MediaManager™
Image and Sound Editing Plug-in for FileMaker Pro

User Guide

Table of Contents

Installing MediaManager

Installation on Macintosh:
Place the MediaManager plug-in into the Extensions folder inside your FileMaker Pro folder.

Installation on Windows:
Place the MediaManager.fmx plug-in into the Extensions folder inside your FileMaker Pro folder.

Remove old versions: Do not forget to remove any older versions of MediaManager that were previously installed. The old plug-in may not be automatically replaced. Having more than one version of MediaManager can cause problems.

Restart FileMaker after installing the plug-in.

Make sure MediaManager is selected in the Plug-Ins section Preferences (found in the Edit Menu on Windows; in the “FileMaker Pro” Application Menu on Mac OS X).

Installation Note: The plug-ins for Windows and Macintosh OS X are different. Please make sure you are using the correct version for your platform.

Installing QuickTime

Much of MediaManager’s functionality is dependent on the presence of QuickTime.

If you do not already have QuickTime on your computer, you can download the free installer for both Macintosh and Windows at:
http://www.apple.com/quicktime/

Registering MediaManager

Registering MediaManager in the Demo File

To register MediaManager enter your user name and license key string into the Registration String field provided in the registration area of the MediaManager demo file. If you have not yet downloaded the current version of the plug-in and associated files from our website, please do so.

http://www.newmillennium.com

Once you have entered your user name and license key, click the Register button in the MediaManager demo file's Register layout. You will know that the plug-in has registered properly if the Registration Pass|Fail field contains the word: "Registered."

Registering MediaManager in Your Solution Files

You can register MediaManager in your own FileMaker Pro solutions by using the Media_Register function in a script. The registration script step can be incorporated into any script in your solution, as long as it is called before you intend the menus to be modified. Therefore it is commonly put into an "On Open" script that is automatically run every time your solution's main file is launched.

You can use a simple Set Field script step to register MediaManager. In the example below we are performing the registration Set Field step into a text field named SFM Response. The registration syntax is:

Media_Register ("My Company|MACCODE|WINDOWSCODE")

The code for the Macintosh platform must appear after the first pipe separator. The code for the Windows platform must appear after the second pipe separator. Even if you are only using the registration code for a single platform, you must still include both pipe separators in the parameter - but only the code for the current platform is checked.

Please note that the user name and license key are case-sensitive.

For a more complete discussion of how to incorporate the registration process into your solutions, see the Media_Register function section below.

Important Registration Note:

Registering MediaManager once does not mean it is permanently registered! It is only registered until FileMaker is shut down. **Each time FileMaker is launched MediaManager must be re-registered.** Registering MediaManager also enables you to test in your solution to make sure that MediaManager is installed.

MediaManager expires after 30 minutes if not registered.

Registration Functions

Media_Register

(regString)

Registers the MediaManager plug-in.

regString – A three part string of the format
“LicenseeName|MacString|WinString”

The parameter must include the user name and the registration code for either the Mac or Windows platform (or both). You can also add an optional password for additional security (more about this function later). Each of the pieces of information are separated by a pipe "|" character:

"LicenseeName|MacCode|WinCode "

The code for the Macintosh platform must appear after the first pipe separator. The code for the Windows platform must appear after the second pipe separator. Even if you are only using the registration code for a single platform, you must still include both pipe separators in the parameter - but only the code for the current platform is checked.

Please note that the user name and license key are case-sensitive.

MediaManager expires after 30 minutes if not registered.

Important Registration Note:

Registering MediaManager once does not mean it is permanently registered! It is only registered until FileMaker is shut down. **Each time FileMaker is launched MediaManager needs to be re-registered.** Calling the Media_Register function when your solution launches enables you to test to make sure that MediaManager is installed.

Registering with the Media_Register function

You can register MediaManager by calling the external "Media_Register" function, usually from a script, though it can also be done from a calculation field or within a validation by calculation. You can include a registration script step in any script in your solution, as long as it is called before you intend the menus to be modified. Therefore it is commonly put into an "On Open" script.

The syntax in the script is:

Media_Register ("My Company|MACCODE|WINDOWSCODE|Optional Password")

To set a script as the file's On Open script, go into FileMaker's File Options (in the File menu). Make sure "Perform Script" is selected under 'When Opening "FILENAME"' in the Open/Close tab, and select your On Open script in the pull-down menu to the right.

If you have a multiple file solution, you must make sure during the opening process that at least one of your files will open and properly register MediaManager in this manner.

Responses

When MediaManager is not loaded or active:
“?”
(In this case, make sure that the MediaManager plug-in is located in FileMaker Pro’s “Extensions” folder, restart FileMaker Pro. Then, if necessary, mark MediaManager’s checkbox in the Plug-Ins section of Preferences.)

When an invalid registration code is used:
“$20001:Invalid registration MediaManager” + version #

A valid registration code will return:
“Registered MediaManager” + version #

Media_RegisterLicensedEncoder

(encoderName ; regString)

Registers additional sound encoder formats that are not automatically registered as part of QuickTime.

Media_RegisterLicensedEncoder can be used to register the iTunes-friendly MPEG-4 AAC encoded sound format.

encoderName – The name of the encoder format being registered: “AAC”
regString – A three part string of the format “LicenseeName|LicenseString”

Most sound format codecs, like WAV and AIFF, are automatically loaded, registered, and available through QuickTime. The AAC encoder format includes iTunes-friendly MPEG-4 audio and video (".mp4", ".m4a", etc.) files. On Windows, AAC/MPEG-4 encoding requires a special registration. Macintosh users can convert to AAC/MPEG-4 format without an AAC license.

You can purchase AAC licenses for a small fee through the New Millennium website <http://www.newmillennium.com>. Just select the last item in the MediaManager purchase popup menu along with the number of licenses you want. Your AAC licenses will be emailed to you.

If you don't properly register the AAC encoder with this function, you may still see MPEG-4 listed by the Media_SoundOutputFormats function, but you will not be able to convert to MPEG-4.

The parameter must include the user name and the registration code, separated by a pipe "|" character:

"LicenseeName|RegistrationCode "

Currently, this function registers only the AAC (MPEG-4) sound encoder. Future versions of MediaManager may be able to register other sound encoder formats.

Important: The Licensee Name must match the License Name used with the Media_Register function, otherwise an error will be returned.

Responses

If the encoderName parameter is blank or an unrecognized type (currently, this function only handles "AAC"):
"$20005:Unknown encoder name (must be AAC)"

When the regString parameter is invalid:
"$20005:Invalid registration" + encoderName format"

A valid licensed encoder registration returns:
"Registered AAC"

Media_Version

(versionFormat)

versionFormat – “”, “short”, “long”, “number” or “platform”

Version function gets a string which may be either empty (“”) or either of words “short”, “long”, or “platform” to specify the output format. If it gets something that makes no sense, the result is empty.

Examples:

To get just the numerical version number:
Set Field [MyTable::Version ; Media_Version (“short”)]

To get the plug-in name and version number together in the response:
Set Field [MyTable::Version ; Media_Version (“long”)]

To get the plug-in version number in the format aabbccdd (eg. version 12.0.1 would return 12000100):
Set Field [MyTable::Version ; Media_Version (“number”)]

To get the platform name – as opposed to the number code returned by Get (SystemPlatform) – use the “platform” parameter:
Set Field [MyTable::Version ; Media_Version (“platform”)]

Image Functions

Demo Note:

When manipulating images with an unregistered version of MediaManager, a white circle will be placed on the resulting image. This behavior disappears when you register MediaManager.

Overview

Images and image files come in many formats, for example, JPEG, GIF, PICT, Bitmap, TIFF, and PNG. Several of these formats are compressed. When an image is compressed, color and detail have been removed resulting in a much smaller file or memory usage when the image is stored. Some image types that support compression are JPEG, GIF, and PICT. Although these images are compressed in such a way that the image looks like the original to the eye, information that was in the original has been lost.

Compressed images present problems for image manipulation programs. For example, suppose you want to rotate to a compressed JPEG. The JPEG must first be decompressed to the pixels and colors it represents, these are rotated, then the image is recompressed back to a JPEG. In the decompress, rotate, recompress sequence, information is lost that existed in the original compressed JPEG. The resulting JPEG shows more visual artifacts, such as jagged lines or blocks of color, than the original. This effect is compounded when several operations are applied, such as rotate, adjust contrast, and sharpen. Each of the three decompress/operate/recompress sequences introduces more artifacts.

MediaManager adopts a strategy that minimizes this loss of information and therefore maximizes the quality of the transformed image. All image transformations and effects operate on uncompressed images at full spatial resolution and color depth. These uncompressed images are stored as a special kind of image, of type 'MMIM', which FMP can store in a container field. MediaManager image transformation functions always return MMIMs. Successive transformations applied to these MMIMs will preserve the maximum image fidelity.

After the desired transformations and effects are applied to the image, Media_ConvertImage is used to render the image into its final form. For example, the image may be converted to a JPEG or GIF suitable for a web page, or reduced in color depth or spatial resolution. By having applied all transformations and effects to a high fidelity image, we insure that the final result is the best possible.

Image Transformations vs. Effects

Operations on images fall into two categories, transformations and effects. Transformations alter the size, shape, and orientation of an image, for example, scale, rotate, skew, reflect, and crop. Effects, in contrast, alter the content of the image itself, for example, brightness, color, or sharpness, and so forth.

See the sections on Image Transformation Functions and Image Effects Functions for further explanation.

Image Location Parameters

Most MediaManager functions require as a parameter the image to be operated on.

Image Stored in Container Field

In most cases this will simply be a container field, and you can choose to leave the parameters at this. However, MM functions provide more flexibility, should you ever need it.

Image in Container Field Stored by Reference

MediaManager handles images stored by reference in container fields the same way as if the image was held directly in the container field. No special parameters are required. Remember the result of any image manipulation is an MMIM that is usually stored directly into a container.

Image File Path

Wherever a container image is expected, you can alternatively use a file path, either relative to the default folder or a full path.

Image File Path in a Text Field

You can also refer to a text field or text calculation that results in a valid file path.

Image URL

Wherever a container image is expected, you can alternatively use a URL, beginning with "http://". The source image will be fetched from its web location. For example—

http://weather.yahoo.com/images/west_cen_sat_440x297.jpg

Image as Function or Calculation Result
An image parameter is not only a container field, but can be the result of another MM function. For example--

Media_EffectApply (Media_TApply (originalImageContainer) ; effectDescription)

--applies a transformation and then an effect without ever storing the intermediate result in a container.

Understanding Image Types

MM uses four letter codes to identify image types. You see these codes when you get info about an image or image stream. You use them when converting an image from one type to another. Some common types are:

JPEG - JPEG
GIFF - GIF
TIFF - TIFF
PICT - PICT
BMP - Windows Bitmap

MediaManager can only convert images to the output types installed with your copy of QuickTime. To get a full list of all available output types, use the Media_ListImageOutputTypes function (see below).

Displayed Image Formats

All image transformation functions return an image in high-fidelity MMIM format which is then set to a container field. Unfortunately FMP is unable to display these images reliably. So in addition to the MMIM, all image functions place an additional JPEG image in the container that is a lower resolution preview of the MMIM. FMP displays the JPEG preview instead, so that you can see the result of the image operation. This preview is normally clearly marked as a preview with white brackets in the corners; however, you can use Media_SetImageDefaults to turn off either the marking or even the creation of the preview. You may choose not to create a preview, saving computation time, when you create intermediate images that the user will never see.

So in addition to the MMIM, all image functions place an additional JPEG image in the container that is a lower quality (compressed) preview of the MMIM. This preview is normally clearly marked as a preview with white brackets in the corners. However, you can use Media_SetImageDefaults to turn off the marking of the preview. You may also adjust the preview quality. You can even choose not to create a preview, saving computation time when you create intermediate images that the user will never see.

Image Stream

The content of a container is technically not a single image, but an "image stream". It can actually contain several versions of the same image. One place we see this is when a MM function returns MMIM as well as a JPEG preview (FMP displays the JPEG.) In another case, FMP itself will sometimes create a JPEG copy when you import or paste certain image types, such as GIF and TIFF. It does this apparently when rendering the image is time consuming. In that case, FMP converts the source image to JPEG once and displays the JPEG whenever it needs to show the image on screen.

When there are several versions of the image in an image stream, how does MM decide which one to use? First, if an MMIM is present, MM always uses it in preference to any other image types. If not, MM determines which of the images is the "native" type that was originally imported and uses it.

There may be occasions when you want MM to pick a different image from the stream than the one it picks by default. The function Media_SetDefaultImageType (type) lets you specify the image type that the MM functions will select from an image stream, overriding MM's default selection. When you specify a type, MM will choose that type only and return an error if it is not present. This function will be used only in exceptional circumstances.

Converting the Image to its Final Format

Once you have created your image as an MMIM, it remains in a completely general format. But there's some purpose intended for it, for example, a web page, catalog, database field, or photo album. The last step is to render the image into the desired format using Media_ConvertImage. Here you will specify the image type (JPEG, PICT, etc.), bit depth, quality, and resolution. The resulting image can be placed in a container or exported to a file.

