off like in previous FM versions under Windows XP.
    - You can now add "editstyle=border", "editstyleex=staticedge" or "editstyleex=clientedge" to the control definitions of standard and slider controls for changing the look of the editboxes.
    - The new **setCtlBuddyStyle**, **setCtlBuddyStyleEx**, **clearCtlBuddyStyle** and **clearCtlBuddyStyleEx** functions were added for changing the style of the labels and edit boxes of standard and slider controls. Use 1 as the middle parameter to change the edit box and 0 to change the label.
    - The trackbar does not have a dark grey background color by default anymore
    - The color of the list box is now set automatically to avoid redraw problems that occurred if you did not explicitly set the color in previous versions
    - New **getSysColor** function for grabbing a system color. You can use the COLOR constants with it, e.g. COLOR_BTNFACE.
    - New **setCtlStyle**, **setCtlStyleEx**, **clearCtlStyle** and **clearCtlStyleEx** functions
    - **setDialogMinMax** works correctly for negative parameters now
    - **setCtlItemTop** and **getCtlItemTop** for setting and getting the top visible item in a combo box or list box
    - **setCtlItemText** and **getCtlItemText** for setting and getting the text of a combo box, list box or tab control item
    - **deleteCtlItem** for deleting an item of a combo box, list box or tab control
    - **deleteCtlItems** for deleting all items of a combo box, list box or tab control

    - **fillDir** fills a combo box or list box with file and folder names
    - **getCtlPos** now also supports the preview.
    - The new **checkDialogFocus** returns true if the FM dialog is the active window and false if it isn't.
- Buffers & Arrays
    - A fourth t-buffer was added. Use **t4get** and **t4set** etc.
    - **t3get** and **t3set** did not work previously. They were fixed.
    - **iget** supports the third and fourth t-buffer now
    - **src** previously returned zero instead of 32768 for the alpha values of 16-bit images whose coordinates were outside the image borders. This was fixed.
    - **allocArray** and **allocArrayPad** now by default allocate memory from Photoshop. Previously they allocated memory from Windows, but if Photoshop had grabbed large portions of the available memory, FM was not able to access it and failed because not enough memory was available to it. If Photoshop cannot provide the necessary memory, FM tries to get it from Windows. (So it is best if you recommend to the users of your plug-ins to set the Photoshop Memory Usage preference setting to a high value.) If FM is not running in Photoshop, the memory allocation will still be done by the host, but that usually means that the memory will be allocated from Windows, because most hosts do not have their own memory management system like Photoshop.
    - You can switch back to the old behavior (Windows allocation) by using **set_array_mode**(0). This is necessary if you want to have reallocatable arrays, because Photoshop allocation does not allow reallocation. **set_array_mode**(1) switches back to Photoshop allocation.
    - The t-buffers are allocated by Photoshop now, which will increase performance in Photoshop.

- Resources
    - New **copyResToArray**(resname, restype, arraynr) function for copying a file resource to an array. Alternatively **getResAddress**(resname, restype) returns a pointer to the specified file resource and **getResSize**(resname, restype) returns the size of the file resource.
    - You can now embed any kind of file by using the Other item, e.g. Embed: Other="Test.txt". This file will be embedded as a "FMDATA" resource type. You can access e.g. with copyResToArray("FMDATA","TEST.TXT",0);
- Events
    - The **FME_DRAWITEM** event now works for the preview. To activate it you need to add "ctl(CTL_PREVIEW):preview(drawitem)" at the top or "**setCtlProperties**(CTL_PREVIEW,CTP_DRAWITEM);" somewhere.
    - The precision for **FME_MOUSEOVER** and **FME_MOUSEMOVE** events was increased from 250 ms to 20 ms. For the preview these events are and always were triggered instantly.
    - New **FME_INIT** event is triggered before the FM dialog is displayed. To activate this event you need to use "Dialog: initevent" or **setDialogEvent**(1). You can use the previous value to check if the plug-in is executed for the first time in the host. In that case previous==false.
    - New **FME_CANCEL** event is triggered when the user exits the plug-in without applying the effect. At this point the dialog already has vanished and changes to global variables are not preserved anymore for the next invocation. To activate this event you need to use "Dialog: cancelevent" or **setDialogEvent**(2).
    - To activate the new **FME_KEYDOWN** and **FME_KEYUP** events you need to use "Dialog:

keyevents" or **setDialogEvent**(4) in your code. The n value of the events contains the VK code of the pressed key. If you want to check for Shift, Ctrl, Alt etc. you need to use **getAsyncKeyStateF**().
    - The new setDialogEvent function lets you activate the **FME_INIT**, **FME_CANCEL**, **FME_KEYDOWN** and **FME_KEYUP** events. To deactivate these events use **clearDialogEvent**. Use 7 as the parameter for activating or deactivating all events.
- Others
    - Tool tips work again under Windows 2000/XP/Vista. To enable the tool tips for statictext, image and frame controls you need to add the notify attribute, e.g. ctl(0): statictext(notify), "Test", tooltip="Here I am"
    - The following floating point constants were added: M_E, M_LOG2E, M_LOG10E, M_LN2, M_LN10, M_PI, M_PI_2, M_PI_4, M_1_PI, M_2_PI, M_2_SQRTPI, M_SQRT2, M_SQRT1_2

Beta 8.4 (1.0 B8) - September 2007

- Improvements
    - DEP problem was fixed. Even if DEP is activated, FM filters and FM plug-ins will be executed.
- T-Buffers
    - Third t-buffer available now (**t3get**,**t3set** etc.)
    - The t-buffers support 16-bit images now. But only **tget**,**t2get**,**tset**,**t2set**,**t3get** and **t3set** work with 16 bit images. The following functions will not work correctly with 16bit data: **tgetp**,**tsetp**, **tgetr**, **tsetr** and their equivalents for the second and third t-buffer.
- Dialog Stuff
    - **getCtlText** for retrieving the text of a control
    - Combo boxes and list boxes now accept text that is longer than 1023 bytes, which means that you can add

    more items to them. However, **getCtlText** will only return a combo box or list box text that is 1023 bytes long.
  - **getCtlPos** for finding out the position and size of a control
  - **setDialogTheme** and **setCtlTheme** for switching XP/Vista themes on and off. Does not work correctly at the moment.
  - **getPreviewCursor** will return the resource number of the current preview cursor
  - **getCtlClass** tells you what type of control a certain control is.
- Others
  - **getSpecialFolder** function
  - **getArrayAddress** and **getBufferAddress** return the memory address of arrays and buffers
  - **blend** now also works with 16-bit values if you use **set_bitdepth_mode**(16)

Beta 8 (1.0 B8) - March 2007

- Improvements
  - Bug Fix: Plug-ins that were created with previous versions of FM crashed under Windows Vista if there was more than 2GB RAM available.
  - The code buffers were increased:
    - Maximum source code size is now 800,000 bytes
    - The **OnCtl** buffer was increased from 256k to 512K
    - Larger compiler buffer to handle more complex algorithms
- SetPixel Stuff
  - **startSetPixel**, **setPixel** and **endSetPixel** work much faster and flicker-free

  - **getPixel** for reading pixel values from the ownerdraw control
  - **setFont** lets you choose the font name, its size, angle, boldness, and italics
  - **setText** lets you draw a certain text with the chosen font, color and alignment (Available alignment constants: TA_BASELINE, TA_BOTTOM, TA_TOP, TA_CENTER, TA_LEFT, TA_RIGHT, TA_NOUPDATECP, TA_RTLREADING, TA_UPDATECP, VTA_BASELINE, VTA_CENTER)
  - **setLine** draws a line
  - **setEllipse** draws an ellipse
  - **setEllipseFill** draws a filled ellipse
  - **setRectFrame** draws a rectangle
  - **setRectFill** draws a filled rectangle
  - **setFill** fills the whole ownerdraw control with a certain color e.g. setFill(getCtlColor(n))
- Other Things
  - **checkCtlFocus** function for checking if a certain control has the focus.
  - Bug Fix: The dialog of created plug-ins is not resizable by default anymore. You need to use a resource editor to make it resizable if you support dialog resizing in your plug-in.
  - AutoLoad button for automatically loading and compiling the .ffp file if it was changed on disk in the meantime. Not sure if it works correctly in Photoshop.

