ScriptFire

FileMaker Plug-In Manual

Dacons

This manual assumes that you have elementary knowledge of FileMaker and you know how to use plug-in functions of this database platform.

Please send feedback to info@dacons.net

Website: http://www.dacons.net

May 01, 2006

CONTENTS

CHAPTER 4

CHAPTER 5

CHAPTER 6

CHAPTER 7

CHAPTER 8

FUNCTION INDEX

FUNCTION OVERVIEW
ScriptFire Features

ScriptFire is a powerful FileMaker plug-in that enables you to trigger scripts anytime and in any situation. Use ScriptFire to schedule scripts for hourly, daily, weekly, monthly or even yearly execution. Open and close FileMaker databases based on schedules. Have ScriptFire trigger your script to check user input after leaving a field or record. Define events that trigger scripts when users access certain layouts or switch windows. By providing script scheduling and event-based script execution, ScriptFire gives you full control over your FileMaker solutions and enables you to create features not seen before.

Function overview

- Trigger scripts based on secondly, minutely, hourly, daily, weekly, monthly and even yearly schedules
- Have scripts triggered once at an absolute or relative point in time
- Install schedules that trigger your scripts repetitively according to custom intervals
- Open and close FileMaker databases automatically based on the same type of schedules – ScriptFire even opens password protected and shared FileMaker files
- Trigger scripts when certain layouts are accessed
- Control FileMaker windows by invoking a script when a certain window comes to front
- Trigger a script when exiting a field or when creating, viewing or exiting records
- Schedule the power management of the system that runs your FileMaker solution – put the computer to sleep or even shut it down at a specific point in time
- Manage installed tasks of all types – suspend, resume or remove tasks and control settings

Possible solutions

- Schedule database backups
- Schedule business process related tasks such as consolidations or notifications
- Implement reminder functionality for calendar solutions
- Schedule email sending and receiving (e.g. using Dacons mail.it)
- Validate user input against a script that is triggered when the user leaves a field
- Refresh content when layouts or windows are accessed
- Control computer power management out of FileMaker

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 ScriptFire using the *Quick Start* file that comes with this plug-in.

About this manual

This manual is written in an easy-to-follow style. **Chapter 1** (→ ScriptFire Basics, p. 9) explains schedule and file formats which are the basics of all ScriptFire functions. Once you understood these basics you can skip to any chapter and focus on the information you currently require.

Chapter 2 (→ Schedule-Based Script Tasks, p. 21) provides all information you need to create your own schedule-based script tasks. **Chapter 3** (→ Event-Based Script Tasks, p. 29) focuses on a different type of tasks that can be installed using ScriptFire. Event-based tasks enable you to trigger scripts when layouts or windows are accessed or fields are left. Read **Chapter 4** (→ Open and Close File Tasks, p. 24) to open and close database files based on schedules. **Chapter 5** through **Chapter 8** provides information on managing ScriptFire tasks and using advanced administration features.

ScriptFire support FileMaker 5.x and 6 as well as FileMaker 7 and 8. These two FileMaker platforms use different styles to format plug-in function calls. Throughout this manual ScriptFire functions will be formatted in the FileMaker 7 and 8 style. However, after reading **Chapter 1** (→ ScriptFire Basics, p. 9), you will be able to apply the explanations and examples of other chapters to FileMaker 5.x and 6 solutions.

Software requirements for Windows

ScriptFire is available in two different versions for Windows. One version supports FileMaker 5.x and 6 (Pro, Developer and Runtime) on the following Windows platforms: Windows XP, Windows 2000 and Windows 98/ME.

The other version of ScriptFire supports the improved plug-in interface of FileMaker 7 and 8. (Pro, Developer/Advanced and Runtime) on the following Windows platforms: Windows XP and Windows 2000.

When running ScriptFire under Windows 2000 you need **Service Pack 4** for Windows 2000.

Software requirements for Macintosh

ScriptFire is available in two different versions for Macintosh. One version supports FileMaker 5.x and 6 (Pro, Developer and Runtime) on the following Macintosh platforms:
Mac OS 9.2 or higher including Mac OS X 10.2 and higher.

The other version of ScriptFire supports the improved plug-in interface for FileMaker 7 and 8 (Pro, Developer/Advanced and Runtime) on the following Macintosh platforms:
Mac OS X 10.2 and higher.

Installing the ScriptFire FileMaker plug-in

Before installing the ScriptFire plug-in, ensure you select the correct version from the download package according to your operating system and FileMaker version.

ScriptFire is available in four versions:

- Windows for FileMaker 5.x and 6
- Windows for FileMaker 7 and 8
- Macintosh for FileMaker 5.x and 6
- Macintosh for FileMaker 7 and 8

Ensure that FileMaker is closed. Next, copy the ScriptFire plug-in into the following folder:

Windows:
FileMaker 5.x and 6: FileMaker *System* folder
FileMaker 7 and 8: FileMaker *Extensions* folder

Macintosh:
FileMaker 5.x and 6: *FileMaker Extensions* folder
FileMaker 7 and 8: *Extensions* folder

The folder you have to look for is located in the FileMaker application folder.

Hands-on examples

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

CHAPTER 1
ScriptFire Basics

This chapter enables you to use ScriptFire functions with FileMaker and explains the concepts of ScriptFire tasks. In addition, you will learn about schedule and file formats which are used for most ScriptFire functions. After reading this chapter advance to any of the other chapters of this manual that provide the information you currently require.

The concept of ScriptFire tasks

When working with ScriptFire everything is a “task”. ScriptFire manages scripts that are scheduled for future execution or events that are attached to windows or layouts as “tasks”. When closing FileMaker, ScriptFire stores tasks automatically and restores them when FileMaker is opened.

ScriptFire plug-in functions are used to install tasks. When installing a schedule-based task you can define whether the task will be set up for one time execution or for repeating execution. In addition, you can set an absolute or relative point in time when this task will expire. If you do not define an expiration point for a repeating task it will ‘live’ forever. **Chapter 6** (→ Managing Tasks, p. 38) provides information on managing ScriptFire tasks. This chapter covers how to suspend, resume and remove installed tasks.

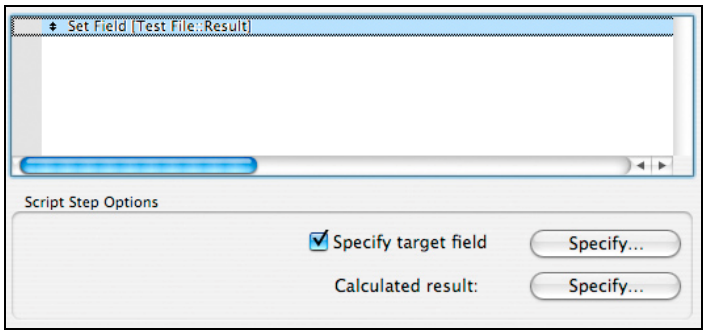
When reading **Chapter 2** through **Chapter 5** just keep in mind that the ScriptFire functions discussed in these chapters install tasks that can be managed using other functions.

How to use ScriptFire functions

Since ScriptFire 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, they are not part of the actual FileMaker application. In order to use the external functions provided by a plug-in it must always be installed and enabled in the FileMaker application preferences.

To access external functions provided by installed plug-ins the FileMaker calculation editor is used. FileMaker shows this dialog whenever a calculation has to be defined (e.g. for calculated fields, validation calculations etc). In most cases you will trigger ScriptFire functions from a script. The easiest way to perform an external function call from a script is to add the script step *Set Field* to your script.

Next, you specify the target field which will contain the result of the operation. The result of ScriptFire operations is a simple result code which tells you if a function could be executed successfully or if errors occurred. Therefore, you should setup a global text field in your solution and call it *Result*. Specify the field *Result* as target field of the calculation.

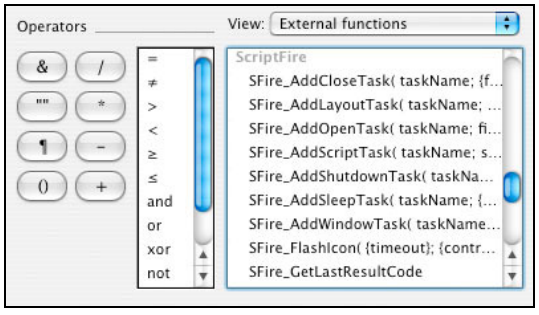


Set Field command in ScriptMaker

Click the *Specify* button (the second *Specify* button in FileMaker 7 and 8) to open the calculation editor.

As mentioned earlier plug-in functions can also be triggered from other places than the ScriptMaker. To trigger a ScriptFire function from a field validation calculation (e.g. to trigger a script that checks user input) open the calculation editor from the field setup dialog. In any case you start working with the plug-in by opening the calculation editor of FileMaker.

