



>

Jazz Utils Plug-in

>

Updated 7 February, 2007

◇

Version 1.5

◇

Introduction

◆ Overview

This plug-in is a collection of miscellaneous utilities which I written on an ad-hoc basis, based on personal or project requirements. They have been written with specific requirements in mind, and are not guaranteed to work under all circumstances.

Version 1.5 of the plug-in has been rewritten to work with the new *FileMaker Pro* plug-in format, and as a Universal Binary, to run on Macintosh PPC and Intel computers. It requires OS-X 10.2 or later, though there is a legacy version available (currently v1.19). At the time of writing there is no version 1.5 for Windows. Windows users should continue to use the earlier version of the plug-in (v1.19) which is still operational. A Windows version will become available at a later date.

The Mac/Windows icons next to function names are used to indicate the platforms supported (or in the case of Windows, will be supported). As I prefer to do development work on a Macintosh computer, several of the functions are Mac only. In cases where a function is cross-platform, the Macintosh version will have had significant more testing than it's Windows counterpart. Similarly, as I don't own an Intel Mac (I borrowed one for testing) there could be minor Intel hiccups which I have not become aware of. If you discover any, please let me know and I will endeavour to fix them.

With the new *FileMaker Pro* plug-in format comes a new (more flexible) naming system. This means existing scripts and calculation formula will need to be updated to work with this version. As this is unavoidable, I have made some alterations to the plug-in names, merged a few, removed a few which are meaningless in an unicode context, and so on.

◆ Shareware or Freeware

This plug-in is freeware. You may copy the plug-in and give it away at no cost. You may not sell this plug-in, but you may include it free-of-charge in products you develop at your own risk. If this plug-in is used in any products you develop, a credit would be appreciated, but is not required.

◆ Disclaimer

These plug-in are provided 'as is', and no warranty or guarantee is made of any kind that it operation as expected, or documented. Use of the plugins is entirely at your own risk. You are entirely responsible for determining the suitability of the plug-in(s) for your purposes.

Calling functions

Version 1.5 (and later) of this extension makes use of the newer *FileMaker Pro* plug-in architecture. Plugins can take multiple arguments (including none) and return a single argument. As some of the functions of this plug-in don't need to return any value, but they still need to be wrapped in a script step as if they did.

Plugins are designed to be used inside formulas. A simple function to calculate the square of a number would be included inside a calculation like:

```
Set Field [Area, 3.1415 * Square(Radius)]
```

could calculate the radius of a circle, where 'Area' and 'Radius' are *FileMaker* fields, and 'Square' is the name of a function. This function accepts the field 'Radius' as its argument, and returns the square of this number, which is used in the formula of the 'Set Field' function.

When an external function does not return a value¹ (or the value is unimportant) there are two easy ways to use it. The first is to create an If-End If pair of functions, with no script steps in between. The return value of the function is thus ignored. If an external were to display a dialog box of information, you could call it this way:

```
If [Display Dialog("Hello there")]  
End If
```

An more compact way (vertically) to call this function is to save the result into a dummy field. I create a global field called 'temp' which I reserve for off loading unwanted function return values. The same function call using this method would look like:

```
Set Field [temp, Display Dialog("Hello there")]
```

This is the method I prefer, because of its compactness, and will be used in the examples to follow.

¹ Actually all external functions return an argument, irrespective of whether it is required or not. If a return value is not require the function will return the empty string ("").

Function Summary

◆ JazzVersion

Returns a string describing the plug-in, its registration status (if applicable) and the version number of the plug-in. The 'TextToNum' function is guaranteed to return the version number on its own (as a decimal number), from which comparisons can be made.

For example, in version 1.50:

```
Set Field ["temp", JazzVersion]
```

sets 'temp' to "Jazz Utils v1.50", for example.

◆ JazzGetClipboard

Usage:

```
JazzGetClipboard(<optional replacement value>)
```

Examples:

```
Set Field [clip1, JazzGetClipboard]  
Set Field [clip1, JazzGetClipboard(clip1)]
```

The first example places the plain-text held in the clipboard into the field "clip1". This is the most common form. If the clipboard does not contain text then the empty string will be returned.

If an argument is provided, as in the second example, then this value will be placed on the clipboard after the current value is retrieved. So the second example swaps the contents of the clipboard with the contents of the *FileMaker* field 'clip1'.

