Release Notes

FM Lite 0.4.1 to Beta 0.4.18

Release Notes for FM Lite 0.4.1

Here's a quick set of release notes for FM Lite beta version 0.4.1.Please note that this list is neither comprehensive nor exhaustive.

0. Caveats
0.1 Release 0.4.1 has been tested only with Photoshop 4.0.1, Paint Shop Pro 4.12, and Paint Shop Pro 5.01 (evaluation version). Use other hosts at your own risk!

0.2 Paint Shop Pro 4.x and 5.x apparently do not preserve a plug-in filter's state across invocations. This means that the settings of controls, the contents of the source edit box, etc., will not be preserved across invocations, which can be a great annoyance. In later versions, FM may use alternate methods to preserve state across invocations, at least for ill-behaved hosts such as PSP.

0.3 This release contains great amounts of debugging code, etc., so it is much bulkier (and somewhat slower) than the retail release
will be.

0.4 This release contains many partially-implemented and/or undocumented features. Use these features at your own risk! Please try to confine your evaluation to *documented* features only (e.g., to features presented in the Tutorial Lessons associated with this release).

0.5 FM runs best at a screen size of 800x600 or higher. To use the Advanced Edit mode, you may need to run your display at 1024x768 or higher, particularly if you are using Large Screen fonts.

1. Changes
The following changes have been made since the previous release:

1.1 The default for the "Use MMX" checkbox has been set to unchecked instead of checked (see 4.1 below). If you set "Use MMX" when using FM with PSP as a host, you will get random GP errors.

1.2 Unused scrollbar, checkbox, and radio button controls in the default filter dialog box will be automatically hidden.

2. New Features
Following are new features in this release:

2.1 The FM dialog box now changes its size, revealing or hiding various controls as appropriate, depending on the editing mode. The modes are: Normal Mode (shows size of final filter that will be created as a stand-alone); Edit Mode (displays source edit box and associated controls for loading, editing, compiling, and saving the filter source code); and Advanced Edit Mode (same as Edit Mode, but changes the Compile button to a series of buttons for the individual compilation steps, and displays the Filter Parameter Block and Compiler Output Log as well as many advanced option check boxes). Advanced Mode is intended for debugging FM - use it at your own risk!

2.2 In Edit Mode, the Compile button performs a "minimal make". The parser is invoked only if the previous parse failed; then the code generator is invoked only if the source was reparsed successfully; then the dialog initialization code is invoked only if code was regenerated successfully. Note that the Compile button does NOT update the proxy preview.

2.3 Load and Save buttons have been implemented to load and save source code in the source edit box. It is no longer necessary to use Cut and Paste (though that still works).

2.4 Loading a *.afs Filter Factory Settings file with the Load button will automatically import the *.afs file as a skeleton FFP program. (This automatic conversion does NOT occur when you Paste thecontents of a *.afs file into the source edit box -- in that case, FM will still parse the text as a (plain) Filter Factory source file.)

2.5 FM now supports if/then/else-statements, for-loops, while-loops, and do-loops as part of the FFP language.

2.6 FM now has many new built-in variables, including several sets of general-purpose user variables (int i0,...i9; int j0,...j9; int k0,...k9; double x0,...x9; double y0,...y9; double z0,...z9;).

2.7 FM now allows a list of channel names when specifying a channel formula. For example, R,G,B:<formula> is equivalent to:

R:<formula>
G:<formula>
B:<formula>

3. Unimplemented Features
The following features are not implemented (or only partially implemented) in this release:

3.1 The following buttons are not yet functional: Find, Help, and Make. In particular, note that this release cannot be used to "Make" a stand-alone filter.

3.2 Filter programs can only access the 10 predefined scrollbar controls. The demo checkbox and radio controls are not yet functional, and will disappear as soon as you compile a source program.

3.3 The Save button is really a "Save As..." button, and will always prompt before overwriting a file, even if it is the current source file. If demand warrants, I will add a separate "Save" button that updates the current source file without prompting.

3.4 JPEG images cannot be used as the dialog background in this release. Only Windows bitmap (*.bmp) files are supported in this release.

3.5 The following FFP language features are not implemented in this release:

- User declaration of variables
- typedef statements
- switch statements
- return statements
- arrays

3.6 The following FFP handlers (and their associated keywords) are not yet implemented:

- onInitialInvocation
- onInitDialog
- onFilterInit
- forEveryTile
- forEveryRow
- forEveryPixel
- onGpError
- onFloatError

3.7 The 'map[n]' dialog key is not yet recognized.

3.8 FM still does not recognize many of the "undocumented" FF variables
such as I, U, V, cmax, etc.

4. Known Problems
Following are some of the known problems in this release:

4.1 Because of a quirk in the way MMX support is implemented, FM may cause random GP errors when run under some hosts. MMX works okay with Photoshop 4.0.1, but causes random crashes in PSP 4.x and 5.x. MMX support will be fixed in a future release to avoid any such GP errors.

4.2 FM does not warn if you exit without saving changes to your source code, nor if you load new source code without first saving any changes to your previous source code.

4.3 The floating-point stack is not emptied across FFP function calls; this can cause floating-point stack overflow errors in some cases.

4.4 The dialog background is not updated properly when switching between edit modes if the background is a gradient or image.

4.5 The FFP region functions interpret coordinates in device pixels rather than in Dialog Box Units (dbu's). This means they will not scale properly across changes to the system Screen Font size.

4.6 User-defined positioning and sizing of controls does not work properly.

4.7 The dialog box title is not saved across filter invocations.

4.8 The Title keyword in an FFW program does not set the dialog box title.

4.9 Updating the Proxy Preview does not always force a recompile if the source code has changed, so you may get a "stale" preview if you do not manually perform a compile.

4.10 The tab order for controls in the FM dialog has not yet been set. If you try to tab from one control to another, you may get strange results.

4.11 The map(n,i) run-time function does not behave the same as FF when the value of the "hi" slider is less than the value of the "lo" slider.

Release Notes for FM Lite 0.4.2

1. Changes
The following changes have been made since the 0.4.1 release:

1.1 The ForEveryPixel and ForEveryTile handlers are now implemented.

1.2 FM now recognizes all "undocumented" (Premiere-compatibility) FF built-in variables (e.g., p, t, total, rmax) and functions (e.g., src0(), src1(), rad0(), rad1(), cnv0(), cnv1()), except rst().

1.3 A problem that prevented FD-generated message boxes from being displayed during final filter application has been corrected.

1.4 Floating-point literals and filter info strings are now correctly preserved across filter invocations.

1.5 pset() now ignores attempts to store pixels outside the current tile.

1.6 A problem that caused a null statement to generate a "bug 2" message has been fixed.

1.7 In Advanced Edit Mode, changing the setting of Use MMX, Use CMOV, or Use inlines now forces a recompile.

2. New Features
Following are new features in this release:

2.1 Two new built-in functions testAbort() and updateProgress() have been implemented.

Release Notes for FM Lite 0.4.3

1. Changes
1.1 Floating-point sin() and cos() now work correctly.

1.2 The Title keyword in an FFW program now correctly sets the dialog box title.

2. New Features
2.1 FM can now directly import a Filter Factory *.8bf file, converting it to FF+ source code in the process.

2.2 FM can now import extended AFS files as produced by PiCo.

Release Notes for FM Lite 0.4.4

1. Changes

1.1 The code generated for the FM pre- and post-increment and decrement operators (++ and --) is now more efficient.

1.2 A 'Track' subkey has been added to the 'Ctl' key, and the default thumb-tracking behavior of a slider control has changed. By default, the proxy preview will NOT be updated while the thumb of a slider control is being moved; the proxy will be updated only when the thumb is released. This default behavior is more acceptable for time-consuming filter operations. If your filter algorithm is fast, you can specify the 'Track' key in the 'Ctl' control definition, which will cause FM to try to update the proxy preview continuously as the thumb is moved. The 'NoTrack' subkey forces the reverse (non-tracking) behavior.

The default thumb-tracking behavior may be temporarily reversed by holding down the SHIFT key while dragging the thumb.

The thumb-tracking behavior for control n may also be changed programmatically by calling setCtlProperties(n, CTP_TRACK) or clearCtlProperties(n, CTP_TRACK) from your filter code or event handler code.

1.3 When you click the '+' (zoom in) or '-' (zoom out) buttons under the proxy preview, FM will now pause for 0.5 seconds before updating the preview. This allows you to click several times in quick succession to change the zoom factor by several increments before the proxy is updated.

Also, holding down SHIFT or CTRL while clicking one of the zoom buttons has special meaning: SHIFT '+' and CTRL '+' cause an immediate zoom to 100%. CTRL '-' causes an immediate zoom to the smallest scale factor (6%). SHIFT '-' causes an immediate zoom to the "best fit" scale factor (i.e., to the largest scale factor that still allows the entire proxy image to be viewed within the preview window).

1.4 FM now has a predefined "tile buffer" called 'tbuf', which is allocated large enough to hold all active planes of each tile as it is processed. To insert a pixel value into the tile buffer, call the tset() function:

```
tset(x, y, z, val);
```

To retrieve a pixel value from tbuf, call the tget() function:

```
val = tget(x, y, z);
```

Note that tset() is analogous to pset(), except pset() sets a pixel in the output (destination) tile; and tget() is analogous to src(), except src() fetches a pixel from the input (source) image.

Tbuf is useful when writing two-pass algorithms within a single ForEveryTile pass: First process pixels from the source image, writing the results to tbuf; then process pixels from tbuf, writing the final result to the output image.

2. Bug Fixes

2.1 A potential "wild" store in FM, that could cause unpredictable behavior, has been eliminated.

3. New Features

3.1 User declaration and initialization of scalar variables of type 'short', 'int', 'long', 'float', and 'double' is now allowed within blocks (i.e., enclosed in braces). For example:

```
{
   int i, j, radius;
   double x, y, z;
   double pi = 3.14;
   int scaledRadius = ctl(0)/scaleFactor;
   double sigma = log(255.0)/4.0;

   <code ...>
}
```

Note, however, that the 'short' and 'long' types are currently implemented as type 'int' (i.e., as a 4-byte signed integer), and that type 'float' is currently implemented the same as type 'double' (i.e., as an 8-byte floating point

number). In future releases, 'short' will be implemented as a 2-byte integer, and 'float' will be implemented as a 4-byte floating point number.

N.B. Array, pointer, structure, union, and enumeration declarations are still not implemented in this release.

Implementation Restriction: You are currently limited to a total of approximately 100 user-defined variables in a filter program. This restriction will be eliminated in future releases.