In the top right corner of the calculation editor dialog you find a drop-down menu called *View*. Change to the section *External functions*. All ScriptFire functions are listed in the section *ScriptFire*. If you have additional plug-ins installed you might have to scroll down to the *ScriptFire* section of the list. If the ScriptFire 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 ScriptFire is installed as described earlier and that it is enabled in the FileMaker application preferences.



ScriptFire functions in the Calculation Editor

Double-click on the ScriptFire function in the list which you would like to use. Next the parameters of the selected function have to be specified.

Function parameters

In order to tell the ScriptFire plug-in what it should do when the function is triggered, its *parameters* need to be specified. Parameters tell ScriptFire which script to trigger or which file to open, for example. Most ScriptFire functions have several parameters.

FileMaker 7 and 8:
After copying a function to the text box of the calculation editor, FileMaker 7 and 8 will show the name of the plug-in function that has been selected and a text label for each parameter of this function.

The following example shows the ScriptFire function *SFire_AddScriptTask* that has been added to the calculation editor:

```
SFire_AddScriptTask( taskName; script; {scriptParameter};
{fileLocation}; {fileAccount}; {filePassword}; {schedule};
{timeout}; {taskState} )
```

FileMaker 5.x and 6:
When working with FileMaker 5.x or 6 you will notice that these versions of FileMaker do not support parameter labels. After copying a ScriptFire function to the calculation editor you will always see a single label that represents *all* parameters of the selected function

```
External ( "SFire-AddScriptTask"; parameter )
```

Under FileMaker 5.x and 6 the *parameter* label will be replaced by different parameter values. Parameters are passed as concatenated text, with each required parameter value separated by a pipe character (“|”).

To type the pipe character (“|”) on an US-keyboard press Shift + ”\”. The location of the pipe key can be different if you are using another keyboard layout (e.g. French or German). However, in most cases you will find the pipe character printed on one of the keys. Sometimes the vertical line is shown as two small lines, almost like a colon (“:”). This does not make any different since the character you type will always look like “|”.

As seen earlier the ScriptFire function *SFire_AddScriptTask* has nine parameters. Since FileMaker 5.x and 6 protect files only by a password the `fileAccount` parameter is not available under FileMaker 5.x and 6. Thus, the function *SFire_AddScriptTask* has only eight parameters on these FileMaker platforms.

To pass a value for each of the parameter to FileMaker 5.x or FileMaker 6 format the function call in the calculation editor as follows:

```
External ( "SFire-AddScriptTask";
taskName          & "|" &
script            & "|" &
scriptParameter   & "|" &
fileLocation      & "|" &
filePassword      & "|" &
```

```
schedule          & "|" &
timeout           & "|" &
taskState         )
```

Note that each parameter like `taskName` and `script` represents a text value. Thus, a parameter value has to be the name of the text field or a text value itself. If it is a text value, the parameter must be enclosed in quotation marks (`"value"`). The example above assumes that the parameters `taskName`, `script` etc. have been defined as text fields in the file.

Important: Ensure that you do not enclose space characters before or after the pipe separators. Always use “|” as separator, neither “ |” nor “| “. Unneeded space characters will be considered as part of a parameter by the plug-in which can result in invalid function calls.

It is recommended that you format the function call as shown above. One line is used for each parameter. This gives a much better overview than a single line of text. However, the plug-in does not require you to format function calls this way.

Required and optional parameters

Most functions have some parameters that are required and some that are optional. When invoking a ScriptFire 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. You will learn about default values later when specific plug-in functions are discussed.

Mandatory 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 which you would not like to use in your function call.

FileMaker 7 and 8:
After copying a function to the calculation editor under FileMaker 7 or 8 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 “”. To skip all optional parameters from the ScriptFire function *AddScriptTask* format the function call as follows:

```
SFire_AddScriptTask( taskName; script; ""; ""; ""; ""; ""; ""; "" )
```

The labels `taskName` and `script` represent text values that are passed to the plug-in for these parameters.

FileMaker 5.x and 6:
When working with FileMaker 5.x or 6 optional parameters can also be skipped by passing empty text valued to the plug-in. The following example skips `scriptParameter` as one of the optional function parameters:

```
External ( "SFire-AddScriptTask";
taskName          & "|" &
script            & "|" &
""                & "|" &
fileLocation      & "|" &
filePassword      & "|" &
schedule          & "|" &
timeout           & "|" &
taskState         )
```

Note that the pipe separator (“|”) is *not* omitted for the skipped parameter. So if all parameters except for the last are to be skipped the function call would look as follows:

```
External ( "SFire-AddScriptTask";
taskName          & "|" &
script            & "|" &
""                & "|" &
""                & "|" &
""                & "|" &
""                & "|" &
""                & "|" &
taskState         )
```

Of course, the function call can also be formatted like this:

```
External ("SFire-AddScriptTask";
taskName & "|" & script & "||||||" & taskState )
```

FileMaker 5.x and 6 simplify the handling of optional parameters with the following rule: Optional parameters at the end of a function call can be skipped. In such a case the pipe separators (“|”) can be omitted as well. So if all optional parameters – including the last one – of the function *AddScriptTask* are to be skipped the function call would look as follows:

```
External ( "SFire-AddScriptTask"; taskName & "|" & script )
```

Throughout this manual ScriptFire functions will be formatted in the FileMaker 7 and 8 style. If you are not sure how to format a function call under FileMaker 5.x and 6 please refer to the example above.

Result codes

Most plug-in functions that are called return a result code and a short description. The result code indicates if the plug-in function succeeded or failed. If it failed the code also provides details about the reason. You can evaluate result codes with a FileMaker script and perform certain activities (e.g. show an error message to the user).

The following table lists all result codes returned by the plug-in.

Result Code	Explanation
000	“OK” (plug-in function succeeded)
-001	"Internal error" (contact the Dacons Support at www.dacons.net/support if you receive this error code)
-002	"Incorrect parameter" (ensure that all parameters of the function you invoke are set and formatted according to the specifications)
-003	"Task not usable, schedule expired” (defined schedule is in the past) or “Task not usable, file location not applicable” (the specified file location cannot be used on the current operating system)
-004	"File not found" (the file indicated in the function call could not be found)
-005	"Script not found" (the script indicated in the function call could not be found)
-006	"Task not found" (the task indicated in the function call could not be found)
-007	"Task event expired" (the task to be executed already expired)
-008	"System does not support power management" (the current system cannot be set to sleep or shut down using ScriptFire)
-009	“No script is executing” (generated by *SFire_GetScriptParameter* if used in a calculation that is not invoked from a script)
-010	“Authorization failed” (Database file could not be opened because user-name/password authorization to open the file failed)
-011	“Incorrect registration data; visit www.dacons.net/support for help”
-012	“Load environment failed” (plug-in preferences could not be read)
-013	“Log file not opened” (ensure that the log file is not opened by another application and that log settings are correct)
-014	“Log file corrupted” (plug-in will erase the log file)
-015	“End of log file reached”
-016	“Log data overflows” (generated by *SFire_ReadLog* if the log file contents returned had to be truncated due to limitations)

Schedule formats

ScriptFire provides tasks that enable you to invoke scripts or open and close FileMaker databases based on a schedule. A schedule is passed to the plug-in by a parameter called `schedule`. This section introduces all schedule formats that are supported by ScriptFire.

Date format:
The `schedule` parameter of ScriptFire functions requires you to pass date and time values in a specified format to the plug-in. This format is the same on all platforms. It is independent from the system’s date and time format settings.

Dates are formatted according to the MM-DD-YYYY pattern (MM = month, DD = day, YYYY = year). Note that a dash character (“-“) is used as separator. The month (MM) can be indicated using the number of the month or the first three characters of the English month’s name.

Examples:
`01-02-2006` (January 2nd, 2006)
`Jan-02-2006` (January 2nd, 2006)

Time format:
Time values are specified in the *24-hour format* according to the familiar HH:MM:SS pattern (HH = hours 0-23, MM = minutes 00-59, SS = seconds 00-59).

Examples:
`06:12:45`
`23:55:02`

If time is used to indicate an interval or a relative point in time instead of a indicating the time of a day a greater maximum greater than 23 is allowed for the hours section (HH).

One time absolute schedules:
Use one time absolute schedule to invoke a task once at a specific point in future which you define independently from the current point in time. To indicate one time schedules to ScriptFire the prefix "`once:`" used.

Example:
`once: 01-02-2006 13:15:00`
(task is scheduled for one time execution on January 2nd, 2006 at quarter past 1 pm)

One time relative schedules:
Use one time relative schedule to invoke a task once at a specific point in future which you define relatively to the current point in time. To indicate one time schedules to ScriptFire the prefix "once:" used. For relative one time schedules this prefix is followed by a plus character ("+") and a period in HH:MM:SS format. Hours (HH) can be greater than 23.

