



>

## Jazz Stack Plug-in

>

Updated 10 September, 2005

◇

---

## Introduction

### ◆ Overview

This plugin provides a mechanism whereby values can be pushed and popped to and from a stack. Other functions allow the stack to be peeked, to be used as a pipe (allows pulling values from the other end) and marking of stack locations for quick “unwinding”.

The stack is available globally; that is, every opened database accesses the same stack. A typical use could be to implement simple “go back” functionality, by recording the history of records a user views, by diligently placing the appropriate record IDs on the stack.

### ◆ Shareware or Freeware

This plug-in is shareware. It will remind you of this fact periodically unless you register. Registration does not cost much, and encourages me to write more plugins. If you use this plug-in regularly, please register it at <http://www.jazzmedia.com.au>.

The simple license is for use by a single user. If you wish to distribute this plugin with your solution there are separate licenses available on a per-product basis. Consult the web-site for more details.

### ◆ Disclaimer

The plug-in is offered “as-is”, and comes with no warranty of any kind. You are responsible for determining its suitability for your particular purpose or purposes, and that it performs accordingly. Please do not register the plugin until you have so-determined the plugin’s suitability and effectiveness as per your requirements.

## Conventions

### ◆ Definitions

The word 'function' and 'command' are used interchangeably in this document. While technically there is a difference between the two, FileMaker only provides a single method of access so differentiation between the terms is not important.

### ◆ Calling functions

This extension makes use of the FileMaker plugin architecture. Plugin functions are designed to take exactly one argument (a single string of text) and return a single argument<sup>1</sup>. As some of the functions of this plugin require multiple or no arguments, and some of them don't return any value, some encoding of the information needs to take place to conform to the FileMaker plugin requirements.

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

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

This script step calculates the radius of a circle, where *Area* and *Radius* are *FileMaker* fields, and *Square* is the name of the function. In this example, the function accepts the field *Radius* as its argument, and returns the square of this number, which is used in the formula as shown.

When an external function does not return a value<sup>2</sup> (or the value is unimportant) there are two easy ways to use it. The first is to create an *If-End If* pair of commands, with no script steps in-between. Because the body of the *If-End If* statement is empty, the return value of the function is ignored. A *display dialog* function could be called this way:

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

An more compact way to achieve the same result is to create a global field called *temp* (for example) which is reserved for offloading unwanted function results. The same *display dialog* function would now be called like this:

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

This is the method I prefer, because of its compactness, and will be used in the examples throughout this document.

<sup>1</sup> Actually plugins accept two arguments (strings), where the first one is the name of the function, and the second one is the data. It is more convenient to consider the first string simply as the name of the function, and thus the second string is the only data passed to this command. When we talk about a single argument being passed to a function, we are referring to this second string.

<sup>2</sup> Actually all external functions return an argument, irrespective of whether it is required or not. If a return value is not required, by convention the function will return an empty string ("").

---

## General use

### ◆ Simple stack use

To be written.

### ◆ Marking the stack and dropping

To be written.

### ◆ Labelling parameters

To be written.

### ◆ Example — Implementing “go back” functionality

To be written.

See example *PHP Math.fp5* to be provided.

---

## Function Summary

### ◆ Stack Version

This function serves two purposes. With no argument (the empty string) the function simply returns a string describing the plugin, its registration status (if applicable) and the version number of the plugin. For example:

```
Set Field ["temp", "External("Stack Version", "")"]
```

will return a string similar to "Jazz Media Stack Routines 1.00". The *TextToNum* function can be used to return the version number on its own (as a decimal number), from which comparisons can be made. (For this reason the version number will contain only a single decimal point, eg "1.21" and not "1.2.1", as has become common practise today). The following example places the numeric version in the field 'plugin version'.

```
Set Field ["plugin version", "TextToNum(External("Stack Version", ""))"]
```

The second purpose of this function is in registration. The serial number provided

to registered users should be given to this function when a database (which uses this plugin) is opened. The returned string indicates whether registration was successful or not. For example, if your serial number is "12345":

```
Set Field ["temp", "External("Stack Version", 12345)"]
```

sets 'temp' to either "Jazz Media Stack Routines 1.00" or "Jazz Media Stack Routines 1.00 - Unregistered".

Even though the registration code need only be run once to register use for all open databases, it is recommended that a script containing the registration code is placed in the start-up script for every database which uses this plugin (enabled in FileMaker's document preferences).

If you accidentally perform a script with a (non-empty) invalid serial number, you will need to quit and re-launch FileMaker before the correct serial number will be accepted. You can check your registration status at any time by providing an empty string to this function, as shown earlier.

### ◆ Stack Push

This function pushes a value onto the top of the stack. The format of the command, followed by two examples of use are:

```
Set Field ["temp", "External("Stack Push", <value>)"]  
Set Field ["temp", "External("Stack Push", "ABC")"]  
Set Field ["temp", "External("Stack Push", user name)"]
```

In the first example, the string "ABC" is pushed onto the stack. In the second example the contents of the FileMaker field called 'user name' is pushed onto the stack.