Be aware of unexpected results if using the clipboard to exchange information with other programs, particularly if this function is called while *FileMaker* is not the front-most application. To understand clipboard behaviour in these circumstances you should be aware that applications maintain their own clipboard, which is only transferred to and from the system clipboard when applications are moved to and from the foreground.

This function is unicode aware. Styles are not transferred from the clipboard.

◆ JazzSetClipboard



Usage:

```
JazzSetClipboard(<new value>)
```

Example:

```
Set Field [temp, JazzSetClipboard("hello")]
```

This example sets the clipboard with the plain text string “hello”. Any existing clipboard contents will be lost.

Be aware of unexpected results if using the clipboard to exchange information with other programs, particularly if this function is called while *FileMaker* is not the front-most application. To understand clipboard behaviour in these circumstances you should be aware that applications maintain their own clipboard, which is only transferred to and from the system clipboard when applications are moved to and from the foreground.

This function is unicode aware. Styles are not transferred to the clipboard.

◆ JazzStripHTML



Usage:

```
JazzStripHTML( <tagged text>)
```

Example:

```
Set Field [plain_text, JazzStripHTML(html text)]
```

This function removes *many* HTML tags surrounding text, and applies minor reformatting to make it readable as plain text. It is by no means comprehensive in operation. I wrote this function when I needed a field to contain HTML text, but wanted the text to be legible at the same time.

An example of the tags removed and processed are:

| | |
|------------------|--|
| <P>, | converted to two paragraph characters (two returns) |
| , <I>, <U>... | just removed |
| <Hn> | two paragraph returns appended after the heading |
| <PRE> | removed, but enclosed contents is not modified |
| | removed, and a bullet character and space prepended. |

HTML has come a long way since this function was written, so it’s usefulness may vary with the HTML of today. This function is not unicode aware. As most non-Roman characters are generally encoded this is not anticipated to be an issue.

◆ JazzClipToHTML



Usage:

```
JazzClipToHTML
```

Example:

```
Go to Field [Select/perform, styled text]  
Copy [Select]  
Set Field [html text, JazzClipToHTML]
```

This function creates simple HTML-encoded text from styled text held on the clipboard. I wanted to retain simple character-style information, such as bold and italic, when exporting as tab-delimited text. This was not possible.

The work around solution was to copy styled text to the clipboard and have a plug-in convert this information into the equivalent tagged text. By placing the function result into another field, and looping through all the records, the desired result could be achieved.

Please note that this function should be considered as deprecated for the following reasons:

- it is limited in its operation (only bold, italic and underline formatting is recognised)
- from version 7 onwards *FileMaker* uses a different clipboard format not known by this function, so styles copied from *FileMaker* fields do not work (however it still works from some other applications)².
- some italic styles are not recognised when they are oblique variations of a font.
- the plugin *JazzHTML* performs the original requirement much better, without the need to copy and paste (*FileMaker* version 7 and later only). It contains more comprehensive style support, and you can customise the HTML produced.

◆ JazzDebug



Usage:

```
JazzDebug(<Debug print text>)  
JazzDebug(<Debug print text>, <target application>)
```

Examples:

```
Set Field [temp, JazzDebug("Entering counting script")]  
Set Field [temp, JazzDebug("Value of i = " & i)]  
Set Field [temp, JazzDebug("Value of i = " & i, "TextWrangler")]
```

² the plugin *JazzStyledText* assists copying styled text into *FileMaker* from other applications, in situations where the style information is not transferred.

This is a Macintosh-only function. The function is designed to aid debugging, by allowing text to be printed to the frontmost window of a third-party application. The text can contain the contents of *FileMaker* fields, or any other contents provided to the function.

An optional second argument allows you to specify which application the text is delivered to. Currently only three options are supported³: “*BBEdit*”, “*TextWrangler*” (or “*TW*” for short) and “*STE*” for *Scriptable Text Editor*. If the second argument is missing the text will be delivered to *BBEdit*.

The function *JazzDebug* sends the argument contents (plus a terminating return) to *BBEdit*'s front-most window, using *AppleEvents*. The application must be open with a document window available for the debug string.

This function is unicode aware.