3.2 The following functions and predefined named constants from the standard C run-time library are now implemented in FM:

```
CLOCKS_PER_SEC
RAND_MAX
NULL
EXIT_SUCCESS
EXIT_FAILURE
_MAX_PATH
_MAX_DRIVE
_MAX_DIR
_MAX_FNAME
_MAX_EXT
EDOM
ERANGE
long clock( void );
long time( long *timer );
dword strlen( const char *string );
char *strcpy( char *strDestination, const char *strSource );
char *strcat( char *strDestination, const char *strSource );
int strcmp( const char *string1, const char *string2 );
char *strncpy( char *strDest, const char *strSource, dword count );
char *strncat( char *strDest, const char *strSource, dword count );
int strncmp( const char *string1, const char *string2, dword count );
double fabs( double x );
double ceil( double x );
double floor( double x );
double fmod( double x, double y );
double exp( double x );
double log( double x );
double log10( double x );
double ldexp( double x, int exp );
double pow( double x, double y );
double sqrt( double x );
double fcos( double x );    //temporary name for cos
double fsin( double x );    //temporary name for sin
double ftan( double x );    //temporary name for tan
double acos( double x );
double asin( double x );
double atan( double x );
double atan2( double y, double x );
double cosh( double x );
double sinh( double x );
double tanh( double x );
double hypot( double x, double y ); //non-ANSI
int rand( void );
void srand( unsigned int seed );
```

4. Known Problems

4.1 There is a slow memory leak in the user-defined symbol table handler which may cause Photoshop to eventually run out of RAM after many FM compilations. If this happens, exit and restart Photoshop. This leak will be fixed in the next release.

Release Notes for FM Lite 0.4.5

1. Changes

1.1 The tan(x) function now generates correct in-line code for real arguments. The valid domain for tan(real) is [-2^63, +2^63]. If you need to compute tan(x) for values outside this domain, use the ftan(x) function, which calls the C run-time library version of tan().

1.2 Run-time exception handling/reporting has been enhanced to include additional information about the environment at the time the exception occurred, including optional disassembly of the X86 machine code at the exception site.

1.3 By default, src(x,y,z) now edge-replicates the alpha channel for values of x and y lying outside the bounds of the input image or selection. (Previously, src() edge-replicated all channels *except* the alpha channel; src() returned 0 for the alpha channel for coordinates outside the image/selection.)

1.4 FM now treats a non-breaking space character (0xA0) as white space. This allows cut-and-paste of FM code examples from Internet Explorer to work correctly.

1.5 When re-invoking FM, the FM dialog box is now made visible *before* re-running the filter algorithm. (Previously, if the filter algorithm was very slow, the user would be presented with a long hourglass wait before the dialog box appeared, which could be confusing and disconcerting.)

2. Bug Fixes

2.1 An obscure bug that could sometimes cause GPF's within R:/G:/B:/A: channel code has been fixed.

2.2 A bug in the updateProgress() processing within the default ForEveryRow handler, which could result in an incorrect progress bar display when the preview image was clipped to fit the proxy window, has been fixed.

2.3 A bug that sometimes caused an incorrect "while looking for <token>" display in the FM syntax error message has been fixed. (Note, however, that there are still some other problems with the syntax error message processing that can also result in incorrect tokens being displayed.)

3. New Features

3.1 The testAbort() function now checks for the ESC key during proxy preview processing as well as during final image filtering. If the user presses ESC during proxy filter processing, testAbort() will return a non-zero value. (Note that the updateProgress() and doForEveryRow() functions also implicitly call testAbort() and will return non-zero if the user presses ESC.)

3.2 The testAbort() function now checks for the PAUSE key during proxy preview processing. If the user presses PAUSE during proxy filter processing, the filter will pause and a message box will be displayed to ask whether the user wishes to continue. Choosing YES causes filter processing to resume. Choosing NO causes testAbort() or updateProgress() or doForEveryRow() to return a non-zero value, which allows the filter code to gracefully terminate the filter.

3.3 A new function abort() has been added, which causes filter processing to stop immediately. The most common use of abort() is to abort filter processing when testAbort() or updateProgress() or doForEveryRow() returns a non-zero value:

```
if (testAbort()) {
   // User requested abort...
   <do any necessary cleanup>
   abort();
}
```

3.4 A new function updatePreview(n) 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. ('n' is the ID of the proxy preview control to be updated. At present, FM supports only one proxy preview, so 'n' is ignored.)

3.5 A new function sleep(msec) causes filter processing to pause for approximately 'msec' milliseconds before resuming. This is useful for controlling the rate of preview display updates when animating iterative algorithms.

3.6 To facilitate coding of iterative algorithms, a second tile buffer (t2buf) has been added. 't2buf' is similar to 'tbuf' and is read/written by the new functions t2get(x,y,z) and t2set(x,y,z,val), which are analogous to the tget/tset functions used to access tbuf.

3.7 Again to facilitate coding of iterative algorithms, a new function pget(x,y,z) has been added as a companion to the pset(x,y,z,val) function. pget(x,y,z) returns the value of the pixel in the output image tile at coordinates (x,y) for channel z. By default, pget() edge-replicates pixel values for coordinates lying outside the bounds of the output image tile. Note that pget/pset is analogous to tget/tset and t2get/t2set.

3.8 To facilitate animation of "classic" Filter Factory *.8bf modules, a new function doForEveryRow() has been added. This function can be called in a ForEveryTile handler to directly drive the ForEveryRow handling (and thence the ForEveryPixel handling). Since the Filter Factory-style R:/G:/B:/A: handlers are driven directly from the default ForEveryPixel handler, this means that you can cause execution of the R:/G:/B:/A: handlers across the entire current tile by each call to doForEveryRow() from within your ForEveryTile handler. Thus, to animate a classic FF filter, set the relevant control values within your ForEveryTile handler; then call doForEveryRow() to run the classic filter; then call updatePreview() to display the results in the proxy preview window. Finally, re-adjust the control values and repeat to perform the next step in the animation. doForEveryRow() will return a non-zero value if the user presses ESC (or PAUSE followed by NO); checking this value allows the ForEveryTile handler to perform any necessary cleanup action before calling abort() to terminate the filter processing.

3.9 To aid in producing high-performance filters (such as blurs) based on separable convolution kernels, two built-in functions for computing 1-D convolutions have been added:

```
val = cnvX(k, off, d, pGetf, x, y, z);
val = cnvY(k, off, d, pGetf, x, y, z);
```

where:

'k' is the "radius" of the kernel and must be >= 0.
'off' is 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' is the denominator by which the convolution sum will be divided, and must be non-zero.
'pGetf' identifies the FM built-in function that will be called to fetch values from an image buffer at the designated coordinates, and must be 'src', 'tget', 't2get', or 'pget'.
'x' is the x-coordinate at which the (center of the) kernel is to be applied.
'y' is the y-coordinate at which the (center of the) kernel is to be applied.
'z' is the channel number to which the kernel is to be applied.
'val' is the integer result obtained by summing the products of the n kernel coefficients with n image pixels in the X or Y direction, and dividing the sum by 'd'.

cnvX() computes the 1-D convolution kernel in the X direction, while cnvY() computes the 1-D convolution kernel in the Y direction. I.e., cnvX() convolves pixels at coordinates:

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

while cnvY() applies the kernel to pixels at coordinates:

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

Release Notes for FM Lite 0.4.6

1. Changes

1.1 The anonymous put/get cells are now initialized to 0 at the start of every filter run. Previously, they retained any values left over from a previous run. This may affect the operation of some filters that depended on such uninitialized values.

1.2 In Advanced Edit Mode, the symbol table dump is now displayed only if the 'Dump SDL' checkbox is checked during the Codegen phase.

2. Bug Fixes

2.1 rnd(a,b) can now return any value in the range a to b, inclusive. Previously, it always returned a value in the range a to b-1. Also, rnd(a,b) now returns a value in the range [b,a] if a > b.

3. New Features

3.1 The Filter Factory built-in function rst(n) is now implemented in FM. Calling rst(n) sets the pseudo-random number generator to a new state which is a function of both the value of 'n' 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(n) with some unpredictable value of 'n'; 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.

3.2 When importing a *.8bf file, FM now honors Michael Johannhanwahr's "Protect" flag in the PARM resource, and will not import the filter if this flag is set. (Plugin Manager 2.0 uses this flag to prevent casual users from viewing the filter source code.)

Release Notes for FM Lite 0.4.7

1. New Features

1.1 The following user control classes are now implemented:

STANDARD	This is the default ("standard") scroll bar style that we all know and love from previous versions of FM.
SCROLLBAR	A bare Windows scroll bar, without the two automatic buddy controls associated with the STANDARD control class.
TRACKBAR	A standard Windows95 track bar control -- a welcome alternative to the time-worn scroll bar control.
PUSHBUTTON	A standard Windows push button control.
CHECKBOX	A standard Windows check box control.
RADIOBUTTON	A standard Windows radio button control
GROUPBOX	A standard Windows group box control. Together with a set of RADIOBUTTON controls, this control can be used to construct a grouping of mutually-exclusive radio buttons.
OWNERDRAW	A standard Windows owner-drawn control, in which the FD is responsible for drawing the appearance. Can be used as a simple push button.
STATICTEXT	A static (read-only) text control. Usually used for displaying fixed text strings, but can also be used as a simple push button
FRAME	A standard Windows frame control, consisting of a styled ectangular frame. Can be used as a simple push button
RECT	A standard Windows rectangle control, consisting of a filled black, gray, or white rectangle. Can be used as a simple push button.

To specify the class of a user control, specify one of the above keywords as the *first* subkey for the Ctl key. For example, to create a check box control:

```
Ctl(3):CHECKBOX,"&amp;amp;Demo",size=(34,14),pos=(214,126)
```

1.2 A new built-in function createCtl() has been added to allow the dynamic creation of controls within the filter code:

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

Dynamically creates a control with index n, class c, text t, coordinates (x,y), width w, height h, style s, extended style sx, properties p, and enable level e. All measurements are in DBUs. For x, y, w, and h, a value of -1 means use the default value.

The class c should be one of the following predefined values:

CC_STANDARD, CC_SCROLLBAR, CC_TRACKBAR, CC_PUSHBUTTON, CC_CHECKBOX,

CC_RADIOBUTTON, CC_GROUPBOX, CC_OWNERDRAW, CC_STATICTEXT, CC_FRAME, or CC_RECT,

1.3 Tool tips are now provided for the OK, Cancel, Edit, Zoom-in, and Zoom-out buttons, as well as the progress bar, the host application static text field, and the preview window. Automatic tracking tool tips are also available for the TRACKBAR control class when the 'TopTip' attribute is specified. A full implementation of FD-specifiable tool tips for user controls will be available in a later release.

1.4 Most of the standard C run-time i/o library routines (stdio) are now exposed in FM The following functions are now built-ins in FM:

clearerr
fclose
fcloseall
feof
ferror
fflush
fgetc
fgets
flushall
fopen
fprintf
fputc
fputs
fread
freopen
fscanf
fseek
ftell
fwrite
getc
putc
remove
rename
rewind
sprintf
sscanf
tmpfile
tmpnam
ungetc

The following stdio functions are not yet implemented. Use ftell() and fseek() instead:

fgetpos
fsetpos

The following stdio functions are not implemented because they implicitly reference one of the standard i/o units (stdin, stdout, or stderr), which are not implemented in FM. Use the corresponding file-oriented function instead.

getchar
gets
perror
printf
putchar
puts
scanf

For documentation of these functions, see any standard C run-time library reference.