Putting It All Together - An Example

Here's an example of using these functions. We start with a photo stored by reference in a container. It's rotated 180 degrees, scaled, and then the user gets a chance to adjust the brightness and contrast. Finally, the edited image is saved in the same folder as the original.

```
Set Field [ errorResult ; Media_TClear ]
        //Clear the current transformation. No result is returned by Media_TClear.
Set Field [ errorResult ; Media_TRotate ( 180 ) ]
        //Set the current transformation to rotate 180 degrees.
Set Field [ errorResult ; Media_TScale ( 0.5 ; 0.5 ) ]
        //Add scale to half size to the current transformation

        //Now apply the current transformation...
Set Field [ newImageContainer ; Media_TApply ( originalImage ) ]

        //newImageContainer now contains the rotated and scaled image (in
        MMIM format).

         //Bring up the contrast/brightness effect dialog, using the transformed
        image as a preview.
Set Field [ effectDescription ;  Media_RunEffectsDialog ( "brct" ;
newImageContainer ; newImageContainer ) ]
        //"brct" selects the brightness/contrast effect dialog
Set Field [ newImageContainer ; Media_EffectApply ( newImageContainer ;
effectDescription ) ]
        //Apply the brightness/contrast adjustment to the image

        //newImageContainer now contains the rotated, scaled, adjusted image
        (still in MMIM format).
        //Now convert the MMIM image to a lower resolution JPEG, suitable for a
        web page: low quality (200), depth 8 bits, 72 dpi.
Set Field [ newImageContainer ;  Media_ConvertImage ( newImageContainer ;
"JPEG" ; 200 ; 8 ; 72 ) ]

        //Next set default path to the original image's folder
Set Field [ errorResult ; Media_SetDefaultPath ( Media_GetPath (
originalImageContainer ) ) ]

        //Save the image as a JPEG image file, with name "Edited Image". The file
        name extension suffix is appended automatically.
Set Field [ errorResult ; Media_ExportContainer ( newImageContainer ; "Edited
Image" ) ]
```

Image Transformation Functions

Transformations alter the size, shape, and orientation of an image, for example, scale, rotate, skew, reflect, and crop.

The QuickTime engine used by MediaManager can apply several transformations to an image at once. In fact, it is best to combine transformations because applying them in one step avoids degrading the image, besides being much faster. MediaManager allows this through the notion of the "current transformation". For example, the following calls set up the current transformation to rotate a quarter turn counterclockwise, scale the image by half, and crop.

```
Media_TClear
Media_TRotate ( -90 )
Media_TScale ( 0.5 ; 0.5 )
Media_TCrop ( 0 ; 0 ; 200 ; 250 )
```

Then this series of transformations is applied to the image in one step:

```
Set Field [ newImageContainer ; Media_TApply ( originalImage ) ]
```

You are then free to apply the current transformation again to any number of different images without setting it up again.

Media_TApply

(container)

Media_TApply applies any currently defined transformations (TRotate, TScale, TSkew, TCrop, TQuad) to the specified image.

container - the location of the source image
The container parameter can be:

- a container field
- file path
- text field with a file path
- image URL
- a function or calculation that results in an image

(See the Image Location Parameter section above.)

To clear transformation settings, use the Media_TClear function (see above).

Inserting Images with Media_TApply

Because the Media_TApply parameter can also be the location of an external image file, you can apply current transformations to the image in that file and place the resulting image into a FileMaker container field. This, in effect, allows you to sidestep the need for a separate process to insert images. If you don't want to actually modify the image before inserting it in this fashion, you can first call the Media_TClear function so the subsequent Media_TApply step does nothing but place the image into the result container.

Transformation Order

If TCrop is one of the transformations defined, the source image will be cropped before other transformations are applied. TQuad is next in priority. When you have several transformations defined at the same time, the will be applied in the order of crop, quad, then other transformations.

Returns:

An MMIM image with preview (see the explanation of image formats above).

Examples:

To take an image from a container field named "Picture 1", rotate it 90 degrees clockwise and place the result into a container named "Picture Result":

```
Set Field [ MyTable::Response ; Media_TClear ]
Set Field [ MyTable::Response ; Media_TRotate ( 90 ) ]
Set Field [ MyTable::Picture Result ; Media_TApply ( MyTable::Picture 1 ) ]
```

To crop and rotate an image at the same time:

```
Set Field [ MyTable::Response ; Media_TClear ]
Set Field [ MyTable::Response ; Media_TCrop ( 45 ; 35 ; 135 ; 145 ) ]
Set Field [ MyTable::Response ; Media_TRotate ( 90 ) ]
Set Field [ MyTable::Picture Result ; Media_TApply ( MyTable::Picture 1 ) ]
```

To insert a picture into a container field from an external image file using the file path and file name:

```
Set Field [ MyTable::Response ; Media_TClear ]
Set Field [ MyTable::Picture ; Media_TApply ( ".D/Images/golden.bmp" ) ]
```

To flip an image upside down as you insert it from an external image file:

```
Set Field [ MyTable::Response ; Media_TClear ]
Set Field [ MyTable::Response ; Media_TScale ( -1 ; 1 ) ]
Set Field [ MyTable::Picture ; Media_TApply ( MyTable::File Location ) ]
```

Media_TClear

Media_TClear clears all of the current transformation function settings. Use this before defining a new set of transformations.

All of the transformation function settings (TRotate, TScale, TSkew, TCrop, and TQuad) remain active until the TClear function is called. This allows you to use the Media_TApply function to apply multiple transformations at once to a single image, or to loop through and transform entire sets of images without having to redefine the transformation settings each time.

This function uses no parameters.

Returns:

Empty string

Media_TCrop

(left ; top ; bottom ; right)

Media_TCrop crops images the next time you call Media_TApply. Cropping is applied to the source image before other transformations (like TRotate, TSkew, TScale).

left ; top ; bottom ; right - Rectangle within the image.

The crop parameter values are rounded to the nearest pixel.

Increasing Image Size

In addition to the ability to crop an image to a smaller canvas size, Media_TCrop can also be used to increase the image canvas size. To do this, all you need to do is set the TCrop parameters outside the original image's dimensions. For example, if you have an image that is 100 x 100 and you want to add 20 pixels of unused space all around the image, use Media_TCrop (-20 ; -20 ; 120 ; 120).

Crop Enacted First

If you call Media_TCrop along with other transformation functions and then activate them all with the same call to Media_TApply, Media_TCrop is always applied first. This is important to be aware of since it can affect the results of other transformation functions.

Activated by Media_TApply.

Returns:

Empty string.

Examples:

To crop an image down to 100 x 100 pixels that starts at an upper left point of 45, 35:

```
Set Field [ Response ; Media_TClear ]
Set Field [ Response ; Media_TCrop ( 45 ; 35 ; 135 ; 145 ) ]
Set Field [ MyTable::Picture Result ; Media_TApply ( MyTable::Picture 1 ) ]
```

To crop an image based on the values in four fields named "Crop Left", "Crop Top", "Crop Bottom", and "Crop Right":

```
Set Field [ Response ; Media_TClear ]
Set Field [ Response ; Media_TCrop ( MyTable::Crop Left ; MyTable::Crop Top ;
MyTable::Crop Bottom ; MyTable::Crop Right ) ]
Set Field [ MyTable::Picture Result ; Media_TApply ( MyTable::Picture 1 ) ]
```

To increase the image canvas size by adding 150 pixels around a 350 x 450 pixel image:

```
Set Field [ Response ; Media_TClear ]
Set Field [ Response ; Media_TCrop ( -150 ; -150 ; 600 ; 500 ) ]
Set Field [ MyTable::Picture Result ; Media_TApply ( MyTable::Picture 1 ) ]
```

Media_TQuad

(sourceScale ; topLeftX ; topLeftY ; topRightX ; topRightY ; bottomRightX ; bottomRightY ; bottomLeftX ; bottomLeftY)

Media_TQuad maps the source image into a quadrilateral of any shape when Media_TApply is called. This transformation can cause the image to appear as it would from any distance or point. Therefore Media_TQuad is often used to create a perspective projection.

sourceScale - the scale of the quadrilateral (in percentage)
topLeftX ; topLeftY ; topRightX ; topRightY ; bottomRightX ; bottomRightY ; bottomLeftX ; bottomLeftY – the X and Y coordinates of the corners of a quadrilateral of any shape.

The scale and coordinates can be fractional. Their full precision is used in the transformation.

Once a TQuad transformation has been applied to an image, it is matched to the dimensions of that image. It should only be re-applied to images of the same size. To apply the TQuad transformation to images of a different size, call TQuad again.

Activated by Media_TApply.

Returns:

Empty string.

Examples:

To create a “Star Wars” title perspective:

```
Set Field [ Response ; Media_TClear ]
Set Field [ Response ; Media_TQuad ( 100 ; 30 ; 0 ; 70 ; 0 ; 100 ; 60 ; 0 ; 60 ) ]
Set Field [ MyTable::Picture Result ; Media_TApply ( MyTable::Picture 1 ) ]
```

To quad an image based on the values in user-entered fields:

```
Set Field [ Response ; Media_TClear ]
Set Field [ Response ; Media_TQuad ( MyTable::Scale ; MyTable::Top Left X ;
MyTable::Top Left Y ; MyTable::Top Right X ; MyTable::Top Right Y ;
MyTable::Bottom Right X ; MyTable::Bottom Right Y ; MyTable::Bottom Left X ;
MyTable::Bottom Left Y ) ]
Set Field [ MyTable::Picture Result ; Media_TApply ( MyTable::Picture 1 ) ]
```

Media_TResize

(xDim ; yDim)

Media_TResize allows you to resize an image to a target size when you next call Media_TApply.

xDim – horizontal dimensions in pixels
yDim – vertical dimensions in pixels

This function is similar to Media_TScale, but instead of scaling the image to a fractional size of the original, you can specify exact target dimensions, horizontally, vertically, or both.

If you specify one dimension as zero, the image will be resized to keep the same proportions. For example, if the source image is 200 x 300, and you specify Media_TResize(100; 0), the image will automatically be resized to 100 x 150.

When using TResize in conjunction with the TCrop transformation, the crop function will be performed first. In other words, in that case it is the cropped image that will be resized. If you are getting confusing results, it may be best to use TApply separately with each function.

Activated by Media_TApply.

Returns:

Empty string.

Examples:

To resize an image to exactly 300 x 250 pixels:

```
Set Field [ MyTable::Response ; Media_TClear ]
Set Field [MyTable::Response; Media_TResize ( 300 ; 250 ) ]
Set Field [ MyTable::Picture Result ; Media_TApply ( MyTable::Picture 1 ) ]
```

To resize an image to a width of 300 pixels, keeping the image in proportion:

```
Set Field [ MyTable::Response ; Media_TClear ]
Set Field [MyTable::Response; Media_TResize ( 300 ; 0 ) ]
Set Field [ MyTable::Picture Result ; Media_TApply ( MyTable::Picture 1 ) ]
```

Media_TRotate

(degrees)

Media_TRotate sets a rotation value in degrees. When you next call Media_TApply the image or images being modified will be rotated.

degrees - the degrees of clockwise rotation

The degree value is for clockwise rotation. You can set a negative value for counterclockwise rotation. For extremely precise rotations, you can define fractional degrees of rotation.

Activated by Media_TApply.

Returns:

Empty string.

Examples:

To rotate an image 90 degrees clockwise:

```
Set Field [ Response ; Media_TClear ]
Set Field [ Response ; Media_TRotate ( 90 ) ]
Set Field [ MyTable::Picture Result ; Media_TApply ( MyTable::Picture 1 ) ]
```

To rotate an image approximately 33 1/3 degrees clockwise:

```
Set Field [ Response ; Media_TClear ]
Set Field [ Response ; Media_TRotate ( 33.33 ) ]
Set Field [ MyTable::Picture Result ; Media_TApply ( MyTable::Picture 1 ) ]
```

To rotate an image 110 degrees counter clockwise:

```
Set Field [ Response ; Media_TClear ]
Set Field [ Response ; Media_TRotate ( -110 ) ]
Set Field [ MyTable::Picture Result ; Media_TApply ( MyTable::Picture 1 ) ]
```

To rotate an image the number of degrees entered into a field named “Rotate”:

```
Set Field [ Response ; Media_TClear ]
Set Field [ Response ; Media_TRotate ( MyTable::Rotate ) ]
Set Field [ MyTable::Picture Result ; Media_TApply ( MyTable::Picture 1 ) ]
```

Media_TScale

(xScale ; yScale)

Media_TScale allows you to change the size of an image to a fractional scale of the original.

xScale - the horizontal scale
yScale - the vertical scale

This function is similar to Media_TResize, but instead of specifying target dimensions in pixels, you indicate a scale value of the original. A value of “1” is full sized. You can define fractional scale values, such as “.5” to get half scale.

Reflecting Images
You can use Media_TScale to reflect images around the x or y axes by using negative values. For example, Media_TScale (-1 ; 1) will reflect the image across the horizontal axis, turning the image upside down. Media_TScale (1 ; -1) will reverse the image left to right.

Activated by Media_TApply.

Returns:

Empty string.