◆ **JazzGetMouse**



Usage:

```
JazzGetMouse
```

Example:

```
Set Field [temp, JazzGetMouse]  
Set Field [xcoord, JazzGetMouse X]  
Set Field [ycoord, JazzGetMouse Y]
```

This function is designed to be used in combination with other “*GetMouse*” functions in this plug-in, as demonstrated in the three lines of above example. The function records the current mouse location within the window and returns immediately.

The function always returns an empty string, but the mouse location is remembered for later retrieval using the functions “*JazzGetMouse X*” and “*JazzGetMouse Y*”.

See the note about the coordinate origin in *JazzGetMouseClick*, below.

³ *BBEdit* is a commercial text editor application, available from *BareBones*' web site at <http://www.barebones.com>. *TextWrangler* is a free text editor also available from *BareBones*. *Scriptable Text Editor* is a small demo application shipping with many *Classic Mac OS* operating systems. It was probably the first text editor to support *AppleEvents*. It is included for backward compatibility only.

◆ JazzGetMouseClicked



Usage:

```
JazzGetMouseClicked
```

Example:

```
Set Field [temp, JazzGetMouseClicked]  
Set Field [xcoord, JazzGetMouseClicked]  
Set Field [ycoord, JazzGetMouseClicked]
```

This function is designed to be used in combination with other functions in this plug-in, as demonstrated in the example, above. This function waits for the next click of the mouse, and records the click location within the window.

The mouse click will *not* get passed on to *FileMaker* or the operating system, it is used purely to define a position. Once called, the function will not return until the mouse is clicked, and the application will not respond to the keyboard.

The function returns an empty string, but the mouse click location is remembered for later retrieval using the functions "JazzGetMouseClicked" and "JazzGetMouseClicked".

This function was written to obtain (x,y) coordinates of towns on maps (pictures) which were contained in a *FileMaker* database. Use of this function allowed the coordinates to be automatically entered into the database. (One database held the towns and recorded locations, while a related one held the maps). You could possibly think of many other uses for this function.

NOTE

The exact origin for the coordinates and the behaviour with scrolled windows is beyond the control of the plug-in. They are dependant upon how *FileMaker* is written, and may change with different versions of *FileMaker Pro*, and / or different operating system versions. Clicks outside the *FileMaker* window are allowed, but the coordinates recorded are undefined, and may be negative.

For many users the origin is the top left of the *Status Area* or the top-left of the window's contents if the *Status Area* is hidden. Similarly, the behaviour of scrolling a layout may or may not adjust the coordinates appropriately.

Despite its shortcomings, for quick plotting of coordinates (eg. from a map shown in a *container* field) this function is invaluable!

◆ JazzGetMouseX



Usage:

```
JazzGetMouseX
```

Example:

```
Set Field [mouse_x , JazzGetMouseX]
```

This function is used to return the x-location recorded at the most recent call of 'JazzGetMouse' or 'JazzGetMouseClicked'. See these functions for further explanation.

◆ JazzGetMouseY



Usage:

```
JazzGetMouseY
```

Example:

```
Set Field [mouse_y, JazzGetMouseY]
```

This function is used to return the y-location recorded at the most recent call of 'JazzGetMouse' or 'JazzGetMouseClicked'. See these functions for further explanation.

◆ JazzWinActivate



Usage:

```
JazzWinActivate(<application name>)
```

Examples:

```
Set Field [temp, JazzWinActivate("readme.txt - NotePad")]  
Set Field [temp, JazzWinActivate("Windows NT Task Manager")]  
Set Field [temp, JazzWinActivate("Wine")]
```

This is a Windows-only function. It was written when a Windows database was required to activate (bring to the front) another application. (The Macintosh version used *AppleScript*, thus it was not required for the Macintosh).

Not being a *Window* guru, I find myself unable to describe the exact parameter requirements for this function, except that it is "a" name for the application you wish to activate. (Some experimentation might be required to determine what the correct name is). Use of an incorrect name can cause the *FileMaker* application to be terminated prematurely (crash). However use of the correct application name (once discovered) proved stable and reliable.

The three examples shown above were found to be reliable to activate certain running applications. The first is the file *readme.txt* when open within the *Note Pad* application. The second is the *Windows NT Task Manager*, and the last is a runtime-application built with *Macromedia Director 7*, called *Wine.exe*.

