ChartFire

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

October 28, 2010

CONTENTS

FUNCTION INDEX

FUNCTION OVERVIEW
ChartFire Features

ChartFire is an indispensable plug-in that lets you go beyond the data representation capabilities of FileMaker. ChartFire provides customizable charts generation facilities. Each chart is produced as a PNG or JPEG image, which can be stored within a container field, or exported to disk.

ChartFire can be used in both FileMaker and Runtime solutions.

Use ChartFire to enrich your solution with supreme charts, not seen in FileMaker before.

Function overview

- Create customizable “Pie”, “Bars” and “Lines” charts
- Create unlimited amount of charts
- Give your FileMaker application a visual data representation that goes beyond the text information

Possible solutions

- Create a professional look for your database
- Use charts with Runtime solutions

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 ChartFire using the files that come with this plug-in.

About this manual

The manual is structured as to explain the plug-in functionality by facility sections.

Chapter 1 (Chapter 1, p. 8) explains how to use ChartFire functions with FileMaker. You will learn about plug-in functions, parameters, and result codes.

Chapter 2 (Chapter 2, p.13) provides all information you need to create, render, and destroy the chart.

Chapter 3 (Chapter 3, p.18) provides all information you need to create graphs and labels.

Chapter 4 (Chapter 4, p. 21) provides all information you need to fulfill the chart with data values.

Finally, **Chapter 5** (Chapter 5, p. 22) provides information about advanced plug-in functions that let you retrieve result codes, configure the log engine and more.

Software requirements

ChartFire 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 (2000 and later) and one for Mac OS X (10.4 and later).

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

Installing the FileMaker plug-in

Before installing the ChartFire 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 ChartFire 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 ChartFire, open the *Quick Start* file that comes with the download package. The Quick Start file demonstrates some of the most powerful ChartFire 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 ChartFire plug-in functions.

Multi-user setups

To use ChartFire 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.

CHAPTER 1
ChartFire Basics

This chapter explains how to use ChartFire functions with FileMaker. After reading this chapter advance to any of the other chapters of this manual that provide the information you currently require.

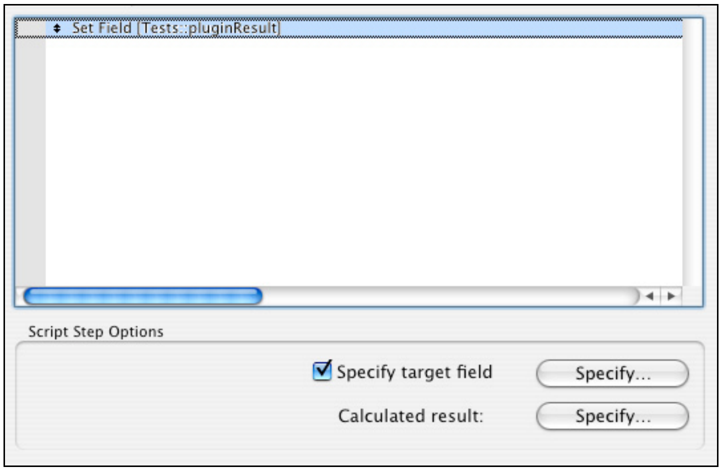
How to use ChartFire functions

Since ChartFire 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 ChartFire 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 ChartFire.

Next, specify the target field which will contain the result of the plug-in operation. There are two types of ChartFire results. Some ChartFire functions return content (such as a Chart image) 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.13) 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

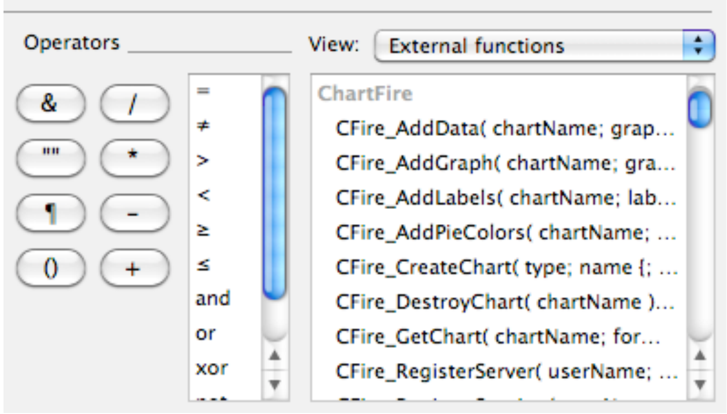
Calculated result:

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 ChartFire functions are listed in the section *ChartFire*. If the ChartFire 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 ChartFire is installed as described earlier and that it is enabled in the FileMaker application preferences.