Examples:

To scale an image to 50 percent of its original size:

```
Set Field [ Response ; Media_TClear ]
Set Field [ Response ; Media_TScale ( .5 ; .5 ) ]
Set Field [ MyTable::Picture Result ; Media_TApply ( MyTable::Picture 1 ) ]
```

To squeeze the width in half while keeping the height the same:

```
Set Field [ Response ; Media_TClear ]
Set Field [ Response ; Media_TScale ( 1 ; .5 ) ]
Set Field [ MyTable::Picture Result ; Media_TApply ( MyTable::Picture 1 ) ]
```

To flip an image upside down:

```
Set Field [ Response ; Media_TClear ]
Set Field [ Response ; Media_TScale ( -1 ; 1 ) ]
Set Field [ MyTable::Picture Result ; Media_TApply ( MyTable::Picture 1 ) ]
```

To scale an image based on the values in two fields named “Scale X” and “Scale Y”:

```
Set Field [ Response ; Media_TClear ]
Set Field [ Response ; Media_TScale ( MyTable::Scale X ; MyTable::Scale Y ) ]
Set Field [ MyTable::Picture Result ; Media_TApply ( MyTable::Picture 1 ) ]
```

Media_TSkew

(xSkew ; ySkew)

Media_TSkew lets you skew images the next time you call Media_TApply. Skew transforms rectangles into parallelograms, allowing you to shift the image’s edges in relationship to each other.

xSkew - amount of horizontal skew
ySkew - amount of vertical skew

An unskewed image is the equivalent of TSkew (1 ; 1).

As a simple example, use Media_TSkew (-0.5 ; 0) to skew the image horizontally so it’s top left corner starts halfway across where the bottom edge stops (making the image look like it’s tilting to the right).

Activated by Media_TApply.

Returns:

Empty string.

Examples:

To skew an image horizontally the full width of the image:

```
Set Field [ Response ; Media_TClear ]
Set Field [ Response ; Media_TSkew ( -1 ; 0 ) ]
Set Field [ MyTable::Picture Result ; Media_TApply ( MyTable::Picture 1 ) ]
```

To skew an image vertically:

```
Set Field [ Response ; Media_TClear ]
Set Field [ Response ; Media_TSkew ( 0 ; 1 ) ]
Set Field [ MyTable::Picture Result ; Media_TApply ( MyTable::Picture 1 ) ]
```

To skew an image based on the values in two fields named “Skew X” and “Skew Y”:

```
Set Field [ Response ; Media_TClear ]
Set Field [ Response ; Media_TSkew ( MyTable::Skew X ; MyTable::Skew Y ) ]
Set Field [ MyTable::Picture Result ; Media_TApply ( MyTable::Picture 1 ) ]
```

Image Effect Functions

Image effects, in contrast to transformations, alter the content of the image itself, for example, brightness, color, or sharpness, and so forth.

QuickTime provides many useful effects that can be applied to images, such as sharpen/blur, contrast/brightness, composite, emboss, and convolve. These effects are sometimes quite complex and involve choosing the appropriate parameters. Fortunately, MediaManager and QuickTime make this easy using the Media_EffectDialog function. This function allows the user to choose an effect and its parameters, and see a preview of the result through a simple dialog. After you set up the effect, Media_EffectDialog returns a full description of the effect as a text string.

You next apply the effect to an image:

```
Set Field [ effectDescription ;  Media_EffectDialog ( "" ; previewImage1 ;
previewImage2 ) ]
Set Field [ newImageContainer ;  Media_EffectApply ( effectDescription ;
originalImage1 ; originalImage2 ) ]
```

Like a transformation, you can apply this saved effect to any number of images without running the effect dialog again. Advanced developers can even edit the effect description (an XML text string) to change its parameters programmatically. In this way, your script can have complete control over the applied effect without requiring user intervention.

Note: Effect Functions Require QT 7

The two image effect functions (Media_EffectDialog and Media_EffectApply) require QuickTime 7.0.0 or later to work.

Media_EffectApply

(container1; effectDescription [; container2])

Media_EffectApply adds the specified effect to an image.

container1 - the location of the source image
The container parameter can be:

- a container field

- file path
- text field with a file path
- image URL
- a function or calculation that results in an image

(See the Image Location Parameter section above.)

effectDescription - the XML description of the effect (returned by the Media_EffectDialog function)

container2 - optional second image used for effects that require two images

The normal sequence of events would be to first call the Media_EffectDialog function to allow the user to select an effect and its settings, and then use the returned effect description text in the Media_EffectApply function to apply the effect to an image.

Examples:

To apply an effect to a picture in “Image Original” with the result appearing in a field named “Image Result” (the effect description text is already assumed to be stored in a field named “Effect Description”):

```
Set Field [ MyTable::Image Result ; Media_EffectApply (MyTable::Image Original
; MyTable::Effect Description ) ]
```

To first call the effect dialog and then apply the selected effect:

```
Set Field [ MyTable::Effect Description ; Media_EffectDialog ( “” ;
MyTable::Image Original ; “” ) ]
Set Field [ MyTable::Image Result ; Media_EffectApply ( MyTable::Image Original
; MyTable::Effect Description ) ]
```

To show the effect dialog and then apply the selected effect for effects that rely on two images:

```
Set Field [ MyTable::Effect Description ; Media_EffectDialog ( “” ;
MyTable::Image 1 ; MyTable::Image 2 ) ]
Set Field [ MyTable::Image Result ; Media_EffectApply ( MyTable::Image 1 ;
MyTable::Effect Description ; MyTable::Image 2  ) ]
```

To apply an effect to a picture in an external image file with the result appearing in a field named “Image Result”:

```
Set Field [ MyTable::Image Result ; Media_EffectApply (
```

".A/images/landscapes/ptreyes.jpg" ; MyTable::Effect Description)]

To apply an effect to a picture in an external image file with the filepath and name stored in a text field called "Image Location":

Set Field [MyTable::Image Result ; Media_EffectApply (MyTable::Image Location ; MyTable::Effect Description)]

Media_EffectDialog

(effectType [; previewImage1 ; [previewImage2]])

Media_EffectDialog displays the effect dialog, allowing the user to choose an effect, specify its settings, and see a preview of the result in a preview thumbnail image.

effectType - a string that specifies the effect or effects to display. Use "" (blank) to allow the user to choose any effect. Use the four character effect type to show the parameter dialog for a specific effect (see below).

previewImage1 - optional image to use as the effect preview in the dialog. If omitted, a default image will be used.

previewImage2 - optional image to use as the effect preview in the dialog, when a two source effect is displayed. If omitted, a default image will be used.

Once the user chooses an effect, a text string is returned that describes the selected effect and its parameters (in XML formatting). This description XML text must be passed to Media_EffectApply to apply the effect to an image. You can edit this string manually or via script, calculation, etc. to change the final effect that is applied.

Effect Type Code

If you want to specify the user's choice of effects, you can use the four character code as the first parameter for the effect you want to allow. You can find this four character code in the returned description XML text. Us the "ostype" value within the atom tag. For example, the code for the Brightness/Contrast effect (from the description XML below) is "brco".

If you want to give the user the full list of effect options, leave this parameter blank "" instead.

Activated by Media_EffectApply.

Returns:

This function returns an XML text description of the effect selected along with the settings specified. As an example, the description response for the Brightness/Contrast effect is:

```
<effect>
     <!-- Brightness and Contrast Effect -->
     <!-- Copyright: Standard QuickTime Effect by Apple
Computer, Inc. -->
     <!-- Allows you to adjust the brightness and contrast
of a single source. -->
     <!-- This effect takes one source as input -->
     <!-- This effect's major class is: Filter -->
     <!-- This effect's minor class is: Adjustments -->
     <atom type="what" id="1" ostype="brco" />

     <!-- __Brightness__ -->
     <!-- Cannot be interpolated -->
     <!-- Value must be between -100 and 100 -->
     <atom type="bryt" id="1" long="16" />

     <!-- __Contrast__ -->
     <!-- Cannot be interpolated -->
     <!-- Value must be between -100 and 100 -->
     <atom type="cntr" id="1" long="30" />
</effect>
```

If the user cancels the dialog, $xxx:Cancel is returned.

Examples:

To display the effect dialog and place the resulting description text into a field named “Effect Description”:

Set Field [MyTable::Effect Description ; Media_EffectDialog (“” ; “” ; “”)]

To call the effect dialog and specify the picture in “Image Original” as the preview image used in the effect dialog:

Set Field [MyTable::Effect Description ; Media_EffectDialog (“” ; MyTable::Image Original ; “”)]

To specify both “Image 1” and “Image 2” as the preview images used in effects that rely on two images:

```
Set Field [ MyTable::Effect Description ; Media_EffectDialog ( “” ;
MyTable::Image 1 ; MyTable::Image 2 ) ]
```

To call the effect dialog with only the Brightness/Contrast effect:

```
Set Field [ MyTable::Effect Description; Media_EffectDialog ( “brco” ;
MyTable::Image Original ; “” ) ]
```

To call the effect dialog and apply the selected effect to an image stored in a field named “Image Original” and place the modified image in a field named “Image Result”:

```
Set Field [ MyTable::Effect Description ; Media_EffectDialog ( “” ;
MyTable::Image Original ; “” ) ]
Set Field [ MyTable::Image Result ; Media_EffectApply ( MyTable::Image Original
; “” ; MyTable::Effect Description ) ]
```

General Image Functions

Media_ConvertImage

(containter ; type)
or
(container ; type ; depth ; hres ; vres ; targetBytes ; quality ; imageIndex; transcode ; packBits ; interlace)

Media_ConvertImage converts an image to the specified type, resolution, image depth, and image quality.

container - the location of the source image
The container parameter can be:

- a container field
- file path
- text field with a file path
- image URL
- a function or calculation that results in an image

(See the Image Location Parameter section above.)
type - the image type to convert the image to (JPEG, TIFF, etc.)
depth - the bit depth of the converted image
hres ; vres - the horizontal and vertical resolution in pixels/inch. You must specify either both or neither.
targetBytes - the size in bytes that the converted image should not exceed. Image quality is reduced as needed so this target is met. (Must be set to "-1" to set the quality parameter that follows.)
quality - the image quality level.
0 = Minimum Quality, Maximum Compression
256 = Low Quality, High Compression
512 = Normal Quality, Medium Compression
768 = High Quality, Low Compression
1023 = Maximum Quality, Minimum Compression
1024 = Lossless Quality
Some image types do not implement all of these compression values. ConvertImage will default to the next larger allowable quality. When targetBytes is specified, this parameter is ignored.
imageIndex - the number of the image layer from the source image to use for multi-image formats. (Default is 1.) TIFF and PhotoShop images can contain multiple image layers. MM cannot create multi-layer images.
transcode - When set to "1", the conversion is attempted directly from the source image, without first decompressing it. This can save processing time.
packBits – *(TIFF images only)* When set to "1", TIFF images are compressed

using PackBits (lossless). When “0”, TIFF images are not compressed.
interlace – *(GIF and PNG images only)* When set to “1”, GIF and PNG images will be interlaced (progressive display).

With the above parameters, use “-1” to specify the default value for that setting. The default is usually the value from the source image.

Long and Short Parameter Options

You can use this function in either its short form where you specify only container and type, or in its long form with all ten of the above parameters. The short form, Media_ConvertImage(container ; type) will re-use the same settings last used in the longer form.

Output Type Notes

MediaManager can only convert images to the output types installed with your copy of QuickTime. To get a full list of all available output types, use the Media_ListImageOutputTypes function (see below).

For output type, you can also pass a pipe separated list for the type parameter. If you do this, only the first element will be used as the output type. This is a convenience for using Media_ListImageOutputTypes in a popup menu.

You must convert MMIM type images to a standard type before they can be exported. (See the discussion of MMIM format and image output types above.)

Note: QuickTime does not include a GIF converter.

Converting Without Changing Image Type

If you want to use Media_ConvertImage to change aspects of an image but not alter the image type, set the Type parameter to empty quotes (“”). You can, for example, change image resolution while keeping the image a JPEG, TIFF, etc.

Converting Images in Demo Mode

When you convert an image with an unregistered copy of MediaManager, the resulting image will be marked with a white circle. This behavior disappears when you register MediaManager.

Parameters for Specific Image Types

The targetBytes, quality, transcode, packBits, and interlace do not apply to every image output type. They will be ignored when they do not apply to a particular conversion operation.

Returns:

The converted image of the specified type.

Examples:

To convert an image held in a container field named “Image 1” to JPEG format and place the result into a container field named “Image Converted”:

```
Set Field [ MyTable::Image Converted ; Media_ConvertImage ( MyTable::Image
1 ; “JPEG” ) ]
```

To convert an image using a hardcoded filepath and name to PNG format:

```
Set Field [ MyTable::Image Converted ; Media_ConvertImage ( “.A/image
examples/logo.gif” ; “PNGf” ) ]
```

To convert an image using a text field that holds the filepath and name to TIFF format:

```
Set Field [ MyTable::Image Converted ; Media_ConvertImage ( MyTable::File
Location ; “TIFF” ) ]
```

To convert an image to JPEG with a resolution of 72 pixels per inch:

```
Set Field [ MyTable::Image Converted ; Media_ConvertImage ( MyTable::Image
1 ; “JPEG” ; “” ; 72 ; 72 ; -1 ; “” ; “” ; “” ; “” ; “” ) ]
```

Media_GetImageInfo

(container)

Media_GetImageInfo returns a text block containing detailed information about the image.

container - the location of the image
- can be a container field
- or text field with a valid filepath and filename
- or the filepath and filename as text

Returns image info, such as:

```
Height (px): 474
Width (px): 396
Depth (bits): 24
Quality: 512
V Resolution (dpi): 72
H Resolution (dpi): 72
Size (bytes): 61521
Compressor Type: jpeg
Compressor Description: Photo – JPEG
```

The values of the response – such as Height, Width, Size, etc. -- can be easily isolated using the Media_ExtractInfo function (see below).

Note that Quality for uncompressed formats will be 1024 and for compressed is 512. Because it is not possible to determine from the result image how much the original image was compressed.