◆ JazzCharToNum



Usage:

```
JazzCharToNum( <single character>)
```

Example:

```
Set Field [temp, JazzCharToNum("C")]
```

This function converts a character to its ascii equivalent. While so trivial as to be used in sample code everywhere, I find this function invaluable. This function is available for Windows, however *extended* ascii will presumably return the corresponding value from the Macintosh character table (since these were the codes are used internally to *FileMaker Pro*).

The example above will return the ascii value for the upper-case letter 'c', which is 67. If more than one character is passed to the function, the ascii value of the first character will be used.

This function is deliberately not unicode aware, however characters in the range 32 to 126 (which includes letters a-z, A-Z, numbers 0-9 and some more common symbols) should not be a problem, as these characters have the same value in ascii and unicode.

◆ JazzNumToChar



Usage:

```
JazzNumToChar(<ascii value>)
```

Example:

```
Set Field [temp, JazzNumToChar(67)]
```

This function is the opposite of 'JazzCharToNum', above. It turns an ascii value to the equivalent character. The example above returns the upper-case character 'C'.

If the value does not lie in the range 0 to 255 then the character will be based on the value modulus 256. Values over 127 will likely return the Macintosh Roman character (since these were the codes are used internally to *FileMaker Pro*). If the ascii value is < 0, the result is undefined.

This function is deliberately not unicode aware, however numbers in the range 32 to 126 should not be a problem, as these characters have the same value in ascii and unicode.

◆ JazzCommonWords



Usage:

```
JazzCommonWords(<lines to compare>)
```

Example:

```
Set Field [temp, JazzCommonWords(field 1, field 2)]
```

This function compares the words in the two parameters provided. It returns a count of the number of words appearing on both lines. The function is case insensitive, so “abc” and “aBc” are considered the same word. For example, if the two fields contain the following:

“the quick brown”

“Quick brown themes”

then the function will return the number ‘2’ — two words match.

◆ JazzSetWindowSize



Usage:

```
JazzSetWindowSize(<width>; <height>)
```

Example:

```
Set Field [temp, JazzSetWindowSize(640, 480)]
```

This is a Windows-only function, because the same functionality can be obtained on the Macintosh using AppleEvents. The function attempts to set the application window size so that it’s content is the width and height provided.

In version 7 FileMaker introduced functions to resize windows, however I found them unreliable when working with a single maximised document window inside Windows’ (non-maximised) application window. So I rolled my own.

◆ JazzShowStatusBar



Usage:

```
JazzCommonWords(<boolean>)
```

Example:

```
Set Field [temp, JazzCommonWords(1)]
```

This is a Windows-only function to show or hide the application window's status bar. Use 1 to show the status bar, or 0 to hide the status bar.

This function is meaningless on a Macintosh as the windows are not constrained inside an application window. FileMaker provides functions to show or hide window status bars.

Appendices

◆ Version History

Version history is sketchy, but the following is known. The list will be elaborated as I dig up past documentation. Not all versions were released to the public.

- 1.03? Distribution inside commercial applications. Not available as a separate item. Included only the clipboard and HTML functions.
- 1.04 Released to the public as freeware.
- 1.14 Added GetMouse and related routines to *Windows* version. (May 2002)
- 1.15 Added the 'CompareLines' function. (Nov 2003)
- 1.16 Mac-only release. Fixed 'Debug-BBEdit'. (Dec 2003)
- 1.19 Updated 'GetMouse' for FMP 8.5 compatibility — PPC only. (Aug 2006)
- 1.50 Reworked plug-in for Intel Macs, using newer plug-in architecture. Added unicode compatibility where necessary. Removed non-unicode text conversion functions. Mac release (Jan 2007).

◆ About Jazz Media

Jazz Media is a small company, with close ties to an earlier company, *Primal Screen Multimedia*. Most of the combined work to date has been creating multimedia CD-ROMs for clients, all of which involve access to large quantities of data (>50,000 records in one title). A slow migration to web-based products is in-hand.

The *James Halliday Interactive Wine Companion* by *HarperCollins Publishers* (Australia) has been one of these titles, which includes a substantial *FileMaker Pro* runtime database. The functions of this and other plugins have come about as direct or indirect requirements of this and other titles.

Presently we have developed some 20 commercial titles, and a similar number plugins for *FileMaker* and *Macromedia Director*, and others. In time more of these may be refined and released to the public as shareware or freeware.

