FileFire

FileMaker Plug-In Manual

Dacons

This manual assumes that you have elementary knowledge of FileMaker.

Please send feedback to info@dacons.net

Website: http://www.dacons.net

April 19, 2012

CONTENTS

FUNCTION INDEX

FUNCTION OVERVIEW

FileFire Features

FileFire is an indispensable plug-in that lets you go beyond the file management capabilities of FileMaker. FileFire comes in two editions, *Express* and *Advanced*. Both editions share a rich set of features that improves the way you import and reference external files. Retrieve all items of specific folders or volumes in FileMaker. Import file attributes such as name, size and even icons. Launch referenced files from within FileMaker together with their parent applications.

In addition to these features, FileFire *Advanced* lets you manage external files from within FileMaker. Rename, copy, move or delete files and folders. Import and export text and work with ZIP archives. Features exclusively provided by FileFire Advanced are marked with an asterisk (*) in this manual.

Use FileFire to enrich your solution with stunning file management features not seen in FileMaker before.

Function overview

- List items of any folder or volume in FileMaker
- Show file attributes like name, icon, size, creation and modification timestamps
- Have users specify external files or folders using system dialogs and retrieve selected paths in ScriptMaker
- Reference or import$^{FM8/9}$ all items of a specific folder, including office documents
- Batch export files or references from container fields to user specified folders^{FM8}
- Launch any external file from FileMaker together with its parent application
- Copy, move or delete files and folders*
- Rename files and folders and change other attributes*
- Create folders and aliases/shortcuts*
- Compress and decompress ZIP archives*
- Import and export text files*
- Find files that match specific criteria*
- Show a status dialog with a progress bar during file operations*

Possible solutions

- Create professional document libraries that link relevant files to FileMaker records
- Add file attributes to solutions that import or reference external files
- Tightly integrate FileFire features with FileMaker script steps such as *Insert File*, *Insert Picture*, *Insert QuickTime* and *Export Field Contents* (requires FileMaker 8 or later) to create dynamic file management solutions
- Manage external files on disk from within FileMaker*
- Archive database backups in ZIP format*

INTRODUCTION
Getting Started

This chapter describes how to use this manual and provides all information you need to install the plug-in. In addition, you will see how to explore the powerful features of FileFire using the files that come with this plug-in.

About this manual

The manual is structured as follows. Chapters 1 through 6 explain functions provided by both editions of FileFire – Express and Advanced. Chapters 7 through 13 show you how to use functions exclusively provided by FileFire Advanced.

Chapter 1 (FileFire Basics, p. 11) explains how to use FileFire functions with FileMaker. You will learn about plug-in functions, parameters, result codes and path formats. **Chapter 2** (Retrieve Item Paths, p. 18) provides all information you need to list folder contents in FileMaker. **Chapter 3** (Retrieve Attributes and Icons, p. 21) teaches you how to retrieve attributes of files and folders. In **Chapter 4** (File and Folder Dialogs, p. 24) you will see how to show dialogs that let users select files or folders. **Chapter 5** (Work with Paths, p. 27) explains how to convert and use the paths that FileFire returns. In **Chapter 6** (Launch Files and Open Folders, p. 31), you will learn how to launch any items.

Chapter 7 (Import and Export Text*, p. 32) provides information about FileFire functions that improve the way you work with external text files. **Chapter 8** (Change Attributes*, p. 35) tells you how to modify file and folder attributes. Need to copy, move or delete files or folders? Check out **Chapter 9** (Copy, Move and Delete*, p. 37). **Chapter 10** (Find Items*, p. 41) shows you how to find files based on search criteria. See how to create folders and aliases in **Chapter 11** (Create Folders and Aliases*, p. 43). **Chapter 12** (Work with ZIP Archives*, p. 45) enables you to operate with archives from within FileMaker. Learn how to show a status dialog for file operations in **Chapter 13** (Status Dialog*, p. 48).

Finally, **Chapter 14** (Additional Functions, p. 50) provides information about advanced plug-in functions that let you retrieve result codes, configure the log engine and more.

Software requirements

FileFire requires FileMaker 7 or later. The FileMaker editions Pro, Advanced/Developer and Runtime are supported. The plug-in is shipped in two different file formats, one for Windows and one for Mac OS X.

When mentioning “FileMaker”, this manual assumes FileMaker 7 or later.

Installing the FileMaker plug-in

Before installing the FileFire plug-in, ensure you select the correct version from the download package according to your operating system (Windows or Mac OS X).

Next, ensure that FileMaker is closed. Then copy the FileFire plug-in for your operating system in the *Extensions* folder which is located in the FileMaker application folder. Now, please launch FileMaker.

Hands-on examples

After installing FileFire, open the *Quick Start* file that comes with the download package. The Quick Start file demonstrates some of the most powerful FileFire features. Take some time to explore the demo tour.

Continue reading this manual to find out how the features shown in the Quick Start file have been implemented in FileMaker using FileFire plug-in functions.

Multi-user setups

To use FileFire functions in multi-user network solutions, the plug-in needs to be installed on every computer that uses its functions. Due to the software architecture of FileMaker Server, plug-ins cannot operate on the server side.

Alternatively, you can set up a copy of FileMaker Pro or Advanced (Developer) with FileFire installed on a dedicated computer (or the server computer) that processes plug-in functions for all users. However, in most cases plug-in functions will be requires on every client computer and thus the first scenario applies.

CHAPTER 1
FileFire Basics

This chapter explains how to use FileFire functions with FileMaker. You will also learn about path formats. After reading this chapter advance to any of the other chapters of this manual that provide the information you currently require.

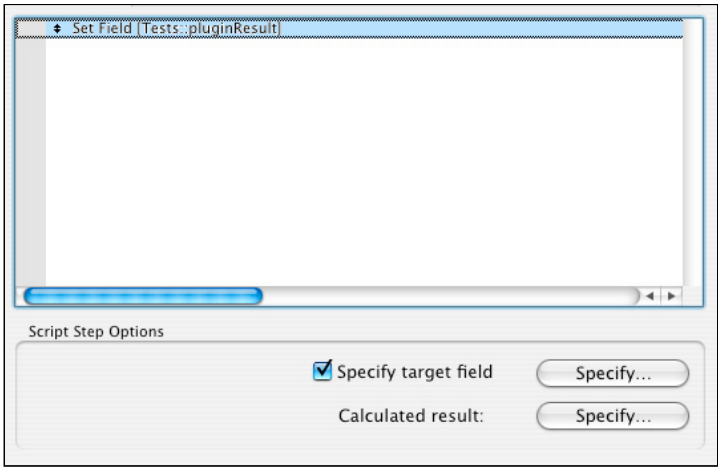
How to use FileFire functions

Since FileFire is a FileMaker plug-in, so-called *external functions* are used to trigger the plug-in from a FileMaker database solution. They are called external functions because these functions are provided by a plug-in and thus are not part of the actual FileMaker application. In order to use the external functions provided by a plug-in it must be installed and enabled in the FileMaker application preferences.

To access external functions provided by a plug-in the FileMaker calculation editor is used. The calculation editor is provided by the FileMaker editions Pro and Advanced (Developer in version 7). FileMaker shows the calculation editor dialog window whenever a calculation has to be defined (e.g. for calculated fields, validation calculations etc). In most cases you will invoke FileFire functions from a script. The easiest bridge between scripts and the calculation editor that invokes plug-in functions is a script step called *Set Field*. Add a *Set Field* script step to your script to start using FileFire.

Next, specify the target field which will contain the result of the plug-in operation. There are two types of FileFire results. Some FileFire functions return content (such as the name of a file) which can be stored directly in a FileMaker field. Other functions do not return content. Instead, result codes are provided by these functions that tell you if an operation succeeded or failed for a certain reason. Click the *Specify* button provided by the *Set Field* script step in ScriptMaker and define the target field of your plug-in command depending on the type of result. Refer to the functions reference (starting on p. 18) for further details about the results returned by each plug-in function.

In the following figure a global field called *pluginResult* (defined in the table *Tests*) has been specified as result field. In most cases, you will choose a global field as result field if the plug-in function invoked does not return content but only a result code. If content is returned by a function choose an appropriate non-global field instead to store results in a database record.


Set Field [Tests::pluginResult]

Script Step Options

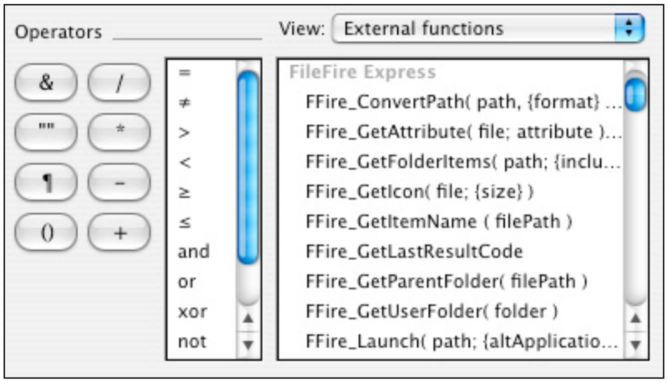
☑ Specify target field

Set Field command in ScriptMaker

If you are working with FileMaker 8 or later you will prefer using script variables over global fields to store non-content plug-in results. Thus, use the script step *Set Variable* instead of *Set Field* for plug-in functions that do not return content.