1.5 Two new built-in functions have been added: RGB(r,g,b) and RGBA(r,g,b,a). RGB() packs three 8-bit RGB color values (r, g, b) into a single 32-bit integer. RGBA() packs three 8-bit RGB color values and an 8-bit alpha channel value into a 32-bit integer. RGB() is useful for creating RGB color values for the 2nd argument of the setCtlColor() and setCtlFontColor() built-ins. For example, setCtlFontColor(12, RGB(255,255,0)) will set the text color for control 12 to yellow.

1.6 A new built-in intrinsic function COLOR(<color>) has been added. COLOR() takes as an argument the same syntax that the Color=<color> subkey takes. For example, <color> can be a simple color name (yellow, red, salmon), aqualified X11, HTML, JAVA, or Netscape color name (X11.AliceBlue, HTML.Aquamarine, JAVA.darkGray, NS.NetscapeGray), a Windows95 UI color name (COLOR_BTNFACE, COLOR_DESKTOP), an RGB triple (RGB(55,255,70)), an HTML-style hexadecimal color specifier in RGB order (#cc00ff), or a 24-bit hexadecimal constant in BGR order (0xff00cc).

As an example, setCtlFontColor(12, COLOR(yellow)) will set the text color for control 12 to yellow.

1.7 Two new built-in functions have been added: setCtlTextv() and setDialogTextv().
These are similar to setCtlText() and setDialogText() but take a C-style i/o format string and a variable list of arguments to perform more sophisticated formatting of output strings. The calling sequences are:

setCtlTextv(n, fmt, arg1, ...)

and

setDialogTextv(fmt, arg1, ...)

where

n is the index of a control,

fmt is a C-style format specifier string,
arg1, ... are optional arguments to be interpolated into the format string.

2. Changes

2.1 FM now asks the FD whether he/she wants to recompile the filter source code whenever the FD tries to run the filter and the compiled code is out-of-date (because either the source has changed or compilation options have changed).

2.2 User-defined dialog attributes (such as the background image and color, the title bar style, etc.) are now reset to their default values after a compilation. This fixes a problem in which some of these attributes were inadvertently "sticky" across compilations.

2.3 Dialog attributes such as the background image and color, the title bar style, the drag mode, etc., are now saved and restored across invocations of FM.

2.4 Coordinates for the Region= subkey of the Dialog key are now specified in Dialog Box Units (DBUs) instead of pixels. This means that the dialog region will now scale automatically across changes to the Windows System Font size. This change also applies to the relevant arguments of the built-in functions createRectRgn(), createRoundRectRgn(), createCircularRgn(), createEllipticRgn(), and createPolyRgn().

2.5 The Color= and FontColor= subkeys of the Ctl key are now functional. Color=<color> sets the background or body color of the control to the specified color (or causes the body/background of the control to be transparent if Color=Transparent is specified). FontColor=<color> sets the color of text for the control. Note that the interpretation and validity of these subkeys varies among control classes. For some classes, the background color is meaningless or unmodifiable. For other classes (such as the PUSHBUTTON class), the FontColor subkey has no effect. The corresponding built-in functions setCtlColor(n, color) and setCtlFontColor(n, color) are now also functional.

2.6 The text buddy control for the default (STANDARD) scroll bar control has been changed from a read-only text field to an editable text field. This means that the user can now enter a numeric value directly in the text field to set the value of the control.

2.7 The built-in function chk(n) is now just a synonym for ctl(n), and the use of chk(n) is deprecated.

2.8 You can use an asterisk (*) instead of the special value -1 in a Size= or Pos= subkey to specify a component which is not to be changed. For example, Size=(40,*) can be used to change the width of a control to 40 DBUs without changing its height.

2.9 Tab stops have been removed from all controls in the expanded Edit and Advanced Edit areas of the FM dialog. These controls can still be accessed from the keyboard in most cases by using their shortcut keys.

3. Bug Fixes

3.1 FM now forces the dialog background to be redrawn whenever the dialog size is changed. This fixes a problem in which the background could be left in a stale state when, for example, the user entered or exited Edit Mode.

3.2 Occasionally FM could cause a GP if the user pressed a button or initiated a zoom-in or zoom-out while simultaneously trying to drag the preview image with the zero-drag option in effect. This bug has been fixed.

3.3 A problem in which the filter algorithm code could sometimes be recursively reentered has been fixed.

3.4 Some ambiguities in parsing FFW-style vs. FFP-style keys (such as the common Ctl key) have been resolved.

4. Known Problems

4.1 For TRACKBAR controls, the Color= subkey has problems setting the background color of the control. The color will not be updated until the first time a user clicks on the track bar control.

4.2 The new dialog size is not always computed correctly when switching between edit modes or when changing title bar attributes.

4.3 Vertically-oriented scroll bars and track bars don't work yet.

4.4 For the STATICTEXT control class, the attributes 'EndEllipsis', 'PathEllipsis', and 'WordEllipsis' do not work.

4.5 Some features of Win95 controls require comctl32.dll version 4.71 or later. (E.g., setting colors for a progress bar requires 4.71 or later). Comctl32.dll is not a redistributable DLL, but can be obtained by installing Microsoft Internet Explorer 4.0 on your system. Earlier versions of comctl32.dll, such as version 4.70, will also work, but you may lose some functionality.

4.6 When the user presses ENTER in the text edit buddy of a STANDARD control, the control will "ding" even if the value entered is perfectly valid.

Release Notes for FM Lite 0.4.8

1. New Features
1.1 The Action= subkey for the Ctl key is now implemented.

Action=a sets the default action for the control to a, where a is 'NONE', 'CANCEL', 'APPLY', 'PREVIEW', or 'EDIT'. The default action is the action that will be taken by the default handler for this control when the control is activated. The default actions are:

NONE	No action. This is the default action for radio buttons and group boxes.
CANCEL	FM exits, leaving the original source image unaltered. This is the default action for the Cancel button,
APPLY	The filter is applied to the original source image, and FM exits. This is the default action for the OK button.
PREVIEW	The filter is applied to the proxy image, and all previews are updated. This is the default action for most user controls, including the STANDARD, SCROLLBAR, TRACKBAR, PUSHBUTTON, and CHECKBOX controls.
EDIT	FM enters or leaves Edit Mode. This is the default action for the Edit control.

The default action for a control can be set programmatically by callingsetCtlAction(n, a), where 'n' is the control index and 'a' is one of the following predefined named constants: CA_NONE, CA_CANCEL, CA_APPLY, CA_PREVIEW, or CA_EDIT.

1.2 The following named constants are now built-ins in FM:

CA_NONE, CA_CANCEL, CA_APPLY, CA_PREVIEW, CA_EDIT,
CC_UNUSED, CC_STANDARD, CC_SCROLLBAR, CC_TRACKBAR,
CC_SPINNER, CC_UPDOWN, CC_PUSHBUTTON, CC_CHECKBOX,
CC_RADIOBUTTON, CC_GROUPBOX, CC_OWNERDRAW, CC_LISTBOX,
CC_COMBOBOX, CC_SLIDER, CC_PROGRESSBAR, CC_EDIT,
CC_STATICTEXT, CC_FRAME, CC_RECT, CC_BITMAP, CC_ICON,
CC_TOOLTIP, CC_ANIMATION, CC_PREVIEW.
CTP_TRACK, CTP_READONLY, CTP_HORZ, CTP_VERT,
CTP_ORIENT_MASK, CTP_TOP, CTP_BOTTOM, CTP_LEFT,
CTP_RIGHT, CTP_SIDE_MASK.

2. Changes
2.1 The default dialog background color is now X11.CadetBlue (#5F9EA0) instead of Black.

2.2 A group box control can now be assigned any arbitrary value, not just 0 or 1.

2.3 String literals and floating point constants are now stored in the same (fixed size) literal pool. If this literal pool overflows during compilation, you will get the message: "Too many floating-point constants or string literals."

3. Bug Fixes
3.1 A couple of small memory leaks have been fixed.

Release Notes for FM Lite 0.4.9

1. New Features

1.1 COMBOBOX and LISTBOX user controls are now implemented.

1.1.1 The syntax to specify a COMBOBOX control with the 'Ctl' keyword is:

Ctl[n]: COMBOBOX (<combobox_style>, ...), <other_control_subkeys>...

where the possible <combobox_style>'s are:

DROPDOWNLIST	Specifies a combo box consisting of a static text control with a drop-down list box which is normally closed up. This is the default style for a COMBOBOX control.
DROPDOWN	Specifies a combo box consisting of an edit control with a drop-down list box which is normally closed up. (Not yet fully supported.)
SIMPLE	Specifies a combo box consisting of an edit control atop a permanently dropped-down list box. (Not yet fully supported.)
SORT	Specifies that the strings in the list box are to be alphanumerically sorted
EXTENDEDUI	Specifies that the combo box will use an alternate keyboard user interface, in which the down arrow drops the list box down and moves the selection, and Enter selects the current list item. With the default keyboard interface, F4 opens and closes the drop- down list, and the cursor navigation keys immediately select a list item.
AUTOHSCROLL	Specifies that the edit control will automatically scroll horizontally to accommodate more characters than can fit in the width of the edit control window. Applies only to SIMPLE and DROPDOWN combo boxes.
HSCROLL	Specifies that the list box will have a horizontal scrollbar if necessary.
VSCROLL	Specifies that the list box will have a vertical scrollbar if necessary.
DISABLENOSCROLL	Specifies that scrollbars in the list box are to be grayed out rather than removed when the number of items in the list box is less than needed to require scrolling. By default, a vertical or horizontal scrollbar is removed entirely if not required to scroll the entire list box contents into view.
INTEGRALHEIGHT	Specifies that the height of the list box will be adjusted to accommodate an integral number of list items. This is the default
NOINTEGRALHEIGHT	Specifies that the height of the list box will be exactly as specified by the 'Size' subkey, and a partial item may be displayed at the bottom of the list box,
UPPERCASE	Specifies that all strings in the list box are to be converted to upper case.
LOWERCASE	Specifies that all strings in the list box are to be converted to lower case.
OEMCONVERT	Specifies that the string in the edit control is to be converted to the OEM character set and then back to the Windows character set to force the string to contain only characters that are legitimate in the OEM character set. This is useful for restricting the character set of file names and path names. Applies only to SIMPLE and DROPDOWN combo boxes.
OWNERDRAWFIXED	Specifies that the list items are of fixed height, and will be drawn by the FD. (Not yet supported.)
OWNERDRAWVARIABLE	Specifies that the list items are of variable height, and will be drawn by the FD.

(Not yet supported.)

HASSTRINGS	Specifies that the list items in an owner-drawn list box consist of strings which are to be stored by the combo box control. (Not yet supported.)
BORDER CLIENTEDGE STATICEDGE MODALFRAME	Places various styles of borders around the combo box control.
TABSTOP	Sets a tab stop on this control.
GROUP	Marks this control as the first (or one past the last) in a group of controls.
ACCEPTFILES	Specifies that the control will accept a set of dragged- and-dropped file names. (Not yet supported.)
TRANSPARENT RIGHTALIGNEDTEXT RTLREADING LTRREADING	(Not yet supported.)