Examples:
`once: +00:00:30` (in 30 seconds)
`once: +48:05:00` (in 48 hours and 5 minutes)

Simple interval schedules:
Interval schedules invoke tasks regularly every couple of seconds, minutes or hours. ScriptFire also supports daily, weekly, monthly and yearly interval schedules. The simplest type of interval schedules use the prefix "`interval:`" followed by an interval period in HH:MM:SS format. Hours (HH) can be greater than 23. Use simple interval schedules to invoke a task every couple of seconds, minutes, or hours.

Examples:
`interval: 00:00:30` (every 30 seconds)
`interval: 12:05:00` (every 12 hours and 5 minutes)

Daily interval schedules:
To invoke a task once per day use a daily interval schedule. Daily interval schedules are in-

dicated by the prefix "`daily:`" followed by the time of the day indicated in HH:MM:SS format (HH: 0-23).

Examples:
`daily: 12:00:00` (every day at noon)
`daily: 20:00:05` (every day at 5 seconds past 8 pm)

Weekly interval schedules:
Use weekly interval schedules to invoke a task once per week. Weekly interval schedules are indicated by the prefix "`weekly:`" followed by an abbreviation of the day of the week this schedule refers to ("Mo", "Tu", "We", "Th", "Fr", "Sa" or "Su") and the time in HH:MM:SS format indicating the time of the day the task is to be triggered.

Examples:
`weekly: Mo 12:00:00` (every Monday at noon)
`weekly: Fr 18:15:00` (every Friday at quarter past 6 pm)

Monthly interval schedules:
This type of interval schedules invokes a task once per month. Monthly interval schedules are indicated by the prefix "`monthly:`" followed by a number indicating the day of the month and the time of the day in HH:MM:SS format.

Examples:
`monthly: 1 09:00:00` (every first day of a month at 9 am)
`monthly: 15 17:00:00` (every 15th of a month at 5 pm)

Note: Use `31` to refer to the last day of a month. Even if a month has only 30, 29 or even 28 days ScriptFire will invoke a task triggered for the 31st on the last day of that month.

Yearly interval schedules:
Use this type of interval schedules to invoke a task once per year. Yearly interval schedules are indicated by the prefix "`yearly:`" followed by a number or an abbreviation indicating the month, another number indicating the day of the month and a time in HH:MM:SS format specifying the time of the day task is to be triggered.

Examples:
`yearly: Dec 25 17:00:00` (every December 25th at 5 pm)
`yearly: 1 1 09:00:00` (every January 1st at 9 am)

Additional schedule options:
Every schedule except for schedules of the type "once" can be amended by "`start:`", "`expire:`" and "`repeat:`" commands that control when a schedule becomes active, when it becomes inactive and how often it is triggered at maximum. These commands are optional, of course.

To control that a schedule should not be active before a certain point in time use the "`start:`" command right after your schedule followed by a date and time indication.

Example:
`weekly: Mo 12:00:00 start: Jan-16-2006 12:00:00`
(every Monday at noon starting January 16th, 2006 at noon)

To deactivate a schedule at a certain point in time us the "`expire:`" command just like the "`start:`" command.

Example:
`weekly: Mo 12:00:00 expire: Jan-01-2007 00:00:00`
(every Monday at noon ending January 1st, 2007)

The "`repeat:`" command enables you to define a maximum number of task invocations for a schedule. When the maximum is reached the schedule will be deactivated automatically.

Example:
`weekly: Mo 12:00:00 repeat: 5`
(every Monday at noon, ending after 5 repetitions)

The commands discussed ("`start:`", "`expire:`" and "`repeat:`") can be combined within the same schedule to set a start and end point and and/or define a maximum number of repetitions.

Example:
```
weekly: Mo 12:00:00 start: Jan-16-2006 12:00:00
expire: Jan-01-2007 00:00:00 repeat: 5
```

In this example you will notice that the repeat condition will be reached before the end condition since there are more than five Mondays between January 16th, 2006 and January 1st, 2007. When combining "`expire:`" and "`repeat:`" conditions within the same schedule ScriptFire will consider whichever command is reached first. So in this case the schedule would be deactivated after 5 repetitions.

Please note that all keywords used in schedules (schedule types, month and week day names etc) are case-insensitive. So the following two examples yield the same result.

```
weekly: mo 12:00:00
WEEKLY: MO 12:00:00
```

File location formats

ScriptFire functions that schedule scripts or install events are related to certain FileMaker files. To tell ScriptFire the location of the file that is to be triggered for a task the `fileLocation` parameter is used.

Just like the *File Reference* dialog of FileMaker 7 and 8 the ScriptFire `fileLocation` parameter supports several file location formats that enable you to specify absolute, relative and even network file locations. The following file location formats are not only supported by

the FileMaker 7 and 8 version of ScriptFire but also by the version of the plug-in that works with FileMaker 5.x and 6.

Relative file location:
This file location format enables you to reference a file relatively to the current FileMaker file. Format the `fileLocation` parameter according to the following pattern:

```
file:fileName
file:directoryName/filename
file:../directoryName/filename
```

The first pattern `file:filename` refers to file that reside in the same directory as the current FileMaker file. To reference a file relatively to the current file that is located in a subdirectory use the second pattern `file:directoryName/filename`. You can specify a path of several directories, of course. The point at upper directories in a relative file location use "../" as demonstrated by the third pattern `file:../directoryName/filename`.

Examples:
`file:MyDatabase.fp7`
(references the file "MyDatabase.fp7" that is located in the same directory as the current FileMaker file)

`file:../Business/MyDatabase.fp7`
(references the file "MyDatabase.fp7" that is located in the directory "Business" which is located in the directory that resides one level above the directory that contains the current FileMaker file)

Absolute Mac file location:
This file location format lets you reference a file on a Mac OS system based on an absolute path that starts at the root level of the file system. Format absolute Mac paths according to the following pattern:

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

Note that the path starts with a forward slash ("/").
This file location type is ignored by ScriptFire under Windows.

Examples:
`filemac:/Harddrive/Business/MyDatabase.fp7`
(references the file "MyDatabase.fp7" that is located in the directory "Business" which resides on the top level of the volume "Harddrive")

Absolute Windows file location:
This file location format lets you reference a file on a Windows system based on an absolute path that starts at the root level of the file system. Format absolute Windows paths according to the following pattern:

```
filewin:/driveletter:/directoryName/fileName
```

Note that the path starts with a forward slash (“/”) and that forward slashes are used instead of backslashes (“\”) to separate directories.
This file location type is ignored by ScriptFire under Mac OS.

Examples:
`filewin:/c:/Business/Tests/MyDatabase.fp7`
(references the file “MyDatabase.fp7” that is located in the directory “Tests” which resides in the directory “Business” that is located on the root level of the volume “C:”)

Windows network file location:
To reference a file that can be accessed via Windows file sharing use this file location type. Format Windows network paths according to the following pattern:

`filewin://computerName/shareName/directoryName/fileName`

Note that the path starts with a double slash (“//”) followed by the name of computer that shares the volume (“shareName”) via Windows file sharing that contains the target file.

Windows network file locations are ignored by ScriptFire on Mac OS.

Examples:
`filewin://Fileserver/Business/MyDatabase.fp7`
(references the file “MyDatabase.fp7” that is located in a directory shared via Windows file sharing under the name “Business” on the computer named “Fileserver”)

FileMaker network file location:
The FileMaker network file location format lets you specify locations of files that are shared by FileMaker host via TCP/IP. Use the following pattern to reference such files:

`fmnet:/hostIPAddress/fileName`

Note that the path starts with a forward slash (“/”). To use this file location type under FileMaker 5.x and 6 ensure that sharing via TCP/IP is enabled in the application preferences of both host and client. FileMaker 7 and 8 always use TCP/IP for sharing.

Examples:
`fmnet:/192.168.0.1/MyDatabase.fp7`
(references the file “MyDatabase.fp7” that is shared via FileMaker TCP/IP sharing by the host with the IP address 192.168.0.1)

Object name location:
ScriptFire enables you to reference any open file by its file name or by one of its window titles (FileMaker 7 and 8 only). Use this file location type to point at files that are currently open in FileMaker. Use the following pattern for this location type:

`fmname:filename`
`fmname:windowName` (FileMaker 7 and 8 only)

Examples:
`fmname:MyDatabase.fp7`
(references the file “MyDatabase.fp7” that is currently open in FileMaker)

`fmname:Customers`
(references the FileMaker file that shows the window with the title “Customers”)

Multiple file locations:
You can set several alternative file locations in search order using the `fileLocation` parameter and separate them by semicolons (“;”). When several file locations are defined ScriptFire will try to locate the target file based on the first location. If the file does not exist it tries the second location and so forth.