Click the *Specify* button (the second *Specify* button when using *Set Field*) to open the calculation editor.

In the top right corner of the calculation editor dialog you find a drop-down menu called *View*. Switch to the section *External functions*. All FileFire functions are listed in the section *FileFire Express* or *FileFire Advanced* depending of the edition of FileFire you are using. If the FileFire functions do not appear in the list, the plug-in is not installed correctly or it is disabled in the FileMaker application preferences. In this case leave the calculation editor and check that FileFire is installed as described earlier and that it is enabled in the FileMaker application preferences.


View: External functions

Operators
&
/
""
*
¶
-
()
+
=
≠
>
<
≥
≤
and
or
xor
not
FileFire Express
FFire_ConvertPath(path, {format} …
FFire_GetAttribute(file; attribute)…
FFire_GetFolderItems(path; {inclu…
FFire_GetIcon(file; {size})
FFire_GetItemName (filePath)
FFire_GetLastResultCode
FFire_GetParentFolder(filePath)
FFire_GetUserFolder(folder)
FFire_Launch(path; {altApplicatio…

FileFire functions in the Calculation Editor

Double-click on the FileFire function in the list which you would like to use. The function is copied to your calculation including a *Quick Reference* text that describes the meaning of the selected function and its parameters. To use the function, parameter values have to be specified.

Function parameters

Most FileFire functions provide parameters that specify details of a function call. Many of these functions have several parameters. After copying a function to the text box of the calculation editor, FileMaker shows the name of the plug-in function that has been selected and a text label for each parameter of the function.

The following example shows the FileFire function `FFire_GetIcon` that has been added to the calculation editor. This function returns the icon of a specified file.

```
FFire_GetIcon ( path {; size })
```

After copying the function to your calculation, you need to define a value for each parameter. This can be *hard-coded* values (i.e. a specific file path for the `path` parameter) enclosed in quotation marks. As in every calculation you may also specify *dynamic* values represented by database field names or script variables that hold the information to be passed to the plug-in as parameter value.

Required and optional parameters

Most functions have some parameters that are required and some that are optional. When invoking a FileFire function you have to pass a value for all required parameters. Optional parameters can be skipped. If no value is passed to the plug-in for an optional parameter the default value for this parameter will be used.

Required parameters are listed in the beginning and optional parameters are listed at the end of a function call. This allows you to omit optional parameters that you would not like to use in your function call.

After copying a function to the calculation editor you will see optional parameters enclosed in curly braces `{}`. To skip an optional parameter use an empty text value represented by two quotation marks in the text editor `""`. Optional parameters at the end of a function call can be skipped completely. After specifying parameter values you need to remove any curly braces from the function call.

In the following example the function `FFire_GetIcon` is invoked. A file path must be specified because the parameter `path` is required. However, the size of the icon to be returned is *not* specified. Since the parameter `size` is optional and there is no further parameter value required for this function call, it can be omitted completely:

```
FFire_GetIcon ( "filewin:/C:/Test/MyDocument.doc" )
```

The plug-in will choose the default value for the parameter `size` (32 x 32 pixels).

A full explanation of path formats will be provided later in this chapter (p. 15).

Result codes

As mentioned earlier, FileFire functions can be separated into two groups. Content functions return values that are stored in database fields (using *Set Field* script step). If a content function fails, an empty result is returned. To find out *why* a content function failed invoke the function `FFire_GetLastResultCode`. It returns a result code and a short description of the error. The second group of FileFire functions is called non-content functions. Non-content functions return a result code directly. However, you may also invoke `FFire_GetLastResultCode` to check the result code of non-content functions.

You can evaluate result codes in your script and perform certain activities (e.g. show an error message to the user). To check in ScriptMaker if the last FileFire operation failed, use the following calculation in an if condition:

```
Left ( FFire_GetLastResultCode ; 1 ) = "-"
```

The following conditions checks for a specific error:

```
Left ( FFire_GetLastResultCode ; 4 ) = "-052"
```

A list of all result codes and their meanings follows on the next page.

Result Code	Description
000	(OK) Function completed successfully, no errors occurred
-007	(Registration failed) Name or registration code incorrect
-008	(Incorrect parameters) Incorrect parameters in function call
-018	(Incorrect encoding) Error applying the encoding specified
-015	(Not supported) The operation is not supported the way it has been configured
-022	(Log file not opened) Error occurred when trying to open the log file
-023	(Log file corrupted) The log file is corrupted
-050	(Incorrect path) The patch specified by the user (file dialog) is invalid
-052	(Failed to open file) The specified file or application could not be launched
-053	(Item not found) The specified file or folder could not be found
-055	(File input/output error) An error occurred while trying to read or write a file
-057	(File or folder is in use) File sharing violation
-058	(Access denied) The accesses to a file or folder has been denied
-066	(Invalid disk drive) Invalid drive has been specified
-067	(Device not ready) The device to be accessed is not ready
-068	(Invalid filter) The specified file type filter is invalid
-071	(Internal error) Contact the Dacons Support at http://www.dacons.net/support
-072	(Dialog cancelled) User has cancelled a dialog
-074	(Loading icon failed) The icon could not be imported

Result Code	Description
-075	(File already exists) File, folder or alias already exists
-080	(Destination inside source) The destination is located inside the source specified
-099	(Unknown error) Contact the Dacons Support at http://www.dacons.net/support

Path formats

FileFire requires paths of files and folder to be specified in a certain format. To make life easier, FileFire uses the same path format as FileMaker script steps that operate with files. However, unlike FileMaker the plug-in does not support relative paths.

On Windows, paths to be used with FileFire functions must be formatted according to one of the following patterns:

```
filewin:/driveletter:/directoryName/fileName.extension
filewin://computerName/shareName/fileName.extension
```

On Mac, please use the following pattern to specify a path:

```
filemac:/volumeName/directoryName/fileName.extension
```

When specifying a file path in FileMaker (*File Location* dialog) you can define alternative files. Paths of alternative files are separated by carriage return. Please note that FileFire does ***not*** support alternative path this way. It allows you to specify exactly one path at a time only to avoid ambiguities.

Some FileFire functions allow you to specify a folder path instead of a file path. Please ensure that folder paths always end with a final slash (“/”). The following patterns refer to folder paths on Windows:

```
filewin:/driveletter:/directoryName/
filewin://computerName/shareName/folderName/
```

On Mac a folder paths has to be formatted as follows:

```
filemac:/volumeName/directoryName/
```

FileFire uses the FileMaker path format not only when the location of files and folder are *specified*. The FileMaker path format is also used when paths are *returned* by the plug-in (e.g. using the function `FFire_GetFolderItems`). Since FileMaker 8 and later support script variables you can use retrieved paths for further processing with FileMaker script steps such as *Insert File*. The script steps *Insert Picture* and *Insert QuickTime* require different path formats. Use the function `FFire_ConvertPath` to prepare retrieved paths for these functions. Please review **Chapter 5** (Work with Paths, p. 27) for further information.

CHAPTER 2

Retrieve Item Paths

FileFire enables you to list the paths of all items that are located in a specific folder. Moreover, FileFire lets you retrieve the paths of all volumes currently mounted. This chapter explains how to retrieve the paths of files, folders and volumes.

List folder items

The function **`FFire_GetFolderItems`** returns the items of a specific folder including files and sub folders.

Syntax:
`FFire_GetFolderItems ( folderPath {; includeSubfolders; control })`

Parameters:
`folderPath` – The path of any local or network folder in FileMaker path format (`filewin` or `filemac`). A file path is not accepted by this function.

`includeSubfolders` – This parameter is optional. It specifies whether items from subfolders will be considered or if the scope of this function will be restricted to the specified path. By default, the value `"includeSubfolders"` is set which includes items of all subfolders recursively. To exclude items of subfolders set this parameter to the value `"excludeSubfolders"`.

`control` – This parameter is optional. It can be used to specify which kind of items are to be returned. Leave this parameter empty (default) to return *visible files and folders* only – neither aliases/shortcuts nor hidden files are returned. The following options can be combined using commas: `"noFolders"`, does not return folders (only files); `"noFiles"`, returns no files (only folders); `"aliases"`, includes aliases/shortcuts and `"hidden"`, returns visible *and* invisible (hidden) items.

Use the function `FFire_ShowFolderDialog` to let the user specify a folder.