Beta 7 (1.0 B7) - June 2005

- New Features
  - Shift clicking the Load button reloads and compiles the current .ffp file
- Improvements

    - **trackPopupMenu** has one more parameter for menu alignment now. Set it to zero for default behaviour or use the following constants: TPM_RIGHTBUTTON, TPM_LEFTALIGN, TPM_CENTERALIGN, TPM_RIGHTALIGN, TPM_TOPALIGN, TPM_VCENTERALIGN, TPM_BOTTOMALIGN, TPM_HORIZONTAL, TPM_VERTICAL, TPM_NONOTIFY, TPM_RETURNCMD.
    - **setCtlProperties** now works for the preview control.
- Bug Fixes
    - **src** lets you read the alpha channel of Greyscale images again
    - The dialog of a compiled plug-in was displayed with a delay in Photoshop if the preview events were activated with "ctl(CTL_PREVIEW): preview (mousemove, mouseover, previewdrag)". Until this problem is solved the preview events are activated again by default.
    - The preview works again for 16-bit Greyscale and CMYK images.
    - **getCtlCoord**, **getPreviewCoordX** and **getPreviewCoordY** work correctly again for zoom levels other than 100%
    - Sliders didn't update the preview in Beta 6 if there was no ctl(0) defined.

Beta 6 (0.4.21) - May 2005

- Alpha Channel Access
    - Transparency is finally correctly displayed as a checkerboard pattern for 16-bit layers.
    - **src** now lets you access the alpha channel of 16-bit layers
    - **src** can now access the transparency information from CMYK images as well as additional alpha channels that

are created and selected in the Channel palette in Photoshop.
    - **Z** had a maximum value of 4 previously. Now it is 5 if it is applied to a CMYK layer. It can even have a higher value if additional alpha channels are selected in the Channel palette in Photoshop.
    - **planesWithoutAlpha** contains the number of image channels without alpha channels. You can now calculate the number of available alpha channels by subtracting planesWithoutAlpha from planes or **Z**.
- Selection Mask
    - **msk** for accessing the selection mask works now for 8-bit and 16-bit images. The selection mask has values from 0 to 255 even for 16-bit images.
    - New functions for calculating the smallest distance to the selection border: **calcSBD**, **freeSBD**, **getSBD**, **getSBDangle**, **getSBDX**, **getSBDY**. They only work for 8-bit images at the moment.
- Controls
    - **getCtlCoord** function for accessing the coordinates of the mouse over a control. It is also a replacement for **getPreviewCoordX** and **getPreviewCoordY**. Both functions are now internally redirected to **getCtlCoord**.
    - **setCtlFocus** for setting the focus to a certain control.
    - **getCtlRange** for checking the min. and max. value of a control.
    - **ctl**, **getCtlVal** and **getCtlRange** can now also be used for CTL_PROGRESS to get the progress bar value and range
- Events
    - drawitem control property now needs to be set to trigger **FME_DRAWITEM** events.
    - mousemove control property can now be set to generate **FME_MOUSEMOVE** events for all controls.

  - FME_VALUECHANGED events are now triggered for trackbars
  - New FME_PREVIEWDRAG event is triggered after the preview was dragged, but before the preview is updated. To activate the FME_PREVIEWDRAG events you need to set the previewdrag control definition, e.g. ctl(CTL_PREVIEW): preview (previewdrag)
  - The **FME_MOUSEMOVE**, **FME_MOUSEOVER** and FME_PREVIEWDRAG events for the preview are now only triggered if you specify them as a control property. For example you can do that by adding "ctl(CTL_PREVIEW): preview (mousemove,mouseover,previewdrag)" at the top of your code.
- Other New Features
  - **requestRect**(x_start, y_start, x_end, y_end, scaleFactor) for requesting an image rectangle. Lets you read pixels outside the preview if isTileable=true. Doesn't work after OK was pressed. **restoreRect** makes sure that the preview is displayed correctly again after requestRect was used.
  - **findFirstFile**, **findNextFile** and **findClose** for reading file and folder names from a certain path.
  - **fmin** and **fmax** for finding the minimum or maximum of two double values.
  - imageHRes and imageVRes double variables were added for getting the dpi value(s) of the image.
  - You can now use the mouse wheel to change slider values. You need to select the slider at first by clicking on it or using the tab key.
- Improvements
  - **fillArray** now lets you fill arrays with 8-bit, 16-bit and 32-bit values.
  - **srcp**, **pgetp** and **psetp** have been made slightly faster
  - **getArrayString** now returns the string "Not Available" if it can't read a string. Previously it returned zero,

    which caused a memory access violation error in some cases.
  - **set__psetp__mode**(1) for making it possible to set the alpha channel to zero with **psetp**
  - **iget** works on 16-bit images now.
  - The Dodge and Burn modes of **blend** have been changed to produce a more correct result.
- Bug Fixes
  - Problem with non-working arrays when running FilterMeister a second time in PhotoPlus and Canvas was fixed.
  - Bug with **getImageTitle** in PhotoPlus was fixed.
  - The algorithm of **saturation** was changed to adjust saturation correctly.
  - Holding down the Shift key while dragging a slider or trackbar won't produce any preview flicker anymore, because the preview is only updated a maximum of 50 times per second.

Beta 5 (0.4.21) - December 2004

- Larger buffers which means larger and more complex filters can be created:
  - Maximum source code size is now 400K
  - onCtl handler: 128K -> 256 K
  - larger compiler buffer to handle more complex algorithms
- Resizable Window
  - The FM dialog is resizable by default, but the windows of the generated plug-ins are not resizable by default. To make them resizable, you need to use a resource editor and change the window style to "Resizable".
  - New related functions: **getDialogWidth**, **getDialogHeight**, **setDialogMinMax**, **PixelsToHDBUs**, **PixelsToVDBUs**

    - New related events: FME_SIZE, FME_EXITSIZE
    - You can now resize and reposition the preview during runtime. You have to use **setCtlPos**(CTL_PREVIEW, ...) to do that. The frame around the preview will be automatically repositioned.
    - To reposition and resize the preview frame, progress bar or zoom controls please use **setCtlPos**(CTL_FRAME, ...), **setCtlPos**(CTL_PROGRESS, ...) or **setCtlPos**(CTL_ZOOM, ...)
    - You can also use **enableCtl** to make the preview frame, progress bar or zoom controls disappear.
    - Less recommended is to use **deleteCtl** to remove the preview frame, progress bar or zoom controls.
    - **setDialogSizeGrip** displays a grip for resizing the window (produces redrawing artifacts in some cases)
    - **setDialogShowState** lets you minimize, maximize or normalize the FilterMeister window
- Processing in Tiles
    - The needPadding variable works now. It lets you set the size of the padding area in pixels, e.g. needPadding = 2 for a 5x5 convolution operation. You need to set isTileable = true to make it work.
    - New **allocArrayPad** (nr, X, Y, Z, bytes, padding) function for allocating padded arrays
    - FilterMeister currently processes 100 full rows at a time when isTileable = true. This value can be adjusted by assigning a new value to the bandWidth variable. For example bandWidth = 200 for processing 200 rows at a time.
    - The Big Gulp check box (Advanced mode) is now deactivated. This avoids that the whole image is loaded to memory when processing in tiles (isTileable = true). As a result there won't be any "Not enough Memory" messages anymore when processing in tiles.
- Drawing on Controls

    - New **startSetPixel**, **setPixel** and **endSetPixel** functions for drawing on controls, usually bitmap, image or ownerdraw controls.
    - New **FME_DRAWITEM** event that is triggered when an ownerdrawn control needs to be redrawn
- Event Triggering and Pseudo-Functions
    - **triggerEvent** lets you execute the code in the **OnCtl** handler that is usually only executed if that event occurs.
    - By checking for the new constant FME_CUSTOMEVENT in the **OnCtl** handler, you can place some custom code there that can be executed with the **triggerEvent** function.
- Context Menus
    - The FME_CONTEXTMENU event is triggered when the user right-clicks on the dialog background
    - **createPopupMenu** creates a context menu and returns the menu handle that is needed for the other menu functions
    - **insertMenuItem** adds a menu item.
    - **trackPopupMenu** displays a context menu and returns the selected menu item
    - **destroyMenu** for deleting the menu handle if it isn't needed anymore.
- Menus (ONLY FOR TESTING -> No Menu events yet, so the selected menu item isn't passed back yet)
    - **createMenu** creates a menu and returns the menu handle that is needed for the other menu functions
    - **insertMenuItem** adds a menu item.
    - **setMenu** for displaying the menu under the title bar
    - **destroyMenu** for deleting the menu handle if it isn't needed anymore.
- Arrays
    - the number of possible arrays was extended from 10 to 100.