Example:
```
fmnet:/192.168.0.1/MyDatabase.fp7;
filewin:/c:/Business/MyDatabase.fp7;
filemac:/Harddrive/Business/MyDatabase.fp7
```

In the example above ScriptFire will first try to locate the file “MyDatabase.fp7” on the FileMaker network by contacting the host 192.168.0.1. If this operation is not successful (e.g. network connection, host or shared file is not available) ScriptFire will try to open the file “MyDatabase.fp7” locally according to the absolute Windows file location specified. If this does not work, either (e.g. file is not available or current system is not Windows) ScriptFire tries the third file location that is specified. Only if the last location does not point to the file “MyDatabase.fp7” as well the function that uses this file location value will fail.

Working with relative location types:
Please note that ScriptFire resolves relative file location types such as `file:filename` and `fmname:filename` to absolute file locations internally. If using these location types ensure that the location of the specified file does not change until ScriptFire triggers it (e.g. the moment a schedule invokes a script).

CHAPTER 2
Schedule-Based Script Tasks

ScriptFire enables you to trigger scripts based on any type of schedule you have seen in **Chapter 1**. You can trigger scripts based on secondly, minutely, hourly, daily, weekly, monthly and even yearly schedules to perform database backups, trigger consolidations or generate notifications.

Schedule script tasks

To schedule FileMaker scripts with ScriptFire the plug-in function ***SFire_AddScriptTask*** is used. It adds a script as a scheduled task to the ScriptFire scheduler. ScriptFire will trigger the script according to the schedule that has been defined. To invoke the function *SFire_AddScriptTask* trigger it from a script that is attached to a button or defined as startup script for your solution. Within such a script use the *Set Field* script step to invoke *SFire_AddScriptTask* as described below.

Syntax:

```
SFire_AddScriptTask( taskName; script; {scriptParameter};
{fileLocation}; {fileAccount}; {filePassword}; {schedule};
{timeout}; {taskState} )
```

Parameters:

`taskName` – This parameter is required. It specifies a unique name of the task to be added to the scheduler. The task name can be used later to manage the task (e.g. change or disable it). To update the settings of an existing task, simply use its name when invoking this function. This parameter is case-sensitive (“Task1” and “TASK1” refer to two different tasks).

`script` – This parameter is required. It specified the name of the script that is to be invoked. This parameter is case-insensitive.

`{scriptParameter}` – This parameter is optional. Use it to pass additional information to the script that will be triggered. When the script is running you can retrieve this information using the function *Get(ScriptParameter)* under FileMaker 7 and 8. Since FileMaker 5.x and 6 do not support script parameters natively you can use the ScriptFire function *SFire_GetScriptParameter* to retrieve the parameter values on these FileMaker platforms.

`{fileLocation}` – This parameter is optional. It specifies the location of the file that contains the script to be invoked. In the moment the script is to be triggered ScriptFire checks if the specified file is already open. If it is not open ScriptFire will try to open it based on the file location specified by this parameter. You can leave this parameter empty to point to the current file. To define a file location all location types shown in **Chapter 1** are supported. Please note that FileMaker must be running for ScriptFire to open database files.

`{fileAccount}` – This parameter is optional and supported under FileMaker 7 and 8 only. Use it to specify an account for the file that contains the script to be triggered in case that ScriptFire might have to open it before invoking the specified script. If no account has been specified and ScriptFire needs to open the target file it will use the default account.

`{filePassword}` – This parameter is optional. Use it to specify the password of the target file in case it has to be opened. Like all other settings, ScriptFire stores passwords in an encrypted data format.

`{schedule}` – This parameter is optional. Use it to specify the schedule for a script task. The schedule can be based on any of the schedule types described in **Chapter 1**. By default the schedule `"once:+0:00:00"` is used which means that the script will be triggered immediately if no value for this parameter is specified.

`{timeout}` – This parameter is optional. FileMaker can only trigger scripts when it is not busy. FileMaker is busy when another script is running, a modal dialog window is open (e.g. *ScriptMaker* or *Define Database* dialog) or other activities are performed (e.g. exporting records). This parameter lets you specify a maximum period of time in `HH:MM:SS` format which ScriptFire waits if FileMaker is busy in the moment the script is to be invoked. If FileMaker becomes idle before the timeout ends ScriptFire will trigger the specified script. By default ScriptFire waits infinitely until FileMaker becomes idle to trigger a scheduled task. Timeouts are also considered when the task is blocked (*SFire_SetTasksBlocking*) at the moment it is to be invoked.

`{taskState}` – This parameter is optional. Use it to specify the initial state of the script task. By default the task state is set to `"enabled"` which means that the task will be active. To disable a task set this parameter to the value `"disabled"`.

Schedule script example

In this example a backup script will be scheduled for daily execution. To ensure that the backup is triggered even if the database is not open the file location, account and password will be specified so that ScriptFire can open the file to trigger the backup script if necessary.

The *Set Field* script steps that installs the ScriptFire task for this example contains the following plug-in function call (placed in the Calculation Editor):

```
SFire_AddScriptTask(
"Backup Task";
"Daily Backup";
"";
```

```
"fmnet:/192.168.0.10/Customers.fp7";
"Admin";
"foo123";
"daily: 01:30:00";
"";
"enabled" )
```

In detail this means that the task that is installed can be referenced later under the name `"Backup Task"` (e.g. to remove it at a later point). The script that will be triggered by ScriptFire is called “Daily Backup”. A script parameter is not needed for the script `"Daily Backup"` so the third parameter is left empty for this example.

If the target database file is not already open when the script is supposed to be invoked ScriptFire will look for it on the FileMaker network and open it. The target file is shared by a FileMaker host with the IP address `192.168.0.10`. The name of the database file is `"Customers.fp7"`. To open the file an account and password is required. The account to be used is called `"Admin"` and the password is `"foo123"`.

The schedule used for this example means that the script `"Daily Backup"` will be triggered every day at 1:30 am (`"daily: 01:30:00"`). A timeout is not set because the backup should be triggered at any later point if FileMaker is not idle at 1:30 am.

Finally, the task state is set to `"enabled"` since this script task should be active from the moment it is installed. Since `"enabled"` is the default value of the `taskState` parameter this value could also be skipped by using an empty value `""`.

Chapter 6 (→ Managing Tasks, p. 38) provides information on how to retrieve settings of installed tasks and manage then (e.g. disable or enable tasks).

Trigger scripts immediately

The plug-in function `SFire_AddScriptTask` can not only be used to schedule scripts. To trigger a script for immediate execution from a calculation leave the schedule parameter empty so that the default value `"once:+0:00:00"` is used.

Example:

```
SFire_AddScriptTask( "Validation"; "Validation Script";
""; ""; ""; ""; ""; ""; "" )
```

This enables you to attach scripts to events like exiting fields or accessing records. Script tasks which are executed immediately (with empty schedule or schedule set to `"once:+0:00:00"`) will not be saved in the ScriptFire scheduler. Such tasks expire directly after execution.

For more information, please review **Record and field events** (p. 32) in **Chapter 4** (→ Event-Based Script Tasks).

CHAPTER 3
Open and Close File Tasks

ScriptFire enables you to schedule tasks that open or close database files. Use the open task type if you do not intent to invoke a script from a file but just open it (once or regularly) at a specific point in time.

If a database should be closed at a specific point in time the close task type comes handy. Unlike a script task that would first open a database file to invoke a script that closes it in such situations, the close task type does not take any actions if the specified file is already closed.

Schedule open file tasks

To open a FileMaker database file once or regularly at a specific point in time the ScriptFire plug-in function ***SFire_AddOpenTask*** is used. It adds a file open task to the ScriptFire scheduler. ScriptFire will open the database file according to the schedule that has been specified. To invoke the function *SFire_AddOpenTask* trigger it from a script that is attached to a button or defined as startup script for your solution. Within such a script use the *Set Field* script step to invoke *SFire_AddOpenTask* as described below.

Syntax:

```
SFire_AddOpenTask( taskName; fileLocation; {fileAccount};
{filePassword}; {openType}; {schedule}; {timeout}; {taskState} )
```

Parameters:

`taskName` – This parameter is required. It specifies a unique name of the task to be added to the scheduler. The task name can be used later to manage the task (e.g. change or disable it). To update the settings of an existing task, simply use its name when invoking this function. This parameter is case-sensitive (“Task1” and “TASK1” refer to two different tasks).

`fileLocation` – This parameter is required. It specifies the location of the database file to be opened. To define a file location all location types shown in **Chapter 1** are supported. Leave this parameter empty to refer to the current database file. Please note that FileMaker must be running for ScriptFire to open database files. If alternative file locations are specified using this parameter ScriptFire will open the first database file it finds starting with the first in the list.

{fileAccount} – This parameter is optional and supported under FileMaker 7 and 8 only. Use it to specify an account for the database that is to be opened. If no account has been specified ScriptFire will use the default account when opening the file.

{filePassword} – This parameter is optional. Use it to specify the password of the target file that is to be opened. Like all other settings, ScriptFire stores passwords in an encrypted data format.