Result:
All items of the specified folder are returned according to the settings provided. Items are

separated by carriage return. If errors occur, an empty result is returned. Invoke the function `FFire_GetLastResultCode` for details in such a case.

Example:
In the following example the function is invoked on a Mac. FileFire is set to list all items of the folder “Letters” which is located on the volume “Hard drive”. According to the parameters specified, the plug-in is to return all items, including aliases/shortcuts and hidden files and folders.

```
FFire_GetFolderItems ( "filemac:/Hard drive/Letters/" ; "" ;
"aliases,hidden" })
```

For this example, the following result is returned:

```
filemac:/Hard drive/Letters/Sandy.doc
filemac:/Hard drive/Letters/Peter.rtf
filemac:/Hard drive/Letters/Pam & George.txt
```

List volumes

Invoke the function **`FFire_GetVolumes`** to retrieve the paths of all volumes currently mounted. This includes local and network volumes shown in “My Computer” (Windows) or on the Desktop (Mac OS X).

Syntax:
`FFire_GetVolumes`

Parameters:
This function does not require any parameters.

Result:
The paths of all volumes currently mounted are returned separated by carriage return. If errors occur, an empty result is returned. Invoke the function `FFire_GetLastResultCode` for details in such a case.

Example:
The following example function call is invoked on a Windows system with three volumes (A, C and D drive):

```
FFire_GetVolumes
```

For this example, the following result is returned:

```
filewin:/a:/
filewin:/c:/
filewin:/d:/
```

Hint:
For further processing of the returned result check the FileMaker functions *MiddleValues* (FileMaker 7 and later) and *GetValue* (FileMaker 8 and later). These functions let you extract values from the result listing.

Retrieve system and user folders

The function **`FFire_GetSystemFolder`** returns the paths of certain system and user folders such as the desktop or the documents folder.

Syntax:
`FFire_GetSystemFolder ( folderType )`

Parameters:
`folderType` – This parameter specifies the folder path is to be returned by the plug-in.

On both *Windows and Mac OS X* this can be one of the following: `"desktop"` returns the desktop folder path. `"applications"` returns the folder which is used as default location when installing applications. `"system"` returns the path of the operating system folder; `"temp"` returns the temp folder path. `"root"` returns the root path of the start-up disk. `"documents"` returns the path of the user's documents folder. `"fonts"` returns the fonts folder. `"userRoot"` returns the path of the user's root folder and `"preferences"` returns the path of the preferences folder.

On *Windows* the following values can be specified in addition: `"startupItems"` returns the folder for start-up items. `"startMenuSystem"` returns the start menu folder that is shared by all users of a system. `"startMenuUser"` returns the start menu folder of the current system user. This start menu folder is different for each system users. `"startProgramsSystem"` returns the folder for application aliases in the start menu that is shared by all users of a system. `"startProgramsUser"` returns the folder for application aliases in the start menu that is different for each system users.

Result:
The path of the specified system or user folder is returned. If errors occur, an empty result is returned. Invoke the function `FFire_GetLastResultCode` for details in such a case.

Example:
In the following example the path of the fonts folder is returned on a Windows system.

```
FFire_GetSystemFolder ( "fonts" )
```

For this example, the following result is returned:

```
filewin:/c:/Windows/Fonts
```

CHAPTER 3

Retrieve Attributes and Icons

FileFire provides functions that enable you to retrieve attributes such as the name and the size of files stored in FileMaker container fields or files stored on disk. You can also retrieve attributes of folders and store the icon of any external file or folder in a container field. Moreover, FileFire enables you to check the existence of a specific file or folder.

Retrieve attributes

The function **`FFire_GetAttribute`** returns a specified attribute of a file, folder or volume.

Syntax:
`FFire_GetAttribute ( path ; attribute )`

Parameters:
`path` – External file or folder referenced by path in FileMaker format (filewin or filemac) or a file stored in a container field.

`attribute` – This parameter indicates which attribute is to be returned by the plug-in. It can be used for files, folders and volume (exceptions apply). `"name"` returns the name of an item. `"parentFolder"` returns the path of the parent folder. For files the extension is included in the name. `"size"` returns the size of an item in bytes. This attribute is supported for files and folder but not for volumes. `"kind"` returns a description of an item (e.g. “Microsoft Word Document” for .doc files). `"readOnly"` returns the value 1 if an item is read-only (Windows) or locked (Mac OS). If an item is not read-only the value 0 is returned. Please note that even though this attribute might not be set for an item it may still be write-protected by privileges or physical restrictions. Therefore, the value `"writable"` returns 1 if an item can be modified or 0 if they cannot be modified . (Please note that items which are exclusively opened by an application are not writable even though user rights may allow modifying them). `"hidden"` returns 1 if an item is hidden and 0 if it is not hidden. `"archive"` (Windows only) returns the value 1 if the archive attribute of an item is set or 0 if it is not set. `"created"` returns the point in time when the item has been created (as FileMaker timestamp). `"modified"` returns the point in time when the item has been modified last time (as FileMaker timestamp, support for file items only). `"accessed"` returns the point in time when the item has been accessed last time (as FileMaker timestamp, support for file items only). `"fmType"` ("FileMaker type") returns "image" if a file is an image in a format supported by FileMaker, "quicktime" if an item is a QuickTime media file in a format supported by FileMaker, "file" if an item is a file (or alias/shortcut) but not an image nor a QuickTime media file, "folder" if an item is a folder, "volume" if an item is a volume or "computer" if the item is a computer. The type can be used to check whether an item can be processed using the script steps *Inset Picture*, *Insert QuickTime* or *Insert File*. Using these script steps with variable paths requires FileMaker 8 or later. The function `FFire_ConvertPath` can be used to convert a path into the required path format.

For files that are stored in a FileMaker container field only the attributes `"name"`, `"size"`, `"kind"` and `"fmType"` are supported.

The following attributes are supported for folders and volumes only. `"fileCount"` returns the number of files (including files in sub folders). `"folderCount"` returns the number of all sub folders. `"itemCount"` returns the number of all files and folders. These attributes need to be computed by the plug-in. Thus it may take some time before the result is returned. The following folder/volume attribute is available immediately, instead. `"isNetwork"` returns 1 if the path points to a network resource and 0 if it points to a local resource.

If the path points to a volume the attribute parameter also supports the following values. `"volumeCapacity"` returns the capacity of a volume in, `"volumeFreeSpace"` returns the space available on a volume and `"volumeUsedSpace"` returns the space used (all in bytes). `"volumeFileSystem"` returns the name of the file system (e.g. "NTFS" or "FAT32" on Windows or "HFS" on Mac OS). Volume attributes are supported for local volumes only.

Instead of using the value `"name"` for this function the function `FFire_GetItemName` can be invoked for paths. To retrieve the parent folder the function `FFire_GetParentFolder` can be invoked instead of using the value `"parentFolder"`.

Please note that the plug-in recognizes folder items by a closing slash ("/"). When processing folder paths, ensure that they contain a closing slash. Otherwise the plug-in may treat them as file items when trying to retrieve attributes.

In order to retrieve different attributes of an item the function `FFire_GetAttribute` has to be invoked several times for the same item with different attribute parameter values.

Result:
Depending on the specified attribute values are returned in text, number or timestamp format. If errors occur, an empty result is returned. Invoke the function `FFire_GetLastResultCode` for details in such a case.

Retrieve icons

Using the function **`FFire_GetIcon`** you can retrieve icons of external files or files stored in container fields. This function also returns icons of folders and volumes.

Syntax:
`FFire_GetIcon ( path {; size })`

Parameters:
`path` – External file or folder referenced by path in FileMaker format (filewin or filemac) or a file stored in a container field. Ensure that folder or volume path end with a slash ("/").

`size` – This parameter is optional. It specifies the size of the icon to be returned. Two values are supported. The value `"32"` (default) returns an icon in a 32x32 pixels format. The value `"16"` returns an icon in a smaller 16x16 pixels format (e.g. for list views).

Result:
Icons are returned as pictures and can be stored in container fields. If errors occur, an empty result is returned. Invoke the function `FFire_GetLastResultCode` for details in such a case.

Check file and folder existence

The function **`FFire_GetItemExistence`** lets you check if a specific file, folder or volume exists. This can be useful if you have a path stored in FileMaker and you need to ensure that the item that path refers to still exists.

Syntax:
`FFire_GetItemExistence ( path )`

Parameter:
`path` – This parameter specifies the path of the file, folder or volume to be checked for existence. The path is specified in FileMaker path format (filewin or filemac). Ensure that folder or volume path end with a slash ("/").

Result:
The function returns a result code which indicates whether the specified file exists. In case it exists, the value "000 (OK)" is returned. If the item cannot be found or other errors occur a different result code is returned.