1.2 The syntax to specify a LISTBOX control with the 'Ctl' keyword is:

Ctl[n]: LISTBOX (<listbox_style>, ...), <other_control_subkeys>...

where the possible <listbox_style>'s are:

SORT	Specifies that the strings in the list box are to be alphanumerically sorted
MULTIPLESEL	Specifies that multiple items may be selected simultaneously in the list box. Items are selected or deselected by clicking or double-clicking. (Not fully supported.) By default, only one item may be selected at a time.
EXTENDEDSEL	Specifies that multiple items may be selected simultaneously in the list box. Individual items are selected or deselected by CTRL-clicking on them. A range of items may be selected by SHIFT-clicking or using the SHIFT-navigation keys. (Not fully supported.) By default, only one item may be selected at a time.
NOSEL	Specifies that items in the list box cannot be selected. Clicking an item will result in an OnCtl() handler event, but the item will not be selected. (Not fully supported.)
MULTICOLUMN	Specifies that the items in the list box may be arranged in multiple columns, which can be scrolled horizontally.
HSCROLL	Specifies that the list box will have a horizontal scrollbar if necessary.
VSCROLL	Specifies that the list box will have a vertical scrollbar if necessary.
DISABLENOSCROLL	Specifies that scrollbars in the list box are to be grayed out rather than removed when the number of items in the list box is less than needed to require scrolling. By default, a vertical or horizontal scrollbar is removed entirely if not required to scroll the entire list box contents into view.
USETABSTOPS	Specifies that tabs in string items will be expanded. By default, tab stops are set every 32 DBUs.
INTEGRALHEIGHT	Specifies that the height of the list box will be adjusted to accommodate an integral number of list items. This is the default.
NOINTEGRALHEIGHT	Specifies that the height of the list box will be exactly as specified by the 'Size' subkey, and a partial item may be displayed at the bottom of the list box,
OWNERDRAWFIXED	Specifies that the list items are of fixed height, and will be drawn by the FD. (Not yet

supported.)

OWNERDRAWVARIABLE	Specifies that the list items are of variable height, and will be drawn by the FD. (Not yet supported.)
HASSTRINGS	Specifies that the list items in an owner-drawn list box consist of strings which are to be stored by the combo box control. (Not yet supported.)
BORDER CLIENTEDGE STATICEDGE MODALFRAME	Places various styles of borders around the list box control. By default, the list box has no border; the BORDER style is recommended
TABSTOP	Sets a tab stop on this control.
GROUP	Marks this control as the first (or one past the last) in a group of controls.
ACCEPTFILES	Specifies that the control will accept a set of dragged-and-dropped file names. (Not yet supported.)
TRANSPARENT RIGHTALIGNEDTEXT RTLREADING LTRREADING	(Not yet supported.)

1.1.3 To populate a list box or combo box control with an initial set of strings, specify the ['text' =] "<string_list>" subkey, where <string_list> is the desired set of strings, separated by newline ('\n') characters. For example, to populate a list box with the names of the following five cities:

Atlanta
San Francisco
Berlin
Mexico City
Paris

use:

text="Atlanta\nSan Francisco\nBerlin\nMexico City\nParis"

as the (initial) text string for the list box control.

Note: The total length of the text string is currently limited to 1023 characters, including one character for each newline ('\n') item separator. For example, this accommodates a list of approximately102 items where each item has an average width of 9 characters (plus one character for the '\n' separator). If the length of the text string exceeds 1023, it will be truncated to 1023 characters without warning.

1.1.4 The value of a COMBOBOX or LISTBOX control (as returned by the ctl(n) or getCtlVal(n) function) is the zero-based index of the currently selected item, or -1 if no items are currently selected.

The value of a LISTBOX control with the MULTIPLESEL or EXTENDEDSEL style is not defined (i.e., these styles are not yet supported).

The value of a COMBOBOX control with the SIMPLE or DROPDOWN style is undefined if the user types a value in the edit control window (i.e., these styles are not yet fully supported).

Note: If the COMBOBOX or LISTBOX control has the SORT style, then the value of the control is the zero-based index of the item in the sorted list, which is not necessarily the same as the index of the item in the text string that was used to initialize the list.

1.1.5 The current selection in a COMBOBOX or LISTBOX control may be set programmatically by calling setCtlVal(n, index), where 'index' is the zero-based index of theitem to be selected, or -1 to deselect all items in the list.

For a LISTBOX with the MULTIPLESEL or EXTENDEDSEL style, you can select only one item at a time by calling setCtlVal(). It is not currently possible to make multiple selections programmatically (i.e., these styles are not yet fully supported).

Note: If the COMBOBOX or LISTBOX control has the SORT style, then the index of the item to be selected is the zero-based index of the item in the sorted list, which is not necessarily the same as the index of the item in the text string that was used to initialize the list.

1.1.6 The default action for a COMBOBOX or LISTBOX control is action=NONE.
If you want to run the filter code immediately when an item is selected, specify action=PREVIEW in the Ctl(n) key definition.

2. Changes

2.1 The evaluation of the sqr() integer square root function has been sped up, and is now approximately twice as fast as the Filter Factory's implementation. Many thanks to Michael Johannhanwahr, who pointed me to a "fast" integer algorithm. It turns out that, at least for a 200 MHz Pentium, this fast integer algorithm is actually a bit slower than using the Pentium FPU FSQRT instruction (including the conversion from integer to floating-point and back). However, further research turned up an algorithm by Ben Discoe (rodent@netcom.COM) at Microsoft, which basically unrolls the loop in the integer algorithm above to achieve a speed significantly faster than the Pentium FPU, even when called as a closed routine instead of generating in-line code. (Of course, these timing results apply only to a 200 MHz Pentium -- your mileage may vary -- but I believe they are a useful indication across most of the P5 and P6 line.)

Note: The c2m(x,y) function also benefits from the speed-up in the integer square root algorithm, but is still significantly slower than the Filter Factory implementation. I finally broke down and disassembled parts of FF, and discovered that FF precomputes tables for both c2m(x,y) and c2d(x,y). The trick is that the tables are computed based on the ratio of x to y, where x <= y. The c2d() function, of course, depends only on this ratio; and the function c2m(x,y) may be computed as: c2m(x,y) = c2m(x/y,1) * y, where x <= y.

I will probably implement this optimization in one of the next few FM releases.

2.2 The sqr() function is now polymorphic. When called with an integer argument, it computes the integer square root (by calling the integer square root routine described in 2.1 above). When called with a floating-point argument, it returns the floating-point square root (by generating an in-line FSQRT instruction).

2.3 The default dialog background image mode is now 'TILED' instead of 'EXACT'. In most cases, the FD should use either 'TILED' or 'STRETCHED' as the dialog background image mode. The 'EXACT' mode can be problematic, and should be used only in special circumstances.

2.4 FM now recognizes the following additional manifest constants:

DIM_EXACT	Used in a call to setDialogImageMode to specify 'EXACT' mode.
DIM_TILED	Used in a call to setDialogImageMode to specify 'TILED' mode.
DIM_STRETCHED	Used in a call to setDialogImageMode to specify 'TILED' mode.
CTP_EXTENDEDUI	Used in a call to set/clearCtlProperties to specified the extended keyboard user interface for a combo box control.
BLACKONWHITE WHITEONBLACK COLORONCOLOR HALFTONE MAXSTRETCHBLTMODE STRETCH_ANDSCANS STRETCH_ORSCANS STRETCH_DELETESCANS STRETCH_HALFTONE	Used as the second argument to setDialogImageMode to specify the stretch mode when the first argument is DIM_STRETCHED.

3. Bug Fixes

3.1 A race condition that could occur after pressing the 'Compile' button has been ameliorated by introducing a 10 millisecond delay between the time the user interface is drawn and the filter code is first run. The symptom of the race condition is that the user interface might be only partially drawn before the filter code begins to run, in which

case the user interface would not finish drawing until after the filter code was executed. This problem may still occur, but should be much rarer. If it becomes a problem again, we can increase the delay or seek another solution.

3.2 In Advanced Edit Mode, toggling the 'Use inlines' checkbox did not force a complete recompile, just a new code generation pass. However, the 'Use inlines' flag is processed in the parse phase, not the code generation phase, so the new setting of 'Use inlines' was not properly recognized. This bug has now been fixed: toggling 'Use inlines' will now force a complete recompile.

3.3 A problem which could cause a GPF within Photoshop's MMXCORE or FASTCORE module has been fixed, However, there may still be some situations in which the GPF can occur, mainly while previewing large (~30MB) images with the Advanced Mode 'Proxy big gulp' option in effect. To aid problem analysis, three temporary error messages have been added to FM. If you get a FilterMeister error message box with any of the following messages, please record the details and send them to me:

a) "Big gulp failed. AdvanceState returned n" where n is a decimal error code. (This is not actually an error, just a warning; execution may still proceed correctly. Turning off the 'Proxy big gulp' option should eliminate this message, but preview dragging will be slower.)

b) "Not enough memory to generate a preview image at this zoom level." In this case, execution cannot proceed; try using a smaller proxy preview zoom level, or increase available memory.

c) "Unknown error (n) while trying to generate a preview image." where n is a decimal error code. A mystery case; please send me the details, including the value of the error code.

4. Known Problems

4.1 The 'bool' data type is not yet fully supported. This can lead to unwarranted error messages when using the Boolean constants "true" and "false", such as:

FM-E-BADREF: Attempt to dereference through a non-pointer or selector

or:

FM-F-BADMIX: Illegal mixing of operand types.

As a temporary workaround, the compiler now treats "true" and "false" as integer constants instead of Boolean constants.

Release Notes for FM Lite 0.4.10

1. New Features

1.1 The "Make..." button, which creates a standalone filter, now works on Windows NT 4.0 (but not on Windows 95). Note, however, that the Make button is still non-functional (by intent) in the "crippled" Beta test version.

1.2 Three new built-in functions have been added to FM:

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

stripEllipsis(s) Returns string s with any trailing ellipsis removed.

formatString(s) Returns string s with the following substitutions mader the designated 2-character substrings (all of which begin with "!"):

substring	is replaced by
!!	"!" (i.e., a verbatim exclamation point)
!A	text specified by the Author key
!a	text specified by the About key
!C	text specified by the Category key
!c	text specified by the Copyright key
!D	text specified by the Description key
!F	text specified by the Filename key
!f	the current Filter Case (e.g., "Flat image, no selection")
!H	the name of the Host application (e.g., "Adobe Photoshop")
!M	the current Image Mode (e.g., "RGB Color")
!T	text specified by the Title key
!t	text specified by the Title key, with any trailing ellipsis removed
!V	text specified by the Version key

Caution: Other substrings beginning with "!" are reserved for future use.

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(), Info(), Warn(), Error(), ErrorOk(), YesNo(), or YesNoCancel() built-in functions,
- when displaying the About text in the About dialog box.

For example, the default Dialog text is "!t (!f) [!M]", which formatString() translates to something like "Filter Title (filter case) [Image Mode]" at run time.