Examples:

To get the image info of a picture stored in a container field named “Picture 1”:

```
Media_GetImageInfo ( MyTable::Picture 1 )
```

To get the info of an external image file named “nirvana.jpg” located on the desktop:

```
Media_GetImageInfo ( “.D/nirvana.jpg” )
```

To get the info of an external image file whose filepath and name are stored in a text field named “Image Location”:

```
Media_GetImageInfo ( MyTable::Image Location )
```

Media_GetMetadata

(container)

This function returns any metadata text associated with the specified image.

container - - the location of the image

- can be a container field
- or text field with a valid filepath and filename
- or the filepath and filename as text

Most image files also have a small block of metadata text associated with them that gives basic information like image creation date and the software used to create it.

A common response might look like this:

```
IPTC
 ©swr: Adobe Photoshop 7.0
 ©day: 2003:11:16 13:07:07
```

Creation software is specified by “©swr” and creation date by “©day”.

You can use the Menu_ExtractInfo function to isolate specific elements of the metadata. See the examples below.

Examples:

To get the metadata for an image in a field named Picture and put it into a field named Meta:

```
Set Field [ MyTable::Meta ; Media_GetMetadata ( MyTable::Picture ) ]
```

To use Menu_ExtractInfo in order to isolate the creation software and place the result in a field named Creator:

```
Set Field [ MyTable::Creator ; Media_ExtractInfo ( Media_GetMetadata (
MyTable::Picture ) ; “IPTC/©swr” ]
```

Media_InsertImage

(filepath ; byReference)

Inserts an image into a FileMaker container field. The image can be stored directly in the container field itself or stored by reference.

filepath - a full file path, or partial file path that picks up with the default path. If

the source is a reference, then the destination can be a path to a folder; otherwise, it must be a path to a file.
The filepath parameter can be:

- file path
- text field with a file path
- image URL

(See the Image Location Parameter section above.)

byReference – “1” or “True” to store by reference; “0” or “False” to store in container field

Inserting Web Images

MediaManager can easily grab web images. All you need to do is use the image's URL (Web address) as the file path parameter with Media_InsertImage.

For example, if you want to insert a web image named "banner.jpg" located in the http://www.MySite.com/images/ directory, you can use a script step:

```
Set Field [ MyTable::Container ; Media_InsertImage (
"http://www.MySite.com/images/banner.jpg" ; 0 )
```

A handy technique is to create a calculated container field that automatically inserts a web image at a URL entered into a separate text field. For example, you may have a text field named URL that allows you to enter the web address of an image. A calculated container field can automatically display the image with:

```
Media_InsertImage (URL Image; 0)
```

Examples:

To insert an image into a container field named Picture 1 using a hardcoded filepath:

```
Set Field [ MyTable::Picture 1 ; Media_InsertImage ( “MyDrive/Pictures/logo1.jpg”
; 0 ) ]
```

To store an image by reference a container field named Picture Ref using a hardcoded filepath:

```
Set Field [ MyTable::Picture Ref ; Media_InsertImage (
“MyDrive/Pictures/logo1.jpg” ; True ) ]
```

To allow the user to select an image file to be inserted by reference:

```
Set Field [ MyTable::FileName ; Media_SetDefaultFolder ( ".V:Select the image to insert" ) ]
Set Field [ MyTable::Picture Ref ; Media_InsertImage ( MyTable::FileName ; True ) ]
```

Media_ListImageOutputTypes

Media_ListImageOutputTypes gives you a list of all output image types available with your installed version of QuickTime. These are the image types that MediaManager can convert to and export.

No parameters used.

Image Output Limitations

QuickTime can read a large number of image types. However, it can convert to and export only a subset of these, due to patent and licensing limitations. For example, basic QuickTime cannot export GIF.

The output image type list can be placed in a popup menu and the selection passed to the Media_ConvertImage function.

Returns:

This function returns a list of available image output formats, including standard image abbreviations. A common response might look like this:

```
BMPf|bmp|BMP
JPEG|jpg|JPEG
jp2 |jp2|JPEG 2000 Image
PNTG|pnt|MacPaint
8BPS|psd|Photoshop
PICT|pct|PICT
PNGf|png|PNG
qtif|qti|QuickTime Image
.SGI|sgi|SGI Image
TPIC|tga|TGA
TIFF|tif|TIFF
```

Examples:

To place the output type list into a global text field named “gTypes”:

Set Field [MyTable::gTypes; Media_ListImageOutputTypes]

To use the output type list as a popup menu in a field, create a text calculation field equal to “Media_ListImageOutputTypes” and then set your selection field to use the calculation field as a value list. You can then call Media_ConvertImage within a script, and reference the selected output type:

Media_SetImageDefaults

(previewQuality ; previewMark)

The Media_SetImageDefaults function allows you to control the preview image quality and whether or not brackets appear in the corners of preview images. You can even choose to bypass the creation of the preview image altogether.

previewQuality - Quality level of the automatically generated preview image, -1 (no preview) to 1024 (maximum quality).

previewMark – “0” or “False” to prevent preview mark brackets; “1” or “True” to restore normal bracket insertion on preview images.

Preview Image Quality

MediaManager always stores image results at maximum quality, but FileMaker Pro cannot display these high-quality images in container fields. In order to display a visual result in container fields, MediaManager normally creates a lower quality preview image that is simultaneously placed in the container field (along with the undisplayed high-quality image).

Once scripts are debugged, you may wish to set the previewQuality parameter to “-1” to reduce execution time – if you don’t need to display a copy of the image in a container field as part of your file’s user interface. When previewQuality is set to -1, FileMaker will display field "Unknown Container Object" as an error message in the container (even though the full quality MM Image is still in the container).

Bracket Marks

To distinguish preview images from the high-quality image, MediaManager normally adds bracket marks to the corners of the preview image. To prevent marking of the previews, set the previewMark parameter to “False”. While MediaManager is is demo mode, preview marking cannot be turned off.

Demo Note: Bracket marks cannot be turned off when MediaManager is in demo mode. You can only turn off bracket mark creation when you have registered your copy of MediaManager.

Examples:

To set the default so a preview image of medium quality is created (with bracket marks):

```
Set Field [ MyTable::Response ; Media_SetImageDefaults ( 512 ; True ) ]
```

To set the default so a preview image of maximum quality is created (with bracket marks):

```
Set Field [ MyTable::Response ; Media_SetImageDefaults ( 1024 ; True ) ]
```

To set the default so no preview image is created:

```
Set Field [ MyTable::Response ; Media_SetImageDefaults ( -1 ; True ) ]
```

To set the default so a preview image of maximum quality is created without bracket marks:

```
Set Field [ MyTable::Response ; Media_SetImageDefaults ( 1024 ; False ) ]
```

File Management Functions

Media_CopyItem

(source ; dest ; replaceFlag)

Media_CopyItem copies a file or folder to a new location.

source - the path or container reference to the folder or file to be copied
dest – the path or container reference to either:

- a folder. The source item will be copied to that folder using its source name.
- a file name. The source will be copied to destination folder as the specified name.

replaceFlag - when “1” or “True”, an item in dest with the same name will be replaced. When “0” or “False”, an error will be generated if copying would replace an existing item.

Caution: Items replaced by this function when replaceFlag is 1 cannot be recovered. Use 0 for the replaceFlag parameter unless you wish to allow this.

Examples:

To copy a file named “Contacts.doc” from the desktop to the root folder:

Set Field [MyTable::Response ; Media_CopyItem (“.D/Contacts.doc” ; “.R” ; 0)]

To copy a file named “Contacts.doc” from the desktop to the root folder, replacing any other file with the same name that might already exist in the root folder:

Set Field [MyTable::Response; Media_CopyItem (“.D/Contacts.doc” ; “.R” ; 1)]

To copy a file named “loka.fmp12” from the default folder to a folder named “My Files” on the desktop:

Set Field [MyTable::Response; Media_CopyItem (“loka.fmp12” ; “.D/My Files” ; 0)]

To copy a file named “fred.gif” from the default folder to the root folder while renaming it to “barney.gif”:

Set Field [MyTable::Response; Media_CopyItem (“fred.gif” ; “.R/barney.gif” ; 0)]

Media_CreateAlias

(targetItem ; alias)

Media_CreateAlias creates an alias (on Mac) or shortcut (on Windows) that points to the target item.

targetItem - path to the alias target. The target must exist.
alias - path to an alias name. This must not exist.

Examples:

To create an alias named “movie alias.mov” on the desktop that points to a file named “cartwheel.mov” in the default folder:

Set Field [MyTable::Response ; Media_CreateAlias (“cartwheel.mov” ; “.D/movie alias.mov”)]

To create an alias named “Start.fmp12” in the default folder that points to a file named “Master.fmp12” in the application folder:

Set Field [MyTable::Response ; Media_CreateAlias (“.A/Master.fmp12” ; “Start.fmp12”)]

Media_CreateFolder

(enclosingFolder ; folderName)

Media_CreateFolder creates a new folder in the location specified.

enclosingFolder - path to the folder inside which to create the new folder.
folderName - the name of the new folder (Unicode aware).

Examples:

To create a folder named “Pictures” in the default folder:

Set Field [MyTable::Response ; Media_CreateFolder (“” ; “Pictures”)]

To create a folder named “Backups” in the Temp folder:

Set Field [MyTable::Response ; Media_CreateFolder (".T" ; "Backups")]

To create a folder named "New User Folder" in the Macintosh Users directory:

Set Field [MyTable::Response ; Media_CreateFolder ("Macintosh HD/Users/" ; "New User Folder")]

Media_DeleteItem

(item ; folderFlag)

Media_DeleteItem permanently deletes a file or folder.

item - the path to the folder or file to be deleted. All the contents of a folder will be deleted.
folderFlag - when "1" or "True", folders can be deleted. When "0" or "False", an error will be generated if you try to delete a folder – in which case only a file can be deleted.

Caution: Items deleted with this function cannot be recovered. To protect against accidentally deleting folders, set the folderFlag parameter to 0. If you want to have the option of recovering deleted items, you may wish to consider using Media_MoveItem to move items to the Trash folder instead.

Examples:

To delete a file named "OldShoe.doc" from the default folder:

Set Field [MyTable::Response ; Media_DeleteItem ("OldShoe.doc" ; False)]

To delete a folder named "Little Feet" from the default folder:

Set Field [MyTable::Response ; Media_DeleteItem ("Little Feet" ; True)]

Media_ExportField

(field ; filepath ; replace)

Exports field contents as an external file.

field - the field containing the image, sound, file, text, or reference to export.
filepath - a full file path, or partial file path that picks up with the default path. If the source is a reference, then the destination can be a path to a folder; otherwise, it must be a path to a file.
replace - use “True” or “1” if you want to replace an existing file with the same name. If “False” or “0”, an error will be returned if the target file already exists.

Media_ExportField is a universal function for exporting any type of container or text field to a file.

File Name Extension

In most cases you do not need to know or specify the file extension in advance.

If the destination extension is not specified in the filename portion of filepath parameter, MediaManager will use the following logic:

- If the contents of a text field are being exported, a text file (.txt) will be created.
- If the source is a reference within a container, the extension of source file is used.
- If the source is stored directly in a container field and is a recognized data type, the appropriate extension is used.
- If the source is not a recognized type, no extension is placed on the destination file name.

If the destination extension is specified, it is used.

If the source is a container reference to another file, you can specify a folder destination without a file name. In that case the referenced file is copied to the destination folder using its original name.

Exported Image Format

When exporting images that have been manipulated by other functions resulting in MMIM format (see the Displayed Image Formats section), convert the image to a standard image type before exporting. MediaManager will give an error if you attempt to export from a container field that holds only an MMIM image.

Container Field Text Note

After using the "Insert Picture" (or QuickTime) command by reference, the container field is temporarily of type text immediately afterwards. Media_ExportField will export the text (a file path) rather than the referenced file itself. If you manually click out of the container filed or use the Commit Record script step, the container field contents are then converted to the correct type.

Returns:

Empty string when no error is detected; otherwise error text is returned to the response field.

Examples:

To export a JPEG image from a container field named “Picture 1” to the desktop with a file name of “exported image.jpg”:

Set Field [MyTable::Response ; Media_ExportField (MyTable::Picture 1 ; “.D/exported image.jpg” ; False)]

To export the image from a container field named “Picture 1” to a folder named “images” located in the current default folder:

Set Field [MyTable::Response ; Media_ExportField (MyTable::Picture 1 ; “images/exported image.jpg” ; False)]

To export an image from a container field named “Image” to the current default folder and using the contents of a field named “Image Title” as the file name:

Set Field [MyTable::Response ; Media_ExportField (MyTable::Image ; MyTable::Image Title ; False)]

To allow the user to select a destination, and then export an image using the contents of a field named “Image Title” as the file name:

Set Field [MyTable::Response ; Media_SetDefaultFolder (“.U:Where do you want to export the image to?”)]
Set Field [MyTable::Response ; Media_ExportField (MyTable::Image ; MyTable::Image Title ; False)]

To export an image, setting the replace parameter so the exported file will replace any file with the same name in the destination location:

Set Field [MyTable::Response ; Media_ExportField (MyTable::Image ; MyTable::Image Title ; True)]

To export a sound from a container field named “Sound Clip” to the current default folder and using the contents of a field named “Sound Name” as the file name:

```
Set Field [ MyTable::Response ; Media_ExportField ( MyTable::Sound Clip ;
MyTable::Sound Name ; False ) ]
```

To export text from a text field named “Notes” to the current default folder and using the contents of a field named “Description” as the file name:

```
Set Field [ MyTable::Response ; Media_ExportField ( MyTable::Notes ;
MyTable::Description ; False ) ]
```

To copy an image from the Web directly to disk, without using a container field:

```
Set Field [ MyTable::Response ; Media_ExportField ( Media_InsertImage (
“http://weather.yahoo.com/images/hisat_440x297.jpg” ; False ) ;
“.D/webimage.jpg” ; False ) ) ]
```

Media_ExtractInfo

(infoList ; retrievalKey)

Media_ExtractInfo allows you to easily extract specific pieces of information from an information list of any format.

infoList - the information list, a text field or expression

retrievalKey – either the retrieval key or the line number of the item to be returned (see below)

This function extracts the value of a field in from an information list, such as the values returned by Media_GetImageInfo, Media_GetSoundInfo, Media_GetItemInfo, Media_ListEffects, Media_ListFolder, etc.