Example:
In the following example the existence of a folder is checked on a Windows system:

```
FFire_GetItemExistence ( "filewin:/c:/Work/Reports/" )
```

Assuming that the specified folder exists, the following result is returned: `000(OK)`

CHAPTER 4
File and Folder Dialogs

FileFire provides functions that let you show system dialogs for file and folder selection. The plug-in returns the paths of items selected by the user for further processing.

Show a file dialog

The function **`FFire_ShowFileDialog`** shows a file dialog which lets the user specify one or several files. The function supports an *open file* and a *save file* mode. A file type filter can be created to let the user specify only certain file types.

Syntax:

```
FFire_ShowFileDialog ( mode {; title ; defaultFileName ;
intialFolder ; filter })
```

Parameters:

`mode` – This parameter specifies whether an Open File dialog or a Save File dialog will be shown using the following values: `"open"` shows an Open File dialog. The name of the default button will be “Open” depending on the language of the operating system. The user will be enabled to select one existing file. If the parameter is set to `"openMultiple"` the user can select multiple existing files from the same folder. The parameter value `"save"` makes this function show a Save File dialog instead. The name of the default button will be “Save”. The user is asked to specify a single file that does not exist yet. If an existing file is selected an warning message will be shown that asks the user to confirm that the existing file should be replaced.

`title` – This parameter is optional. It specifies the title (text value) to be shown in the file dialog window. According to the mode which has been set for the dialog (open or save) the title should indicate why the user is to specify a file.

`defaultFileName` – This parameter is optional. It specifies a default file name (including extension) that can be set if the user is to select (for opening or saving) a file with a specific name. Users will be able to ignore a default file name.

`initialFolder` – This parameter is optional. It specifies the path of the initial folder which will be shown in the dialog. If this parameter is empty or if the path indicated can not be found the folder from the last dialog session is shown.

`filter` – This parameter is optional. It specifies file type filters for the dialog. The following example shows the filter syntax:

```
"[Adobe Photoshop; CompuServe GIF] *.psd; *.gif¶
 [Microsoft Word; Microsoft Excel] *.doc; *.xls¶
 [Movie Files] *.mov; *.avi"
```

This will show three file type filters that users can select when working with the file dialog. Filters are separated by carriage return so use the “¶” character to separate filters in calculations. In the example above the first filter will be named “Adobe Photoshop; CompuServe GIF”. This filter will allow users to specify any file with the extension *.psd or *.gif. Since it is the first filter it will be the default one. The other two filters can be selected by users to show other file types in the dialog. The filter descriptions (in [ ] brackets above) are optional for filters. If they are not used the file extensions will be shown as filter descriptions in the dialog.

If the user is to specify a file of any file type one of the following alternatives can be used: `[All Files] *.*` shows “All Files” as filter description; `*.*` shows `*.*` as filter description.

Leave the filters parameter empty to let the user specify any file.

Note: If the mode parameter is set to`"save"` every filter must contain one extension only. Filters with multiple extensions are accepted for the modes `"open"` and `"openMultiple"` only.

Result:
This function returns the path of the selected or specified item in FileMaker path format (filewin or filemac). If several items have been specified or selected in the dialog their paths are separated by carriage return. If the dialog is cancelled by the user or if errors occur, an empty result is returned. Invoke the function `FFire_GetLastResultCode` for details in such a case.

Example:
The following function calls shows an Open File dialog that lets the user select multiple files. The dialog title indicates that the file selected will be imported. A default file name is not set and an initial folder is not specified. By default, files of all types are shown in the dialog so that the user can select any files for importing. However, to improve the overview in for folders that contain many items various filters are provided.

```
FFire_ShowFileDialog ( "openMultiple" ; "Select Files for Import" ;
"" ; "" ; "[All Files] *.*¶[FileMaker Databases] *.fp7; *.fp5;
*.fp3; *.fmj¶[Office Files] *.doc; *.xls; *.ppt" )
```

Show a folder dialog

The function **`FFire_ShowFolderDialog`** shows a system dialog which lets the user select a single existing folder. The dialog also provides and option to create a new folder first and then select it.

Syntax:
`FFire_ShowFolderDialog ({ title ; initialFolder })`

Parameters:
`title` – This parameter is optional. It sets a title (text value) shown in the Select Folder dialog window.

`intialFolder` – This parameter is optional. It specifies the path of a folder in FileMaker format (filewin or filemac) which will be shown in the dialog when it appears. If this parameter is empty or if the specified path cannot be found the dialog will show the root level in the folder dialog.

Result:
This function returns the path of the folder selected by the user in the FileMaker path format (filewin or filemac). If the user cancels the dialog or if errors occur an empty result is returned. Invoke the function `FFire_GetLastResultCode` for details in such a case.

Example:
The following function call shows a folder dialog with the title “Select a Folder for Export”. An initial folder is not specified for the dialog. Thus, the root level will be shown. Since the parameter `intialFolder` is optional it can be omitted.

```
FFire_ShowFolderDialog ( "Select a Folder for Export" )
```

CHAPTER 5
Work with Paths

Many FileFire functions such as `FFire_GetFolderItems`, `FFire_GetVolumes`, `FFire_ShowFileDialog` and `FFire_ShowFolderDialog` return paths of files, folders or volumes. The plug-in provides functions that support you when processing retrieved paths. FileFire enables you to convert path formats, extract the parent folder of any given path or extract the name of an item from a path.

Convert path formats

FileFire returns paths in the FileMaker path format. This means, paths begin with a filewin or a filemac prefix depending on the current operating system. You may want to convert retrieved paths into a different format to either display them to user in a native system format or to use them for script steps (e.g. *Insert Picture* and *Insert QuickTime*) that require different path formats. The function **`FFire_ConvertPath`** lets you convert path into any supported format.

Syntax:
`FFire_ConvertPath ( path {; targetFormat })`

Parameters:
`path` – This parameter specifies the source path which is to be converted. The source path must be in filewin, filemac, moviewin, moviemac, imagewin imagemac format or in a native Windows or Mac path format (depending on the current system).

`targetFormat` – This parameter is optional. It specifies the format of the output path. The value `"file"` (default) returns the path in filewin format (Windows) or filemac format (Mac OS). The value `"image"` returns the path in the FileMaker imagewin format (Windows) or the imagemac format (Mac OS) which is required for the *Insert Picture* script step. The value `"movie"` returns the path in the FileMaker moviewin format (Windows) or in the moviemac format (Mac OS) which is required for the *Insert QuickTime* script step. The value `"system"` returns the path in the native system format depending on the current operating system (Windows or Mac OS).

Use the function `FFire_GetAttribute` to check the `fmType` ("FileMaker type") of a file. Items of the fmType "file" can be processed using the script step *Insert File*, items of the fmType "picture" can be processed using the script step *Insert Picture* after converting their path into the picturewin or picturemac format and items of the fmType "quicktime" can be processed using the script step *Insert QuickTime* after converting their path into the moviewin or moviemac format.

Please note that processing variable paths with the script steps *Insert File*, *Insert Picture* and *Insert QuickTime* require script variables and thus FileMaker 8 or later.

For more information about FileMaker path formats please refer to the FileMaker Online Help available from the FileMaker Help menu.

Result:
This function returns the converted path according to the settings specified. If errors occur an empty result is returned. Invoke the function `FFire_GetLastResultCode` for details in such a case.

Example 1:
In the first example a file path has been retrieved using the function FFire_GetFolderItems. Since this path points to a picture file, it can be processed with the script step *Insert Picture*. This script step requires the path to be converted into the
imagewin or imagemac path format (depending on the current operating system).

```
FFire_ConvertPath ( "filewin:/c:/pictures/Hawaii0001.jpg" ;
"image" )
```

The function call above results in the following image path:

```
imagewin:/c:/Pictures/Hawaii0001.jpg
```

Example 2:
In the second example, the path given above should be converted into the native system path format. The converted path will be displayed to the user.

```
FFire_ConvertPath ( "filewin:/c:/Pictures/Hawaii_0001.jpg" ;
"system" )
```

The function call above results in the following system path:

```
c:\pictures\hawaii_0001.jpg
```

Retrieve the parent folder

The function **`FFire_GetParentFolder`** returns the parent folder of a specified path. The specified path can point to a file or another folder. This function can be used to navigate to a parent folder (e.g. using the function `FFire_Launch`) or to create a folder navigator in FileMaker using database relationships as shown in the Quick Start file.

Syntax:
`FFire_GetParentFolder ( path )`

Parameter:
`path` – This parameter specifies the path of a file or folder in FileMaker format (filewin or filemac).

Please note that instead of using this function you can also use the function `FFire_GetAttribute ( "…" ; "parentFolder" ).`

Result:
This function returns the parent folder of the provided path. If the errors occur an empty result is returned. Invoke the function `FFire_GetLastResultCode` for details in such a case.