    - New **fillArray** function for filling an array with a certain byte value, e.g. for initializing all cells of the array to zero.
    - New **putArrayString** and **getArrayString** functions for storing strings in arrays.
- New Events
    - FME_COMBO_DROPDOWN, FME_COMBO_CLOSEUP are triggered when the drop down list displays and closes again.
    - FME_SETEDITFOCUS, FME_KILLEDITFOCUS are triggered when the focus is set on the edit box of a slider and when the focus is removed again from it.
- Sliders
    - **setCtlEditSize** function and editsize property for changing the size of the edit box of standard sliders
    - **setCtlGamma** function for changing the slider behaviour
- Other
    - **set__bitdepth__mode**(mode) function. Setting mode to 16 will make the **rgb2hsl**, **hsl2rgb** and other color conversion function treat the passed color values as 16-bit color values. Setting mode to 8 will activate the default behaviour of these functions.
    - **doEvents** pauses the current code to update the window and process events. Useful inside a loop that needs longer to finish.
    - Several value interpolation functions: **linearInterpolate**, **cosineInterpolate**, **cubicInterpolate** and **hermiteInterpolate**
    - **quickFill** fills the put/get cells quickly with image values and **quickMedian** calculates the median of the values from the put/get cells quickly.
    - **quickSort** function for sorting put/get cells
    - **getImageTitle** retrieves the path or file name of the currently processed image. Only works in Photoshop

    (returns full path) and PSP (only returns file name without file extension).
  - **egw** function was updated
  - New **egm**(a,b,value) function for edge mirroring - NOT YET AVAILABLE! (mwvdlee)
- Bug Fixes:
  - ForEveryPixel and RGB handlers work in PSP 8 again without causing a memory error
  - The preview displays fine and not half transparent anymore when applying FilterMeister to a 16-bit layer in Photoshop CS

Beta 4 (0.4.21) - June 30, 2003

- Bug Fixes:
  - Message box that can occur under XP was removed
  - If compilation failed, the dialog doesn't shrink anymore, but gets its original size back.

Beta 3 (0.4.21) - June 16, 2003

- Bug Fixes:
  - A bug in the **freeArray**() function prevented the memory of the array from being freed and even worse kept FilterMeister itself from automatically freeing the memory on exit

Beta 2 (0.4.21) - June 3, 2003

- Dialog Changes:
  - All dialog controls are now displayed in classic style under Win XP. Currently you can't activate XP styles, but it will be possible later.
  - Advanced button in the editor was added again

    - The size of the standard slider text boxes has been increased to fully display decimal values
- Bug Fixes:
    - FilterMeister welcome message isn't displayed anymore in created plug-ins.
    - Stack overflow problem that crashed FM was fixed
    - Problem with disabled editor control was fixed
    - Problem with freeing of allocated arrays was fixed
    - **srcp**, **pgetp**, **tgetp**, **t2getp** check for image boundaries now to avoid memory error messages
    - The getPreviewCoord functions work precisely now
    - **rgb2ycbcr** and **ycbcr2rgb** work more precisely now
- New Attributes:
    - when defining a standard control you can use the new divisor attribute to make the slider display decimal values. E.g. set divisor=1000 for three decimal places. The default divisor value is 1.
- New Functions:
    - **setCtlDivisor**(n,value) for having standard slider with decimal places. *value* can be a value from 1 to 1000. The default is a value of 1.
- New Events:
    - FME_VALUECHANGED is triggered when a slider value was changed. Returns the previous slider value in a variable called previous.
    - FME_CLICKED events now return the previous value of combo boxes in a variable called previous for combo boxes, list boxes, check boxes and radio buttons

Beta 1 (0.4.20) - March 22, 2003

- Dialog Changes:
    - Larger main dialog
    - 200% larger preview
    - system color is used for the dialog background

    - system color is used for slider labels and statictext controls
    - The editor window is now automatically displayed when you start FilterMeister
    - The Edit >>> button was removed and instead a Minimize button was added to the title bar of the editor
    - When disabling a standard slider control, the slider label is now disabled, too.
- Larger code buffer which means that filters with a larger and more complex code can be created:
    - onCtl: 64K -> 128K
    - ForEveryTile: 128K -> 256K
- New Color Modes:
    - Lab48Mode
    - CMYK64Mode
    - DeepMultichannelMode
    - Duotone16Mode
- New Functions:
    - **pget** and **pgetr** support 16 bit images
    - for registry access
    - for color space conversion (YUV, Lab, HSL, YCbCr)
    - **iget** for interpolated image access with 5 methods (nearest, bisquare, bicosine, bilinear, bicubic)
    - **srcp**, **pgetp** etc. for accessing a whole pixel at once. Using these functions lets you read and write image data almost twice as fast as with **src**() and **pset**().
    - for drawing rectangles, circles and triangles easily
    - for locking the window redraw and for redrawing the whole window, a control or a region
    - for setting the preview zoom (**setZoom**)
    - for reading the coordinates of the mouse pointer above preview
    - for using up to 10 arrays with one, two or three dimensions

    - for performing 20 different **blend** operation with two image sources
    - for drawing a dozen two-dimensional gradients
    - for edge-wrapping values
    - for checking if keys were pressed
    - for getting the current Window Version, the screen resolution and screen refresh rate.
    - for triggering up to 10 different timer events e.g. to perform multi-threading-like activities (**setTimerEvent**).
- New Events:
    - FME_TIMER: for asynchronous activities (timer)
    - indicating left and right clicks on the preview (FME_LEFTCLICKED_DOWN, FME_LEFTCLICKED_UP, FME_RIGHTCLICKED_DOWN, FME_RIGHTCLICKED_UP)
    - **FME_MOUSEMOVE**: indicating a mouse movement above the preview (mousemove)

FM mt5c07 "As-Is" Release Notes 11/12/2015

This is an "as-is" release for early adopters. It contains specific fixes and enhancements needed by several filter designers. It also contains widespread but incomplete changes to the FM compiler infrastructure code, which have not yet been tested and will not be documented for this release. It is possible that these infrastructure changes may introduce new bugs, so if you do not require the following specific changes, it is recommended that you continue your present development with the current mt5c06 "stable" release until such time that a fully regression-tested version (mt5c08) is released.

Release mt5c07 contains the following changes

- The size of the literal pool for string and floating-point constants has been doubled from 64K bytes to 128K bytes.
- A bug in the fillArray and ffillArray functions which would crash on Arrays bigger than 2G-1 bytes has been fixed.

Release mt5c07 still contains the following previously unreported bugs in the allocArray function

- A crash may occur if an Array is allocated with fewer than 2G bytes from the malloc pool, and is then reallocated with the same number of bytes from the Host memory pool; or vice versa.
- A crash may occur if an Array is first allocated from the Host memory pool, and then the same array is reallocated from the malloc pool. (This can happen inadvertantly if you first allocate an Array with fewer than 2G bytes, and then reallocate the same array with 2G or more bytes.)
- In X32 mode, allocArray cannot allocate an Array larger than 2G-1 bytes.

Contents of this Zip file

README_mt5c07.txt This README file

AfhFM04dx32mt5c07.8bf The FM32 mt5c07 plug-in file

AfhFM04dx64mt5c07.8bf The FM64 mt5c07 plug-in file

DISCLAIMER

This is an "As-Is" release. Please use these files at your own risk.

Filter Templates

Here are some templates. Please feel free to edit and enhance them. But please try to retain their style. Also feel free to add your own templates.

Rules:

1. Try to comment as much code as possible with easy to understand explanations.

2. Add a larger comment at the top that explains the purpose of the template.

3. Make sure that all code and comments are visible in the FilterMeister Editor, so that the user doesn't need to vertically scroll to read it.

4. Add as many empty lines or empty space as possible to make the templates easy to read.

5. Don't add any "complex" code to the templates. Try the easiest approach that is possible. For example, your templates include some code that isn't really necessary, e.g. the total_cost stuff and the Zend variable.