Media_ExtractInfo works with both simple lists and key-value pairs.

Simple Lists

Simple lists are just lists separated by carriage returns. Here’s an example of a simple list:

Apples
Oranges
Lychees
Papayas
Fruit Salad

If the above list is contained in a field named “Fruit Options,” you can easily retrieve “Oranges” – the second value – by passing a retrieval key value of 2:

```
Set Field [ MyTable::Response ; Media_ExtractInfo ( MyTable::Fruit Options ; 2 ) ]
```

Key – Value Pairs (Basic)

Media_ExtractInfo can also work with key and value pairs. Simple example would be:

Height: 6 ft. 2 in.
Weight: 155 lbs

Suppose in this example you want to get the weight value. Instead of using a retrieval key of “2” (which would return the entire second line), you can send a retrieval key of “Weight” to get the response “155 lbs”:

```
Set Field [ MyTable::Response ; Media_ExtractInfo ( MyTable::Size ; “Weight” ) ]
```

This is especially useful to extract data from Media_GetImageInfo function’s results.

Key – Value Pairs (Hierarchical Structures)

In addition, Media_ExtractInfo can also handle more complex hierarchical structures. For example, the Media_GetSoundInfo function returns an indented hierarchical response, such as:

```
Format: AIFF
Track 1
  Volume: 256
  Duration (s): 000:00:01.207
  Creation Date (ts): 22-08-2005 12:39:43
  Modification Date (ts): 22-08-2005 12:39:43
  Media
    Media Duration (s): 000:00:01.207
    Media Creation Date (ts): 22-08-2005 12:39:43
    Media Modification Date (ts): 22-08-2005 12:39:43
    Media Language: 0
    Media Quality: 0
      Sample Description 1
      Sample Format: 74776F73
      Number of Channels: 2
      Sample Size (b): 16
      Sample Rate (Hz): 44100
```

If you just want to know that the format is “AIFF”, all you need to do is send a basic retrieval key:

Set Field [MyTable::Response ; Media_ExtractInfo (MyTable::Sound Info ; “Format”)]

But suppose you want to find out the volume for Track 1. In that case, you would need a hierarchical retrieval key of “Track 1/Volume”:

Set Field [MyTable::Response ; Media_ExtractInfo (MyTable::Sound Info ; “Track 1/Volume”)]

This approach works the same even for deeper levels of the hierarchy. For example, to get the number of channels (whether it is a mono or stereo sound file), you would use “Track 1/Media/Sample Description 1/Number of Channels” for your retrieval key.

Units Designator

In some cases, the key in the info list may include a units or type designator, such as “(s)” for seconds, etc. Media_ExtractInfo does not require you to list the designator as part of your retrieval key. For example, in the above sound info example, you can use—

Set Field [MyTable::Response ; Media_ExtractInfo (MyTable::Sound Info ; “Track 1/Duration”)]

--to get the length of the sound clip. You do not need to use “Track 1/Duration (s)” as the retrieval key.

List Count

If you want Media_ExtractInfo to give you a count of the number of items in the list, just leave the retrieval key parameter empty. For example:

Set Field [MyTable::Response ; Media_ExtractInfo (MyTable::Sound Info ; “”)]

Non-Carriage Return Separators

Many information lists, of course, do not use carriage returns as separators. They may instead be separated by commas, pipe characters, etc. In order to allow Media_ExtractInfo to work with these lists, all you have to do is use FileMaker Pro's Substitute function to substitute a carriage return for the separator character.

For example, suppose you have a field named Items that contains a list separated by commas, and you want to extract the third item. You can use the Substitute function like this:

```
Set Field [ MyTable::Response ; Media_ExtractInfo ( Substitute ( MyTable::Items
; "," ; "¶" ) ; 3 ) ]
```

Returns:

Media_ExtractInfo returns the value associated with the key or index. If the value represents a time, date, or timestamp, the appropriate FileMaker type is returned. Otherwise, text is returned.

Media_GetDefaultFolder

Media_GetDefaultFolder returns the path to the default folder.

No parameters are used.

See the documentation on Media_SetDefaultFolder for an explanation of the default folder.

Example:

To store the default folder file path in a text field named "Path":

```
Set Field [ MyTable::Path ; Media_GetDefaultFolder ]
```

Media_GetItemInfo

(item)

Media_GetItemInfo gives you system information about an external file, returning information such as creation, modification, type, extension, and size.

item - path to the item (file or folder)

Use Media_ExtractInfo to extract individual values from the response.

Returns:

This function returns text containing a list of information about the item. For example, a response for an MPEG-4 audio file might look like this:

```
Created: 13-12-2005 22:25:23
Modified: 13-12-2005 22:26:14
Invisible: 0
Directory: 0
Data size (b): 473264
Resource size (b): 286
Locked: 0
Alias: 0
Type: mpg4
Creator: TVOD
```

Examples:

To check the information on a file on the desktop named “wolf.jpg”:

```
Set Field [ MyTable::Response ; Media_GetItemInfo ( “.D/wolf.jpg” ) ]
```

To isolate just the file size into a field named Size, you can use this function in conjunction with Media_ExtractInfo:

```
Set Field [ MyTable::Size ; Media_ExtractInfo ( Media_GetItemInfo ( “.D/wolf.jpg”
) ; “Data size” ) ]
```

Media_GetMouseUp

Returns the pointer coordinates when the mouse button was last clicked. Set an entire layout or container field as a single button and know precisely where the user has clicked.

Media_GetMouseUp returns the mouse pointer's horizontal and vertical coordinates from the last 'mouse up' -- that is, the last time the mouse button was clicked.

The response of Media_GetMouseUp contains four values separated by pipe characters "|". The response might look like "20|45|221|300". The first two values are the horizontal and vertical coordinates relative to the front window's upper left corner. The 3rd and 4th values are relative to the bottom right corner.

So, if the response is:

> 20|45|221|300
>
> 20 = Horizontal, from upper left corner
> 45 = Vertical, from upper left corner
> 221 = Horizontal, from bottom right corner
> 300 = Vertical, from bottom right corner

If the mouse has not yet been clicked, Media_GetMouseUp returns "-1|-1|-1|-1".

You can also use Media_ExtractInfo to easily get a specific value from the returned value list.

OS X Note:
On Mac OS X, the coordinates returned by Media_GetMouseUp are the current mouse pointer's position, not the position when the mouse button was last clicked. In normal usage (when the mouse click triggers the script that contains the Media_GetMouseUp function), the current mouse position is the same or nearly the same as the last clicked position. If you have several script steps before Media_GetMouseUp, however, and if the user is still moving the mouse, then a discrepancy can occur. For this reason, Media_GetMouseUp should be in the first script step.

Parameters:

No parameter is used. Use empty quotes "".

Example:

To get the pointer coordinates the last time the mouse was clicked--

Set Field[Table::Response Field ; Media_GetMouseUp]

Media_GetName

(container)

Returns the name of an item or container reference.

container - either a container reference, or a file or folder path.

Returns:

The item name at the end of the path, a file or folder name. For example, if a container refers to "/Macintosh HD/Users/username/Desktop/Pictures/Ghost.jpg", then Media_GetName will return "Ghost.jpg".

Media_GetPath

(container [; parent])

Returns the full path to an item or image stored in a container by reference.

container - either a container reference, or a file or folder path.

parent - Optional argument. If blank or omitted, the full path is returned. If a number, the full path to the parent directory at that level (see below).

For example, let's assume we have an image stored by reference in a container field named Image, and the path refers to "/Macintosh HD/Users/Fred/Desktop/Pictures/book.gif".

If we want the full path returned, we can use:

Set Field [MyTable::Response ; Media_GetPath (MyTable::Image)]

If we just want the enclosing folder without the file name included, we can set the parent parameter to ‘1”:

Set Field [MyTable::Response ; Media_GetPath (MyTable::Image ; 1)]

If we just want to know which user directory it is in, we can set the parent parameter to “3”:

Set Field [MyTable::Response ; Media_GetPath (MyTable::Image ; 3)]

Returns:
The path to the specified file or folder (down to the specified parent directory level).

Media_InsertFile

(filepath ; byReference ; asText)

Inserts a file directly or by reference into a container field. You can also insert a file’s text into a text field.

filepath - a full file path, or partial file path that picks up with the default path. If the source is a reference, then the destination can be a path to a folder; otherwise, it must be a path to a file.
The filepath parameter can be:

- file path
- text field with a file path
- file URL

byReference – “1” or “True” to store by reference; “0” or “False” to store in container field

asText – “1” or “True” to insert only the file’s text into a text field; “0” or “False” to insert the entire file into a container field.

Media_InsertFile gives you the ability to quickly and easily insert files of any type (even other FMP databases) into a FileMaker container. As with an image, the inserted file can be stored directly in the container or as a reference.

This function can also insert the contents of text files directly into text fields

The filepath parameter can also be a URL, giving you the ability to insert the text web pages, giving you the ability to store and catalog the HTML of favorite web pages. And, if you want, you can then use FileMaker's built-in text parsing functions to isolate specific information – image or link URLs, new posting dates, updated stock information, etc.

For example, if you want to grab the HTML text of a file named "new.htm" located in the http://www.MySite.com/ directory, you can use a script step:

```
Set Field [ MyTable::HTML Text ; Media_InsertFile (
"http://www.MySite.com/new.htm" ; False ; True )
```

(View the demo files for in-depth examples of how to do this.)

Examples:

To insert "customerltr.doc" into a container field named Container using a hardcoded filepath:

```
Set Field [ MyTable::Container ; Media_InsertFile ( "/c:/documents
/customerltr.doc" ; False ; False ) ]
```

To insert a file named "tester.fmp12" into a container field by reference named Example:

```
Set Field [ MyTable::Example ; Media_InsertFile ( "tester.fmp12" ; True ; False ) ]
```

To insert the text of a file named "notes012606.txt" into a text field named Notes:

```
Set Field [ MyTable::Notes ; Media_InsertFile ( "notes012606.txt" ; False ; True ) ]
```

To allow the user to select a file to be inserted by reference:

```
Set Field [ MyTable::FileName ; Media_SetDefaultFolder ( ".V:Select the file to
insert" ) ]
Set Field [ MyTable::Container ; Media_InsertFile ( MyTable::FileName ; True ;
False ) ]
```

Media_ListDisks

Media_ListDisks returns a list of all available mounted disks.

No parameters are used.

You can use Media_ExtractInfo to quickly extract disk names from the list.

Returns:

All mounted volumes separated by the pipe character “|”.

For example, a Mac response might be:
Macintosh HD|Network

A Windows response might be:
A:\|C:\|D:\|E:\

Example:

To get a list of all mounted disks:

```
Set Field [ MyTable::Response ; Media_ListDisks ]
```

Media_ListFolder

(folder ; listAll)

Media_ListFolder gives you a list of the files and folders in the specified folder.

folder- path to the folder to be listed.

listAll - If “1” or “True”, the list includes invisible files and folders

You can use Media_ExtractInfo to retrieve individual items from the list of files and folders.

Folders and Special Characters

When the listAll parameter is set to “1” or “True”, folder names are preceded by the “ƒ” character. Invisible files are often prefixed with a period “.”

Examples:

To get a list of all files in the default folder:

Set Field [MyTable::Response ; Media_ListFolder (“” ; False)]

To list all files and folders in the Applications folder:

Set Field [MyTable::Response ; Media_ListFolder (“.A” ; True)]

Media_MoveItem

(source ; dest)

Moves a file or folder to a new location.

source - the path to the folder or file to be copied
dest –a path to a folder. The source item will be copied to that folder using its source name.

The source and dest parameters must be on the same volume.

Unlike Media_CopyItem, Media_MoveItem will not replace an existing item with the same name in the destination folder.

Examples:

To move a file named “Tree Houses.doc” from the desktop to the root folder:

Set Field [MyTable::Response ; Media_MoveItem (“.D/Tree Houses.doc” ; “.R”)]

To move a file named “logo_thumb.jpg” from the default folder to a folder named “Images” on the desktop:

Set Field [MyTable::Response ; Media_MoveItem (“logo_thumb.jpg” ; “.D/Images”)]

Media_OpenItem

(item)

Media_OpenItem opens the file, application, or folder specified. Documents and applications are launched; folders are opened in the Finder.

item – path or container reference to the item to be opened.

Examples:

To open a PDF file named “Help File.pdf” in the default folder:

Set Field [MyTable::Response ; Media_OpenItem (“Help File.pdf”)]

To open a Word document named “Correspondence.doc” in a folder named “MySolution” on the desktop:

Set Field [MyTable::Response ; Media_OpenItem (“.D/MySolution/Correspondence.doc”)]

Media_RenameItem

(item ; newName)

Media_RenameItem renames a file or folder.

item - path to the item
newName - the new name (Unicode aware).