Example:
In the following example the function `FFire_GetParentFolder` is invoked to retrieve the parent folder of a specific file path.

```
FFire_GetParentFolder ( "filemac:/Hard drive/Letters/Sandy.doc" )
```

The above function call will return the following result:

```
filemac:/Hard drive/Letters/
```

Retrieve item name

The function **`FFire_GetItemName`** returns the name of an item. This can be the path of any file, folder or volume.

Syntax:
`FFire_GetItemName ( path )`

Parameter:
`path` – This parameter specifies the path of a file, folder, or volume in FileMaker format (filewin or filemac).

Please note that this function does not support files that are stored in FileMaker container fields. To retrieve the name of such items use:

```
FFire_GetAttribute ( "…" ; "name" )
```

Result:
This function returns the name of the specified item (including the extension of file names).

If the errors occur an empty result is returned. Invoke the function `FFire_GetLastResultCode` for details in such a case.

Example:

In the following example the function `FFire_GetItemName` is invoked to retrieve the name of a folder.

```
FFire_GetItemName ( "filemac:/Hard drive/Letters/ " )
```

This function call returns:

```
Letters
```

CHAPTER 6
Launch Files and Open Folders

Using FileFire you can launch any file by its path. A file will be opened together with its parent application. Moreover, FileFire lets you open folders.

Launch anything

The plug-in function **`FFire_Launch`** opens a specified external file together with its parent application. If the application is already running FileFire will try to set the application window to front. If the path of a folder is specified it will be shown in an Explorer window under Windows or in a Finder window under Mac OS X.

Syntax:

```
FFire_Launch ( path {; altAppPath })
```

Parameters:

`path` – This parameter specifies the path of a file or folder which is to be launched. The item to be opened must be specified in the FileMaker path format (filewin or filemac).

`altAppPath` – This parameter is optional. It is used to open a file with a specific application (other than the default application). The path of the application must be specified in the FileMaker path format (filewin or filemac). When opening a folder instead of a file this parameter is ignored.

Result:

This function returns a result code to report about success or error reasons.

Example:

In the following example a text file is launched. However, it is not opened with the default application for text files. Instead, an alternative application is specified.

```
FFire_Launch ( "filewin:/c:/Letters/Peter.txt" ;
"filewin:/c:/Program Files/MyTextApplication/App.exe" )
```

CHAPTER 7
Import and Export Text*

FileFire Advanced lets you import text from text files with a single script step. In addition, it improves the way you export text from FileMaker to text files

Import text

The function **`FFire_GetTextFile*`** returns the contents of a specified text file in Unicode format so it can be processed in FileMaker and stored in a text field. This function supports Western, Asian and international text encoding standards.

Syntax:
`FFire_GetTextFile ( filePath {; encoding })`

Parameters:
`filePath` – This parameter specifies the plain text file to be imported in FileMaker path format (filewin or filemac).

`encoding` – This parameter is optional. It specifies the encoding of the text file to be imported. If the parameter is empty the plug-in tries to detect the encoding of the file automatically. However, to ensure correct importing, specify the file's encoding using this parameter. The following values are supported: UTF-8, ISO8859-1 through ISO8859-16, KOI-8R, KOI-8U, Windows1250 through Windows1258, ISO-2022-JP, ISO-2022-JP-1 through ISO-2022-JP-3, CSISO-2022-JP, EUC-JP, ISO-2022-CN, ISO-2022-CN-EXT, GB2312, HZ-GB-2312, ISO-2022-KR, KSC5601-1987, EUC-KR and Big5.

Result:
The function returns the text of the specified file in Unicode format (FileMaker text standard). Invoke the function `FFire_GetLastResultCode` to retrieve a result code which indicates whether the operation succeeded or failed. If it failed details can be referred from the error code.

Export text

FileMaker provides the script step *Export Field Contents* that lets you export text to a file. However, using the FileFire function **`FFire_WriteTextFile*`** introduces advanced text export features. This function lets you append text to an existing file and works with target encoding standards not natively supported by FileMaker.

Syntax:

```
FFire_WriteTextFile ( text ; filePath {; appendControl ;
encoding })
```

Parameters:

`text` – This parameter specifies the text to be written to the text file. Usually you will specify the field name or script variable that holds the text to be exported.

`filePath` – This parameter specifies the path of the target text file in FileMaker path format (filewin or filemac). This file will store the exported text. If the file does not exist yet it will be created by the plug-in. You may specify any file type extension that is based on plain text such as *.txt*, *.xml* and *.html*. However, please note that certain file types require a specific encoding which is to be set using the `encoding` parameter.

`appendControl` – This parameter is optional. It specifies the behavior in case the given file is not empty. The value `"append"` (default) makes the plug-in append the text content. The value `"appendLine"` adds a carriage return to the existing text file (if necessary) and then adds the specified content. The value `"overwrite"` replaces any existing content of the file.

`encoding` – This parameter is optional. It specifies the encoding of the target text file. If the parameter is empty the plug-in chooses the system's default text encoding. However, to ensure correct exporting, specify the file's encoding using this parameter. The following values are supported: UTF-8, ISO8859-1 through ISO8859-16, KOI-8R, KOI-8U, Windows1250 through Windows1258, ISO-2022-JP, ISO-2022-JP-1 through ISO-2022-JP-3, CSISO-2022-JP, EUC-JP, ISO-2022-CN, ISO-2022-CN-EXT, GB2312, HZ-GB-2312, ISO-2022-KR, KSC5601-1987, EUC-KR and Big5.

Result:

This function returns a result code to report about success or error reasons.

Example:

This example assumes that the log file `filewin:/c:/Examples/MyTextFile.txt` contains the following lines:

```
Green Grass
Blue Sky
```

Now, the following function is invoked:

```
FFire_WriteTextFile ( "Clean Water" ;
"filewin:/c:/Examples/MyTextFile.txt" ; "appendLine" ;
"ISO8859-1" )
```

This results in the following contents of the text file:

```
Green Grass
Blue Sky
Clean Water
```

CHAPTER 8

Change Attributes*

FileFire Advanced lets you import text from text files with a single script step. In addition, it improves the way you export text from FileMaker to text files

Change file and folder attributes

The function **`FFire_SetAttribute*`** lets you rename files and folder and change other attributes such as creation and modification timestamps.

Syntax:
`FFire_SetAttribute ( path ; attribute ; value )`

Parameters:
`path` – External file or folder referenced by a path in FileMaker path format (filewin or filemac). Please keep in mind that the paths of folders and volumes end with a final slash ("/").

`attribute` – This parameter indicates which file attribute is to be changed by the plug-in: "`name`" changes the name an item (can be a file, folder or volume). "`created`" changes the creation timestamp of an item. "`modified`" changes the modification timestamp of an item. "`accessed`" changes the access timestamp of an item. "`readOnly`" changes the read-only attribute of an item. "`hidden`" changes the hidden status of an item. "`archive`" (Windows only) changes the archive attribute of an item.

`value` – This parameter specifies the attribute value to be set. To change the "`name`" attribute of an item set a text value for this parameter. Please note that the file names must include an extension (such as MyText.*txt*). The read-only attribute is controlled using the values 1 (enable read-only) or 0 (disable read-only). To hide an item set the value to 1; to unhide it set the value to 0. The archive attribute (Windows only) is enabled by the value 1 and disabled by the value 0.

When processing folder items, the value 1 enables a specific attribute only for the folder itself but not for enclosed items (even though in many cases the effect will be the same; i.e. when a folder is hidden all enclosed items will be hidden). The same restrictions apply when disabling an attribute using the value 0 for a folder. When working with folder items the value 2 disables an attribute for all enclosed items recursively (if possible). The value 3 enables an attribute for all enclosed items recursively (if possible).

Result:
This function returns a result code to report about success or error reasons.

Example:
The following function call enables the read-only attributes for all files stored in the folder *Letters*:

```
FFire_SetAttribute ( "filemac:/Harddisk/Letters/ ; "readOnly" ; 3 )
```

Show an info dialog for attributes

Depending on the version of the operating system and the file system you cannot change all attributes using the function `FFire_SetAttribute`. Therefore, the function **`FFire_ShowInfoDialog*`** is provided. It lets you show an Explorer (Windows) or Finder (Mac) info dialog for a file, folder or volume. User can view and modify attributes of an item using that dialog.

Syntax:
`FFire_ShowInfoDialog ( path )`

Parameter:
`path` – Specifies the path of an item in FileMaker path format (filewin or filemac). The plug-in will show an info dialog with all attributes of the specified item.

Result:
This function returns a result code to report about success or error reasons.

Please note: Users may change item attributes using the info dialog. To retrieve changed attribute values in FileMaker, other plug-in function (such as `FFire_GetAttribute`) have to be used.