View: External functions

ChartFire
CFire_AddData(chartName; grap…
CFire_AddGraph(chartName; gra…
CFire_AddLabels(chartName; lab…
CFire_AddPieColors(chartName; …
CFire_CreateChart(type; name {; …
CFire_DestroyChart(chartName)…
CFire_GetChart(chartName; for…
CFire_RegisterServer(userName; …

ChartFire functions in the Calculation Editor

Double-click on the ChartFire 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 ChartFire 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 ChartFire function `CFire_AddLabels` that has been added to the calculation editor. This function attaches the (set of) labels to the chart object.

```
CFire_AddLabels (chartName; labels)
```

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 chart name for the `chartName` 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 ChartFire 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 `CFire_AddGraph` is invoked. Chart and graph names must be specified because parameters `chartName` and `graphName` are required. However, title and display color parameters are *not* specified. Since parameters `title` and `displayColor` are optional, and there is no further parameter value required for this function call, it can be omitted completely:

```
CFire_AddGraph ( "My Chart"; "My Graph" )
```

The plug-in will choose the default value for parameters `title` and `displayColor` (empty title and random color).

Result codes

As mentioned earlier, ChartFire 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 `CFire_GetLastResultCode`. It returns a result code and a short description of the error. The second group of ChartFire functions is called non-content functions. Non-content functions return a result code directly. However, you may also invoke `CFire_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 ChartFire operation failed, use the following calculation in an if condition:

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

The following conditions checks for a specific error:

```
Left (CFire_GetLastResultCode; 4) = "-007"
```

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
-001	(Reserved) Result code reserved
-002	(Invalid registration information) Enter registration exactly as supplied
-003	(Incorrect option) One of the parameters passed in external function has incorrect value
-004	(Reserved) Result code reserved
-005	(Unknown error) Contact the Dacons Support at http://www.dacons.net/support
-006	(File i/o error) Plug-in was unable to write the log file

CHAPTER 2

Create, Retrieve and Destroy Charts

ChartFire plug-in enables you to create an unlimited amount of chart images. After chart image is created and obtained, it should be destroyed. This chapter explains how to create and destroy charts accordingly.

A basic chart construction procedure consists of several steps. First of all the `"CFire_CreateChart"` function should be used, to define the general parameters for the new chart. On this stage the chart does not include any data, and therefore can't be displayed. The next step is to add the necessary *graphs*. In case if the chart has a `"bars"` or `"lines"` style, multiple *graphs* can be created using the `"CFire_AddGraph"` function. When the *graphs* are ready, it is time to add the *labels* and data values using the `"CFire_AddLabels"` and `"CFire_AddData"` functions. As soon as the chart has at least one *graph*, *label*, and at least one data value, the plug-in is ready to render the chart. Chart rendering is invoked using the `"CFire_GetChart"` function, which should return an image object as a function call result. After the chart image has been successfully rendered and retrieved, it should be destroyed using the `"CFire_DestoryChart"` function, in order to free the memory resources.

Any color specification should be supplied within an RGB format, for example:

```
192,255,192
```

Hexadecimal format is supported as well, for example:

```
C0FFC0
```

Single function call may combine different color formats.

Create Chart

The function **`CFire_CreateChart`** creates a chart, based on the parameters specified.

In case if a "`pie`" chart is created, there should be only one *graph* used, considering that pie-chart represents a percentage split of a single volume.

In case if a "`bars`" or "`lines`" chart is created, the user may create unlimited amount of *graphs*, which shell be drawn in a sequence of creation.

The plug-in does not limit the maximum size of the picture, however if no size is provided, the plug-in will use a 640x480 dimensions.

In case if no background color is specified, the plug-in will use transparency, which is supported by PNG image format. However, in case if a JPEG format is used, transparent areas will be converted to white color.

Syntax:

```
CFire_CreateChart ( type; name {; format; pictureWidth;
pictureHeight; bkColor; title; titleFontName; titleFontSize;
titleFontColor; labelPosition; labelFontName; labelFontSize;
labelFontColor; historyPosition; historyFontName; historyFontSize;
historyFontColor; showTags; tagsBkColor; showGrid} )
```

Parameters:

`type` – Type of the *Chart* to be created. Parameter values can be `"pie 2d"`, `"pie 3d"`, `"bars 2d"`, `"bars 3d"`, `"lines"`, `"lines soft"`, and `"lines dotted"`.