1.3 An "About" key has been added to allow the Filter Designer to specify the design and content of the About box that is displayed when selected from Photoshop's Help/About Plug-in... menu item. The syntax for the About key specification is similar to the syntax for the Dialog key, and allows the FD to specify the size, background, and text of the About box. However, in this release all subkeys are ignored except the Text subkey, so only the Text subkey is documented here. The background of the About box is set to a default bitmap image in this release, and the About text is horizontally centered within a static text box which has room for approximately 5 lines of 40 characters each. The About text will be formatted by the formatString() function described above before being displayed.

Syntax

About: [Text =] "quoted string" [other subkeys, ...]

The default About text is "!t Plug-in !V\n!C\n!c\n!A", which translates to something like:

Filter Title Plug-in 1.0
Category
Copyright Notice
Author's Name

Note that clicking the FM default logo also displays the About box.

1.4 An "Embed" key has been added to allow the Filter Designer to specify a set of resources that are to be embedded in the standalone filter generated by the Make button. Embeddable resources include bitmap files, metafiles, and wave files.

Syntax

Embed: <resource> = "filename" [, ...]

where:

<resource> specifies the type of resource to be embedded, and may be "Bitmap", "Metafile", or "Wave" (only "Bitmap" is implemented in this release).

For example, if your filter uses "crazy glue.bmp" as a backgroundimage for the dialog, your Dialog specification might look something like this:

Dialog: ... image="crazy glue.bmp" ...

If you "Make" this into a standalone filter, the standalone filter will search for the external bitmap file "crazy glue.bmp" in order to draw the background. If you move the standalone filter to another location without also copying the "crazy glue.bmp" file to the same directory, then the standalone filter might not find the "crazy glue.bmp" file, in which case it will revert to using the specified (or default) background color or gradient instead of the bitmap. Even worse, the standalone filter might find a *different* file named "crazy glue.bmp", with unexpected results.

To avoid such problems, use the Embed key to tell FM to embed any such external files as resources in the standalone filter. Then you need distribute only the single .8bf file, which will be completely self-contained.

In the above example, you would add an Embed specification following the Dialog specification, similar to:
Dialog: ... image="crazy glue.bmp" ...
Embed: bitmap="crazy glue.bmp"

In a multi-filter example, where the background bitmap is changed for each individual filter by calls to setDialogImage(), you would specify a complete list of all the bitmaps you want to embed:

Dialog: image="patternwav.bmp",stretched

Embed: bitmap="patternwav.bmp",
bitmap="pattern2.bmp",
bitmap="zigzag.bmp"

1.5 A new FF+ statement, _eval_FFP{}, has been added to simplify the creation of a multi-filter from a collection of Filter Factory source codes.

Syntax

<FFP_statement> ::=
"_eval_FFP" "{" <FFP_eval_specification>* "}"

<FFP_eval_specification> ::=
<category_specification> |
<title_specification> |
<copyright_specification> |
<author_specification> |
<filename_specification> |
<description_specification> |

<version_specification> |
<dialog_specification> |
<control_specification> |
<control_val_specification> |
<FFP_channel_list> ":" <FFP_formula>

In other words, within the braces you can include any or all of the following FF+ key specifications, and these will be evaluated at filter run time rather than at compile time.:

```
_eval_FFP {
   Category: ...
   Title: ...
   Copyright: ...
   Author: ...
   Filename: ...
   Description: ...
   Version: ...
   Dialog: ...
   Ctl[n]: ...
   Val[n]: ...
   R: ...
   G: ...
   B: ...
   A: ...
}
```

For example, the statement:

```
_eval_FFP {
   Dialog: "My Filter", Color=blue, NoTitlebar, Drag=background,  Pos=Screen(Center)
}
```

will be translated into the appropriate set of calls to the FM built-in functions setDialogText(), setDialogColor(), clearDialogStyle(),
setDialogDragMode(), and setDialogPos(), which may then be executed at run time as desired.

1.6 FM can now display Windows Metafiles (with or without an Aldus placeable header) and Win32 Enhanced Metafiles on the surface of the dialog box.

2. Changes

2.1 The FilterMeister Lite/Beta plug-in now appears in the host application's plug-in filter menu under the category "FilterMeister", not "AFH".

2.2 The default Category text is now "FilterMeister" rather than the null string. The default Title text is now "My Filter" rather than the null string.

2.3 The default Dialog text is now "!t (!f) [!M]" (see 1.2 above).

2.4 Ellipses are automatically added or stripped from the filter Title text as necessary for display in various contexts.

2.5 'switch' statements are now implemented.

2.6 'return' statements are now implemented (but only for integral/Boolean return types).

2.7 Advanced Edit Mode is deleted from the Beta test version.

2.8 The "Find..." button has been disabled for now.

2.9 The "Help..." button has been renamed "About...".

2.10 FM should now recognize the host signatures for Corel Xara and Adobe Illustrator 7.

2.11 Elementary casts of the form (short), (int), (long), (bool), (float), and (double) are now implemented. (More complex casts are still not allowed.)

2.12 The FD is now prompted to save his/her modified source code when necessary.

2.13 The source image is now always copied to the destination image before each filter run.

2.14 FM-generated filters can now be applied to empty layers in the Editable Transparency cases.

2.15 If FM can't load the file specified for the dialog background bitmap, you are no longer given a choice of retrying or ignoring the error. Instead, an error message is displayed and the bitmap is ignored. The background is painted using the specified or default dialog color or gradient, and no further attempt is made to load the bitmap file. (In a standalone filter, the action is the same, but no error message is displayed.)

2.16 When importing a .8bf Filter Factory filter module, FM no longer tries to load the secondary dynamic link libraries required by that module. In particular, it is not necessary to have plugin.dll installed on your system when importing a 48KB Filter Factory module.

3. Bug Fixes
3.1 In a STANDARD slider control, the numedit buddy control wasn't initialized properly when the initial control range did not include 0. This has been fixed in 0.4.10.

3.2 Specifying the MaxBox subkey in a Dialog specification allowed a user to show the Advanced Edit Mode controls in the Beta version. To prevent this, the Beta version now ignores the MaxBox subkey.

4. Known Problems
4.1 Non-constant specifications such as "range=(0,X)" do not work as expected in standalone filters.

4.2 ToolTips don't work on Windows NT.

4.3 When importing a .8bf Filter Factory filter module, special characters such as '\' are not properly escaped in the filename contained in the generated Description specification.

4.4 Filters with no user-adjustable controls should not display a filter dialog box.

4.5 The 'return' statement should be allowed in a ForEveryPixel handler.

Release Notes for FM Lite 0.4.11

1. New Features
1.1 The "Make..." button now works on Windows 95 (and Windows 98?) as well as on Windows NT 4.0. You will be prompted for your FM username and password the first time you press Make in each host session.

1.2 The generated x86 code and all FD-specified control label text embedded in a standalone filter is now encrypted to discourage pilfering and reverse engineering.

1.3 Each generated standalone filter contains embedded information that allows it to be traced back to the original copy of FM which generated it. Modifying this embedded information will render the filter inoperable.

Release Notes for FM Beta 0.4.12

1. New Features
1.1 New Control Classes

This release introduces four new user controls: BITMAP, IMAGE, METAFILE, and ICON.

1.1.1 BITMAP Control Class

The BITMAP control class is based on the Win32 static bitmap control, and is used to display a (possibly scaled) bitmap in a rectangular region of the dialog box. The format for the BITMAP control is:

ctl[n]:BITMAP(<bitmap_properties>),image="filename",<general_properties>

where '(<bitmap_properties>)' is an optional comma-separated list of bitmap-specific properties, including:

NOTIFY - enables the control to send messages when it is clicked. You must specify this if the control is to process tooltips, cause an action, or invoke a handler; otherwise, this control is purely passive

CENTERIMAGE - centers the bitmap within the rectangle instead of scaling it to cover the entire rectangle

SUNKEN - draws a sunken-looking frame around the rectangle

BORDER - these properties draw other various frames around the specified rectangle
CLIENTEDGE
STATICEDGE
MODALFRAME

Note that the BITMAP control does not support transparency; i.e., the bitmap will always have a solid background color. For a control that displays a pseudo-transparent bitmap, use the IMAGE control below.

1.1.2 IMAGE Control Class

The IMAGE control class is similar to the BITMAP control, but is based on the Win32 owner-drawn static class. The principal difference is that the IMAGE control supports a form of pseudo-transparency: The color of the first pixel in the bitmap (i.e., the pixel in the upper left corner) is taken to be the "transparency" color for the bitmap. All pixels of this color in the bitmap will be replaced with the underlying background pixels in the dialog box, thus giving the appearance of transparency. The format for the IMAGE control is:

ctl[n]:IMAGE(<image_properties>),image="filename",<general_properties>

where '(<image_properties>)' is an optional comma-separated list of image-specific properties, including:

NOTIFY - enables the control to send messages when it is clicked. You must specify this if the control is to process tooltips, cause an action, or invoke a handler; otherwise, this control is purely passive

SUNKEN - draws a sunken-looking frame around the rectangle

BORDER - these properties draw other various frames around the specified rectangle
CLIENTEDGE
STATICEDGE
MODALFRAME

1.1.3 METAFILE Control Class

The METAFILE control class is similar to the BITMAP control, but is based on the Win32 ENHMETAFILE static control. The METAFILE control is used to display a metafile-based image (rather than a bitmap image) in the specified rectangle. The image to be displayed may be either a 16-bit Windows Metafile (.wmf), with or without an Aldus placeable header, or a 32-bit Enhanced
Metafile (.emf). The format for the METAFILE control is:

ctl[n]:METAFILE(<metafile_properties>),image="filename",<general_properties>

where '(<metafile_properties>)' is an optional comma-separated list of metafile-specific properties, including:

NOTIFY - enables the control to send messages when it is clicked. You must specify this if the control is to process tooltips, cause an action, or invoke a handler; otherwise, this control is purely passive.

SUNKEN - draws a sunken-looking frame around the rectangle

BORDER - these properties draw other various frames around the specified rectangle
CLIENTEDGE
STATICEDGE
MODALFRAME

1.1.4 ICON Control Class

The ICON control class is similar to the BITMAP control, but is based on the Win32 ICON static control. The ICON control is used to display a standard Windows icon (a special form of bitmap image) in the specified rectangle. The format for the ICON control is:

ctl[n]:ICON(<icon_properties>),image="filename",<general_properties>

where '(<icon_properties>)' is an optional comma-separated list of icon-specific properties, including:

NOTIFY - enables the control to send messages when it is clicked. You must specify this if the control is to process tooltips, cause an action, or invoke a handler; otherwise, this control is purely passive.

CENTERIMAGE - centers the icon within the rectangle instead of scaling it to cover the entire rectangle

SUNKEN - draws a sunken-looking frame around the rectangle

BORDER - these properties draw other various frames around the specified rectangle
CLIENTEDGE
STATICEDGE
MODALFRAME

1.2 New Control Pseudo-classes

This release also introduces two new user control pseudo-classes: NONE and MODIFY.

1.2.1 NONE Control Class

Specifying NONE for the class of a control simply deletes the control. This is useful, for example, to delete one of the predefined usercontrols, or to delete unused controls in a multi-filter. The format for using the NONE class is:

ctl[n]:NONE

To delete control 'n' at run time, call the new built-in function
deleteCtl(n).