CHAPTER 9

Copy, Move and Delete*

Using FileFire Advanced you can copy, move and delete any external files. The following functions give great power to your FileMaker solutions and let you implement anything from simple batch processing scripts that help to organize database backups to sophisticated file commander tools.

Copy files and folders

Use the function **`FFire_Copy*`** to copy file or folders.

Syntax:

```
FFire_Copy ( path ; destinationPath {; replaceControl ;
deleteControl })
```

Parameters:

`path` – Specifies the source file or folder to be copied in FileMaker path format (filewin or filemac).

`destinationPath` – The target path of the file or folder to be copied. Please note that if the name of the file or folder does not match the name indicated by the `path` parameter the item will be renamed after copying.

`replaceControl` – This parameter is optional. It specifies the behavior of the plug-in for situations when the destination file or folder already exists. The value `"replaceError"` (default) makes the plug-in return an error code when the destination file or folder already exists. The specified item is not processed. The value `"replaceSkip"` skips all existing destination items automatically and only replaces items that do not exist in the destination yet. The value `"replaceAsk"` shows a Replace File or Replace Folder dialog box and lets the user choose whether to replace conflicting originals. The value `"replaceUpdate"` compares conflicting items by their modification timestamp and replaces outdated items. Items that do not cause a conflict are processed as well. The value `"replaceSilently"` replaces all conflicting items without error or warning.

`deleteControl` – This parameter is optional. It specifies whether to move replaced items to trash or erase them. By default, this parameter is set to `"trash"`. Set this parameter to `"erase"` to delete replaced items without moving them to trash.

Result:
This function returns a result code which indicates if it succeeded or if errors occurred. When copying a folder that contains several files please note the following: The plug-in copies all files contained within the folder sequentially. Thus, it may happen that not all files are copied due to replace errors (depending on the `replaceControl` parameter value) or other limitation such as privileges.

The result returned is formatted according to the following pattern. Please not that the “failed item” may be a source or destination item.

```
XXX (Description)
(Failed item follows, if any)
…
<start position> <length> (Processed items)
<start position > <length> (Skipped items)
<start position > <length> (Replaced items)
(Items processed follow, if any)
…
(Items skipped before any error follow, if any)
…
(Items replaced before any error follow, if any)
…
```

Warning:
To prevent unwanted loss of information, be careful with the `replaceControl` parameter values `"replaceUpdate"` and `"replaceSilently"`. Moreover, do NOT set the `deleteControl` parameter to the value `"erase"`.

Move files and folders

The function **`FFire_Move*`** lets you move files and folders.

Syntax:
```
FFire_Move ( path ; destinationPath {; replaceControl ;
deleteControl})
```

Parameters:
`path` – Specifies the source file or folder to be moved in FileMaker path format (filewin or filemac).

`destinationPath` – The target path of the file or folder to be moved. Please note that if the name of the file or folder does not match the name indicated by the `path` parameter the item will be renamed.

`replaceControl` – This parameter is optional. It specifies the behavior of the plug-in for situations when the destination file or folder already exists. The value `"replaceError"` (default) makes the plug-in return an error code when the destination file or folder already exists. The specified item is not processed. The value `"replaceSkip"` skips all existing destination items automatically and only replaces items that do not exist in the destination yet. The value `"replaceAsk"` shows a Replace File or Replace Folder dialog box and lets the user choose whether to replace conflicting originals. The value `"replaceUpdate"` compares conflicting items by their modification timestamp and replaces outdated items. Items that do not cause a conflict are processed as well. The value `"replaceSilently"` replaces all conflicting items without error or warning.

`deleteControl` – This parameter is optional. It specifies whether to move replaced items to trash or erase them. By default, this parameter is set to `"trash"`. Set this parameter to `"erase"` to delete replaced items without moving them to trash.

Result:
This function returns a result code which indicates if it succeeded or if errors occurred. When moving a folder that contains several files please note the following: The plug-in moves all files contained within the folder sequentially. Thus, it may happen that not all files are moved due to replace errors (depending on the `replaceControl` parameter value) or other limitation such as privileges.

The result returned is formatted according to the following pattern. Please note that the “failed item” may be a source or destination item.

```
XXX (Description)
(Failed item follows, if any)
…
<start position> <length> (Processed items)
<start position > <length> (Skipped items)
<start position > <length> (Replaced items)
(Items processed follow, if any)
…
(Items skipped before any error follow, if any)
…
(Items replaced before any error follow, if any)
…
```

Warning:
To prevent unwanted loss of information, be careful with the `replaceControl` parameter values `"replaceUpdate"` and `"replaceSilently"`. Moreover, do NOT set the `deleteControl` parameter to the value `"erase"`.

Delete files and folders

The function **`FFire_Delte*`** lets you delete files and folders. Before using this function, please ensure that you read all information in this section including the warning below.

Syntax:
`FFire_Delete ( path {; deleteControl })`

Parameters:
`path` – Specifies the path of a file or folder to be deleted in FileMaker path format (filewin or filemac). If a folder is specified all files (including files in sub folders) will be deleted.

`deleteControl` – This parameter is optional. It specifies protection features to prevent unwanted loss of information. The value "`trashAsk`" (default) shows a Delete File or Delete Folder dialog box. Specified items are moved to trash if the user confirms this. The value "`trashSilently`" moves selected items to trash without showing a warning dialog. The value "`eraseAsk`" shows a warning dialog and erases specified items *without* moving them to trash if the user confirms this. The value "`eraseSilently`" erases specified items *without* moving them to trash and without showing a warning dialog.

Result:
This function returns a result code which indicates if it succeeded or if errors occurred. When deleting a folder that contains several files please note the following: The plug-in deletes all files contained within the folder sequentially. Thus, it may happen that not all files are deleted due to various reasons (user decides not to delete specific files or user account does have sufficient privileges).

The result returned is formatted according to the following pattern.

```
XXX (Description)
(Failed item follows, if any)
…
<start position> <length> (Processed items)
<start position > <length> (Skipped items)
<start position > <length> (Replaced items)
(Items processed follow, if any)
…
(Items skipped before any error follow, if any)
…
(Items replaced before any error follow, if any)
…
```

Warning:
Please be very careful when using this function. To prevent unwanted loss of information, do not use this function to delete entire folders. Instead, delete selected files sequentially. Do not use the `deleteControl` parameter values "`eraseAsk`" and "`eraseSilently`" unless you are absolutely sure about what you are doing.

CHAPTER 10
Find Items*

FileFire Advanced lets you find files and folders that match specific criteria. Use this feature to provide file find tools to users or import/reference files that match specific criteria (e.g. import all new files from a project folder that have been created or modified since last database update).

Find files and folders

The function **`FFire_GetFindResult*`** returns all items that match specific find criteria.

Syntax:
`FFire_GetFindResult ( criteria {; path })`

Parameters:
`criteria` – This parameter specifies find criteria. Find criteria are formatted according to the following pattern: `Criterion = Value`. Several find criteria can be concatenated using AND/OR operators.

Pattern example 1: `(Criterion1 = Value1) OR (Criterion2 = Value2)`
Pattern example 2: `(Criterion1 = Value1) AND (Criterion2 = Value2)`
Pattern example 3: `((Criterion1 = Value1) AND (Criterion2 = Value2)) OR ((Criterion3 = Value3) AND (Criterion 4 = Value4))`

In addition, this function supports a NOT operator (`!=`).

Pattern example 4: `(Criterion1 != Value1)`

The value string supports the following wildcards: ? is used to specify any single character whereas * is used to specify any sequence of characters. For expressions, all of the following comparators are supported: =, ≠, !=, <>, >, >=, ≥, <, ≤ and <=.

The following listing provides all supported criterion/value pairs.

- Criterion: `"type"`, values: `"file"` or `"folder"` (includes volumes).
- Criterion: `"fmType"`, values: `"image"`, `"quicktime"`, `"file"`, `"alias"`, `"folder"` or “`volume`”.
- Criterion: `"name"`, value: item name, wildcards `?` and `*` are supported.
- Criterion: `"size"` (files only), value: file size in bytes.
- Criterion: `"kind"`, value: file kind description (e.g. `"Microsoft Word Document"`).
- Criterion: `"readOnly"`, values: `1` (read-only enabled) or `0` (read-only disabled).
- Criterion: `"hidden"`, values: `1` (hidden) or `0` (not hidden).
- Criterion: `"archive"` (Windows only), values: `1` (archive enabled) or `0` (archive disabled).
- Criterion: `"created"`, value: creation date or timestamp.
- Criterion: `"modified"`, value: modification date or timestamp.
- Criterion: `"accessed"`, value: access date or timestamp.

Dates and timestamps can be passed to the plug-in in the operating system format or in the default format of the database file. Value strings should be quoted properly using the FileMaker function *Quote*. Example:

```
FFire_GetFindResult ( "name = " & Quote(*Dacons*) ; "" )
```

`path` – This parameter is optional. It is used to restrict the find operation to a certain folder or volume. Leave this parameter empty to search for files on all volumes currently mounted (default). Please note that finding files by criteria my take some time. Thus, restricting the find operation to certain folders is highly recommended. The find operation will include all sub folder of the specified path.