{openType} – This parameter is optional. It specifies whether the file should be opened hidden or not. Hidden files are opened without a window. User can show the window of hidden files by selecting them in the *Window* menu. By default, ScriptFire uses the value `"Open Visible"` to open a database file with a window. To open a file hidden without a window set this parameter to `"Open Hidden"`. Please note that FileMaker 5.x and 6 under Windows do not support the hidden window state of database files. Instead, these FileMaker versions show hidden database files as minimized icons.

{schedule} – This parameter is optional. Use it to specify the schedule for an open task. The schedule can be based on any of the schedule types described in **Chapter 1**. By default the schedule `"once:+0:00:00"` is used which means that the database file will be opened immediately if no value for this parameter is specified.

{timeout} – This parameter is optional. FileMaker can only open database files when it is not busy. FileMaker is busy when a script is running, a modal dialog window is open (e.g. *ScriptMaker* or *Define Database* dialog) or other activities are performed (e.g. exporting records). This parameter lets you specify a maximum period of time in `HH:MM:SS` format which ScriptFire waits if FileMaker is busy in the moment the database file is to be opened. If FileMaker becomes idle before the timeout ends ScriptFire will open the specified file. By default ScriptFire waits infinitely until FileMaker becomes idle to open scheduled database files. Timeouts are also considered when the task is blocked (*SFire_SetTasksBlocking*) at the moment it is to be invoked.

{taskState} – This parameter is optional. Use it to specify the initial state of the open task. By default the task state is set to `"enabled"` which means that the task will be active. To disable a task set this parameter to the value `"disabled"`.

Schedule open file example

In this example ScriptFire will be scheduled to open a database file every Monday morning on the host computer that shares this file with other FileMaker users. By scheduling a file open task the availability of the database file is ensured for all FileMaker clients who use it for operations during office ours.

The *Set Field* script steps that installs the ScriptFire task for this example contains the following plug-in function call (placed in the Calculation Editor):

```
SFire_AddOpenTask(
"Open Database Task";
"fmwin:/c:/Project Databases/ProjectX.fp7";
```

```
"Admin";
"foo123";
"weekly: Mo 06:00:00";
"";
"enabled" )
```

In detail this means that the task that is installed can be referenced later under the name `"Open Database Task"` (e.g. to remove it at a later point). The file to be opened is named `ProjectX.fp7`. It is located in the `Project Databases` folder on the c-drive of the local Windows file system.

To open the file an account and password is required. The account to be used is called `"Admin"` and the password is `"foo123"`.

The schedule used for this example specifies that the database file will be opened every Monday morning day at 6:00 am (`"weekly: Mo 06:00:00"`). A timeout is not set because the database should be opened at any later point if FileMaker is not idle Mondays at 6:00 am.

Finally, the task state is set to `"enabled"` since this open task should be active from the moment it is installed. Since `"enabled"` is the default value of the `taskState` parameter this value could also be skipped by using an empty value `""`.

Schedule close file tasks

To close a FileMaker database file once or regularly at a specific point in time the ScriptFire plug-in function ***SFire_AddCloseTask*** is used. It adds a file close task to the ScriptFire scheduler. ScriptFire will close the database file according to the schedule that has been specified. Unlike a script task that would first open a database file to invoke a script that closes it in such situations, the close task type does not take any actions if the specified file is already closed.

To invoke the function *SFire_AddCloseTask* trigger it from a script that is attached to a button or defined as startup script for your solution. Within such a script use the *Set Field* script step to invoke *SFire_AddCloseTask* as described below.

Syntax:

```
SFire_AddCloseTask( taskName; fileLocation; {schedule};
{timeout}; {taskState} )
```

Parameters:

`taskName` – This parameter is required. It specifies a unique name of the task to be added to the scheduler. The task name can be used later to manage the task (e.g. change or disable it). To update the settings of an existing task, simply use its name when invoking this function. This parameter is case-sensitive (“Task1” and “TASK1” refer to two different tasks).

`fileLocation` – This parameter is required. It specifies the location of the database file to be closed. To define a file location all location types shown in **Chapter 1** are supported. Leave this parameter empty to refer to the current database file. Use the location type `"fmname:"` to point at a specific window that is to be closed instead of the entire database (FileMaker 7 and 8 only). If alternative file locations are specified using this parameter ScriptFire will close the first database file (or window) it finds starting with the first in the list.

`{schedule}` – This parameter is optional. Use it to specify the schedule for a close task. The schedule can be based on any of the schedule types described in **Chapter 1**. By default the schedule `"once:+0:00:00"` is used which means that the database file will be closed immediately if no value for this parameter is specified.

`{timeout}` – This parameter is optional. FileMaker can only close database files when it is not busy. FileMaker is busy when a script is running, a modal dialog window is open (e.g. *ScriptMaker* or *Define Database* dialog) or other activities are performed (e.g. exporting records). This parameter lets you specify a maximum period of time in `HH:MM:SS` format which ScriptFire waits if FileMaker is busy in the moment the database file is to be closed. If FileMaker becomes idle before the timeout ends ScriptFire will close the specified file. By default ScriptFire waits infinitely until FileMaker becomes idle to close scheduled database files. Timeouts are also considered when the task is blocked (*SFire_SetTasksBlocking*) at the moment it is to be invoked.

`{taskState}` – This parameter is optional. Use it to specify the initial state of the close task. By default the task state is set to `"enabled"` which means that the task will be active. To disable a task set this parameter to the value `"disabled"`.

Schedule close file example

In this example ScriptFire will be scheduled to close a database file every Friday night (e.g. for security reasons).

The *Set Field* script steps that installs the ScriptFire task for this example contains the following plug-in function call (placed in the Calculation Editor):

```
SFire_AddCloseTask(
"Close Database Task";
"fmwin:/c:/Project Databases/ProjectX.fp7";
"weekly: Fr 23:00:00";
"";
"enabled" )
```

In detail this means that the task that is installed can be referenced later under the name `"Close Database Task"` (e.g. to remove it at a later point). The file to be closed is named `ProjectX.fp7`. It is located in the `Project Databases` folder on the c-drive of the local Windows file system.

The schedule used for this example specifies that the database file will be closed every Friday night at 11:00 pm (`"weekly: Fr 23:00:00"`). A timeout is not set because the database should be closed at any later point if FileMaker is not idle Fridays at 11:00 pm.

Finally, the task state is set to `"enabled"` since this close task should be active from the moment it is installed. Since `"enabled"` is the default value of the `taskState` parameter this value could also be skipped by using an empty value `""`.

CHAPTER 4
Event-Based Script Tasks

In addition to schedule-basked tasks that trigger scripts or open files at a specific point in time, ScriptFire supports event-based script tasks. Event-based functionality is provided by window and layout tasks.

Window tasks enable you to trigger a script when a specific window is activated. An activation event for window tasks is invoked whenever a specified window is shown or focused. This can happen because of a script command or a user interaction.

Layout tasks provide the same functionality for FileMaker layouts. Use a layout task to trigger a script whenever a specific layout is accessed (e.g. by the user from the layout menu or by a script).

Please note that both window and layout tasks are only supported for FileMaker 7 and later.

Window tasks

To install a window task for FileMaker using ScriptFire the plug-in function ***SFire_AddWindowTask*** is used. The task installed by this function ensures that a specific script is triggered whenever a specified window comes to front. To invoke the function *SFire_AddWindowTask* trigger it from a script that is attached to a button or defined as startup script for your solution. Within such a script use the *Set Field* script step to invoke *SFire_AddWindowTask* as described below.

Syntax:

```
SFire_AddWindowTask( taskName; window; script;
{scriptParameter}; {fileLocation}; {taskState} )
```

Parameters:

`taskName` – This parameter is required. It specifies a unique name of the task to be added. The task name can be used later to manage the task (e.g. change or disable it). To update the settings of an existing task, simply use its name when invoking this function. This parameter is case-sensitive (“Task1” and “TASK1” refer to two different tasks).

`window` – This parameter is required. It specifies the title of the window to be observed. To observe all windows of the specified file(s) use the value “-All Windows” for this parameter. This parameter is case-insensitive.

`script` – This parameter is required. It specified the name of the script that is to be invoked. This parameter is case-insensitive.

`{scriptParameter}` – This parameter is optional. Use it to pass additional information to the script that will be triggered. When the script is running you can retrieve this information using the function *Get(ScriptParameter)* under FileMaker 7 and 8.

`{fileLocation}` – This parameter is optional. It specifies the file(s) that will be observed for the defined window. The handling of this parameter is a little different compared to schedule-based tasks. For event-based tasks ScriptFire considers *all* file locations indicated by this parameter. This means that file locations set by this parameter are not alternatives – every file will be observed for the specified window to become active. Of course, file location types that are not supported by a specific platform (e.g. filemac-type on Windows) will be ignored. In addition, the fmname-type is not supported by this function. Leave this parameter empty to point at the current file.