Simple Template 1

Simple Template 2

Simple Template 3

FF in FET Template

Simple Template 1

```
%ffp

// This is a simple template for a
// FilterMeister filter. To get started
// simply replace the current values with
// your own ones. You can also try to
// replace parts of the current code or
// even add your own code.


//-----------------------------------------
// Filter Infos

// The name of the sub menu on which your
// filter will appear in the graphics
// application
Category:     "Harry's Filters"

// The name of the filter and the
// sub menu item
Title:        "Colorize"

// Your plug-in will be saved under
// this name
Filename:     "Colorize.8bf"

// Other values
Copyright:    "Freeware"
Author:       "Harald Heim"
Organization: "The Plugin Site"
URL:          "http://thepluginsite.com"
Description:  "Colorize the Image\n"
```

```
Version:      "1.0"

// Determines what will be displayed on
// the About dialog of your plug-in:
// Title (!T), Version (!V), Description
// (!D), Copyright (!c) and URL (!U) each
// in a separate line
About:         "!T !V\n!D\n!c\n!U"


//------------------------------------------
// Filter Control Definitions

ctl(0): "Red",    // The slider label
  Range=(-128,128), // Value range of the slider
  Pos=(220,20),   // Position of the slider on the dialog
  Size=(*,7),     // Size of the slider
  Val=0     // Initial value of the slider

ctl(1): "Green", Range=(-128,128),
        Pos=(220,30), size=(*,7), Val=0
ctl(2): "Blue", Range=(-128,128),
        Pos=(220,40), size=(*,7), Val=0

// Hide the Edit button and the logo
ctl[CTL_EDIT]: none
ctl[CTL_LOGO]: None


//------------------------------------------
// Here comes the filter code

// In this case the value of the sliders
// are added to the r, g and b color
// values and saved in the R, G and B
// channels
```

```

R = r + ctl(0)
G = g + ctl(1)
B = b + ctl(2)
```

Simple Template 2

```
%ffp

// This is a simple template for a
// FilterMeister filter. To get started
// simply replace the current values with
// your own ones. You can also try to
// replace parts of the current code
// or even add your own code.


//------------------------------------------
// Filter Infos

// The name of the sub menu on which your
// filter will appear in the graphics
// application
Category:     "Harry's Filters"

// The name of the filter and the sub
// menu item
Title:        "Colorize"

// Your plug-in will be saved under
// this name
Filename:     "Colorize.8bf"

// Other values
Copyright:    "Freeware"
Author:       "Harald Heim"
Organization: "The Plugin Site"
URL:          "http://thepluginsite.com"
Description:  "Colorize the Image\n"
```

```
Version:              "1.0"

// Determines what will be displayed on
// the About dialog of your plug-in:
// Title (!T), Version (!V), Description
// (!D), Copyright (!c) and URL (!U) each
// in a separate line
About:       "!T !V\n!D\n!c\n!U"


//-------------------------------------------
// Filter Control Definitions

ctl(0):         "Red",          //The slider label
        Range=(-128,128),       //Value range of the
slider
        pos=(220,20),   //Position of the slider on the
dialog
        size=(*,7),             //Size of the slider
        Val=0           //Initial value of the slider

ctl(1): "Green", Range=(-128,128),
          Pos=(220,30), Size=(*,7),Val=0
ctl(2): "Blue", Range=(-128,128),
          Pos=(220,40), Size=(*,7),Val=0

// Hide the Edit button and the logo
ctl[CTL_EDIT]: none
ctl[CTL_LOGO]: None


//-------------------------------------------
// Here comes the filter code

ForEveryPixel:
{

  // In this case the value of the
  // sliders are added to the r, g and b
  // color values and saved in the
  // R, G and B channels

  R = r + ctl(0);
  G = g + ctl(1);
  B = b + ctl(2);

}
```

Simple Template 3

```
%ffp

// This is a simple template for a
// FilterMeister filter. To get started
// simply replace the current values with
// your own ones. You can also try to
// replace parts of the current code or
// even add your own code.


//------------------------------------------
// Filter Infos

// The name of the sub menu on which your
// filter will appear in the graphics
// application

Category:       "Harry's Filters"

// The name of the filter and the sub
// menu item
Title:          "Colorize"

// Your plug-in will be saved under this
// name
Filename:       "Colorize.8bf"

// Other values
Copyright:      "Freeware"
Author:         "Harald Heim"
Organization:   "The Plugin Site"
URL:            "http://thepluginsite.com"
```

```
Description:    "Colorize the Image\n"
Version:        "1.0"

// Determines what will be displayed on
// the About dialog of your plug-in:
// Title (!T), Version (!V), Description
// (!D), Copyright (!c) and URL (!U)
// each in a separate line
About:          "!T !V\n!D\n!c\n!U"


//-----------------------------------------
// Filter Control Definitions

ctl(0): "Red",            //The slider label
        Range=(-128,128), //Value range of the slider
        Pos=(220,20),     //Position of the slider on the
dialog
        Size=(*,7),       //Size of the slider
        Val=0             //Initial value of the slider

ctl(1): "Green", Range=(-128,128),
          Pos=(220,30), Size=(*,7), Val=0
ctl(2): "Blue", Range=(-128,128),
          Pos=(220,40), Size=(*,7), Val=0

// Hide the Edit button and the logo
ctl[CTL_EDIT]: None
ctl[CTL_LOGO]: None


//-----------------------------------------
// Here comes the filter code

ForEveryTile:
{
```

```
int c;

// loop through all rows
for (y = y_start; y < y_end; ++y) {

  // Update progress bar and cancel
  // if ESC key was pressed
  if ( updateProgress(y,Y) ) break;

  // loop through all columns
  for (x=x_start; x<x_end; ++x) {

    // loop through all channels
    for (z=0; z<Z; ++z){

      // Read a color value
      c = pget (x,y,z);

      // Process the color value.
      // In this case the value of
      // the appropriate slider is
      // added to the color value
      c = c + ctl(z);

      // Write color value
      pset ( x, y, z, c );

    } // for z

  } // for x

} // for y

// Stop processing and apply the effect
```

```
  return true;
}
```

```
%ffp

// This template gets you started
// when you plan to enhance one
// of your Filter Factory
// plug-ins with FilterMeister.
// For this purpose it is
// important to be put the FF
// source code into
// FilterMeister's ForEveryTile
// handler as it gives you a lot
// of new possibilities.


//-------------------------------
// Filter Infos

Category    : "FilterMeister"
Title       : "FF in FET"
Copyright   : "None"
Author      : "Harald Heim"
Organization: "The Plugin Site"
URL:"http://thepluginsite.com"
Filename    : "FFinFET.8bf"
Description: "Use your FF code in the ForEveryTile
handler"
Version     : "1.01"
About       : "!T !V\n!D\n"
              "!c\n!U"

// Only apply filter to RGB and Grayscale images
SupportedModes: RGBMode,GrayScaleMode
```

```
//--------------------------------
// Filter Control Definitions

// Determines how the filter dialog will look like
Dialog: color=#C0C0C0, NoTitlebar, Drag=Background, size=
(348,145) //,Pos=Client(CENTER)

ctl(13): groupbox, Color=#C0C0C0, Fontcolor=#000000, Pos=
(180,0), Size=(165,95)

// Insert you own slider label names and comment out the
slider that
// you don't need
ctl(0): "Slider 1", Val=28, Size=(*,6), Pos=(225,12),
fontcolor=black
ctl(1): "Slider 2", Val=201, Size=(*,6), Pos=(225,22),
Fontcolor=black
ctl(2): "Slider 3", Val=0, Size=(*,6), Pos=(225,32),
Fontcolor=black
ctl(3): "Slider 4", Val=76, Size=(*,6), Pos=(225,42),
Fontcolor=black
ctl(4): "Slider 5", Val=82, Size=(*,6), Pos=(225,52),
Fontcolor=black
ctl(5): "Slider 6", Val=149, Size=(*,6), Pos=(225,62),
Fontcolor=black
ctl(6): "Slider 7", Val=8, Size=(*,6), Range=(0,255), Pos=
(225,72), Fontcolor=black
ctl(7): "Slider 8", Val=0, Size=(*,6), Range=(0,255), Pos=
(225,82), Fontcolor=black

// Hide the Edit button and the logo
ctl[CTL_EDIT]: none
ctl[CTL_LOGO]: None



//--------------------------------
```

```
// Here comes the filter code

ForEveryTile:{

  int x,y,r,g,b,i,u,v,m,d,M,D;

  // Calculation of unsupported FF expressions
  // Comment out the not needed variables!
  M = c2m(X,Y)/2;
  D = c2d(X,Y)/2;

  // loop through all rows
  for (y=y_start; y<y_end; ++y) {

    // Update progress bar and
    // cancel if ESC key was
    // pressed
    if ( updateProgress(y, Y) ) break;

    // loop through all columns
    for (x=x_start; x<x_end; ++x) {

      // Calculation of unsupported FF expressions
      // Comment out the not needed variables!
      r = src(x,y,0);
      g = src(x,y,1);
      b = src(x,y,2);
      i = ((76 * r) + (150 * g) + (29 * b))/256;
      u = ((-19 * r) + (-37 * g) + (56 * b))/256;
      v = ((78 * r) + (-65 * g) + (-13 * b))/256;
      m = c2m(x-X/2,y-Y/2);
      d = c2d(x-X/2,y-Y/2);



      // Replace "255-r/g/b" with your own FF formulas
here
```

```
      /* If the FF code should include puts and gets,
      please replace the gets with the statement inside
the
      put command and remove the put command itself */

      pset(x,y,0, 255-r ); //red channel
      pset(x,y,1, 255-g ); //green channel
      pset(x,y,2, 255-b ); //blue channel
    }
  }

  // Apply calculations to image
  return true;

}//ForEveryTile
```