Result:
The function returns paths of all files that match the specified find criteria separated by carriage return. Invoke the function `FFire_GetLastResultCode` to retrieve a result code which indicates whether the operation succeeded or failed. If it failed details can be referred from the error code.

Example:
The following function call finds all files that contain the word `Memo` in their name, have been created on or after January 1st, 2007 and are located in the folder `Business` or a sub folder of it:

```
FFire_GetFindResult ( "(type = " & Quote(file) & ") AND (name = " &
Quote(*Memo*) & ") AND ( created ≥ 01.01.2007)" ;
"filemac:/Harddisk/Business/" )
```

CHAPTER 11
Create Folders and Aliases*

Using FileFire Advanced you can create folders. Moreover, FileFire Advanced lets you create aliases (Mac OS) and shortcuts (Windows) to files and folders.

Create folders

The FileFire Advanced function **`FFire_CreateFolder*`** creates a new folder in a specific location.

Syntax:
`FFire_CreateFolder ( folderPath )`

Parameters:
`folderPath` – This parameter specifies the path of the folder to be created in the FileMaker path format (filewin or filemac). If necessary, the plug-in creates all folders of the specified path. Please ensure that the folder path specified ends with a slash (“/”).

Result:
This function returns a result code which indicates if it succeeded or if errors occurred. Please note that this function fails if a folder at the specified location already exists. In such cases the specified folder is present. However, you cannot be sure that it is empty. Invoke the function `FFire_GetFolderItems` to check this.

Example:
In this example the following folder exists on a system:

```
filemac:/Harddisk/Business/
```

The following function call results in two folders being created. The folder `Letters` will be created in the folder `Business`. The folder `Q107` will be created in the folder `Letters`.

```
FFire_CreateFolder ( "filemac:/Harddisk/Business/Letters/Q107/" )
```

Create aliases

An alias (Mac OS) or shortcut (Windows) is a link to file or folder stored elsewhere on a disk or network volume. The FileFire Advanced function **`FFire_CreateAlias*`** crates such an alias/shortcut.

Syntax:

```
FFire_CreateAlias ( sourcePath {; aliasPath })
```

Parameters:

`sourcePath` – This parameter specifies the path of the file or folder the alias points to in FileMaker path format (filewin or filemac).

`aliasPath` – This parameter is optional. It specifies the path of the alias/shortcut including its name. By default, the alias is named by the operating system (in most cases it will be named like the source) and placed in the same folder as its source. Please note that depending on user settings the operating system may or may not keep track of aliases when sources are moved or renamed.

Result:

This function returns the path of the alias created. Invoke the function `FFire_GetLastResultCode` to retrieve a result code which indicates whether the operation succeeded or failed. Please note that the function will fail if an alias with at the specified location (parameter `aliasPath`) already exists.

Example:

This example assumes that a source folder with the following path is available:

```
filewin:/C:/Holiday Photos/Cancun06/
```

The following function call places an alias/shortcut to this folder on the Desktop and names it `Recent Holiday Photos`:

```
FFire_CreateAlias ( "filewin:/C:/Holiday Photos/Cancun 06/" ;
FFire_GetSystemFolder ( "desktop" ) & "Recent Holiday Photos" )
```

CHAPTER 12
Work with ZIP Archives*

FileFire Advanced lets you compress any files and folders to ZIP archives. Moreover, you can decompress any ZIP archive to a specified location from within FileMaker. ZIP is a cross-platform compression format. You can deploy ZIP archives to Mac OS X, Windows XP and Windows Vista users.

Archive files and folders

The function **`FFire_Zip*`** compresses a specified file or folder and creates a ZIP archive.

Syntax:

```
FFire_Zip ( path {; archivePath ; compressionRate ;
replaceControl })
```

Parameters:

`path` – This parameter specifies the path of the file or folder to be archived in FileMaker path format (filewin or filemac).

`archivePath` – This parameter is optional. It specifies the destination path of the archive to be created. By default, the zip archive will be located in the same folder as the item (file or folder) that has been archived and it will be named like the archived item (extension will be *.zip*).

`compressionRate` – This parameter is optional. It specifies the compression rate of the ZIP archive. The compression rate can be `"maxCompression"` (default), `"normalCompression"` or `"lowCompression"`. Decrease the default compression rate to increase the compression speed.

`replaceControl` – This parameter is optional. It specifies the behavior of the plug-in in situations when an archive with the specified name already exists at the specified location. The value `"replaceError"` (default) makes the plug-in return an error code in such cases. The archive is not created. The value `"replaceAsk"` shows a Replace File or Replace Folder dialog box and lets the user choose whether to replace the existing archive. The result returned by the plug-in function indicates whether the archive has been created or not. The value "`replaceSilently`" replaces all an existing archive without error or warning.

Result:
This function returns the path of the archive that has been created. Invoke the function `FFire_GetLastResultCode` to retrieve a result code which indicates whether the operation succeeded or failed.

Warning:
When replacing an archive please note that the existing archive is erased and NOT moved to trash! To prevent unwanted loss of information, do not use the `replaceControl` parameter value "`replaceSilently`".

Example:
In the following example the folder `Letters` is archived. The archive will be named like the folder and located in the same folder. If an archive with the same name already exists in this location the user will be asked whether to replace the existing archive.

```
FFire_Zip ( "filemac:/Harddisk/Letters/ ; "" ; "maxCompression" ;
"replaceAsk" )
```

Decompress archives

The FileFire Advanced function **`FFire_Unzip*`** decompresses the contents of a ZIP archive to a specified location.

Syntax:
```
FFire_Unzip ( archivePath {; destinationPath ; replaceControl })
```

Parameters:
`archivePath` – This parameter specifies the path of the archive to be decompressed in FileMaker path format (filewin or filemac).

`destinationPath` – This parameter is optional. It specifies the destination path for the archive content. If no destination path is indicated the plug-in acts according to the following default rules: Archives that contain only one file (without any relative path information stored for that file in the archive) are decompressed to the same folder that stores the archive. Archives that store several files (or contain relative path information for a single stored file) are decompressed to a new sub folder that is created in the folder that stores the archive. This folder is named like the archive. Specifying a destination path is recommended.

`replaceControl` – This parameter is optional. It specifies the behavior of the plug-in in situations when destination files already exist. The value "`replaceError`" (default) makes the plug-in return an error upon the first existing destination file that is detected. The value “`replaceSkip`" skips all existing destination items automatically and only decompresses items that do not exist in the destination yet. The result returned by the plug-in function indicates which items have not been decompressed. The value `"replaceAsk"` shows a Replace File or Replace Folder dialog box and lets the user choose whether to replace the destination file. The result returned by the plug-in function indicates which files have been replaced. The value `"replaceUpdate"` compares conflicting files by their modification timestamp and replaces outdated files only. The value `"replaceSilently"` replaces all items without error or warning.

Result:
This function returns a result code that indicates whether the operation succeeded or failed. When decompressing archives with several files please note the following: The plug-in decompresses all files contained within the archive sequentially and then copies the files to their destination. Thus, it may happen that not all files are copied due to replace errors (depending on the `replaceControl` parameter value or restrictions). If not all items could be copied, the result contains a list of items that have not been copied. If files have been replaced they are provided as a separate result listing.

Warning:
When replacing files please note that they are deleted and NOT moved to trash! To prevent unwanted loss of information, do not use the `replaceControl` parameter values `"replaceUpdate"` and `"replaceSilently"`.

Example:
In the following example the archive `Tools.zip` is decompressed to a folder `Tools` on the desktop. To prevent unwanted loss of information the user is asked in case of file name conflicts.

```
FFire_Unzip ( "filewin:/C:/Downloads/Tools.zip" ;
FFire_GetSystemFolder ( "desktop" ) & "Tools/" ; "replaceAsk" )
```

CHAPTER 13
Status Dialog*

File operations may require a couple of seconds or even minutes to complete depending on the type of operation and the system performance. FileFire Advance lets you show a status dialog during file operations. This status dialog contains a text label (English) and a progress bar.

The status dialog can be used for the following FileFire functions: `FFire_Copy`, `FFire_Move`, `FFire_Delete`, `FFire_Zip` and `FFire_Unzip`. The following functions support an “intermediate” progress bar that keeps moving during the operation but does not indicate the progress: `FFire_GetFindResult`, `FFire_GetFolderItems`, `FFire_GetTextFile` and `FFire_WriteTextFile`.

Setup the status dialog

Prior to showing the status dialog using the function `FFire_ShowStatusDialog` it is configured by **`FFire_SetupStatusDialog*`**. You can also change the settings of the status dialog with this function while it is shown. Therefore, set a non-text (e.g. 0) value for every parameter that should *not* be updated.