An item cannot be renamed to the same name as an existing item in the same folder.

Examples:

To rename a file named “Contacts.fmp12” (in the default folder) to “Contacts 2.fmp12”:

Set Field [MyTable::Response ; Media_RenameItem (“Contacts.fmp12” ; “Contacts 2.fmp12”)]

To rename a file named “Seville1.mp3” on the desktop to “Sevilla.mp3”:

```
Set Field [ MyTable::Response ; Media_RenameItem ( “.D/Seville1.mp3” ;
“Sevilla.mp3” ) ]
```

Media_SetDefaultFolder

(path)

Sets the default folder for all MediaManager functions that depend upon a file path.

path - any absolute path or path relative to the current default folder. See also special values below.

<u>Special Values for Path</u>

.D	- Desktop
.A	- the Application folder
.F	- The folder containing the currently active FileMaker Pro file (Mac only)
.U:prompt	- user selects a folder from the folder dialog
.V:prompt	- user selects a file from the file dialog. Returns filename, sets default folder to the file's enclosing folder.
.T	- Temp folder
.R	- The root folder of the boot disk
.P	- Preferences folder
.S	- System folder
.O	- Documents folder (Mac only)
.X	- Trash folder

The above codes can be used anywhere a file path is expected. Special values can be combined with relative paths. For example, ".D/foobar/image.jpg" refers to the file image.jpg inside the images folder on the desktop.

At startup, the default folder is set to the root disk.

Path Syntax

All paths to files and folders used by MediaManager follow the conventions of FileMaker Pro. An absolute path starts with slash; relative paths do not. A path to a folder ends in slash; a path to a file ends with the file name. Here are some examples:

An absolute path to a user directory named “Frank” on the Mac:

/Macintosh HD/Users/Frank/

An absolute path to a user directory named “Frank” on Windows:
/c:/Documents and Settings/Frank/

A relative path to a file named “MyFile.txt” located on the desktop:
.D/MyFile.txt

The Default Folder

MediaManager uses the convention of the "default folder". This path is set using Media_SetDefaultFolder and is automatically prepended to every file or folder path specified in MediaManager functions. In other words, you can set the default path to any folder, and then simply set all paths to be relative to that default location.

For example, suppose the default path is set to the Desktop folder, with either one of the following:

```
Media_SetDefaultFolder ( “/Macintosh HD/Users/Frank/Desktop/” )
Media_SetDefaultFolder ( “.D” )
```

If you then use a MediaManager function like Media_ExportField, you can simply list the file name in the path parameter, and the file will be exported to the desktop (the default folder):

```
Set Field [ MyTable::Response ; Media_ExportField ( MyTable::Picture 1 ;
“image.jpg” ; False ) ]
```

Similarly, if you want to specify a folder within the default folder, you only need to list the folder and file name, and MediaManager understands that you are specifying a location relative to the default folder (again, the desktop):

```
Set Field [ MyTable::Response ; Media_ExportField ( MyTable::Picture 1 ;
“MyPictures/image.jpg” ; False ) ]
```

Of course, you can specify a full path at any time and the default path is ignored.

Other Special Notations

../	- Refers to the enclosing or “parent” folder. It is the equivalent of going “up one level” in the current default path.

For example, if you set the default path is set to a user directory like this--

Media_SetDefaultFolder (“/c:/Documents and Settings/Frank/”)

--you can then specify the Documents and Settings directory with “../”, such as in a Media_ExportField function like:

Set Field [MyTable::Response ; Media_ExportField (MyTable::Picture 1 ; “../image.jpg” ; False)]

Returns:

An empty string if successful. Otherwise, returns an error message.

With either of the user prompt options (.U and .V), if a user clicks on the Cancel button, MediaManager returns “$0:Cancel”.

Examples:

To set the default folder to the desktop:

Set Field [MyTable:Response ; Media_SetDefaultFolder (“.D”)]

To set the default folder to the boot disk:

Set Field [MyTable:Response ; Media_SetDefaultFolder (“.R”)]

To allow the user to select the default folder with the message “Select a default folder”:

Set Field [MyTable:Response ; Media_SetDefaultFolder (“.U:Select a default folder”)]

Sound Functions

Demo Note

When modifying sounds with an unregistered version of MediaManager, a series of tones will be mixed with the resulting sound. This behavior disappears when you register MediaManager.

Important: SecureFM Solves the QuickTime 7/FileMaker Pro Bug on Windows

There is a significant bug in how QuickTime 7 interacts with FileMaker Pro on Windows. **If you try to manually insert QuickTime sound files into a FileMaker Pro container field on Windows, FileMaker Pro will unexpectedly quit.**

MediaManager gives you a safe workaround with the Menu_InsertQuickTime function, allowing you to insert sound references on Windows as well as Mac.

Sound Formats

The sound formats available through functions like Media_ConvertSound vary based on the version of QuickTime and additional sound codecs you may have installed on your computer. Common sound formats include:

MPEG-4 (".mp4", ".m4a")

- Works in iTunes.
- Good file size compression.
- MPEG-4 is a compression format for both audio and video files. For audio only files, best to give file names the extension ".m4a". (Video files are often named with ".m4v" or the generic ".mp4".)
- On Windows, you need an AAC license to convert to MPEG-4 format. (See below for information on how to do this.) On Mac, you can convert to MPEG-4 without a special license.

MPEG-3 (".mp3")

- Widely used format.

- QuickTime cannot normally convert to .mp3 format unless you have installed a special MPEG-3 codec. (See http://en.wikipedia.org/wiki/LAME for explanation and links.)

WAVE (".wav)

- Popular format on Windows machines.

AIFF (".aif")

- Popular format on Macs.
- Larger file sizes, but good sound quality.

AAC License and MPEG-4 Sound Files

Most sound format codecs, like WAV and AIFF, are automatically loaded, registered, and available through QuickTime. The AAC encoder format includes MPEG-4 audio and video (".mp4", ".m4a", etc.) files. On Windows, AAC/MPEG-4 encoding requires a special registration.

If you want to use MediaManager to convert sound files to MPEG-4 format on Windows, you must first register the AAC encoder with the Media_RegisterLicensedEncoder function. You can purchase an inexpensive AAC license from New Millennium <http://www.newmillennium.com>.

If you don't properly register the AAC encoder with this function, you may still see MPEG-4 listed by the Media_SoundOutputFormats function, but you will not be able to convert to MPEG-4. Macintosh users can convert to AAC/MPEG-4 format without an AAC license.

Media_Amplify

(container; factor {; limit })

Media_Amplify amplifies the audio waveform of a sound file, adjusting its volume.

container – location of the sound
factor – (any number) the amplification factor that multiplies the audio waveform, changing its playback volume
limit – the peak or limit to the greatest amplification of the audio waveform

The result must be returned into a container field.

Essentially, Media_Amplify multiplies the audio waveform (factor) without exceeding the limit value.

Setting the factor to “2” will double the amplitude, for example. In other words, it multiplies every point along the audio wave by two.

If you also set the limit to “1”, for example, after amplification no point on the waveform will exceed the highest point of the original wave form.

A good way to become more familiar with this function is to run the function with several values, and follow by calling the Media_DrawWaveform function to get a visual representation of the changes.

Use Media_GetLastError to check for any errors of this function.

Media_AppendSound

(container1 ; container2)

Media_AppendSound attaches one sound to the end of another sound, resulting in a single long sound clip.

container1 – location of the initial sound
container2 – location of the sound to be appended

The result must be returned to a container field.

Returns:

The result is the first specified sound followed by the second sound.

In the case of an error, nothing is returned to the container field. Use Media_GetLastError to check for any errors of this function.

Example:

To add an introductory sound clip to your interview audio file:

Set Field [MyTable::Podcast Complete; Media_AppendSound (MyTable::Intro Clip ; MyTable::Interview)]

Media_ConvertSound

(container ; filename ; soundType ; {parameters})

Media_ConvertSound converts a sound file to different sound format.

container – location of the sound to be converted
filename – output file name
soundType – output sound file format (hex code from Media_ListSoundOutputFormats)
{parameters} – optional hexadecimal convert settings

The available list of sound formats can be obtained using the Media_ListSoundOutputFormats function. Media_ConvertSound can use either the full line description returned by Media_ListSoundOutputFormats or simply the 8 digit hex code at the beginning of the line.

The optional format settings parameter is a hexadecimal value that is obtained using the Media_SoundFormatDialog function. If this parameter is left empty "" the user is shown a format settings dialog.

Important: Extra "ref.mov" File

When MediaManager converts a sound file, it produces two sound files as a result. First is the sound file itself in the converted format. A second file is also created with "ref.mov" added to the end of the file name. For example, if you are outputting a file named "sound1.m4a", a second file named "sound1_ref.mov" will also be created. This second file is necessary to resolve a problem in how FileMaker Pro handles QuickTime sound files.

- The first sound file is the actual converted sound file. This file can be played in other sound applications, like iTunes or QuickTime Player. It does not require the "ref.mov" to work on its own. It has no direct connection to the converted result referenced in your FileMaker Pro file.

- The "ref.mov" file is the file that is inserted into your result container field in your FileMaker Pro solution. If you delete or move this file, the sound will no longer be referenced in your result container field.

AAC License and MPEG-4 Sound Files

Most sound format codecs, like WAV and AIFF, are automatically loaded, registered, and available through QuickTime. The AAC encoder format includes MPEG-4 audio and video (".mp4", ".m4a", etc.) files. On Windows, AAC/MPEG-4 encoding requires a special registration. You can purchase an inexpensive AAC license from New Millennium <http://www.newmillennium.com>. Macintosh users can convert to AAC/MPEG-4 format without an AAC license.

MP3 Audio Format
Beginning with version 8.2, MediaManger is able to handle MP3 audio files. If you want to convert to MP3 format, however, you must first install a third party MP3 encoder, such as LAME (see http://en.wikipedia.org/wiki/LAME for explanation and links). Once you've installed the MP3 encoder and rebooted your system, you should see MP3 as one of the options listed by the Media_ListSoundOutputFormats function and incorporate it into the Media_ConvertSound function.

Examples:

To convert a sound to WAV format, giving the user the option to set the convert settings when the script runs:

Set Field [MyTable::Sound Converted ; Media_ConvertSound (MyTable::Sound Original ; "soundConverted.wav" ; "57415645" ; "")]

To convert a sound using the type format stored in a field (which uses the result of a Media_ListSoundOutputFormats calc as the value list):

Set Field [MyTable::Sound Converted ; Media_ConvertSound (MyTable::Sound Original ; "soundConverted" ; MyTable::Sound Formats ; "")]

To convert a sound to WAV format, with the optional settings preset with Media_SoundFormatDialog in a preference field:

Set Field [MyTable::Sound Converted ; Media_ConvertSound (MyTable::Sound Original ; "soundConverted.wav" ; "57415645" ; MyTable::WAV Settings)]

Media_CreateSound

(type ; {duration ; {frequency}})

Media_CreateSound allows you to create a tone, or white noise, or silence for a specified duration.

type – created sound wave format
sine

square
sawtooth
silence
noise
sin (same as sine)
saw (same as triangle)
rectangle (same as square)
none (same as silence)

duration – created sound duration in seconds
frequency – created sound wave frequency in Hz

The function returns a sound to a container field.

Sine, square, and sawtooth types can give you precise tones on the musical scale, as well as various beeps and buzzes, depending on the frequency setting you use.

Silence simply creates an empty sound result for the specified duration.

Noise creates a static-like white noise for the length of time you want.

Media_DrawWaveform

(container; trackID; width; height {; startTime {; endTime {; fgcolor {; bgcolor } } } })

The Media_DrawWaveform function creates a waveform graph of a sound file and places the resulting image in a container field. The waveform image is generated in the colors and dimensions you specify. You can focus the waveform on a specific time segment of the audio file or cover its full length.

container – sound location (container field or file path)
trackID – track of the audio file to be graphed (typically “1”)
width – width of the waveform image to be generated (in pixels)
height – height of the waveform image to be generated (in pixels)
startTime – time in sound file to begin graph (HH:MM:SS.S)
endTime – time in sound file to end graph (HH:MM:SS.S)
fgcolor – foreground color for the sound graph (in hexadecimal)
bgcolor – background color for the sound graph (in hexadecimal)

The result must be returned to a container field.

The resulting image shows two waveform graphs, one above the other. The top waveform is for the left channel, the bottom is the right channel.

Graphing the Full Sound File
If you want to graph the entire sound file, set both the startTime and endTime parameters to “00:00:00”.

Use Media_GetLastError to get possible error of this function.

Examples:

To create a waveform at 500x200 pixels for a an audio file in a field named soundContainer:

Media_DrawWaveform (myTable::soundContainer ; 1; 500 ; 200 ; “00:00:00” ; “00:00:00” ; “#000000” ; “#FFFFFF”)

To get the same waveform for a an external audio file on the desktop named song.mp3:

Media_DrawWaveform (“.d/song.mp3” ; 1; 500 ; 200 ; “00:00:00” ; “00:00:00” ; “#000000” ; “#FFFFFF)

To create a waveform graph covering the first 10 seconds of the sound file::

Media_DrawWaveform (“.d/song.mp3” ; 1; 500 ; 200 ; “00:00:00” ; “00:00:10” ; “#000000” ; “#FFFFFF)

To generate a waveform with a red foreground and light violet background:

Media_DrawWaveform (“.d/song.mp3” ; 1; 500 ; 200 ; “00:00:00” ; “00:00:10” ; “#FF0000” ; “#CCCCFF”)

Media_ExtractSound

(container ; beginTime ; endTime)

Media_ExtractSound extracts a specified segment of a sound.

container – sound location
beginTime – time in original sound to begin extraction (HH:MM:SS.S)

endTime – time in original sound to end extraction (HH:MM:SS.S)

Use Media_GetLastError to get possible error of this function.