`{taskState}` – This parameter is optional. Use it to specify the initial state of the task. By default the task state is set to `"enabled"` which means that the task will be active. To disable a task set this parameter to the value `"disabled"`.

Window task example

The following example installs a window task that observes a specific database file for a defined window to be activated.

The *Set Field* script steps that installs the ScriptFire task for this example contains the following plug-in function call (placed in the Calculation Editor):

```
SFire_AddWindowTask(
"Window Task";
"Contacts";
"Set Contacts Focus";
"";
"fmwin:/c:/Business/Customers.fp7";
"enabled" )
```

In detail this means that the task installed can be referenced later under the name `"Window Task"` (e.g. to remove it at a later point). The title of the window that will be observed is `"Contacts"`. Whenever this window is activated the script with the name `"Set Contacts Focus"` will be invoked. A script parameter is not specified.

ScriptFire will observe a specific database file for this event – it is called `Customers.fp7`. Finally, the task state is set to `"enabled"` since this window task should be active from

the moment it is installed. Since `"enabled"` is the default value of the `taskState` parameter this value could also be skipped by using an empty value `""`.

Layout tasks

To install a layout task for FileMaker using ScriptFire the plug-in function ***SFire_AddLayoutTask*** is used. The task installed by this function ensures that a specific script is triggered whenever a specified layout is accessed. To invoke the function *SFire_ AddLayoutTask* trigger it from a script that is attached to a button or defined as startup script for your solution. Within such a script use the *Set Field* script step to invoke *SFire_ AddLayoutTask* as described below.

Syntax:

```
SFire_AddLayoutTask( taskName; layout; window; script;
{scriptParameter}; {fileLocation}; {taskState} )
```

Parameters:

`taskName` – This parameter is required. It specifies a unique name of the task to be added. The task name can be used later to manage the task (e.g. change or disable it). To update the settings of an existing task, simply use its name when invoking this function. This parameter is case-sensitive (“Task1” and “TASK1” refer to two different tasks).

`layout` – This parameter is required. It specifies the name of the layout to be observed. To observe all layouts use the value “-All Layouts” for this parameter. This parameter is case-insensitive.

`window` – This parameter is required. Use it if ScriptFire is supposed to only observe a specific window for layout changes. To observe all windows of the specified file(s) use the value “-All Windows” for this parameter. This parameter is case-insensitive.

`script` – This parameter is required. It specified the name of the script that is to be invoked. This parameter is case-insensitive.

`{scriptParameter}` – This parameter is optional. Use it to pass additional information to the script that will be triggered. When the script is running you can retrieve this information using the function *Get(ScriptParameter)* under FileMaker 7 and 8.

`{fileLocation}` – This parameter is optional. It specifies the file(s) that will be observed. For event-based tasks ScriptFire considers *all* file locations indicated by this parameter. This means that file locations set by this parameter are not alternatives – every file will be observed for the specified layout to become active. Of course, file location types that are not supported by a specific platform (e.g. filemac-type on Windows) will be ignored. In addition, the fmname-type is not supported by this function. Leave this parameter empty to point at the current file.

`{taskState}` – This parameter is optional. Use it to specify the initial state of the task. By default the task state is set to `"enabled"` which means that the task will be active. To disable a task set this parameter to the value `"disabled"`.

Layout task example

The following example installs a layout task that observes a specific database file for a defined layout to be activated.

The *Set Field* script steps that installs the ScriptFire task for this example contains the following plug-in function call (placed in the Calculation Editor):

```
SFire_AddLayoutTask(
"Layout Task";
"Invoices";
"-All Windows";
"Update Invoices";
"";
"fmwin:/c:/Business/Customers.fp7";
"enabled" )
```

In detail this means that the task installed can be referenced later under the name `"Layout Task"` (e.g. to remove it at a later point). The layout to be observed is named `"Invoices"`. All windows of the database that is specified in the next step will be observed for this layout to become active. This is achieved by setting the `window` parameter to the value `"-All Windows".` The script to be invoked when the specified layout is entered is called `"Update Invoices".`This script is triggered without a script parameter.

ScriptFire will observe the database file `Customers.fp7` for this layout task. Finally, the task state is set to `"enabled"` since this layout task should be active from the moment it is installed. Since `"enabled"` is the default value of the `taskState` parameter this value could also be skipped by using an empty value `""`.

Record and field events

Window and layout tasks are installed once and managed by ScriptFire independently of FileMaker.

For other events such as exiting fields or records FileMaker provides calculations that can be used in combination with ScriptFire to invoke scripts. To trigger a script immediately from a calculation the ScriptFire function `SFire_AddScriptTask` (introduced in **Chapter 2**) is triggered with an empty schedule parameter so that the default schedule `"once:+0:00:00"` is used.

The following shows how to invoke scripts when users exit a field, create, view or delete a record.

Invoke a script when exiting a field (simple field validation):
To invoke a script that validates user input of a field a script is triggered from the Validation Calculation of that field. To specify the Validation Calculation for a field open the *Define Database* dialog, navigate to the table that contains the field, select the field and click the *Op-*

tions button. In the following dialog click on the *Validation* tab and enable the checkbox *Validated by calculation*.

For example, to validate the input of a field called `Email` the ScriptFire function `SFire_AddScriptTask` would be placed inside this field's Validation Calculation as follows:

```
SFire_AddScriptTask( "FieldValidationSimple";
Validate Email Address"; ""; ""; ""; ""; ""; ""; "" )
```

This would result in the script `Validate Email Address` to be invoked every time the user leaves the field `Email` after changing its value. If the validation of the field `Email` fails the script `Validate Email Address` could move the input cursor back into the field (script step *Go to Field*) and inform the user (e.g. via a text label below the field) that the input has to be corrected.

Under FileMaker 6 and earlier this simple approach triggers the validation script when the user leaves the field by clicking or by using the Tab key. Under FileMaker 7 and 8 this simple validation method only works for clicking. To ensure that users cannot bypass a validation script under FileMaker 7 and 8 use the advanced field validation approach described hereafter.

Invoke a script when exiting a field (adv. field validation, FileMaker 7 and 8 only):
To ensure that a validation script is triggered under FileMaker 7 and 8 no matter which way users leave a field (by clicking or using a control key such as Tab or Return) use the field's Auto-Enter Calculation instead of its Validation Calculation. In the *Define Database* dialog select the field and click the *Options* button. On the *Auto-Enter* tab enable the checkbox *Calculated value* and specify the calculation according to the following example:

```
Email &
Left( SFire_AddScriptTask( "FieldValidationAdvanced";
"Validate Email Address"; ""; ""; ""; ""; ""; ""; ""); 0 )
```

This ensures that the actual value of the field `Email` will not be changed by the calculation. However, the script `Validate Email Address` is invoked when leaving the field after updating its value. The script may then change the field value, notify the user or take other actions.

FileMaker 7 and 8 support simple and advanced field validation as shown here also for global fields.

Invoke a script when creating a record:
To have a script triggered automatically when a new record is created set up a *Result* field (or any other name you choose) of type *Text* (not global) in the target table and place a ScriptFire function call like the following inside the field's Auto-Enter Calculation:

```
SFire_AddScriptTask( "RecordCreation"; "Record Creation Script";
""; ""; ""; ""; ""; ""; "")
```

Invoke a script when viewing a record (FileMaker 7 and 8 only):
To invoke a script when a record is viewed (in any type of layout including List and Table layouts) make use of the FileMaker Access Privileges under FileMaker 7 and 8 as follows.

In the *Define Accounts & Privileges* dialog switch to the *Privilege Sets* tab and create a new privilege set that will be assigned to all user accounts. Alternatively, you may also modify an existing set that is already assigned to user accounts. In the *Edit Privilege Set* dialog choose *Custom privileges* in the *Records* pull-down field. The *Custom Record Privileges* dialog pops up. Select your target table from the list and choose the entry *limited* from the *View* pull-down field; the Calculation Editor is shown. Insert a calculation according to the following example:

```
1 &
Left( SFire_AddScriptTask( "AccessRecord";
"Access Record Script"; ""; ""; ""; ""; ""; ""; ""); 0 )
```

This ensures that ScriptFire invokes the script `Access Record Script` every time a record of the target table is viewed. As the calculation always returns the value 1, record viewing is granted in any case. Of course, you may customize the formula to return 0 if record viewing should be denied.

Invoke a script when editing a record (FileMaker 7 and 8 only):
To invoke a script when a record is edited (user enters any field of the record) adapt the Access Privileges as mentioned above. Therefore, navigate to the *Custom Record Privileges* dialog as shown above. However, this time choose the entry *limited* from the *Edit* pull-down field and specify a ScriptFire function call according to this example:

```
1 &
Left( SFire_AddScriptTask( "EditRecord";
"Edit Record Script"; ""; ""; ""; ""; ""; ""; ""); 0 )
```

Just as shown earlier for record viewing, this ensures that ScriptFire invokes the `Edit Record Script` every time a record of the target table is edited.