Demonstration Source Codes

Using the Color Dialog

RGB to Grayscale Conversion

Writing "DEMO" on an image

Using The Color Dialog

```
%ffp

OnFilterStart:{

  int color = 0;

  // Displays the color dialog with white as
  // the default color
  color = chooseColor(RGB (255,255,255),
  "Please choose a color:");

  // Display the RGB values of the chosen color
  Info ("The following color was chosen: "
    "RGB (%d, %d, %d)",
    Rval(color), Gval(color), Bval(color) );

  return false;
}
```

RGB To Grayscale Conversion

```
OnFilterStart:
{
  if (imageMode != GrayScaleMode && imageMode != RGBMode)
  {
    ErrorOk("This filter works only with 8-bit images in
RGB or Grayscale mode.");
    doAction(CA_CANCEL);
  }
  return false;
}


ForEveryTile:
{

  if (imageMode == RGBMode)
  // Put Grayscale version into channel 0
  // Grayscale conversion similar to Photoshops RGB Mode
to Grayscale Mode conversion
  // app. 30% Red, 59% Green, 11% Blue
  {
    for (y=y_start; y<y_end; y++)
      for (x=x_start; x<x_end; x++)
        pset( x, y, 0, ( src(x, y, 0) * 76 + src(x, y, 1)
* 150 + src(x, y, 2) * 29 ) / 255 );
}

// FILTER CODE that does its weird line art or grayscale
thing on channel 0...

if (imageMode == RGBMode)
// Copy result to channels 1 and 2
// because a Grayscale image in RGB mode has three
identical channels
{
  for (y=y_start; y<y_end; y++)
    for (x=x_start; x<x_end; x++)
    {
      pset(x, y, 1, pget(x, y, 0));
      pset(x, y, 2, pget(x, y, 0));
    }
  }

  return true;
}
```

Writing DEMO on an image

This code example shows how to render a small watermark with the word "DEMO" randomly across on an image when the user clicks the OK button. You can use this code for public demo versions of your Photoshop plug-ins, so the customer can try out your plug-in before buying it.

Example

```
%ffp

ForEveryTile:
{
  int i, j, offset, demo, xstart, step;

  if (ctl(CTL_OK)) {
    if (Y > X) step=(X)/8; else step=(Y)/8;

    // Avoid too many "DEMO" on small images
    if (step < 200) step=200;

    // Set maximum "DEMO" distance
    if (step > 400) step=400;

    for (z=0; z < Z; z++) {
      xstart = 0;
      for (y=step/2; y < Y; y+=step) {
        if (xstart > 0){
          xstart=0;
        } else {
          xstart=step/2;
        }
        for (x=xstart; x < X; x+=step) {
          // Set background for text
          for (j=-5; j < =10; j++)
          for (i=-5; i < =30; i++)
            // blend with image
            pset(x+i, y+j, z,
            blend( pget(x+i, y+j, z),
              255, z, 0, 70));

            //Set "DEMO" Text
            //D
            offset=0;
            pset(x+offset,y+0,z,0);
            pset(x+offset,y+1,z,0);
            pset(x+offset,y+2,z,0);
            pset(x+offset,y+3,z,0);
            pset(x+offset,y+4,z,0);
            offset+=1;
            pset(x+offset,y+0,z,0);
            pset(x+offset,y+4,z,0);
            offset+=1;
            pset(x+offset,y+0,z,0);
            pset(x+offset,y+4,z,0);
            offset+=1;
            pset(x+offset,y+1,z,0);
            pset(x+offset,y+2,z,0);
            pset(x+offset,y+3,z,0);

            //E
            offset=7;
            pset(x+offset,y+0,z,0);
            pset(x+offset,y+1,z,0);
            pset(x+offset,y+2,z,0);
            pset(x+offset,y+3,z,0);
            pset(x+offset,y+4,z,0);
            offset+=1;
            pset(x+offset,y+0,z,0);
```

```
        pset(x+offset,y+2,z,0);
        pset(x+offset,y+4,z,0);
        offset+=1;
        pset(x+offset,y+0,z,0);
        pset(x+offset,y+2,z,0);
        pset(x+offset,y+4,z,0);
        offset+=1;
        pset(x+offset,y+0,z,0);
        pset(x+offset,y+2,z,0);
        pset(x+offset,y+4,z,0);

        //M
        offset=14;
        pset(x+offset,y+0,z,0);
        pset(x+offset,y+1,z,0);
        pset(x+offset,y+2,z,0);
        pset(x+offset,y+3,z,0);
        pset(x+offset,y+4,z,0);
        offset+=1;
        pset(x+offset,y+1,z,0);
        offset=16;
        pset(x+offset,y+2,z,0);
        offset+=1;
        pset(x+offset,y+1,z,0);
        offset+=1;
        pset(x+offset,y+0,z,0);
        pset(x+offset,y+1,z,0);
        pset(x+offset,y+2,z,0);
        pset(x+offset,y+3,z,0);
        pset(x+offset,y+4,z,0);

        //O
        offset=21;
        pset(x+offset,y+1,z,0);
        pset(x+offset,y+2,z,0);
        pset(x+offset,y+3,z,0);
```

```
            offset+=1;
            pset(x+offset,y+0,z,0);
            pset(x+offset,y+4,z,0);
            offset+=1;
            pset(x+offset,y+0,z,0);
            pset(x+offset,y+4,z,0);
            offset+=1;
            pset(x+offset,y+1,z,0);
            pset(x+offset,y+2,z,0);
            pset(x+offset,y+3,z,0);
          }//x
        }//y
      }//z

  }//if OK Button

  return true;
}
```

VK codes

This is a list of all available VK codes in FilterMeister

```
VK_LBUTTON
VK_RBUTTON
VK_RBUTTON
VK_CANCEL
VK_MBUTTON
VK_BACK
VK_TAB
VK_CLEAR
VK_RETURN
VK_SHIFT
VK_CONTROL
VK_MENU
VK_PAUSE
VK_CAPITAL
VK_ESCAPE
VK_SPACE
VK_PRIOR
VK_NEXT
VK_END
VK_HOME
VK_LEFT
VK_UP
VK_RIGHT
VK_DOWN
VK_SELECT
VK_PRINT
VK_EXECUTE
VK_SNAPSHOT
VK_INSERT
VK_DELETE
```

```
VK_HELP
VK_0
VK_1
VK_2
VK_3
VK_4
VK_5
VK_6
VK_7
VK_8
VK_9
VK_A
VK_B
VK_C
VK_D
VK_E
VK_F
VK_G
VK_H
VK_I
VK_J
VK_K
VK_L
VK_M
VK_N
VK_O
VK_P
VK_Q
VK_R
VK_S
VK_T
VK_U
VK_V
VK_W
VK_X
VK_Y
```

```
VK_Z
VK_LWIN
VK_RWIN
VK_APPS
VK_NUMPAD0
VK_NUMPAD1
VK_NUMPAD2
VK_NUMPAD3
VK_NUMPAD4
VK_NUMPAD5
VK_NUMPAD6
VK_NUMPAD7
VK_NUMPAD8
VK_NUMPAD9
VK_MULTIPLY
VK_ADD
VK_SEPARATOR
VK_SUBTRACT
VK_DECIMAL
VK_DIVIDE
VK_F1
VK_F2
VK_F3
VK_F4
VK_F5
VK_F6
VK_F7
VK_F8
VK_F9
VK_F10
VK_F11
VK_F12
VK_F13
VK_F14
VK_F15
VK_F16
```

```
VK_F17
VK_F18
VK_F19
VK_F20
VK_F21
VK_F22
VK_F23
VK_F24
VK_NUMLOCK
VK_SCROLL
VK_LSHIFT
VK_RSHIFT
VK_LCONTROL
VK_RCONTROL
VK_LMENU
VK_RMENU
VK_PROCESSKEY
VK_ATTN
VK_CRSEL
VK_EXSEL
VK_EREOF
VK_PLAY
VK_ZOOM
VK_NONAME
VK_PA1
VK_OEM_CLEAR
```

See Also

getAsyncKeyState, getAsyncKeyStateF, FME__KEYDOWN, FME__KEYUP

Windows UI Color Constants

COLOR_3DDKSHADOW

COLOR_3DFACE

COLOR_3DHIGHLIGHT

COLOR_3DHILIGHT

COLOR_3DLIGHT

COLOR_3DSHADOW

COLOR_ACTIVEBORDER

COLOR_ACTIVECAPTION

COLOR_APPWORKSPACE

COLOR_BACKGROUND

COLOR_BTNFACE - Standard window/dialog background color.