Example:

To extract the first 2 minutes of a sound:

Set Field [MyTable::Sound Result; Media_ExtractSound (MyTable::Original Sound; “00:00:00” ; “00:02:00”)]

Media_FadeSound

(container ; startTime ; endTime ; level ; inOrOut)

Media_FadeSound will apply a fade in or out to a specific portion of a sound file.

container – location of the sound file
startTime - time in original sound to begin fade (HH:MM:SS.S)
endTime - time in original sound to end fade (HH:MM:SS.S)
level – final fade level (1 = full sound at start; 0 = no sound at start)
inOrOut – 1 = fade out; 0 = fade in

You can use negative values in both the startTime and endTime parameters to refer to a position from the end of the file. For instance, when startTime is 00:00:00 and endTime is -00:00:00, the entire sound is referenced. To reference the last five seconds for example, use startTime = -00:00:05 and endTime = -00:00:00.

If startTime is invalid, it is set to 00:00:00. If endTime is invalid, it is set to the end of the sound. Level is reset to 0 for fade out and 1.0 for fade in.

Use Media_GetLastError to get possible error of this function.

Examples:

To have a sound fade out beginning at 5 seconds and finishing within 10 seconds:

Set Field [MyTable::Sound Result; Media_FadeSound (MyTable::Original Sound; “00:00:05” ; “00:00:10” ; 1 ; 1)]

To begin fade out 5 seconds from the end of the sound clip:

Set Field [MyTable::Sound Result; Media_FadeSound (MyTable::Original Sound; “-00:00:05” ; “-00:00:00” ; 1 ; 1)]

To fade in for the first 3 seconds of a sound clip:

Set Field [MyTable::Sound Result; Media_FadeSound (MyTable::Original Sound; “00:00:00” ; “00:00:05” ; 0 ; 0)]

Media_Get

(tag)

The Media_Get function allows you to check the version of QuickTime, and whether or not sound recording is taking place.

tag – one of several values:
- “Volume”
- “Tempo”
- “Balance”
- “PlayStatus”
- “recording”
- “quicktime-version”

Volume
Media_Get (“Volume”) returns the overall volume level for all asynchronously playing files.

Tempo
Media_Get (“Tempo”) returns the tempo rate of the first file in the asynchronous playback queue.

Balance
Media_Get (“Balance”) returns a value between -1 and 1 representing the balance between left and right audio channels during playback.

PlayStatus
Media_Get (“PlayStatus”) returns “1” if if anything is in the playback queue; otherwise, “0”.

Recording
Media_Get (“recording”) returns “1” if recording is taking place or “0” when no recording is occurring.

QuickTime Version
Sending Media_Get (“quicktime-version”) will return the QuickTime version currently running as a numerical value. For example, if QuickTime 7.0.3 is active, the value returned is “703”.

Examples:

To get the current volume level:

```
Set Field [MyTable::Volume; Media_Get ( “Volume” ) ]
```

To get the tempo of currently playing sound files:

```
Set Field [MyTable::Tempo; Media_Get ( “Tempo” ) ]
```

To check if anything is playing:

```
Set Field [MyTable::Playing?; Media_Get ( “PlayStatus” ) ]
```

To check if anything is playing:

```
Set Field [MyTable::Playing?; Media_Get ( “PlayStatus” ) ]
```

To check if MediaManager is recording sound:

```
Set Field [MyTable::Recording?; Media_Get ( “recording” ) ]
```

To check the version of QuickTime:

```
Set Field [MyTable::QTversion; Media_Get ( “quicktime-version” ) ]
```

Media_GetMetadata

(containerOrFile {; tag })

Media_GetMetadata lets you view the ID3 metadata stored in an MP3 sound file. You can either view the entire list of ID3 tags, or a specific tag, like “Artist,” etc.

containerOrFile – location of the sound file
tag – optional parameter if you want to view only a specific piece of information

When the optional “tag” parameter is left blank or unused, Media_GetMetadata returns all of the ID3 metadata, such as:

```
Name: Poème, Op. 41, No. 6
Artist: Richard Steiner
Album: Piano Encores: 50 Favourite Pieces (Disc 2)
Year: 1995
Track: 23/25
Disc Number: 2/2
Genre: Classical
```

Extracting Values from the List

To extract the value of a particular tag, use the Media_ExtractInfo function. Note: It is not recommended to parse information list by hand as it may contain encoded characters.

Returning a Single Tag

If you don’t want the entire metadata list and you know the name of a particular value you want, you can specify the tag directly with Media_GetMetadata:

```
Media_GetMetadata( container ; "Artist" )
```

For a sound file with the metadata shown above, MediaManager will respond with “Richard Steiner”.

Follow this function with Media_GetLastError to check for any error responses.

See Media_SetMetadata for information on how to change the metadata tags.

Examples:

To get the full metadata list of an external sound file named “Bop.mp3” and stored on the desktop:

Set Field [MyTable::Metadata List; Media_GetMetadata (“.d/Bop.mp3”)]

To get metadata of a sound file in a container field named SoundContainer:

Set Field [MyTable::Metadata List; Media_GetMetadata (MyTable::SoundContainer)]

To get only the value of the “Genre” tag from an external sound file named “Allegro.mp3” in the current default folder:

Set Field [MyTable::Metadata Genre ; Media_GetMetadata (“Allegro.mp3” ; “Genre”)]

To store the full metadata list in a field and, in a separate field, store the “Name” value:

Set Field [MyTable::Metadata List; Media_GetMetadata (“.d/sounds/pavane.mp3”)]
Set Field [MyTable::Metadata Name ; Media_ExtractInfo (MyTable::Metadata List ; “Name”)]

Media_GetSoundInfo
(container)

Media_GetSoundInfo gives you detailed information about the specified sound file, including format, duration, sample size, sample rate, and bit rate.

container – location of the sound file

To pick out specific pieces of information, use the Media_ExtractInfo function.

Because the information returned by this function is hierarchical and indented, it can be helpful to display the returned information with a font that uses equal character spacing, such as Courier.

Commit Record Note

If Media_GetSoundInfo does not immediately return a value, such as after a sound has just been recorded, you may need to click outside of a field or use FileMaker Pro’s Commit Records/Requests script step.

Returns:

Media_GetSoundInfo returns a hierarchical list of information on the specified sound file, such as:

```
Format: AIFF
Track 1
  Volume: 256
  Duration (s): 000:00:01.207
  Creation Date (ts): 22-08-2005 12:39:43
  Modification Date (ts): 22-08-2005 12:39:43
  Media
    Media Duration (s): 000:00:01.207
    Media Creation Date (ts): 22-08-2005 12:39:43
    Media Modification Date (ts): 22-08-2005 12:39:43
    Media Language: 0
    Media Quality: 0
    Sample Description 1
      Sample Format: 74776F73
      Number of Channels: 2
      Sample Size (b): 16
      Sample Rate (Hz): 44100
      Bit Rate (kbps): 64
```

Examples:

To place all of the sound info in a field named “Sound Data”:

Set Field [MyTable::Sound Data; Media_GetSoundInfo (MyTable::Container)]

To put the sample size and sample rate values into fields named “Size” and “Rate”:

Set Field [MyTable::Size; Media_ExtractInfo (Media_GetSoundInfo (MyTable::Container) ; “Track 1/Media/Sample Description 1/Sample Size”)]
Set Field [MyTable::Rate; Media_ExtractInfo (Media_GetSoundInfo (MyTable::Container) ; “Track 1/Media/Sample Description 1/Sample Rate”)]

To isolate the bit rate:

Set Field [MyTable::Rate ; Media_ExtractInfo (Media_GetSoundInfo (MyTable::Container) ; “Track 1/Media/Sample Description 1/Bit Rate”)]

Media_InsertQuickTime

(path)

Media_InsertQuickTime inserts a sound file or other QuickTime element into a container field.

path – the file path or URL for the sound or QuickTime file to be inserted

You can also use a Web URL for the path. In that case, the file is downloaded and a reference to it is returned.

Important: SecureFM Solves the QuickTime 7/FileMaker Pro Bug on Windows

There is a significant bug in how QuickTime 7 interacts with FileMaker Pro on Windows. **If you try to insert QuickTime objects (sound files, movies, etc.) into a FileMaker Pro container field on Windows, FileMaker Pro will unexpectedly quit.** This occurs whether you insert the QT objects manually. ***The Menu_InsertQuickTime function solves that problem.***

“This File Has No Preview” Note on Windows

When you use Media_InsertQuickTime to insert a sound file reference on Windows, a text note will appear in the container field that reads “This File Has No Preview.” **This does not indicate an error with inserting the sound reference.** The text is part of how MediaManager resolves the QuickTime bug on Windows. The normal interaction between QuickTime and FileMaker Pro on Windows does not produce the required “preview” for the container field, so MediaManager performs this task for them.

Important: QuickTime References

Inserted sounds are only temporary references and will disappear when the computer is shut down. To make the reference permanent, convert/export the sound.

Even then, be aware that FileMaker Pro only stores a reference to the external file. **Any time you move the external file in relationship to your FMP database, the reference will be lost and the sound or movie will no longer play.**

Inserting Remote Sound Files

MediaManager uses a workaround that sidesteps a bug in QuickTime on Windows that causes a FileMaker Pro crash when inserting sound files from a remote volume.

See the Media_Set function for more information on controlling the workaround and its warning dialog.

Remote File Warning Dialog

The first time (after launching FileMaker Pro) that you call Media_InsertQuickTime to insert a sound file from a remote volume, the following dialog will be displayed.

"MediaManager's Media_InsertQuicktime is about to us a workaround for a bug in QuickTime while accessing remote media. This workaround may cause the FileMaker script to fail. Please see the documentation for Media_InsertQuickTime for complete information."

You can use the Media_Set function if you wish to prevent the dialog from appearing.

Examples:

To insert a sound file named "flute.wav" located in the default folder:

```
Set Field [ MyTable::Container ; Media_InsertQuickTime ( "flute.wav" ) ]
```

To insert "drums.m4a" located on the desktop:

```
Set Field [ MyTable::Container ; Media_InsertQuickTime ( ".D/drums.m4a" ) ]
```

To allow users to select a sound file to be inserted:

```
Set Field [ MyTable::FileName ; Media_SetDefaultFolder ( ".U:Select a sound file" ) ]
Set Field [ MyTable::Container ; Media_InsertQuickTime ( MyTable::FileName ) ]
```

To make the inserted sound reference stable (so it won't be lost on shut down) by converting and outputting it as a WAV file:

```
Set Field [ MyTable::FileName ; Media_SetDefaultFolder ( ".U:Select a sound file" ) ]
Set Field [ MyTable::Sound Original ; Media_InsertQuickTime (
MyTable::FileName ) ]
Set Field [ MyTable::Sound Converted ; Media_ConvertSound ( MyTable::Sound
Original ; “soundConverted.wav” ;  “57415645” ; MyTable::WAV Settings ) ]
```

Media_ListSoundOutputFormats

Media_ListSoundOutputFormats gives you a list of the available output formats.

No parameter is used.

If you are giving users the option to select from all possible output formats, a useful technique is to create two fields: a calculation text field equal to this function, and another text field to be used as the selection field. Set the selection field to display a popup menu value list that is generated by the calculation field.

Only the first part of the format (8 digit hex code) is required by Media_ConvertSound or Media_SoundFormatDialog, though the entire line can be used, as well.

Returns:

A list of all installed sound output format codecs, such as:

```
41494646|AIFF|AIFF|Exports into an AIFF sound file
4D6F6F56|MooV|QuickTime Movie|Exports file into a QuickTime
Movie
554C4157|ULAW|AU|Exports into an AU sound file
56665720|VfW |AVI|Exports file into an AVI file
57415645|WAVE|Wave|Exports into a WAV sound file
33677070|3gpp|3G|Exports file to a 3G file
6D706734|mpg4|MPEG-4|Exports file into a MP4 Movie
```

Media_MixSounds

(container1 ; container2)

Media_MixSounds mixes the contents of two sound files so they play simultaneously.

Container1 – location of sound 1
Container2 - location of sound 2

The result must be returned to a container field.

If you want to mix two sounds together but with one of the sounds starting later than the other, you can first add a length of silence to the later sound (using Media_CreateSound and Media_AppendSound) before calling Media_MixSounds.

Use Media_GetLastError to get possible error of this function.

Returns:

The result is first sound and the second sound overlapping each other.

In the case of an error, nothing is returned to the container field.

Example:

To mix Sound A with Sound B:

```
Set Field [ MyTable::Sound Result ; Media_MixSounds ( MyTable::Sound A ; MyTable Sound B ) ]
```

Media_Normalize

(container)

Media_Normalize amplifies sound volume as much as possible without exceeding the loudest element in the sound sample. In otherwise, it "normalizes" the volume around a unified volume level. Very useful volume rises and falls within a sound file.

Container – location of the sound file

The result must be set into a container field.

Media_Normalize makes the sound as loud as possible without clipping.