For convenience the function returns the value of the item pushed, otherwise it returns the empty string (eg. if not enough memory is available to save the value).

### ◆ Stack Pop

This function removes and returns the top-most element from the stack. That is, the most recent value placed on the stack with a *Stack Push* command is removed from the stack and returned to the user. The format of the command, followed by an example is shown below:

```
Set Field [<field>, "External("Stack Pop", "")"]  
Set Field ["name", "External("Stack Pop", "")"]
```

The example places the popped value into the FileMaker field called 'name'. If there is no value on the stack (the stack is empty) then the empty string will be returned.

### ◆ Stack Pull

This function removes the bottom-most element from the stack. In conjunction

with the *Stack Push* command, a pipe can be simulated (whereby the first element pushed on the stack is the first to be removed, or *pulled*). The format of the command, followed by an example is shown below:

```
Set Field [<field>, "External("Stack Pull", "")"]  
Set Field ["name", "External("Stack Pull", "")"]
```

The example places the pulled value into the FileMaker field called 'name'. If there is no value on the stack (the stack is empty) then the empty string will be returned.

### ◆ Stack Peek

This function returns the value at the location on the stack as specified by *index*. The format of this command is:

```
Set Field ["temp", "External("Stack Peek", <index>)"]
```

The stack is unaltered by this command. If *index* is 0, the top of the stack is returned. If *index* is 1, the element second from the top of the stack is returned, and so on. If the element specified does not exist, the empty string is returned.

### ◆ Stack Size

This function returns the size of the stack (the count of elements on the stack). An empty stack returns 0. The following example places the stack size into the FileMaker field 'temp':

```
Set Field ["temp", "External("Stack Size", "")"]
```

### ◆ Stack Mark

This function places a mark next to the current top of stack. (You can use any name as the mark). Placing a mark does not affect the size or contents of the stack in any way, it simply gives an element a name, and flags it as *marked*. The mark will follow the element as the stack grows and shrinks. The mark is lost when the element is removed from the stack.

Use this function when you wish to remember the current top-of-stack for later, in combination with the *Stack Get Mark* and *Stack Drop To Mark* routines.

Usage and example:

```
Set Field ["temp", "External("Stack Mark", "<name>")"]  
Set Field ["temp", "External("Stack Mark", "fred")"]
```

Calls to *Stack Mark* and *Stack Drop To Mark* are often paired. The latter lets you quickly return to the stack as-it-was when it was previously marked.

### ◆ Stack Get Mark

This function returns the name of the mark applied to the stack at the location specified by *index*. This function is similar to *Stack Peek*, except that it returns the mark placed at that stack element (if any) rather than the value.

The format of this command is:

```
Set Field ["temp", "External("Stack Get Mark", <index>)"]
```

The stack is unaltered by this command. If *index* is 0, the mark at the top of the stack (if any) is returned. If *index* is 1, the mark of the element second from the top of the stack is returned, and so on. If there is no mark at the specified element, or the element does not exist (the index is bigger than the stack), the empty string is returned.

### ◆ Stack Drop

This command removes the top N elements from the stack. It is similar to calling *Stack Pop* several times, and discarding the values. The function returns *true* (1) if an item existed to drop, or *false* (0) if the full requested number of elements could be dropped (leaving the stack empty).

Usage and example:

```
Set Field ["temp", "External("Stack Drop", <count>)"]  
Set Field ["temp", "External("Stack Drop", 2)"]
```

The example drops two elements from the top of the stack.

### ◆ Stack Drop To Mark

This function drops elements from the top of the stack, up to but not including the element containing the mark of the given name.

If more than one mark exists with this name, the first (top-most) location will be used. If a mark by the given name can not be found anywhere on the stack, the function does nothing, and returns *false* (0). Names are case-sensitive.

Usage:

```
Set Field ["temp", "External("Stack Drop To Mark", "<name>")"]  
Set Field ["temp", "External("Stack Drop To Mark", "fred")"]
```

### ◆ Stack Clear

This routine clears the entire stack.

Usage and example:

```
Set Field ["temp", "External("Stack Clear", "")"]
```

---

## How it works

This plugin is implemented by creating a linked list of elements, operating as a stack. Each element can contain a value and a name (a string of up to 31 characters). Named elements are said to be *marked*.

More to be written.

---

## Appendices

### ◆ Version History

- 1.06 Released to the public as shareware (June 2002)
- 1.07 Fixed bug where no result was returned when the shareware dialog was displayed. Decreased dialog frequency.  
Corrected version string version number in *Stack Version* function.
- 1.08 Internal codebase update only. Issues with previous codebase are not believed to impact on *Jazz Stack*.

### ◆ About Jazz Media

Jazz Media is an Australian-based company, who's main business is in creating MultiMedia products for clients. It specialises in CD-ROMs and dynamic web sites requiring a database back-end. Use of an in-house database engine for CD-ROMs provides extremely fast access to data, and means that clients can distribute solutions royalty free.

In the course of normal business plugins have been developed, either to solve specific developmental problems, or to add functionality to programs and bundled as part of the commercial products. After years of successful operation, these plugins have been re-packaged and released to the public as shareware or freeware.