COLOR_BTNHIGHLIGHT

COLOR_BTNHILIGHT

COLOR_BTNSHADOW

COLOR_BTNTEXT

COLOR_CAPTIONTEXT

COLOR_DESKTOP

COLOR_GRAYTEXT

COLOR_HIGHLIGHT

COLOR_HIGHLIGHTTEXT

COLOR_INACTIVEBORDER

COLOR_INACTIVECAPTION

COLOR_INACTIVECAPTIONTEXT

COLOR_INFOBK

COLOR_INFOTEXT

COLOR_MENU

COLOR_MENUTEXT

COLOR_SCROLLBAR

COLOR_WINDOW

COLOR_WINDOWFRAME

COLOR_WINDOWTEXT

Filter Specification Keys

(case insensitive)

About

Author

Category

Copyright

Description

Dialog

Embed

EnableInfo

Filename

FilterCaseInfo

Organization

SupportedModes

Title

URL

Version

formatString descriptors

(These are built-in 256-character string arrays.)

filterAuthorText

filterCase

filterCaseText

filterCategoryText

filterCopyrightText

filterDescriptionText

filterFilenameText

filterHostText

filterImageModeText

filterInstallDir

filterOrganizationText

filterTitleText

filterURLText

filterVersionText

Named modes for SupportedModes Specification

(case insensitive, may be abbreviated)

AllModes

BitmapMode

GrayScaleMode

IndexedColorMode

RGBMode

CMYKMode

HSLMode

HSBMode

MultichannelMode

DuotoneMode

LabMode

Gray16Mode

RGB48Mode

Lab48Mode

CMYK64Mode

DeepMultichannelMode

Duotone16Mode

Named modes for EnableInfo Specification

(case sensitive, may NOT be abbreviated)

BitmapMode

GrayScaleMode

IndexedColorMode

RGBMode

CMYKMode

HSLMode

HSBMode

MultichannelMode

DuotoneMode

LabMode

Gray16Mode

RGB48Mode

Lab48Mode

CMYK64Mode

DeepMultichannelMode

Duotone16Mode

Named variables for EnableInfo specification

(case sensitive)

PSHOP_ImageMode

PSHOP_ImageDepth

PSHOP_HasLayerMask

PSHOP_HasSelectionMask

PSHOP_HasTransparencyMask

PSHOP_NumTargetChannels

PSHOP_NumTrueChannels

PSHOP_IsTargetComposite

PSHOP_ImageWidth

PSHOP_ImageHeight

Functions specific to EnableInfo specification

(case sensitive)

dim

in

max

min

Named mode Constants for use in source code

(case sensitive; this is an upper-case version, CamelCase above is also supported)

BITMAPMODE

GRAYSCALEMODE

INDEXEDCOLORMODE

RGBMODE

CMYKMODE

HSLMODE

HSMODE

MULTICHANNELMODE

DUOTONEMODE

LABMODE

GRAY16MODE

RGB48MODE

LAB48MODE

CMYK64MODE

DEEPMULTICHANNELMODE

DUOTONE16MODE

Filter Case Handling Specifications

(case insensitive)

FlatImageNoSelection

FlatImageWithSelection

FloatingSelection

EditableTransparencyNoSelection

EditableTransparencyWithSelection

ProtectedTransparencyNoSelection

ProtectedTransparencyWithSelection

inCantFilter

inStraightData

inBlackMat

inGrayMat

inWhiteMat

inDefringe

inBlackZap

inGrayZap

inWhiteZap

inBackgroundZap

inForegroundZap

outCantFilter

outStraightData

outBlackMat

outGrayMat

outWhiteMat

outFillMask

copySourceToDestination

doNotCopySourceToDestination

doesNotWorkWithBlankData

worksWithBlankData

doesNotFilterLayerMasks

filtersLayerMasks

doesNotWriteOutsideSelection

writesOutsideSelection

Plug-in Compatible Hosts

These list compatibility of the Plug-ins generated from FilterMeister with graphics hosts.

Fully compatible

Vendor	Application	Version
ACD Systems	Canvas	X
ACD Systems	Photo Canvas	2
ACD Systems	Photo Editor	3
Adobe	After Effects	4.1, 5, 5.5
Adobe	Illustrator	7
Adobe	ImageReady	2
Adobe	PhotoDeLuxe	2, 3
Adobe	PhotoShop	3, 4, 5, 5.5, 6, 7, CS, CS2, CS3
Adobe	PhotoShop Elements	1, 2, 3, 4, 5, 6
Alexander Sabov	PicMaster	1.25
Antonio Da Cruz	PhotoFiltre Studio	7.0.x
Arcadia	PhotoPerfect	2.90
Aurora Borealis	Mandala Painter	3
CADLink	SignLab	5 and higher
CDH Productions	Image Explorer Pro	4
CiEBV Computerinsel	PhotoLine 32	5, 6, 7, 8, 9, 10, 11

Corel	Bryce	4
Corel	Draw	9
Corel	Painter	IX
Corel	PhotoPaint	9, 10, 11, 12
Corel	Paint Shop Pro	X, XI, X2
Deneba	Canvas	6, 7, 8, 9
Equilibrium	DeBabelizer Pro	5
Irfan Skiljan	IrfanView	3.85, 4
JASC	Paint Shop Pro	4.12, 5, 6, 7, 8, 9
KnowledgeAdventure	HyperStudio	4.2
Macromedia	Freehand	7, 8
Macromedia	Fireworks	8
Mediachance	PhotoBrush	1
MeeSoft	Image Analyzer	1 (with the 8bf Interface plug-in)
Megalux	Ultimate FX	
Megalux	Ultimate Paint	2
MetaCreations/Corel	Painter	6
Microfrontier	Digital Darkroom	1.2
MicroSoft	PhotoDraw 2000	1
New World	Focus PhotoEditor	4
Newave	Chaos Fx: Twilight'76	1.2
Pierre-emmanuel Gougelet	XnView	1.70
PluginMaster	PluginMaster	1
PS	ImageN	1.0z
Right Hemisphere	Deep Paint	

Ron Scott	QFX (LE)	7.0
Satori	PhotoXL	2.29
Serif	PhotoPlus	6, 7, 8, 9, 10
SigmaPi	NiGulp	1.5
SigmaPi	Pixopedia 24	1.0.5
Stoik	ImageMan Pro	5
The Plugin Site	Plugin Commander Pro	1.5, 1.6
ThinkTank	Ameri-Imager	2
ULead	GIF Animator	4
ULead	PhotoImpact	4, 4.2, 5, 8, 10, 11
WebSuperGoo	Achroma	
Xara	Xara	X
Zoner	Photo Studio	10

Partially compatible

Vendor	**Application**	**Version**	**Limitations**
ArcSoft	PhotoStudio 2000	5.5	Preview incorrect
Corel	PhotoPaint	6, 7, 8	Preview and result are inverted and rotated
Discreet	Combustion	2	Blue cast in preview
GIMP	GIMP	1.2.4	Preview incorrect
Macromedia	Fireworks	2, 3, MX	Transparency not correct in preview
Magix	Xtreme Photo Designer	6	Applying to a selection or object does not work; Random crashes
Micrografx	Picture	8, 9	Preview zoom doesn't work

	Publisher		
MicroSoft	Image Composer	1.5	Cannot drag preview
MicroSoft	Picture It! Digital Image Pro	7	100% zoom may not work, Cancel sometimes produces crash
VicMan	VCW VicMan Photo Editor	6.9	Preview drag causes crash

