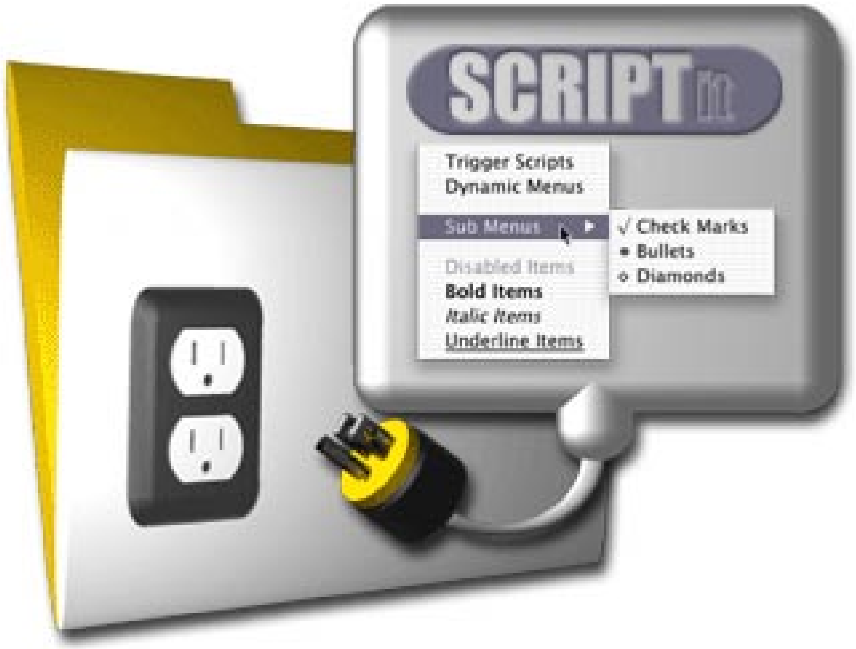


SCRIPTit

A Plug-in for FileMaker Pro

- Call scripts from any calculation
- Schedule scripts for any time
- Create dynamic pop-up menus
- Design menus with powerful XML

Comm-Unity Networking Systems
8652 Camp Bowie West
Fort Worth, Texas 76116

http://www.cnsplug-ins.com/
info@comm-unity.net

Phone: 817-560-4226
Fax: 817-244-0340

Contents

Index of Figures

Introduction

SCRIPTit is an interface enhancement plug-in that uses the power of XML for defining script events and dynamic popup menus. With this plug-in you can define script events that happen immediately, at any future time, or on any repeating interval. These script events are defined using easy to use, easy to read, and easy to understand XML Markup. This plug-in's main feature, though, is its ability to create dynamic popup menus.

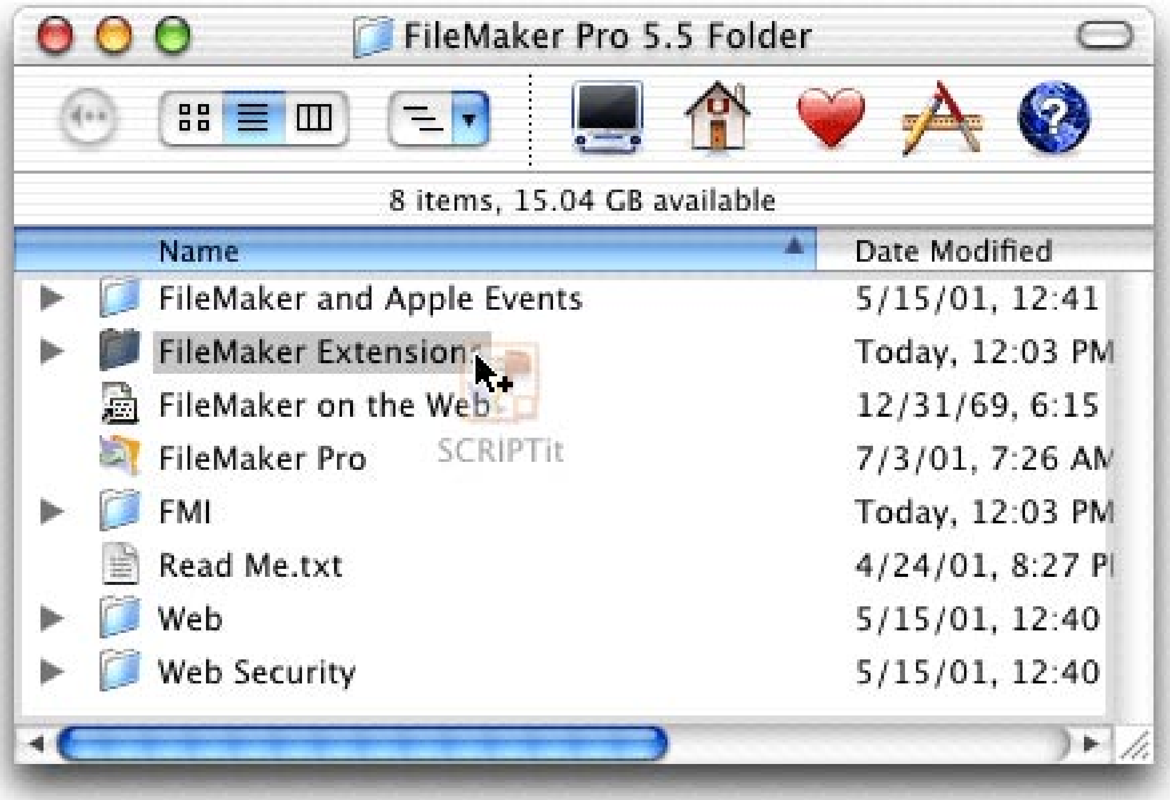
What are dynamic popup menus? Well, have you ever wanted to be able to click on a button on your layout and be presented with a popup menu of choices to perform? Have you ever wanted to have a list of values with hiearchial sub lists cascading off the main list? Have you ever wanted to have a value list where some of the items appear disabled part of the time? Have you ever wanted popups to look like Mac OS 9 menus on Mac OS 9, Mac OS X menus on OS X, and Windows menus on Windows? Well, you can have all these things, and much more with SCRIPTit's powerful dynamic popup menus!