A good way to become more familiar with this function is to run the function with several values, and follow by calling the Media_DrawWaveform function to get a visual representation of the changes.

Use Media_GetLastError to check for any errors of this function.

Media_PlayPause

Media_PlayPause allows you to temporarily halt the Media_PlaySound function.

No parameters are used.

When Media_PlaySound is playing a sound, calling Media_PlayPause will halt the playback. Calling Media_PlayPause a second time will resume play.

This is a useful function to call from a script triggered by a button.

Example:

To pause an audio file currently being played:

Set Field [MyTable::Response ; Media_PlayPause]

Media_PlaySound

(container ; {inBackground ; {start ; {end}}})

This function allows you to play a sound in a container field or from a sound file, giving you precise control over playback.

container – location of the sound to be played
inBackground – "1" to play asynchronous in background, or "0" to play synchronous in FMP
start – point in the sound file to begin playing
end – point in the sound file to end playing

The Media_PlaySound function can play all of the follow sound files:

a) QuickTime reference [Created with right Insert QuickTime...]
b) 'snd ' resource, create with double click at the container
c) file reference eg.

filemac:/Macintosh HD/Users/tomas/Desktop/file.mp3
file:./file.mp3
filewin:/C/test.mp3

Foreground or Background Playback

The sound can be played synchronously or asynchronously.
Synchronous playback pauses other FileMaker operations until the sound is finished playing. (You can use ESC or CMD+period to stop playback.)

Asynchronous playback plays in the background, even when FileMaker is no longer the selected application.

To stop playback before the sound file is completed, use the Media_StopSound function.

Media_RecordSoundStart

(path ; { inBackground ; { parameters } })

This function gives you the ability to record sounds of any length either directly into a container or as an external sound file stored as a reference.

path – location and file name of the external sound file to be created. With synchronous (non-background) recording, leave this parameter blank if you want to store the recorded sound directly in the container field.
inBackground – “1” or “True” to record asynchronous in background, or “0” or “False” to record synchronous within FMP
parameters – (not yet implemented in the current version of MediaManager) When available, this parameter will allow you to override the computer’s default sound input settings. (Note that the preceding semicolon “;” divider is still required.)

FileMaker Pro’s built-in sound recording capability only records directly into the container field and limits the recorded sound duration to a maximum of about one minute in length. MediaManager’s record sound functions (Media_RecordSoundStart, Media_RecordSoundPause, Media_RecordSoundStop) entirely expand your ability to record sounds through FileMaker Pro. They give you the ability record sounds of any length, as well as the capability to store the recording as an external sound file that is stored in the container field as a QuickTime reference.

Synchronous or Asynchronous Recording

Just as with Media_PlaySound, this function can record synchronously or asynchronously.

Synchronous recording pauses other FileMaker operations until you are done recording. To manually stop recording, click outside of the destination container field, type ESC, or use CMD+period.

Asynchronous recording occurs in the background, even when FileMaker is not longer the frontmost active window. To stop asynchronous recording, use the Media_RecordSoundStop function.

Synchronous and Asynchronous File References

It is also important to understand that the QuickTime file references created by the two methods of recording are different. An asynchronous recording results in a temporary QuickTime file reference that is valid only until the computer is shut down. In order to make the reference to the sound file stable, you must convert/export the recorded sound file.

Doing a synchronous recording to an external file, however, results in a stable file reference that does not break when the computer is shut down.

File Formats

Synchronous recordings directly into a field (without a file path specified) are in “sfil” format. All other recorded sounds are initially generic QuickTime movie format. If you are saving as an external file, the file name extension should be “.mov”.

Returns:

Returns the recorded sound (or QuickTime reference) to the specified container field.

Use Media_GetLastError to check for errors.

Examples:

To record within FMP directly into a container field:

```
Set Field [ MyTable::Sound ; Media_RecordSoundStart ( “” ; 0 ; ) ]
```

To record within FMP to a file named “NewSound.mov” (to be placed in the default folder):

Set Field [MyTable::Sound ; Media_RecordSoundStart (“NewSound.mov” ; False ;)]

To record to a file named “NewSound.mov” on the desktop, regardless of whether or not FMP is active:

Set Field [MyTable::Sound ; Media_RecordSoundStart (“.D/NewSound.mov” ; 1 ;)]

Media_RecordSoundPause

Toggles the pause/resume state of the current recording.

No parameter is used.

It is often useful to call this function from a button/script.

Returns:

Returns an error if the function is called when no recording is in progress.

Use Media_GetLastError to check for errors.

Example:

To pause or resume recording:

Set Field [MyTable::SoundContainer ; Media_RecordSoundPause]

Media_RecordSoundStop

This function stops the current asynchronous recording if any and returns a

QuickTime reference to that file.

No parameter is used.

It is often useful to call this function from a button/script.

Returns:

Returns an error if the function is called when no recording is in progress.

Use Media_GetLastError to check for errors.

Example:

To stop recording:

```
Set Field [ MyTable::SoundContainer ; Media_RecordSoundStop ]
```

Media_Set

(tag {; value})

The Media_Set function allows you to control the Media_InsertQuickTime function's interaction with QuickTime. MediaManager uses a couple of workarounds to sidestep some known bugs in QuickTime.

tag – one of several values:
- "Volume"
- "Tempo"
- "Balance"
- "lmworkaround"
- "rfworkaround"

value – (see below)

Both workarounds are set to being "on" by default – and under most circumstances you will want them to remain on. These workarounds exist primarily for circumstances when you may want to turn them off, especially if QuicktTime eventually fixes the bugs and you want to rely on the standard QT process.

Volume
Media_Set ("Volume" ; value)

Sets the playback volume for Media_PlaySound.

1 = normal playback volume

Setting the volume to greater than 1, such as “5”, will result in a louder playback. A volume of less than one, such as “.7”, gives a quieter playback result.

Tempo
Media_Set ("Tempo" ; value)

Sets the tempo or audio speed of the Media_Playsound playback.

1 = normal playback speed

A tempo of less than one, such as “.5”, gives a slower playback speed. A tempo greater than one, such as “2”, will result in a faster playback speed.

To reverse playback, set the tempo to a negative value.

Balance
Media_Set (“Balance” ; balanceValue)

This value sets the balance between left and right audio channels during Media_PlaySound playback.

0 = equal balance
-1 = full left channel
1 = full right channel

The balanceValue parameter should be a number between -1 and 1.

Compatibility with Earlier Versions of QuickTime
Media_Set ("lmworkaround" ; True)

The “lmworkaround” tag – linking media workaround – allows you to work with earlier versions of QuickTime. Because QuickTime 7.1.5 incorporates significant “linking media” bug fixes that affect how FileMaker Pro inserts QuickTime sound files, MediaManager by default expects to work with QT 7.1.5. If you want to be able to work with earlier versions of QuickTime, …

value parameter -

True or 1 - This allows the end user to have earlier versions of QuickTime installed. MediaManager checks the version of QT at startup against 7.1.5. At each function call, if it is earlier than 7.1.5, then the workaround is applied.

False or 0 - Media linking is never done. The developer is responsible for checking the version of QT and refusing to run if it is not recent enough.

"never set" – The version of QuickTime is checked at startup. During any relevant function call, QT version is tested against 7.1.5. If earlier, then the user gets this one-time dialog:
"The MediaManager plug-in requires the latest version of QuickTime. Please quit FileMaker now, and install QuickTime from quicktime.apple.com." (OK only)

Inserting Remote Sound Files

Media_Set ("rfworkaround" ; True)

With the “rfworkaround” tag – remote file workaround – MediaManager sidesteps a bug that causes FileMaker Pro to crash on Windows when attempting to insert QuickTime sound files from a remote volume.

In order to avoid the FileMaker crash, MediaManager essentially copies the remote file to the local volume’s temporary directory before inserting it.

Important: Because the inserted sound file actually resides in the temporary folder, that means that the sound file will be lost when the temporary file resets on shut down. In order to avoid this, you must relocate the source file. An easy way to do this is to immediately follow the insert steps with exporting the sound file to a stable location.

Remote File Warning Dialog

The first time (after launching FileMaker Pro) that you call Media_InsertQuickTime to insert a sound file from a remote volume, the following dialog will be displayed.

“MediaManager's Media_InsertQuicktime is about to us a workaround for a bug in QuickTime while accessing remote media. This workaround may cause the FileMaker script to fail. Please see the documentation for Media_InsertQuickTime for complete information.”

You can prevent this dialog from appearing by setting “rfworkaround” to True before calling Media_InsertQuickTime.

value parameter -

True or 1 – Use the remote file copying workaround on Windows. Also suppresses the remote file warning dialog.

False or 0 – Disable the workaround

Media_SetMetadata

(containerOrFile ; informationListOrTag1 {; value1IfTag1 {; tag2; value2 {; tag3; value3...}}})

This media allows you to modify the ID3 metadata stored in an MP3 sound file.

containerOrFile – location of the sound file
informationListOrTag1 –
- when this is the only other parameter, this value completely overwrites all metadata;
- otherwise, this is the tag name for the value that follows

value1 – the value to be set for tag1 (defined in the previous parameter.
tag2; value2; tag3; value3; etc…

To give you the greatest possible flexibility, Media_SetMetadata can work in several possible ways:

Clear All Metadata

Media_SetMetadata (containerOrFile ; "")

In this case MediaManager will remove all metadata from the sound file.

Clear a Single Tag

Media_SetMetadata(containerOrFile ; tag)

In this format, you list a valid tag name (such as “Name,” “Artist,” “Year,” etc.) but do not follow with a new value, MediaManager will removes the specified tag from the sound file's metadata.

Set a Metadata Information List

Media_SetMetadata(containerOrFile ; information list)

When the second parameter is not recognized as a valid tag that already exists in the file's metadata, then MediaManager will overwrite the old metadata with this value. The assumption is that this parameter contains a completely new set of metadata with both tag names and values.

Modify a Single Tag

Media_SetMetadata (containerOrFile ; tag1 ; value1)

If you want to modify only a specific tag in the metadata, you can easily do this by giving the tag name as the second parameter (such as "Artist") and the value it should hold immediately following it (such as "Richard Steiner").

Modify a Series of Tags

Media_SetMetadata(containerOrFile ; tag1; value1 ; tag2; value2 ; tag3; value3; etc.)

This works the same as with modifying a single tag, but you can also define an entire series of tags to be individual modified.

File Metadata vs. Container Metadata

MediaManager distinguishes between the metadata of the external source sound file and the metadata of sound within the FileMaker Pro container field. If the first parameter ("containerOrFile") is a container field name, only the container's metadata will be affected. If the first parameter is a file name (or file path and file name) then only the external file's metadata will be modified. To affect both, you must call the function twice, once for the file and once for the container

Follow Media_SetMetadata with Media_GetLastError to check for any error responses.

See Media_GetMetadata to view metadata tags.

Examples:

To add a "Comment" tag to the metadata in a container field:

```
Set Field [ MyTable::Response ; Media_SetMetadata ( MyTable::SoundContainer
; “Comment” ; “Great audio take!  Use in the final set.” ) ]
```

To add the same “Comment” tag to the metadata the external source file:

```
Set Field [ MyTable::Response ; Media_SetMetadata ( “.a/audio/take13.mp3” ;
“Comment” ; “Great audio take!  Use in the final set.” ) ]
```

To change the “Artist” tag of an external file to read “The Flying Mozart Quartet”:

```
Set Field [ MyTable::Response ; Media_SetMetadata ( “Aria.mp3” ; “Artist” ;
“Flying Mozart Quartet” ) ]
```

To completely replace the metadata in a container with your own list:

```
Set Field [ MyTable::Response ; Media_SetMetadata ( MyTable::SoundContainer
;
“Name: Sevilla/Seville
Artist: Albeniz
Album: Spanish Nights
Track: 2/13”) ]
```

To modify a series of tags and also check for errors:

```
Set Field [ MyTable::Response ; Media_SetMetadata ( “.d/scatological.mp3” ;
“Artist” ; “Trashcan Jeff” ; “Date” ; “Late last night” ; “Description” ; “You don’t
want to know.” ) ]
If [ not IsEmpty ( Media_GetLastError ) ]
        Beep
        Show Custom Dialog [ “Metadata Error” ; “The metadata could not be set.”
]
End If
```

To clear all metadata in a container:

```
Set Field [ MyTable::Response ; Media_SetMetadata ( MyTable::SoundContainer
; “” ) ]
```

Media_SoundFormatDialog

(soundOutputFormat)

This function displays a sample sound format setting dialog and returns a hexadecimal value that corresponds to the settings you select.

soundOutputFormat – sound format that defines which dialog is displayed

The returned hexadecimal value can be used to store preset sound format preference settings which can then be used with the Media_ConvertSound function.

Returns:

The result of this function is a <hex>....</hex> text, such as:

```
<hex>000000AC789C636040010B8A5313F38034231483404F717E294C8C
152A26515C9C5C82A486A1A43CBF18225E8422BEC6054C89150301B2388
300443439194594098865CA528B50D5323308812800106610C3</hex>
```

Media_StopSound

This function stops all currently playing asynchronous sounds.

No parameter is used.

New Millennium
COMMUNICATIONS

New Millennium Communications is a software development company located in Boulder, Colorado. We specialize in making tools and advanced templates for FileMaker Pro developers.

New Millennium Communications
1332 Pearl Street

Boulder, CO 80302 USA
303-444-1476

www.newmillennium.com plug-ins@nmci.com