Untested

Vendor	Application	Version
Ability	PhotoPaint Studio	any
Adobe	Illustrator	8
Adobe	Image Ready	1
Adobe	LiveMotion	any
Adobe	Pagemaker	6 and higher
BananaSoft	TwistedPixel	any
Corel	PhotoHouse	2 and higher
Corel	Xara	2
DigisoftDirect	ImagePro	2K1
Equilibrium	DeBabelizer Pro	4.5
Fractal Design	Detailer	any
Macromedia	Director	5, 6 and higher
MetaCreations	Painter	5.5, 7
MetaCreations/Corel	Art Dabbler	2.1
MGI	PhotoSuite	4 and higher
Micrografx	Picture Publisher	4ak, 5, 6a, 7
Microsoft	PhotoDraw 2000	2
Newtek	Aura	any

Newtek	Inspire 3D	any
Newtek	Lightwave	5.6 and higher
PM	Imagic	any
Presto!	ImageFolio	all
Serif	PhotoPlus	5
Ulead	PhotoImpact	2, 3.01 and higher

Incompatible

Vendor	**Application**	**Version**	**Limitations**
Adobe	After Effects	6.5	
Adobe	ImageStyler		Doesn't support PhotoShop plug-ins
Adobe	PhotoDeLuxe	1	
Adobe	Premiere	4.2, 5, 5.1, 5.5	Renders only black to image
ArcSoft	PhotoStudio 2000	4.1	Doesn't support PhotoShop plug-ins
AutoDesk	Combustion	2008	Crashes and quits on trying to invoke plug-in
Datatech	ImageMan		Crashes when dragging preview and when applying final render
Discreet	3D Studio Max	4.2	Background not displayed, color dialog doesn't work, renders distorted red/green pattern
Macromedia	xRes	2	Crashes on loading plug-in
MetaCreations	Painter	5	Does not apply final render

MicroSoft	Image Composer	1.0	Doesn't recognize plug-ins
SPG	ColorWorks: Web	4	Does not apply final render

FilterMeister Compatible Hosts

These list compatibility of the FilterMeister plug-in itself with graphics hosts.

Fully compatible

Vendor	Application	Version
Adobe	PhotoShop	3, 4, 5, 5.5, 6, 7, CS, CS2
JASC	Paint Shop Pro	7, 8, 9

Untested

Vendor	Application	Version
Ability	PhotoPaint Studio	any
ACD Systems	Canvas	9, X
Adobe	After Effects	4.1, 6.5
Adobe	Illustrator	7, 8
Adobe	Image Ready	1, 2
Adobe	ImageStyler	
Adobe	LiveMotion	any
Adobe	Pagemaker	6 and higher
Adobe	PhotoDeLuxe	1, 2, 3
Adobe	PhotoShop Elements	1, 2, 3
Adobe	Premiere	4.2, 5, 5.1, 5.5
Alexander Sabov	PicMaster	1.25
Antonio Da Cruz	PhotoFiltre Studio	7.0.x
ArcSoft	PhotoStudio 2000	4.1, 5.5
BananaSoft	TwistedPixel	any

CADLink	SignLab	5 and higher
CDH Productions	Image Explorer Pro	4
CiEBV Computerinsel	PhotoLine 32	5, 6
Corel	Bryce	4
Corel	Draw	9
Corel	Painter	IX
Corel	PhotoHouse	2 and higher
Corel	PhotoPaint	6, 7, 8, 9, 12
Corel	Xara	2
Datatech	ImageMan	
Deneba	Canvas	6
DigisoftDirect	ImagePro	2K1
Discreet	3D Studio Max	4.2
Discreet	Combustion	2
Equilibrium	DeBabelizer Pro	4.5
Equilibrium	DeBabelizer Pro	5
Fractal Design	Detailer	any
GIMP	GIMP	1.2.4
Harald Heim	Plugin Commander	1.5
Irfan Skiljan	IrfanView	3.85
JASC/Corel	Paint Shop Pro	4.12, 5, 6
KnowledgeAdventure	HyperStudio	4.2
Macromedia	Director	5, 6 and higher
Macromedia	Fireworks	2, 3, 8
Macromedia	Freehand	7, 8
Macromedia	xRes	2
Mediachance	PhotoBrush	
MegaLux	Ultimate FX	
MegaLux	Ultimate Paint	2.1

MetaCreations/Corel	Art Dabbler	2.1
MetaCreations/Corel	Painter	5, 5.5, 6, 7
MGI	PhotoSuite	4 and higher
Microfrontier	Digital Darkroom	1.2
Micrografx	Picture Publisher	4ak, 5, 6a, 7, 8, 9
MicroSoft	Image Composer	1.0, 1.5
MicroSoft	PhotoDraw 2000	1, 2
MicroSoft	Picture It! Digital Image Pro	7
New World	Focus PhotoEditor	4
Newave	Chaos Fx: Twilight'76	1.2
Newtek	Aura	any
Newtek	Inspire 3D	any
Newtek	Lightwave	5.6 and higher
Pierre-emmanuel Gougelet	XnView	1.70
PM	Imagic	any
Presto!	ImageFolio	all
PS	ImageN	1.0z
Right Hemisphere	Deep Paint	
Ron Scott	QFX (LE)	7.0
Satori	PhotoXL	2.29
Serif	PhotoPlus	5, 6, 7, 9
SigmaPi	NiGulp	1.5
SigmaPi	Pixopedia 24	1.0.5
SPG	ColorWorks: Web	4
Stoik	ImageMan Pro	5
ThinkTank	Ameri-Imager	2
ULead	GIF Animator	4

ULead	PhotoImpact	2, 3.01, 4, 4.2, 5, 8, 10, 11
VicMan	VCW VicMan Photo Editor	6.9
WebSuperGoo	Achroma	
Xara	X	

Incompatible

Autodesk	Combustion	2008	Crashes and quits on trying to invoke plug-in

FilterMeister Programming Style Guide

Spelling

Correct spelling is as much to be admired in source code comments and software documentation as in any highly literary work. A spell-checker can be very helpful when checking text documents, but is often less useful for vetting comments buried within cryptic source code lines (without proper tools, of course). So please take extra time and care to review your source code for correct spelling and grammar. You might even turn up a few program bugs along the way -- it has happened to me!

The following spelling rules and suggestions are based on American, not British, usage. Since many of our European members are more familiar with British English, we won't harp on "color" versus "colour" here and there. It's more important simply to keep the usage consistent within a single document. I've used the **[Merriam-Webster]** (an abridged work) as my primary source of reference to resolve spelling issues for the principal reason that it tends to agree with me more often than does, say, the American Heritage Dictionary.

Most of the examples below were gathered from actual occurrences in FilterMeister documentation, postings, and code listings. If you have your own personal "pet peeves", please add them here! -Alex

- **a lot**, not alot. But distinguish from the verb **allot**.
- **abbreviate**, not abreviate.
- **acknowledgment** or **acknowledgement**, but **acknowledgment** is preferred in American English. Ditto **judgment** versus **judgement**.

- **all right**, not alright. But **already** and **all ready** are okay when correctly used.
- **backward** or **backwards**. A slight preference for **backward** as an adverb, but always **backward** as an adjective. Ditto for **forward**, **downward**, **upward**, **westward**, and **toward**.
- **beveler**, **beveled** and **beveling**. Also British **beveller**, **bevelled** and **bevelling**.
- **Boolean**, not boolean. Named after George Boole, "Boolean" should be capitalized per **[Merriam-Webster]**. However, the FM type keyword `bool` is lowercase. Go figure.
- **category**, not catagory.
- **compatible** and **compatibility**, not compatable or compatability.
- **definite**, not definate.
- **different from**, **different than**, **different to**. This is a matter of usage, not spelling. According to **[Alt-Usage-English]**, "different to" is used chiefly in British speech, while "different than" is chiefly American usage. The most widely accepted usage is "different from", which we hereby recommend.
- **disastrous**, not disasterous.
- **discriminate**, not descriminate.
- **duplicate**, not dublicate.
- **e.g.** and **i.e.**, not eg or ie. These are abbreviations of *exempli gratia* and *id est*, resp., and require periods.
- **Filter Factory**, not FilterFactory. Two words, per Adobe.
- **FilterMeister**, not Filtermeister or Filter Meister, and certainly not FilterMiester! "FilterMeister™" is (or will become) a trademark, and should be spelled exactly this way (one word, camel-case). The single word form "FilterMeister" also works better in search engines (fewer spurious hits) than the two word phrase "Filter Meister".
- **formatted**, not formated.
- **gauge**, not guage. **gage** is also allowed, but not preferred.