Just like the script events, you define these popup menus with easy to learn and easy to use XML markup. You can have any number of menu items on your menus, andy any number of cascading sub menus (limited only by OS issues). The menu items can be bold, italic, underline, or any combination of those styles. You can have check marks, round bullets, and diamond shaped bullets. You can enable and disable menu items easily. In fact, you can change the complete appearance of any menu item at any time. Finally, whenever the user chooses a menu item, SCRIPTit will run a script in your database. Think of the power!

Installation & Registration

Installation

Installing SCRIPTit is very easy. If FileMaker Pro is open, then close it. Next, locate the folder on your hard drive that contains your FileMaker Pro application. For macintosh, this is usually in the Applications folder; while on windows, it is usually in the Program Files folder. Next to the FileMaker Pro application, you should see a folder named FileMaker Extensions (on Macintosh; See Figure 1) or System (on Windows; See Figure 2). Copy the plug-in file into this folder to install it into FileMaker Pro.

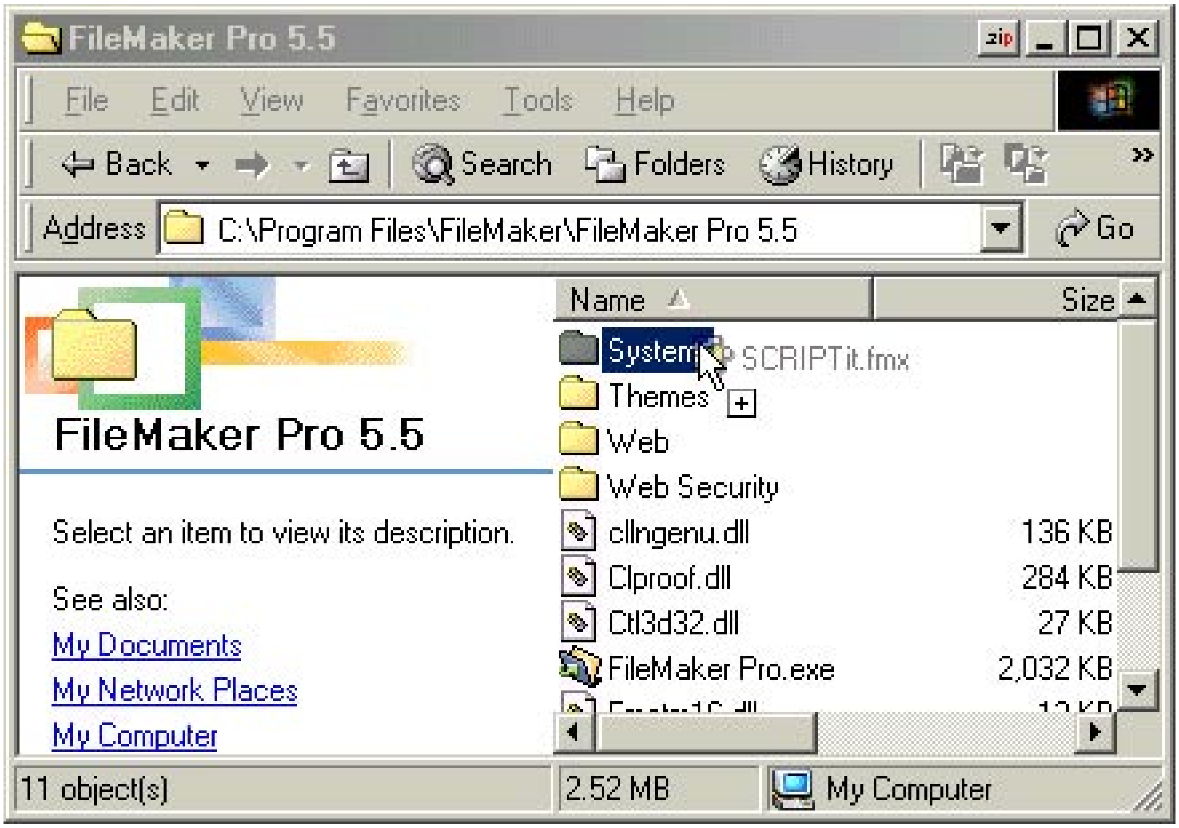


FileMaker Pro 5.5 Folder

8 items, 15.04 GB available

Name	Date Modified
FileMaker and Apple Events	5/15/01, 12:41
FileMaker Extensions	Today, 12:03 PM
FileMaker on the Web	12/31/69, 6:15
FileMaker Pro	7/3/01, 7:26 A
FMI	Today, 12:03 PM
Read Me.txt	4/24/01, 8:27 P
Web	5/15/01, 12:40
Web Security	5/15/01, 12:40

Figure 1. Installing on Macintosh



Address: C:\Program Files\FileMaker\FileMaker Pro 5.5

Select an item to view its description.

Name	Size
System	
Themes	
Web	
Web Security	
cllngenu.dll	136 KB
Clproof.dll	284 KB
Ctl3d32.dll	27 KB
FileMaker Pro.exe	2,032 KB

11 object(s) | 2.52 MB | My Computer

Figure 2. Installing on Windows

After installing the plug-in as described above, you can open FileMaker Pro again and start exploring the example databases that came with SCRIPTit.

Registration

If you find SCRIPTit useful and would like to register it, you can purchase a license to use it from our secure online purchase form. Simply go to http://www.cnsplug-ins.com/ and click on the Purchase link. Once you have purchased a license to use SCRIPTit, you will be sent a registration code to register your copy of the plug-in. To do that, simply open the Register_SCRIPTit.fp5 example database and enter in your registration information.

SCRIPTit XML Definitions