`name` – The name of the chart object to be created. This name is used with other function calls, in order to specify the object name to be addressed, and therefore must be unique.

`format` – Format of the picture to be used. Parameter values can be `"png"` (default) and `"jpeg"`.

`pictureWidth` – The width of the image in pixels. Default value is `"640"`.

`pictureHeight` – The height of the image in pixels. Default value is `"480"`.

`bkColor` – Background color in RGB format. Empty value stands for a transparent background.

`title` – Chart title to display. Default value is empty.

`titleFontName` – Font name to be applied, if available, to render the chart title. If empty, system default font name will be used.

`titleFontSize` – Font size to be applied. If empty, system default font size will be used.

`titleFontColor` – Font color to be applied in RGB format. If empty, black color will be used.

`labelPosition` – Labels axis positioning. Parameter values can be `"auto"` (default), `"horizontal",` and `"vertical"`.

`labelFontName` – Font name to be applied, if available, to render the chart labels. If empty, system default font name will be used.

`labelFontSize` – Font size to be applied. If empty, system default font size will be used.

`labelFontColor` – Font color to be applied in RGB format. If empty, black color will be used.

`historyPosition` – History location. Parameter values can be `"left",` and `"right"` (default).

`historyFontName` – Font name to be applied, if available, to render the chart history. If empty, system default font name will be used.

`historyFontSize` – Font size to be applied. If empty, system default font size will be used.

`historyFontColor` – Font color to be applied in RGB format. If empty, black color will be used.

`showTags` – Include indication tags ("`pie`" charts only). Parameter values can be `"percentage", "values"` and `"off"` (default).

`tagsBkColor` – Color to be applied as a background color for pie tags information bars ("`pie`" charts only). In case if no color is specified, a common tooltip color will be used.

`showGrid` – Show grid for ("`bars`" or "`lines`" charts only). Parameter values can be `"showGrid"` (default) and `"hideGrid"`.

Result:
Returns result code (see result code table).

Example:
In the following example a simple "`lines`" charts is created:

```
CFire_CreateChart ( "lines"; "My Chart" )
```

For this example, the following result should be returned:

```
000 (OK)
```

Retrieve Chart

The function **`CFire_GetChart`** renders the chart, and returns an image object in requested format.

Syntax:
`CFire_GetChart ( chartName )`

Parameters:
`chartName` – Name of the chart object to be retrieved.

Result:
Returns an image object.

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

Example:
In the following example renders and retrieves a chart named "My Chart":

```
CFire_GetChart ( "My Chart" )
```

For this example, an image result shell be returned.

Destroy Chart

The function **`CFire_DestoryChart`** destroys the chart, and any associated data.

Syntax:
`CFire_DestoryChart ( chartName )`

Parameters:

`chartName` – Name of the chart object to be destroyed.

Result:

Returns result code (see result code table).

Example:

In the following example destroys a chart named "My Chart":

```
CFire_DestoryChart ( "My Chart" )
```

For this example, the following result should be returned:

```
000 (OK)
```

CHAPTER 3
Graphs and Labels

Each separate data series represented on chart is called "`Graph`". In order to add each graph, a single "`CFire_AddGraph`" function call is required.

Multiple graphs can be created only for "`bars`" and "`lines`" charts. Where "`pie`" chart requires only one graph.

Step markers available for "`bars`" and "`lines`" charts are called "`Labels`". Labels are common for all graphs, as they represent a “milestone” for each data series.

Add Chart

The function **`CFire_AddGraph`** adds a new graph to the chart. In case if the chart has a "pie" type, a single function call is permitted. Any subsequent function call will return an error.

Syntax:
`CFire_AddGraph ( chartName; graphName {; title; displayColor} )`

Parameters:
`chartName` – The name of the chart object to be updated.

`graphName` – The name of the graph object to be created. This name is used with other function calls, in order to specify the object name to be addressed, and therefore must be unique.

`title` – Graph title to be displayed within the History area. In case if this parameter is empty, only a color indication will be used.

`displayColor` – Color to be applied to render this graph. In case if this parameter is empty, a random color will be used. This parameter is meaningful for "`bars`" or "`lines`" charts only. In order to set a custom color for each "`pie`" peace, a "`CFire_AddPieColors`" function should be used.

Result:
Returns result code (see result code table).