- **gray** or **grey**. American English slightly prefers **gray** and Microsoft uses this spelling in its technical documentation and API names, so let's prefer **gray** in this context.
- **height**, not heighth or heigth.
- **its** versus **it's**, **your** versus **you're**, etc. These and many other *possessive/contraction* homonym pairs are easily mistyped without careful proofreading.
- **kudos**, not **kudo**. Per [Merriam-Webster], **kudos** is a singular word in Greek.
- **label**, not lable. Also **labeler**, **labeled** and **labeling** (or British **labelled** and **labelling**).
- **license** or **licence**. American English prefers the former.
- **millennium**, not millenium.
- **misspell**, not mispell. Something oddly recursive about this one.
- **NULL** versus **NUL**. "NULL" is a zero-valued pointer to nothing. "NUL" is a zero-valued character constant ('\0').
- **occasion**, not ocassion.
- **occurred**, not occured. Per [Merriam-Webster]. Also: **occurring**, **occurrence**, but **occurs**.
- **Paint Shop Pro**, not PaintShop Pro or Paintshop pro. "Paint Shop Pro®" is a registered trademark of Corel®.
- **Photoshop**, not PhotoShop or Photo Shop. "Photoshop®" is trademarked by Adobe.
- **plug-in**, not plugin. Per [Merriam-Webster] for both adjective and noun usages. But note: **plug in** when used as a verb.
- **precede**, not preceed. But **succeed** and **supersede**.
- **preferred**, not prefered. Per [Merriam-Webster]. Also: **preferring**, but **preference** and **prefers**.
- **principle** and **principal**. [Merriam-Webster] says: Although nearly every handbook and many dictionaries warn against confusing principle and principal, many people still do. Principle is only a noun; principal is both adjective and noun.
- **privilege**, not priviledge or privelege.

- **propagate**, not propogate (*my personal nemesis -Alex*). Ditto **propaganda**, not propoganda. But note **propose** and **proponent**.
- **publicly** or **publically**. [Merriam-Webster] permits either, but prefers **publicly**.
- **recommend**, not reccommend.
- **resource**, not ressource.
- **separate**, not seperate.
- **their**, **there**, **they're**. Another set of homonyms which are easily confused when typing rapidly.
- **threshold**, not threshhold or treshold. Who would have guessed?
- **tileable**, not tilable. Not in most dictionaries, but Google lists 18,500 hits on "tilable" versus 107,000+ hits on "tileable". Also, if you search Google for "tilable" it asks "Did you mean: tileable". 'Nuff said.
- **usage**, not useage. Per [Merriam-Webster]. However, either **usable** or **useable** is acceptable per [Merriam-Webster]. Enough to tear one's hair out.
- **weird**, not wierd. An egregious exception to the rule: [i] before [e] except after ?.

Information

Describe how the information stuff should be formatted.

Controls

Describe how the control definitions should be formatted.

Code

Expressions

- Keep one expression on one line; each line should contain only one `;` symbol. The *only* exception being the `for` statement.

Parentheses

- Use spaces on the inside of parentheses; this is practically required to do decent Boolean expression formatting and should thus be used everywhere for consistency. (*I disagree. -Alex.*)

Brackets

- Only use brackets when necessary, not for single lines of code. Commonly it is advised to always use brackets but most of the time this is ridiculous and only causes bloated source files. (*I disagree. -Alex.*, *Strongly disagree. -Martijn.*)
- Place brackets on their own lines always, never combined with anything else, this makes for easy recognition.

```
ForEveryTile():
{
    if( ctl( 0 ) )
        Info( "Control 0 is set" );
    else
    {
        Info( "Control 0 is not set" );
        return false;
    }
}
```

Boolean expressions

- Split logical expressions over multiple lines whenever they concern independent variables.

- Let the code layout represent the logical structure of the expression.
- Never compare Booleans to true or false explicitly; this makes code more intuitive to read.
- Use parentheses whenever and only when mixing comparable operators such as Boolean operators, comparison operators or mathematical operators.

```
// Bad
if(((x>10) && (x<2+4*5)) || ((y==0) && do_first_row))
    // ...

// Good
if( ( x > 10 && x < 2 + ( 4 * 5 ) )
 || ( y == 0
   && do_first_row ) )
    // ...
```

Indenting

- Use a tab size of 4 characters; any smaller will clash with logical expressions, any more is useless.

Ternary operator

- Use a ternary operator whenever an if..else would normally be used which only assigns value to a single variable.
- Split the ternary operator over two lines. (*I disagree. -Alex.*)
- Use direct assignment of Boolean values whenever possible.

```
// Bad
if( a > 10 )
    b = true;
else
    b = false;
```

```

// Better
b = a > 10? true
          : false;

// Best (because we assign Boolean values)
b = a > 10;
```

Variable names

There are several options available and decisions to be made:

- Capitals
- Underscores
- Abbreviation
- Structuring
- Type

When combined, there are a few common standards in the C/C++ world:

- Hungarian/Microsoft notation. This is the one we encounter in the Windows API. It uses Capitals for each word and prepends the first letter of each structure level before the word describing the variable.
  - width of the size of a circle: csWidth.
  - Uses no underscores, has no notion of type, practically never abbreviates but is highly structured.
  - Quite unreadable and likely to be quite ambiguous, structuring has practically no use in FM.
- OpenGL-like notation. This is basically caps for all words and only one underscore before the type identifier.
  - width of the size of a circle: CircleSizeWidth_f (assuming floating point)
  - Long names, using type can be quite redundant and long names :)

    - Forces you to know the type which is a good thing.
- C++ style notation. Using no caps anywhere, underscores to separate words.
    - width of the size of a circle: circle_size_width
    - Long names but at least it's easy and pleasant to read.
    - Has my vote - Martijn

Then there's the leading underscore thing. Some consider these to be reserved for the compiler, others (including me) like them to distinguish between normal variables and those passed in as arguments. FM has no user-defined functions yet so this issue is of no hurry.

Efficiency

- Use prefix increment and decrement whenever possible instead of postfix; postfix causes a temporary variable to be created which is practically never needed. (*Not necessarily true, depends on how well the compiler optimizes. -Alex.*)
- Place the most discriminating (failing) Boolean expressions first; any consecutive expressions will only be tested if it evaluates to true.

```
// Bad
if ( x > 0 && y == 1 )
    // ...

// Good
if ( y == 1 && x > 0 )
```

Syntax of the FM Language

FilterMeister actually supports several plug-in filter design languages, including

- Adobe's original Filter Factory (FF) language,
- Mario Klingemann's Filter Factory Wizard (FFW),
- the extended Filter Factory Plus (FFP) language, and
- the all-encompassing FilterMeister Language (FML)

As such, the syntax of the total union of all languages is rather complex, but may be more readily understood by examining the syntax of each component language individually.

Conventions

Case Sensitivity

Extended BNF Description

Terminal Symbols

Examples of terminal symbols (e.g., a keyword or a reserved word):

- `Dialog`
- `ForEveryTile`
- `int`, `for`

Non-terminal Symbols

Examples of non-terminal symbols:

- <FM_program></FM_program>

- <statement></statement>
- <constant_expression></constant_expression>
- <decimal_number></decimal_number>

Top Level Symbol

The top-level non-terminal symbol, from which all programs recognized by FilterMeister are derived, is:

- <FM_program> </FM_program>

Click the link for <FM_program> to begin delving into the depths of the various FM Language syntaxes, or click one of the following links for the syntax of a specific FM sublanguage : </FM_program>

- <AFS_program> </AFS_program>
- <FFW_program> </FFW_program>
- <FFP_program> </FFP_program>
- <FML_program> </FML_program>

See Also

Command Reference

Alphabetic Index

enumRegValue
Error
ErrorOk
Events
exp
expand

F

fabs
false
fc2d
fc2m
fCallLib
fCallLibFmc
fclose
fcloseall
feof
ferror
FF in FET Template
ffillArray
fflush
fgColor
fgetArray
fgetc
fgetpos
fgets
fillArray
Filter Specifications
Filter Templates
filterCase
filterInstallDir
FilterMeister compatible hosts
filterUniqueID
findClose
findFirstFile

ftell
fwrite

G

gamma
get
getAppTheme
getArray
getArrayAddress
getArrayDim
getArrayString
getAsyncKeyState
getAsyncKeyStateF
getc
getCpuReg
getCtlClass
getCtlColor
getCtlCoord
getCtlDivisor
getCtlHandle
getCtlItemCount
getCtlItemText
getCtlPos
getCtlRange
getCtlTab
getCtlText
getCtlVal
getcwd
getDialogHandle
getDialogHeight
getDialogPos
getDialogWidth
getDisplaySettings
getImageTitle
getLibFn