Syntax:

```
FFire_SetupStatusDialog ({ dialogTitle ; allowCancel ;
cancelButtonLabel ; textMode ; customText ; progressUnits })
```

Parameters:

`dialogTitle` – This parameter is optional. It specifies the title of the dialog. By default, there is no text in the title bar of the dialog.

allowCancel – This parameter is optional. It specifies whether users are allowed to cancel the current plug-in operation using a Cancel button. By default, this parameter is set to `"allowCancelOn"` so that a Cancel button is shown. Please note that certain plug-in operations cannot be cancelled. The Cancel button will be disabled in such situations automatically. If you allow users to cancel operations plug-in function may not complete and return an error code accordingly. Set this parameter to `"allowCancelOff"` to hide the Cancel button.

`cancelButtonLabel` – This parameter is optional. By default, the Cancel button shows the label “Cancel”. Use this parameter to rename the button. However, please note that the button’s behavior does not change. Use this parameter to translate the label into a language other than English.

`textMode` – This parameter is optional. The status dialog supports three different text modes. By default, this parameter is set to `"autoText"`. In this text mode, an automatic status text is shown in the dialog providing information about the current plug-in operation. Set this parameter to `"customText"` and use the `customText` parameter to show a custom status text in the dialog. The default status text can also be combined with a custom text. Therefore, set the parameter `textMode` to `"combinedText"`.

`customText` – This parameter is optional. Use it to specify a custom status text that is shown in the dialog. Set the `textMode` parameter to `"customText"` or `"combinedText"` when using this parameter.

`progressUnits` – This parameter is optional. Plug-in functions that support the progress bar of the status dialog window manage the progress bar automatically. However, to group several plug-in function calls into *one* operation use this parameter to specify the number total progress bar units (e.g. total number of files to be copied or moved from different sources). This parameter is also used to reset the progress bar of the dialog from earlier dialog sessions and specify the number of progress bar units for the following dialog session. After dialog setup, the plug-in will manage the progress bar automatically.

Result:
This function returns a result code which indicates if it succeeded or if errors occurred.

Show the status dialog

The function **`FFire_ShowStatusDialog*`** shows the status dialog.

Syntax:
`FFire_ShowStatusDialog`

Result:
This function returns a result code which indicates if it succeeded or if errors occurred.

Hide the status dialog

To hide the status dialog invoke the function **`FFire_HideStatusDialog*`**.

Syntax:
`FFire_HideStatusDialog`

Result:
This function returns a result code which indicates if it succeeded or if errors occurred.

CHAPTER 14
Additional Functions

FileFire provides additional plug-in functions that are described in this chapter. These functions enable you to retrieve result codes and configure the log engine of the plug-in that writes errors or activities to a log file. Moreover, FileFire enables you to check the exact version number of the plug-in installed and unlock the trial version using with a registration name and a registration code from script so that no end-user interaction is required to unlock a trial version. In addition, FileFire Advanced provides a debug mode that assists you with debugging complex file management solutions.

Retrieve last result code

The function **`FFire_GetLastResultCode`** returns the result code of the last FileFire function that has been invoked. Please refer to p. 14 for a table of all result codes.

Syntax:
`FFire_GetLastResultCode`

Parameter:
No parameter needed.

Parameter:
This function returns the result code of the last plug-in function that has been invoked prior to `FFire_GetLastResultCode`.

Result:
The result code for the last plug-in operation is returned together with a short description. If the result returned is empty no plug-in function been invoked prior to `FFire_GetLastResultCode`.

Example:
The function `FFire_GetAttribute` returns content (attribute value). It does not return error codes. If an error occurs, this function returns an empty result. In the following example, the specified file cannot be found.

```
FFire_GetAttribute ( "filewin:/c:/Letters/Mary.doc" ; "size" )
```

Since the file cannot be found an empty result is returned. To find out why the function `FFire_GetAttribute` failed the function `FFire_GetLastResultCode` is invoked.

```
FFire_GetLastResultCode
```

In this example case it returns:

```
-053 (Item not found)
```

Configure the log engine

FileFire provides a log engine that can be configured using this function. Using the function **`FFire_SetupLog`** the log engine can be disabled or enabled. Moreover, it can be configured to log all plug-in activities or errors only. Use the log engine to debug FileMaker solutions that use the FileFire plug-in. When contacting the Dacons Support with a technical request, please include the log file.

Syntax:
`FFire_SetupLog ( control {; mode ; maxSize })`

Parameters:
`control` – This parameter is required. It switches logging on or off using one of the following commands: `"off"` disables the log engine completely. `"overwrite"` switches logging on and replaces the log file for every FileMaker session. "`append`" switches logging on as well, and appends content to the log file for every session.

`mode` – This parameter is optional. It specifies the log level. To log plug-in errors only set this parameter to `"errors"` (default). To log all activities of the plug-in, set the value `"activities"`.

`maxSize` – This parameter is optional. It specifies the maximum size of the log file in KB. By default (empty parameter), the value 256 is used. During a session, the maximum size of the log file may be greater than specified maximum (up to 1.5 times) to avoid speed degradation. On shutdown the plug-in adjusts the log file size according to specified maximum.

Result:
This function returns a result code which tells you if the operation succeeded or if errors occurred.

Example:
The following function call makes the log engine start a new log file for every FileMaker session. It specifies that not only errors but all plug-in activities should be logged and that the maximum size of the log file should not be more than 1 MB.

```
FFire_SetupLog ( "overwrite" ; "activities" ; 1024 })
```

Retrieve plug-in version

The function **`FFire_GetVersion`** returns the version of the installed FileFire plug-in. Use this function to check if the plug-in is installed when your database solution starts (start-up script). The version information provided by this function can also be used for the AutoUpdate plug-in which pushes updated versions of other plug-ins to all FileMaker network clients automatically. Review the FileMaker documentation for more information about the AutoUpdate plug-in.

Syntax:
`FFire_GetVersion ({ control })`

Parameters:
`control` – This parameter is optional. It can be used to customize the content returned by the plug-in function. Leave this parameter empty to retrieve all of the following items. To retrieve only a specific version information item, set this parameter to one of the following values: `"version"` returns the exact version of the plug-in installed. In most cases you will pass this value to the plug-in to retrieve the version number. `"product"` returns the name of the product which is “FileFire Express” or “FileFire Advanced” depending on the edition of the plug-in currently installed. `"platform"` returns the operating system the plug-in is running on and `"copyright"` returns copyright information of the plug-in.

Result:
This function returns the plug-in version description including all items specified by the `control` parameter.

Register the plug-in

To remove all trial limitations from the plug-in it has to be registered using the registration data you receive from Dacons after purchasing a FileFire license. The plug-in can be registered manually using the preferences dialog (Go to: FileMaker Application Preferences ► Plug-Ins ► FileFire Express or FileFire Advanced).

To avoid manual plug-in registration when shipping your database solution to a client or distribute a FileMaker Runtime application with FileFire you can also register the plug-in from the start-up script of your solution by invoking the function **`FFire_RegisterSession`** with your registration data. This function has to be invoked ***prior to any other function***. Note that registration data from script will not be stored so registration from script has to be invoked every time a solution starts.

Syntax:
`FFire_RegisterSession ( username ; userCode )`

Parameters:
`userName` – Sets the user name you receive from Dacons after purchasing a FileFire license.

`userCode` – Use this parameter to pass the registration code to the plug-in which you receive from Dacons after purchasing a FileFire license.

Please note that you cannot use a FileFire Express license with FileFire Advanced and vice versa. You can upgrade from FileFire Express to FileFire Advanced anytime.

Result:
This function returns a result code which tells you if the operation succeeded or if errors occurred. Please contact the Dacons Support at http://www.dacons.net/support if you registration code is rejected for any reason.

For permanent plug-in registration on the FileMaker Server side **`FFire_RegisterServer`** function should be used. This function has to be invoked on the FileMaker Server side ***only***, using the FileMaker Server Console script scheduler. Function syntax and result codes are identical to **`FFire_RegisterSession.`**

Syntax:
`FFire_RegisterSever ( username ; userCode )`

Configure debug mode*

The FileFire Advanced function **`FFire_SetupDebugMode*`** lets you specify whether the plug-in will show a message box whenever an error code is generated. Enable this feature to debug complex file management solutions.

Syntax:
`FFire_SetupDebugMode ( control )`

Parameter:
`control` – This parameter controls the debug behavior of the plug-in. Set this parameter to `"on"` to enable debugging. This means that a message box is shown whenever an error code is generated. The plug-in will not continue before the message is confirmed. Set this parameter to `"off"` to disable debugging. By default, the debug mode is disabled.

Result:
This function returns a result code which indicates if it succeeded or if errors occurred.