1.2.2 MODIFY Control Class

Specifying MODIFY as the control class does not create a new control, but modifies the general properties of an already existing control. This is useful, for example, to customize the general properties of a predefined control. The format for using the MODIFY class is:

ctl[n]:MODIFY,<general_properties>

1.3 Tooltips

You can now define a pop-up tooltip for any user control by using the tooltip="text" property specifier. For example:

ctl[0]:CHECKBOX,"Wrap",tooltip="Wrap effect at edge of image"

For static control classes (IMAGE, BITMAP, ICON, METAFILE, RECT, FRAME, and STATICTEXT), you must specify the 'notify' class-specificproperty so the control can process tooltip messages.

To change the tooltip for a control at run time, call the built-in function setCtlToolTip(c,"new tooltip text",s), where 'c' is the index of the control and 's' specifies any style bits (usually 0). For example:

```
setCtlToolTip(0, "Replicate pixels at edge of image", 0x02);
```

where the style 0x02 (TTF_CENTERTIP) causes the tooltip to be centered beneath the control rather than positioned to the right of the cursor.

To delete the tooltip for a control, specify tooltip="" or specify NULL instead of a character string for the second argument in the call to setCtlToolTip().

1.4 New Function setCtlImage()

Use the new built-in function setCtlImage(c,"filename",'X') to change the image displayed by an IMAGE, METAFILE, BITMAP, or ICON control at run time. 'X' is a 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

Note: Only the 'B', 'W', 'E', and 'I' file types are recognized in the current release. The remaining types are reserved for future use.

1.5 New Function ErrorOk()

A new built-in function, ErrorOk(fmt,...), has been added. This is similar to the Error() built-in function, but displays only one pushbutton (OK), and always returns IDOK.

1.6 Action=ABOUT

A new default control action ('action=about') has been added. Specifying this action for a control causes the FM 'About' dialog box to be displayed when the control is activated. (For static control classes, you must include the 'notify' class-specific property if this action is to have any effect.) To set this action for a control at run time, call the setCtlAction(n,a) function with CA_ABOUT as the second argument.

1.7 TRACKBAR Tick Marks

You can now set the frequency of tick marks in a TRACKBAR control with the 'ticfreq=n' control property, or by calling the setCtlTicFreq(c,n) built-in function. The control will draw a tick mark at an interval of every 'n' units; the default is n=1. (You must specify the 'autoticks' property for the TRACKBAR control if this to have any visible effect.)

2. Changes

2.1 New Registration Scheme (Full version only)

In the full versions of FM, authorization to generate a standalone filter is now based on a 'registration code' rather than a password. You will be asked for your user name, organization, and registration code the first time you try to Make a standalone filter. Thereafter, the registration information is stored in a secure manner in the Windows registry, and you will normally never be required to reenter your registration information.

2.2 Configurable Predefined Controls

The OK, Cancel, Edit, and FM Logo predefined controls are now fully configurable by the Filter Designer. They are defined the same as an ordinary user control, using special dedicated control indexes at the high end of the index range. Since these special control indexes may be assigned different values in future releases, always refer to them by their symbolic names: CTL_OK, CTL_CANCEL,
CTL_EDIT, and CTL_LOGO. The default definitions for these controls are:

```
ctl[CTL_OK]:PUSHBUTTON(default),"OK",pos=(306,126),size=(34,14),
        action=apply,tooltip="Apply filter to main image"

ctl[CTL_CANCEL]:PUSHBUTTON,"Cancel",pos=(269,126),size=(34,14),
        action=cancel,tooltip="Exit without applying filter"

ctl[CTL_EDIT]:PUSHBUTTON,"&amp;amp;Edit >>>",pos=(231,126),size=(34,14),
        action=edit,tooltip="Edit filter source code"

ctl[CTL_LOGO]:IMAGE(notify),image="logo2.bmp",pos=(352,97),size=(32,43),
        action=about,tooltip="About this filter"
```

If you want to customize these predefined controls in your filter, you can completely redefine or delete them. If you want to change only the general properties of a predefined control, you can simply use the MODIFY pseudo-class specifier. For example, to change the logo 'image' and 'size' general properties:

```
ctl[CTL_LOGO]:MODIFY,image="MyLogo.bmp",size=(34,40)
```

To delete a predefined control altogether, use the NONE pseudo-class specifier. For example, if you want to delete the OK control from your demo filter, specify:

```
ctl[CTL_OK]:NONE
```

Note that the CTL_EDIT control is handled specially. You cannot change the predefined text labels and tooltips for this control, since they are assigned dynamically by FM. If you really want to change the label and/or tooltip for an EDIT control, delete the predefined control and define your own edit control using some other available control index. For example:

```
//delete the predefined edit control
ctl[CTL_EDIT]:NONE

//now define my own custom version
ctl[49]:PUSHBUTTON,"Edit",action=edit,tooltip="Open/Close edit window"
```

Finally, note that the CTL_EDIT control is *never* defined in a standalone filter; any references to the CTL_EDIT control index will be ignored in a standalone filter.

In future releases, the predefined proxy preview, progress bar, and Host Application static text controls will also be fully configurable.

In the current release, the range of valid control indexes is 0 - 63,inclusive. The predefined controls are assigned indexes in the range 50 - 63. To avoid conflict with the predefined controls, use indexes in the range 0 - 49 for your own control definitions.

2.3 New Default FM Logo

The default FM Logo in the filter dialog box now uses the IMAGE control to display a transparent scaleable bitmap of Wern's Gartenzwerg logo. You are free to redefine the logo in any way you like. To revert to the metafile- based "butterfly" logo used in release 0.4.11, use the following control and Embed specifications:

```
ctl[CTL_LOGO]:METAFILE(notify),image="buttrfly.wmf",pos=(329,90),size=(56,47),action=about

Embed:metafile="buttrfly.wmf"
```

Note: "buttrfly.wmf" is supplied as an external file in the FM installation package.

2.4 Host Application Trademark Symbols

To be legally correct, appropriate trademark (tm) and registered trademark (®) symbols have been inserted into the names of various host applications as displayed by the Host Application static text control in the FM dialog box.

2.5 Use CMOV and Use MMX

The 'Use CMOV' checkbox in Advanced Edit mode is now turned off by default, even if the CMOV instructions are available on the host processor. If either the 'Use CMOV' or 'Use MMX' checkbox is checked when you try to MAKE a standalone filter, FM warns that the resulting filter may not execute correctly on all systems.

2.6 Warning to Embed Resources

FM now warns you when making a standalone filter if you have specified a background image for the dialog box, or you have specified an image for any of the static image controls (IMAGE, BITMAP, ICON, METAFILE), but no files have been embedded with the Embed: specifier. This is not necessarily an error, but a reminder to the FD to embed any resources as desired.

2.7 Reminder to use Return Statement

FM now warns you if your ForEveryTile handler does not contain a return statement. The previous "kludge" of specifying a bare 'true' or 'false' expression as the return value is now obsolescent, and will be phased out altogether in a future release. This warning will serve as a reminder to you to replace all such bare 'true' or 'false' return values with the proper return statement (i.e., 'return true' or 'return false').

This warning also applies to other handlers which expect a return value, such as OnZeroDivide.

2.8 New Control Default Values

The default size, position, and action for certain user controls has changed. This may affect your dialog layout if you depended on these defaults.

2.9 New Embeddable File Types

In addition to bitmaps, you can now embed metafiles, icons, and wave files in your standalone filter with the Embed: key. To embed a metafile, specify:

```
Embed:metafile="filename"
```

To embed an icon, specify:

```
Embed:icon="filename"
```

To embed a wave file, specify:

```
Embed:wave="filename"
```

Note: Wave files are reserved for future use.

2.10 New End-of-file Token

The optional FF+ end-of-file token has been changed from %EOF to %%EOF to avoid a syntactic ambiguity. You can think of %%EOF as marking the start of a "super" comment that extends to the physical end of the source file.This is useful, for example, to comment out large sections of source code when tracking down the location of a compilation error.

3. Bug Fixes

3.1 A scrollbar with an inverted range (min > max) did not display the initial value correctly. This has been fixed.

3.2 Vertical scrollbars and trackbars (e.g., STANDARD(vert), SCROLLBAR(vert), or TRACKBAR(vert)) now function correctly.

3.3 The 'bottomtip' and 'righttip' properties for a horizontal or verticalTRACKBAR control (respectively) now operate correctly. (Previously, 'bottomtip' functioned the same as 'toptip' and 'righttip' functioned the same as 'lefttip'.)

3.4 The background of a STATICTEXT control is now updated properly when changing the text for the control.

3.5 A TRACKBAR control is now updated correctly whenever its background color is changed.

3.6 The default action for a scrollbar or trackbar was ignored in previous releases, and the 'apply' action was always performed. This has been fixed.

3.7 Referencing a user-declared array no longer causes a GP fault. (However, note that arrays are still not implemented, and should *not* be declared in your current FF+ code.)

3.8 A few resource leaks have been fixed.

4. Known Problems
4.1 A TRACKBAR control with an inverted range (min > max) does not functioncorrectly.

4.2 The TTF_CENTERTIP style is not always applied correctly when specified in a call to setCtlToolTip().

Release Notes for FM Beta 0.4.13

1. Changes
1.1 The text string for a tooltip is now formatted with the FM formatString() function as described below (see the release notes for 0.4.10).

2. Bug Fixes
2.1 A problem with parsing the 'bitmap' subkey for the 'Embed' resource embedding key has been fixed.

3. Known Problems
3.1 The background for a transparent ICON control is not properly updated when the background color is set to 'transparent' (which is the default). For now, always specify a solid background color for an ICON control.

3.2 The background color for the static text label field of a STANDARD (default) control is not updated properly if a solid background color is specified. For now, always use color=transparent (which is the default) for a STANDARD control.

Release Notes for FM Beta 0.4.14

1. New Features

1.1 Two new FM specifications have been added (Organization and URL).

1.1.1 The format of the Organization specification is:

Organization:"<name of your organization>"

Use this specification to supply the name of your organization for use in the plug-in's VS_VERSION_INFO resource and elsewhere. The string containing the Organization specification is available in the new FM built-in variable filterOrganizationText, or as format specifier '!O' within an FM-formatted string.

1.1.2 The format of the URL specification is:

URL:"<http://www.your-company.com>"

Use this specification to supply the URL of your website for use in the plug-in's VS_VERSION_INFO resource and elsewhere. The string containing the URL specification is available in the new FM built-in variable filterURLText, or as format specifier '!U' within an FM-formatted string.

1.2 FM now sets the following fields in the VS_VERSION_INFO resource in a standalone filter. (These fields can be viewed by selecting the plug-in in Windows Explorer and choosing File/Properties/Version from the menu bar.)

File version:	Set from the FM Version: specification.
Description:	Set from the FM Description: specification.
Copyright:	Set from the FM Copyright: specification.
Author:	Set from the FM Author: specification.
Comments:	Set to "Generated by FilterMeister®".
Company Name:	Set from the FM Organization: specification.
Internal Name:	Set to "AfhFM04".
Language:	Set to "English (United States)".
Legal Trademarks:	Set to "FilterMeister is a trademark of AFH Systems Group".
Original Filename:	Set to the base file name to which the plug-in was saved.
Product Name:	Set from the FM Title: specification.
Product Version:	Set from the FM Version: specification.
URL:	Set from the FM URL: specification if present.