Example:
In the following example we shell add a new graph, named "First Graph", to the chart named "my chart":

```
CFire_AddGraph ( "My Chart"; "First Graph" )
```

For this example, the following result should be returned:

```
000 (OK)
```

Add Pie Colors

The function **`CFire_AddPieColors`** adds a series of custom colors to the "pie" chart, separated by carriage return. Subsequent calls will also add the colors to the available set. Therefore the user may either add the full set of color, separated by carriage return, within a single function call, or perform a series of function calls, adding a single color per call.

In case if a graph will contain more items then custom colors available, the plug-in will generate a random color for each item that is missing a custom color specification.

This function is meaningful for "`pie`" charts only.

Syntax:
`CFire_AddPieColors ( chartName; color )`

Parameters:
`chartName` – The name of the chart object to be updated.

`labels` – Color value to be added, or a set of color values, separated by carriage return.

Result:
Returns result code (see result code table).

Example:
In the following example we shell add two colors to the chart named "My Chart":

```
CFire_AddLabels ( "My Chart"; "127,145,240¶247,111,110" )
```

For this example, the following result should be returned:

```
000 (OK)
```

Add Labels

The function **`CFire_AddLabels`** adds a series of labels to the chart, separated by carriage return. Subsequent calls will also add the labels to the available set. Therefore the user may either add the full set of labels, separated by carriage return, within a single function call, or perform a series of function calls, adding a single label per call.

This function is meaningful for "`bars`" or "`lines`" charts only.

Syntax:
`CFire_AddLabels ( chartName; labels )`

Parameters:
`chartName` – The name of the chart object to be updated.

`labels` – Label name to be added, or a set of labels, separated by carriage return.

Result:
Returns result code (see result code table).

Example:
In the following example we shell add three labels to the chart named "My Chart":

```
CFire_AddLabels ( "My Chart"; "First¶Second¶Third" )
```

For this example, the following result should be returned:

```
000 (OK)
```

CHAPTER 4
Data

In order to render the chart values properly, each record value should be attached to appropriate graph, using the "`CFire_AddData`" function.

Add Data

The function **`CFire_AddData`** adds a new data value to the chart graph.

Syntax:
`CFire_AddData ( chartName; graphName; data {; options } )`

Parameters:
`chartName` – Name of the chart object, to be updated.

`graphName` – Name of the graph object, to be updated.

`data` – Numeric data to be applied. This function may accept a series of values, separated by carriage return, or can be triggered sequentially, where each call adds a value (or set of values, separated by carriage return) to the graph.

`options` – This parameter can be used to highlight a particular data item for the "`pie`" charts only. Parameter values can be `"exploded"` or empty (default).

Result:
Returns result code (see result code table).

Example:
In the following example we shell add a single value, to the chart named "My Chart", and graph named "My Graph":

```
CFire_AddData ( "My Chart"; "My Graph"; 10 )
```

For this example, the following result should be returned:

```
000 (OK)
```

CHAPTER 5

Additional Functions

ChartFire provides additional plug-in functions that are described in this chapter. These functions enable you to retrieve result codes, 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.

Retrieve last result code

The function **`CFire_GetLastResultCode`** returns the result code of the last ChartFire function that has been invoked. Please refer to **p. 12** for a table of all result codes.

Syntax:
`CFire_GetLastResultCode`

Parameter:
No parameter needed.

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

Example:
The function `CFire_GetChart` renders and returns a chart image. It does not return error codes. If an error occurs, this function returns an empty result. In the following example, the specified parameter is incorrect:

```
CFire_GetChart ( "Unknown Chart" )
```

Since mandatory parameter value is incorrect (assuming that `"Unknown Chart"` has never been created), an empty result is returned. To find out why the function `CFire_GetChart` failed the function `CFire_GetLastResultCode` is invoked.

```
CFire_GetLastResultCode
```

In this example case it returns:

```
-003 (Incorrect Option)
```

Retrieve plug-in version

The function **`CFire_GetVersion`** returns the version of the installed ChartFire 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:
`TFire_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. `"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 ChartFire license. The plug-in can be registered manually using the preferences dialog (Go to: FileMaker Application Preferences ► Plug-Ins ► ChartFire).

To avoid manual plug-in registration when shipping your database solution to a client or distribute a FileMaker Runtime application with ChartFire you can also register the plug-in from the start-up script of your solution by invoking the function **`CFire_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:
`CFire_RegisterSession ( username ; userCode )`

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

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

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.