Invoke a script when deleting a record (FileMaker 7 and 8 only):
The same approach can be used to invoke a script when a record is deleted. Therefore, choose the entry *limited* from the *Delete* pull-down field in *Custom Record Privileges* dialog and specify a ScriptFire function call as demonstrated by the previous examples.

CHAPTER 5
Power Management Tasks

In addition to schedule-based and event-based tasks, ScriptFire supports the scheduling of computer power management. Using sleep and shutdown tasks you can put the computer to sleep according to a specific schedule (e.g. every day at night) or even shut it down automatically.

Sleep tasks

To install a sleep task for FileMaker using ScriptFire the plug-in function ***SFire_AddSleepTask*** is used. The task installed by this function ensures that the current computer is set to sleep mode according to a specified schedule. To invoke the function *SFire_AddSleepTask* trigger it from a script that is attached to a button or defined as startup script for your solution. Within such a script use the *Set Field* script step to invoke *SFire_AddSleepTask* as described below.

Syntax:
`SFire_AddSleepTask( taskName; {schedule}; {timeout}; {taskState} )`

Parameters:
`taskName` – This parameter is required. It specifies a unique name of the task to be added. The task name can be used later to manage the task (e.g. change or disable it). To update the settings of an existing task, simply use its name when invoking this function. This parameter is case-sensitive (“Task1” and “TASK1” refer to two different tasks).

`{schedule}` – This parameter is optional. Use it to specify the schedule for a sleep task. The schedule can be based on any of the schedule types described in **Chapter 1**. By default the schedule `"once:+0:00:00"` is used which means that the computer will be set to sleep mode immediately if no value for this parameter is specified.

`{timeout}` – This parameter is optional. It specifies a timeout for the sleep task to be performed in the format `HH:MM:SS`. By default no timeout is specified for this operation. The timeout is considered when FileMaker is busy or the task is blocked at the moment it is to be executed.

{taskState} – This parameter is optional. Use it to specify the initial state of the task. By default the task state is set to "enabled" which means that the task will be active. To disable a task set this parameter to the value "disabled".

Sleep task example

The following example sets the computer to sleep every day at 6 pm.

The *Set Field* script steps that installs the ScriptFire task for this example contains the following plug-in function call (placed in the Calculation Editor):

Syntax:
```
SFire_AddSleepTask( "Sleep Task"; "daily: 18:00:00"; ""; "enabled")
```

In detail this means that the task installed can be referenced later under the name "Sleep Task" (e.g. to remove it at a later point). The value of the parameter schedule which is "daily: 18:00:00" sets the sleep time to 6 pm (daily). A timeout is not set in this example. Finally, the task state is set to "enabled" since this script task should be active from the moment it is installed. Since "enabled" is the default value of the taskState parameter this value could also be skipped by using an empty value "".

Shutdown tasks

Using the shutdown tasks is similar to sleep tasks. To install a shutdown task for FileMaker using ScriptFire the plug-in function ***SFire_AddShutdownTask*** is used. The task installed by this function ensures that the current computer is shut down according to a specified schedule. To invoke the function *SFire_AddShutdownTask* trigger it from a script that is attached to a button or defined as startup script for your solution. Within such a script use the *Set Field* script step to invoke *SFire_AddShutdownTask* as described below.

Syntax:
```
SFire_AddShutdownTask( taskName; {schedule};
{timeout}; {taskState} )
```

Parameters:
taskName – This parameter is required. It specifies a unique name of the task to be added. The task name can be used later to manage the task (e.g. change or disable it). To update the settings of an existing task, simply use its name when invoking this function. This parameter is case-sensitive (“Task1” and “TASK1” refer to two different tasks).

{schedule} – This parameter is optional. Use it to specify the schedule for a shutdown task. The schedule can be based on any of the schedule types described in **Chapter 1**. By default the schedule "once:+0:00:00" is used which means that the computer will be shut down immediately if no value for this parameter is specified.

{timeout} – This parameter is optional. It specifies a timeout for the shutdown task to be performed in the format HH:MM:SS. By default no timeout is specified for this operation.

The timeout is considered when FileMaker is busy or the task is blocked at the moment it is to be executed.

`{taskState}` – This parameter is optional. Use it to specify the initial state of the task. By default the task state is set to `"enabled"` which means that the task will be active. To disable a task set this parameter to the value `"disabled"`.

Shutdown task example

In the following example the computer is shut down every Saturday at midnight.

The *Set Field* script steps that installs the ScriptFire task for this example contains the following plug-in function call (placed in the Calculation Editor):

Syntax:

```
SFire_AddSleepTask( "Shutdown Task";
"weekly: Sa 00:00:00"; ""; "enabled")
```

In detail this means that the task installed can be referenced later under the name `"Shutdown Task"` (e.g. to remove it at a later point). The computer will be shut down every Saturday at midnight since the `schedule` parameter is set to the value `"weekly: Sa 00:00:00"`. A timeout is not set in this example. Finally, the task state is set to `"enabled"` since this script task should be active from the moment it is installed. Since `"enabled"` is the default value of the `taskState` parameter this value could also be skipped by using an empty value `""`.

CHAPTER 6
Managing Tasks

ScriptFire provides function to manage installed tasks. Using the following managing functions, you can remove installed task, enable or disable them, and retrieve settings of tasks.

Note: To update settings of an existing task just install another task with the same name. ScriptFire will then overwrite existing settings for the specified task.

Task filters

The functions mentioned below use the parameter `taskFilter`. It lets you apply certain task managing activities to all installed tasks that match a specified filter. To apply a function call to a specific task, simply pass its name (case-sensitive) to ScriptFire using this parameter. In order to apply a command to a group of tasks use one of the following filters:

- `"-all"`
  Refers to all installed tasks.
- `"type=task-type"`; task-type can be one of the following values:
  `"Script"`, `"Open"`, `"Close"`, `"Sleep"`, `"Shutdown"`, `"Layout"` or `"Window"`.
  Refers to all tasks that match the specified type.
- `"class=task-type"`; task-type can be one of the following values:
  `"Schedule-based"` or `"Event-based"`
  Refers to all tasks of the specified task class.
- `"state=task-state"`; task-state can be one of the following values:
  `"Enabled"` or `"Disabled"`
  Refers to all enabled or disabled tasks.

All filter keywords mentioned above are case-insensitive.

Remove tasks

To remove any ScriptFire task that has been installed earlier the plug-in function ***SFire_RemoveTask*** is used. This function supports filters that enable you to remove a specific task or an entire group of tasks that match certain criteria. To invoke the function

SFire_RemoveTask trigger it from a script that is attached to a button. Within such a script use the *Set Field* script step to invoke *SFire_RemoveTask* as described below.

Syntax:
`SFire_RemoveTask( taskFilter )`

Parameters:
`taskFilter` – This parameter is required. It supports different criteria that can be used to indicate the tasks that are to be removed. To remove a specific task, simply provide its name (case-sensitive). To remove a group of tasks, specify a task filter as shown in the introduction of this chapter.

Toggle task state

To change the state (enabled or disabled) of a specific task or of a group of tasks the plug-in function ***SFire_ToggleTask*** is used. To invoke the function *SFire_ToggleTask* trigger it from a script that is attached to a button. Within such a script use the *Set Field* script step to invoke *SFire_ ToggleTask* as described below.

Syntax:
`SFire_ToggleTask( taskFilter; taskState )`

Parameters:
`taskFilter` – This parameter is required. It supports different criteria that can be used to indicate the tasks that are to be toggled. To toggle the state of a specific task, simply provide its name (case-sensitive). To toggle a group of tasks, this parameter supports the filters shown in the introduction of this chapter.

`taskState` – This parameter is required. Use it to set the state for the tasks to be toggled. This parameter can have the value `"Enabled"` (enabled all selected tasks) or `"Disabled"` (disables all selected tasks). This parameter is case-insensitive.

Retrieve the number of tasks

Sometimes you need to retrieve the number of installed tasks that match a certain criterion. Therefore, invoke the function ***SFire_GetTaskCount*** from a script (*Set Field* script step) or from any other calculation that requires the number of installed ScriptFire tasks.

Syntax:
`SFire_GetTaskCount( {taskFilter} )`

Parameters:
`{taskFilter}` – The `taskFilter` parameter is optional for this function. By default it returns the number of all installed tasks. To retrieve the number of a certain task group, apply one of the filters described in the introduction of this chapter.

Retrieve a list of installed tasks

To retrieve a list of the names of all installed tasks or a list of all installed tasks that match a certain criterion invoke the plug-in function ***SFire_GetTaskList*** from a script (*Set Field* script step) or from any calculation. This function returns the names of all installed tasks that match the specified criterion separated by carriage return.