1.3 A new built-in function, chooseColor(), invokes the host application's default color picker dialog to allow the user to select an RGB color. The form of a call to chooseColor() is:

rgb = chooseColor(initialColor, "Prompt string" [, args]...);

where:

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

"Prompt string" specifies the prompt string for the color picker.
This string may contain printf-style format descriptors, which will be expanded using the succeeding arguments.

args are optional arguments to be expanded into the prompt string.

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

1.4 The host application's current foreground and background colors are now available as RGB triples in the built-in variables fgColor and bgColor, respectively.

1.5 The individual components of an RGB triple or an RGBA quadruple may now be conveniently accessed with new built-in functions Rval, Gval, Bval, and Aval, which select the Red, Green, Blue, or Alpha components, respectively.
The format for calling these functions is:

```
r = Rval(rgbTriple);
g = Gval(rgbTriple);
b = Bval(rgbTriple);
a = Aval(rgbaQuad);
```

1.6 Three new built-in functions allow your filter to play sound (.wav) files upon demand.

The specified wave file name is first searched for in the filter's embedded resources. If not found among the resources, the wave file name is searched for in the local file system, using the customary search paths (see 2.9 below).

Each function returns true if successful or false otherwise. Any waveform sounds that are played asynchronously will be automatically stopped when the filter exits.

1.6.1 To play a wave file asynchronously (i.e., the wave file starts playing and the function returns immediately, without waiting for the waveform sound to finish playing):

```
success = playSoundWave("wavefilename.wav");
```

To stop any currently playing waveform sound, call this function with a NULL argument:

```
success = playSoundWave(NULL);
```

1.6.2 To play a wave file synchronously (i.e., the function waits for the waveform sound to finish playing before returning):

```
success = playSoundWaveSync("wavefilename.wav");
```

1.6.3 To start playing a wave file repeatedly:

```
success = playSoundWaveLoop("wavefilename.wav");
```

To stop a looping waveform sound:

```
success = playSoundWave(NULL);
```

1.7 A new built-in string variable, filterInstallDir, will contain the full path name of the directory from which your filter was loaded (which is presumably the directory in which it was installed). This is useful for locating files (such as HTML help files) which were installed as separate resources in your filter's installation directory. (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.)

1.8 A new built-in function, shellExec(), lets your filter execute a Windows shell command. The calling format is:

```
retcode = shellExec("Verb", "Filename", "Params", "DefDir");
```

where retcode will be -1 if successful, or an error code otherwise. "Verb" is the shell operation to be performed (e.g., "open", "print", or "explore"). "Filename" is 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. "DefDir" specifies the default directory for the execution of the command. Any unneeded parameter may be set to NULL. If NULL is specified for "Verb", the function opens the file or URL specified by "Filename".

For 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);
```

1.9 Four new built-in functions allow you to set and retrieve pixel values based on radial (polar) coordinates instead of rectangular (Cartesian) coordinates

pgetr(d, m, z) - returns the destination (output buffer) pixel value at angle d, distance m, from channel z
(analogous to the pget(x,y,z) rectangular function)

psetr(d, m, z, v) - sets the pixel value at angle d, distance m, in channel z of the destination (output) buffer to the clamped value of v, and returns this clamped value (analogous to the rectangular function pset(x,y,z,v))

tgetr(d, m, z) - returns the pixel value at angle d, distance m, from channel z of the first tile buffer (analogous to the tget(x,y,z) rectangular function)

tsetr(d, m, z, v) - sets the pixel value at angle d, distance m, in channel z of the first tile buffer to the clamped value of v, and returns this clamped value(analogous to the rectangular function tset(x,y,z,v))

t2getr(d, m, z) - returns the pixel value at angle d, distance m, from channel z of the second tile buffer (analogous to the t2get(x,y,z) rectangular function)

t2setr(d, m, z, v) - sets the pixel value at angle d, distance m, in channel z of the second tile buffer to the clamped value of v, and returns this clamped value(analogous to the rectangular function t2set(x,y,z,v))

Note that care must be exercised when using the *setr() functions, since there is no guarantee that they can be used to densely populate the plane; i.e., due to integer rounding of coordinates, etc., there may be points in the target buffer which cannot be addressed via integer polar coordinates (d,m). Thus, you may want to set the destination plane to all black (for example) before using the *setr() routines to cover any "holes" in the mapping.

1.10 Two new built-in functions allow you to efficiently apply gamma correction to an image:

setGamma(g) - precomputes a gamma correction table for gamma value g, where g is a float or double value. (The gamma table is initially set to gamma = 1.0.) Returns true if successful, else false.

gamma(i) - returns the gamma-corrected value of i, where 0 <= i <= 255. The result is undefined if i is out of range.

Example: To copy the input buffer to the output buffer, applying a gamma correction of 1.8:

```
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)));
```

1.11 Three new event handlers have been added.

1.11.1 The **OnFilterStart** handler is invoked at the start of each filter run, just before the ForEveryTile, 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. The OnFilterStart handler should 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 0.4.14).

1.11.2 The **OnFilterEnd** handler is invoked at the end of each filter run, after all calls to the ForEveryTile, 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. The OnFilterEnd handler should 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 currentlyignored in FM 0.4.14). The default OnFilterEnd handler calls playSoundWave(NULL) to terminate any wave file that may still be playing.

1.11.3 **The OnCtl(n)** handler is invoked whenever the user clicks on or changes the value of a control. Within the OnCtl(n) handler, built-in variable n will be set to the index of the control which was activated, and built-in variable e will be set to one of the following symbolic constants to indicate the specific event that occurred:

Symbolic Constant	FM Event
FME_UNKNOWN	Unknown or unspecified event.
FME_CLICKED	Control was clicked (push buttons, etc.).
FME_DBLCLK	Control was double-clicked.
FME_PAGEUP(*)	Control was incremented by a large step (scrollbars, trackbars).

FME_PAGEDOWN(*)	Control was decremented by a large step (scrollbars, trackbars).
FME_LINEUP(*)	Control was incremented by a small step (scrollbars, trackbars).
FME_LINEDOWN(*)	Control was decremented by a small step (scrollbars, trackbars).
FME_MOUSEOVER	Mouse cursor entered the control window.
FME_MOUSEOUT	Mouse cursor exited the control window.

(*) - These events are currently not processed by FM 0.4.14.

The following FM event codes are also defined, but cannot occur as the value of e in the OnCtl(n) handler:

FME_ZERODIVIDE	An attempt was made to divide an integer by 0.
FME_DIVIDEOVERFLOW	Overflow occurred while trying to divide two integers.

1.12 A new built-in function doAction(a) performs one of several predefined control actions specified by its integer argument a, where a 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 (not yet implemented in FM 0.4.14).
CA_NONE	Performs no action.

1.13 The following additional built-in (read-only) variables have been added to expose information provided by the host application. (Note that these values are guaranteed to be valid only when Adobe Photoshop is the host; other so-called Photoshop-compatible hosts may or may not provide useful information in these variables.)

Variable Name	Type	Value
hostSerialNumber	Int	The host application's serial number.Your plug-in filter can use this value as part of a copy-protection scheme. (Note: The Mac version of Photoshop apparently sets this field to a valid serial number, but on the Windows platforms Photoshop always sets this field to 0. Use the hostSerialString variable instead, which will be implemented in a later version of FM.)
imageWidth	Int	The source image's width in pixels. If the selection is floating, this variable instead holds the width of the floating selection.
imageHeight	Int	The source image's height in pixels. If the selection is floating, this variable instead holds the height of the floating selection.
wholeWidth	int	The width in pixels of the entire source image, regardless of any floating selection.
wholeHeight	int	The height in pixels of the entire source image, regardless of any floating selection.
planes	int	For Photoshop 4+, this variable contains the total number of active planes in the image, including alpha channels. The image mode should be determined by looking at the imageMode variable. For earlier versions of Photoshop, this variable will be equal to 3 if filtering the RGB channel of an RGB color image, or 4 if filtering the CMYK channel of a CMYK color image. Otherwise it will be equal to 1.
maxSpace	int	The maximum number of bytes of information the plug-in can expect to be able to access at once (input area size + output area size + mask area size + bufferSpace).
isFloating	bool	This variable is true if and only if the selection is floating
haveMask	bool	This variable is true if and only if a non- rectangular area has been selected

bgColor	int	The current background color as a packed triple or quadruple in the color space native to the source image. Use the Rval, Gval, Bval, and Aval functions described in section 1.5 above to access the individual components of bgColor
fgColor	int	The current foreground color as a packed triple or quadruple in the color space native to the source image. Use the Rval, Gval, Bval, and Aval functions described in section 1.5 above to access the individual components of fgColor
hostSig	int	The host application provides its signature in this variable. Adobe Photoshop's signature is '8BIM'. 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'. if (hostSig == '8BIM') { // Host is Adobe Photoshop... }
imageMode	int	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, and 48-bit RGB Color = 11.
filterCase	int	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.
samplingSupport	int	Indicates whether the host supports non-1:1sampling 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 integralsampling steps; future versions may support non-integral sampling steps.

1.14 A new integer built-in variable, zoomFactor, has been added. This will contain a value 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.)

1.15 Two new built-in functions, getCtlColor(c) and getCtlVal(c), have been added. getCtlColor(c) returns the current color of control c, as an RGB triple. getCtlVal(c) returns the current integer value of control c, and is simply a synonym for the ctl(c) function.

1.16 Two new built-in integer functions, HDBUsToPixels(n) and VDBUsToPixels(n),have been added. HDBUsToPixels converts horizontal DBUs (Dialog Box Units) to pixels; VDBUsToPixels converts vertical DBUs to pixels. The conversion factor depends on the current system font, and can be different in the horizontal and vertical directions.

2. Changes

2.1 The number of anonymous put/get cells has been increased from 256 to 1024.

2.2 The maximum number of local variables that the user can declare in a handler has been doubled from 200 to 400 (float and double variables count as two variables each).

2.3 The size of the compiler's semantic descriptor stack has been doubled, allowing the compilation of very complex expressions. Various code buffer sizes have also been increased to allow for the compilation of very long code sequences. If a code buffer nevertheless overflows, the name of the code buffer will now be identified in the overflow error message. Please report this buffer name when submitting a bug/performance report against FM.

2.4 Control values are now properly cached before the OnFilterStart handler is invoked.

2.5 Two alternatives to the built-in mix() function have been added, to handle rounding errors in different ways: mix1() and mix2(). The original mix() function retains its previous behavior for compatibility with Filter Factory.

mix(a,b,n,d) is computed as a*n/d + b*(d-n)/d, per Filter Factory.

mix1(a,b,n,d) is computed as (a*n + b*(d-n))/d, per Werner Streidt, to avoid some of the rounding errors inherent in the FF formula.

mix2(a,b,n,d) is computed using a more complex formula to avoid other rounding errors that are introduced when (b - a)*n < 0.

The bottom line: Experiment with mix, mix1, and mix2 to find out which version works best in your filter. When in doubt, mix1 is probably the best choice.

2.6 The size of the static text label associated with a standard scrollbar control is now dynamically adjusted to exactly fit the displayed text rather than using a fixed-size field. This should help eliminate some problems with truncation or overlap of label fields. Also, tooltips have been re-enabled for the static text label.

2.7 Setting the border style for a standard scrollbar control now sets the border style of the associated static text label.

2.8 The NOTIFY attribute has been implemented for button class controls. Setting this attribute allows the control to generate FME_DBLCLK events, which may then be processed by the OnCtl(n) handler. Note that an FME_DBLCLK event will always be preceded by an FME_CLICKED event for the same control.

2.9 FM now searches for a file in the following order when looking for a resource file such as a bitmap, icon, metafile, or sound wave file:

1. The current working directory is searched;
2. All directories listed in the PATH environment variable are searched;
3. All directories listed in the FM_PATH environment variable (if defined) are searched;
4. If the filename specifies an absolute pathname, that pathname is searched.

2.10 An error message is now issued when a case label (FM-E-ILLCASELAB) or a default label (FM-E-ILLDEFLAB) is found outside of any switch statement.

2.11 A warning is now issued (FM-I-IGNLAB) when a statement label is encountered, to warn that the label will be ignored.

2.12 Fatal "X86 Emit Error" message boxes during compilation now correctly abort the compilation when acknowledged, instead of trying to continue.

2.13 The CANCEL button in an "X86 Emit Warning" message box now correctly aborts the compilation.

2.14 FM now recognizes 'UP20' as the host signature for Megalux Ultimate Paint 2.0.

2.15 The host signature 'PSAd' is now reported as "Adobe Photoshop Adapter" instead of "ImageReady", since this signature is used by several Adobe products, not just ImageReady.

2.16 The body color of a scrollbar (STANDARD or SCROLLBAR) control can now be set with the color= property or the setCtlColor() function.

2.17 In Advanced editing mode, the following additional fields are now displayed in the Filter Parameter Block window if the Property Suite is available:

propertyProcsVersion, numPropertyProcs, getPropertyProc, setPropertyProc

If the getPropertyProc (or older getProperty) procedure is available, it is used to obtain and display the following additional information:

propNumberOfChannels	- Number of channels in the document.
propCopyright	- Whether the current image is considered copyrighted (1) or not (0).
propInterpolationMethod	- Current interpolation method: 1=point sample, 2=bilinear, 3=bicubic.
propTitle	- The title of the current image (usually its file name, or "Untitled-n" if not yet saved).
propURL	- The URL for the current image.

propSerialString - Serial number of the plug-in host as a string.

2.18 The syntax for <color> can now include any integer expression; e.g., Dialog:color=100*256 will set the dialog background color to a dark green, and ctl[0]:fontcolor=fgColor will set the font color of control 0 to the currently selected foreground color.

2.19 The temporary tile buffers (tbuf and t2buf) are now set to the size of the input tile instead of the size of the output tile.

2.20 Several additional format descriptors are now recognized by the formatString() function and all other functions which implicitly call formatString: !h, !m, !O, !U, !w, and !z.

The complete list of descriptors recognized by formatString is now:

descriptor	is replaced by
!!	"!" (i.e., a verbatim exclamation point)
!A	text specified by the Author key (filterAuthorText)
!a	text specified by the About key (filterAboutText)
!C	text specified by the Category key (filterCategoryText)
!c	text specified by the Copyright key (filterCopyrightText)
!D	text specified by the Description key (filterDescriptionText)
!F	text specified by the Filename key (filterFilenameText)
!f	the current Filter Case as a text string (filterCaseText) e.g., "Flat image, no selection"
!H	the name of the Host application (filterHostText) e.g., "Adobe Photoshop"
!h	the source image height as a decimal integer (imageHeight)
!M	the current Image Mode as a text string (filterImageModeText) e.g., "RGB Color"
!m	the current Image Mode as a decimal integer (imageMode)
!O	text specified by the Organization key (filterOrganizationText)
!T	text specified by the Title key (filterTitleText)
!t	text specified by the Title key, with any trailing ellipsis removed
!U	text specified by the URL key (filterURLText)
!V	text specified by the Version key (filterVersionText)
!w	the source image width as a decimal integer (imageWidth)
!z	the proxy zoom factor as a decimal integer (zoomFactor)

3. Bug Fixes

3.1 The built-in val() function now works correctly when the control range is other than the default [0, 255].

3.2 A setCtlText() redraw problem has been fixed for check box and radio button controls.

3.3 A bug which occurred when a switch statement had no default case has been fixed.

3.4 A problem which caused spurious error messages for 'short' and 'long' type specifiers has been fixed.

3.5 A bug which could cause a General Protection fault when the proxy preview was zoomed larger than the size of the preview window has been fixed.

3.6 A bug which caused incorrect code to be generated for Duff's device (and other cases in which a case label was nested within a loop statement enclosed by the corresponding switch statement) has been fixed.

3.7 A bug in the deletion of control fonts which could cause a crash on Windows NT has been fixed.

3.8 A problem which caused an incorrect syntax error message while looking for a specific character during compilation has been fixed.

3.9 A retry bug when tile buffer allocation failed has been fixed.

3.10 A workaround has been added for hosts such as PiCo (which uses imhost32.dll) and several others which expect the propertyLength field in the PiPL resource to always be a multiple of 4.

3.11 Several redundant stores to the Premiere-compatible variables i0, i1, u0, u1, v0, and v1 have been eliminated.

4. Known Problems

4.1 The 'MaxBox' subkey in a Dialog specification is currently ignored. This fixes a problem in which the dialog box could be inadvertently maximized but not restored.

4.2 HDBUsToPixels() or VDBUsToPixels() can cause a divide error in USER.EXE if the result would be greater than approximately 32K pixels. This appears to be a limitation of the MapDialogRect Win32 API function.

4.3 Returning a value of true from the OnFilterStart handler does not abort the filter run as it should.

4.4 Returning a value of true from the OnFilterEnd handler does not cause the default OnFilterEnd processing to be skipped as it should.

4.5 The OnCtl(n) handler does not get invoked when a scrollbar or trackbar control is activated.

Release Notes for FM Beta 0.4.15

1. Changes

1.1 The internal name in the VS_VERSION_INFO resource was changed from "AfhFM04" to "AfhFM04s".

2. Bug Fixes

2.1 A quick Y2K fix was implemented to allow FM Beta Testers to continue to use their current registration keys through the end of year 2000.

Release Notes for FM Beta 0.4.16

1. Changes

1.1 The PiPL resource has been temporarily modified to allow FM and FM-generated standalone filters to be invoked in any image mode.

2. Bug Fixes

2.1 A quick fix was implemented to prevent an infinite loop when re-initializing a group of mutually-exclusive radio buttons.
2.2 Several problems with embedding Bitmap and Icon resources have been fixed.
2.3 A small resource leak in SysErrorMessage was fixed.
2.4 Correct binary file and product versions are now generated in the VS_VERSION_INFO resource.

3. Known Problems

3.1 Embedding a Cursor resource is not yet supported.

Release Notes for FM Beta 0.4.17

1. New Features

1.1 Mouseover/mouseout events are now supported for user controls. To enable mouseover/mouseout notification for a control, specify the "mouseover" property for the control. For example:

ctl[1]:STATICTEXT(notify,mouseover),text="My Home Page", fontcolor=cyan

When the mouse cursor enters the window of a control with the mouseover property enabled, the OnCtl(n) handler will be invoked with event code e set to the symbolic value FME_MOUSEOVER. When the mouse cursor leaves the control's window, the OnCtl(n) handler will be called again with event code e set to FME_MOUSEOUT.

For example, you can change the font color of a STATICTEXT control when the mouse cursor passes over it by coding an OnCtl(n) handler similar to the following:

```
OnCtl(n):{
   if (n == 1) {
      switch (e) {
      case FME_MOUSEOVER:
         setCtlFontColor(n, COLOR(brightred));
         break;
      case FME_MOUSEOUT:
         setCtlFontColor(n, COLOR(cyan));
         break;
      case FME_CLICKED:
         shellExec("open", "http://www.filtermeister.com",
               NULL, NULL);
         break;
      }        }
   return false;
}
```

2. Changes

2.1 The MB_TASKMODAL flag is now forced in most message boxes to prevent some problems with top-level windows not being disabled.

3. Bug Fixes

3.1 A refresh problem in IMAGE controls has been fixed.

Release Notes for FM Beta 0.4.18

1. New Features

1.1 The FM source code editor and advanced mode controls have been split off into a separate window from the main FM dialog window. The two windows are "modeless" (i.e., you can freely switch back and forth between them) and run as separate tasks. The principal benefit of this change is that the main FM dialog now works in truly WYSIWYG mode. In particular, you can use the Dialog:size and Dialog:region specifications in the FM editor window to control the size and region of the FM dialog window, and the effects will be accurately reflected in the dialog window at design time.

1.2 Preliminary support has been added for 16-bit deep image modes (i.e., 16-bit Grayscale and 48-bit RGB). The proxy preview now displays correctly in these modes although, of course, the preview is only 8 bits deep, since few if any display drivers support 16-bit color depth. The src() and pset() functions now operate correctly in 16-bit mode. However, other functions such as tset(), t2set(), tget(), and t2get() do *not* yet support 16-bit mode; and the built-in variables r, g, b, R, G, and B also do *not* recognize 16-bit mode, which means that the R/G/B handlers do not yet support 16-bit mode. While you can now write a simple ForEveryTile handler to process a 16-bit image using just the src() and pset() functions, we suggest you wait for full 16-bit support to be implemented in a subsequent release before expending any serious effort on writing 16-bit filters.

1.3 Ten new predefined global string variables str0, str1,..., str9 have been added. Each of these variables is of type char[256].

1.4 The FM source editor now recognizes Ctrl-A as a short cut for the "Select All" operation.

1.5 The FM source editor now accepts the Tab character as a valid source code character. Default tab stops are set at every multiple of 4 characters.

2. Changes

2.1 Several internal changes have been made to the host/plug-in interface for better compatibility with the AVIedit host application; however, FM still does not operate correctly with AVIedit.

2.2 The "Big gulp failed" error message is now issued only in debug versions of FM.

2.3 The maximum number of temporary variables in the FM evaluation stack has been increased from 100 to 200, thus greatly increasing the complexity of expressions that can be compiled.

2.4 The previous limit of ~100 arguments per function call has been doubled to ~200. However, the number of floating-point arguments in a function call is still limited to 8.

2.5 FM now recognizes '8B)M' as the signature for the Tryout version of Adobe Premiere 5.0.

3. Bug Fixes

3.1 A memory leak that could occur whenever a compilation was aborted has been fixed.

4. Known Problems

4.1 Message boxes issued by the main FM dialog window can sometimes be hidden behind the FM Source Editor window. If FM just "dings" when you click either the FM window or the Editor window, try moving the windows around to discover if there is a hidden message box waiting for your input.

4.2 Specifying Dialog:drag=background causes the titlebar ToolTip to appear anywhere in the draggable body of the FM dialog.