Syntax:
`SFire_GetTaskList( {taskFilter} )`

Parameters:
`{taskFilter}` – The `taskFilter` parameter is optional for this function. By default it returns the names of all installed tasks. To retrieve the names of a certain task group only, apply one of the filters described in the introduction of this chapter.

Retrieve task settings

To retrieve specific settings of an installed task, invoke the plug-in function ***SFire_GetTaskSettings***. It returns specific settings of a single installed task at a time. Invoke this function from including the *Set Field* script step.

Syntax:
`SFire_GetTaskSettings( taskName; {items} )`

Parameters:
`taskName` – This parameter is required. It specifies the name of the task of which settings are to be retrieved.

`{items}` – This parameter is optional. Use it to specify the properties you would like to retrieve. This can be one or several of the following items: `"Name"`, `"Type"`, `"Class"`, `"State"`, `"Schedule"`, `"File"`, `"Script"`, `"Timeout"`. In addition, ScriptFire running under FileMaker 7 or 8 lets you specify the following items: `"Window"` and `"Layout"`. Use semicolons (";") to combine several items within this parameter. By default (empty parameter) this function returns the most common settings according to the following request pattern: `"Name; Type; Class; State"`

Note: The order of items in the request does not have an influence of the order of items in the result. The result is formatted according to a pre-defined pattern.

Block certain task types

The function ***SFire_SetTasksBlocking*** blocks or unblocks the execution of certain task types. Invoke this function from a script (*Set Field* script step) or from any calculation. ScriptFire stores a blocking counter for each task type (Script, Open, Close, Sleep, Shutdown, Layout and Window). Blocking counters can be controlled using this function. If a blocking counter for a task type is greater than zero, ScriptFire does not execute tasks of this type.

If a section of a script requires a certain task type to be blocked (e.g. Window type) you can block this type for the execution of this script section and unblock it afterwards without changing the state property (enabled or disabled) of the assigned tasks.

Syntax:
`SFire_SetTasksBlocking( control; {types} )`

Parameters:
`control` – This parameter is required. It specifies the blocking control that is to be executed. The value `"Block"` increments the blocking counter by 1. The value `"Unblock"` decrements the blocking counter by 1. The value `"Force block"` sets the blocking counter to 1 and the value `"Force unblock"` sets the counter to 0. This parameter is case-insensitive.

`{types}` – This parameter is optional. Use it to apply the blocking control to certain tasks types. By default, the control is applied to all task types. The following values are supported: `"Script"`, `"Open"`, `"Close"`, `"Sleep"`, `"Shutdown"`, `"Layout"` and `"Window"`. To apply the blocking control to several types at a time separate the types by semicolon (e.g. `"Layout; Window"`). This parameter is case-insensitive.

CHAPTER 7
Logging Plug-In Activities

ScriptFire provides sophisticated features to log plug-in activities. The plug-in writes the log to a file named *ScriptFire.log*. It is stored in the same folder as the plug-in. Use logging to track plug-in activities. This can be especially useful when debugging your ScriptFire solutions. Of course, logging is optional and can be switched off completely.

Set logging preferences

To set logging preferences the plug-in function ***SFire_SetupLog*** is used. It is recommended to invoke this function in a start-up script of your solution (using the *Set Field* script step).

Syntax:
`SFire_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. The value `"Append"` switches logging on as well, and appends the log file for every session. This parameter is case-insensitive.

`{mode}` – This parameter is optional. It specifies the log level. To log plug-in errors only set this parameter to `"Errors"`. To log all activities of ScriptFire, set the value `"Activities"`. By default (empty parameter), only errors are logged. This parameter is case-insensitive.

`{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 ScriptFire adjusts the log file size according to specified maximum.

Read from the log file

Using the function ***SFire_ReadLog*** ScriptFire lets you read information from the log file into FileMaker. This can be useful to integrate log functionality or administration control into a FileMaker solution.

Syntax:
```
SFire_ReadLog( control; {lineCount} )
```

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. The value `"Append"` switches logging on as well, and appends the log file for every session. This parameter is case-insensitive.

`{lineCount}` – This parameter is optional. It specifies the number of log file lines that are returned. By default, only the current line is returned. Set this parameter to `"-All"` to retrieve all lines starting at the current reading position. This parameter is ignored if the `control` parameter is set to `"Goto end"`. The total log content returned is limited to 1 MB under FileMaker 7 and 8 and to 64 KB under FileMaker 6 and earlier.

To retrieve all log entries that were written during the execution of a specific script, invoke *SFire_ReadLog* and set the `control` parameter to `"Goto end"` in the beginning of the script. At the end of the script invoke this function again and with the `control` parameter set to `"From marker"` and the `lineCount` parameter set to `"-All"`.

Write custom log entries

The function ***SFire_WriteLog*** enables you to extend the logging functionality beyond the command mentioned earlier. Use this function to log custom entries (e.g. to debug your FileMaker solution).

Syntax:
```
SFire_WriteLog( msgId; msgDescription; {msgExtension} )
```

Parameters:
`msgId` – This parameter is required. Use it to specify an ID of the custom log entry. This help to identify log entries at a later point.

`msgDescription` – This parameter is required. Provide a short description of the log entry using this parameter.

`{msgExtension}` – This parameter is optional. Use it to log additional information.

CHAPTER 8
Additional Plug-In Functions

ScriptFire provide additional functions that are described in this chapter. They enable you to retrieve script parameters, result codes and the version of the plug-in. Moreover, you can flash the application icon to notify users about events.

Retrieve script parameter

Use the function ***SFire_GetScriptParameter*** under FileMaker 6 and earlier to retrieve the script parameter that has been passed to the script that has been passed to the current script. This function is not available under FileMaker 7 and 8. When working with these FileMaker versions simply use the native function *Get (ScriptParameter)* to retrieve the script parameter of the current script.

Retrieve last result code

The function ***SFire_GetLastResultCode*** returns the result code of the last ScriptFire function that has been invoked. ScriptFire functions that are not expected to return content provide a result code out-of-the-box. However, functions that are expected to return content do not provide results codes. To find out about the operation status of these functions *SFire_GetLastResultCode* has to be invoked. Please refer to p. 13 for a list of all ScriptFire result codes and their meanings.

ScriptFire functions that return result codes (*SFire_GetLastResultCode* not needed but can still be used):

- SFire_AddScriptTask
- SFire_AddOpenTask
- SFire_AddCloseTask
- SFire_AddSleepTask
- SFire_AddShutdownTask
- SFire_AddWindowTask
- SFire_AddLayoutTask
- SFire_RemoveTask
- SFire_ToggleTask

- SFire_SetTasksBlocking
- SFire_SetupLog
- SFire_WriteLog
- SFire_FlashIcon
- SFire_RegisterSession

ScriptFire functions that do **not** return result codes (*SFire_GetLastResultCode* has to be used to find out about the operation status):

- SFire_GetTaskCount
- SFire_GetTaskList
- SFire_GetTaskSettings
- SFire_ReadLog
- SFire_GetScriptParameter
- SFire_Version

Flash the application icon

When FileMaker executes scripts it is usually busy. During time intensive operations users may switch to other applications. Invoke the ScriptFire function ***SFire_FlashIcon*** at the end of time intensive operations to flash the FileMaker icon (in the task bar on Windows or in the Dock on Mac OS X).

Syntax:
`SFire_FlashIcon ( {timeout}; {control} )`

Parameters:
`{timeout}` – This parameter is optional. It specifies a timeout in HH:MM:SS format that defines the maximum duration of the icon flashing. Set this parameter to `"00:00:05"` to have the application icon flashed for a maximum of five seconds. The value `"00:00:00"` stops flashing immediately. Flashing is also stopped when FileMaker was in background and is clicked to foreground. By default, this function sets an infinite flashing timeout.

`{control}` – This parameter is optional. It specifies whether the application icon should only be flashed when FileMaker is in background (value `"Background"`) or if the icon should be flashed in any case (value `"Always"`). By default (empty parameter) the option `"Always"` is used.

Retrieve plug-in version

The function ***SFire_Version*** returns the version of the installed ScriptFire 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 cli-

ents automatically. Review the FileMaker documentation for more information about the AutoUpdate plug-in.

Syntax:
`SFire_Version ( {option} )`

Parameters:
`{option}` – 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 “ScriptFire”), `"Platform"` (returns the operating system the plug-in is running on) or `"Copyright"` (returns copyright information of the plug-in).

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 ScriptFire license. ScriptFire can be registered manually using the preferences dialog (FileMaker Application Preferences ► Plug-Ins ► ScriptFire).

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

Syntax:
`SFire_RegisterSession ( userName; userCode )`

Parameters:
`userName` – This parameter is required. It is used to set the user name you receive from Dacons after purchasing a ScriptFire license.

`userCode` – This parameter is required. Use it to pass the registration code to the plug-in which you receive from Dacons after purchasing a ScriptFire license.