XML Conventions

First, here are a few things to remember about XML: XML is case-sensitive. A tag with the name `<menu>` is not the same as a tag with the name `<MENU>` or `<Menu>`. In the SCRIPTit XML Definitions, everything is lowercase except for the root <SCRIPTit> tag. This includes attributes; all attributes are in lowercase. Attributes are of the form:

`name="value"`

or

`name='value'`

The `"value"` part must have quotes around it, although they can be single or double quotes.

You can put comments in your XML markup. Comments start with "<!--" and end with "-->", and cannot contain two sequential dashes anywhere in the comment.

Now on to the XML tags found in SCRIPTit.

<SCRIPTit> Tag

This is the Root Tag of the SCRIPTit XML documents. It is the equivalent of the <HTML> tag in HTML documents. The only tags that can appear inside the root `<SCRIPTit>` tag are the <script>, <menu>, <updatemenuitem>, <insertmenuitem>, and <deletemenuitem> tags. This tag has no attributes.

<script> Tag

This tag allows you to define script events for SCRIPTit to call. You need to define at least the `db` and `script` attributes to have a valid `<script>` tag. This is an empty tag, so you can either use a forward-slash before the closing angled bracket ("`/>`") or follow the tag immediately with a closing `</script>` tag. This tag has eight attributes, two of which are required:

`db` **Attribute** - Defines the database that contains the script to be called. Specify only the FileName of the database. You can use the `Status(CurrentFileName)` function in a FileMaker Pro calculation to obtain the FileName of the database. This attribute is required.

`script` **Attribute** - Defines the name of the script as it appears in your database. This attribute is required.

`name` **Attribute** - Defines a user-supplied name of this script event. You can name all of your script events and use the Scrp-LastScriptName function to retrieve that name when the actual script is called. This attribute is required if you also define the `date`, `time`, `interval`, `repeat`, or `persistent` attributes (so that you can later delete them if you no longer need them); the default value is `""`.

`value` **Attribute** - Defines a user-supplied value for this script event. You can store any value you want in this attribute and retrieve the value with the Scrp-LastScriptValue function. This attribute is optional; the default value is `""`.

`date` **Attribute** - Defines the date the script event should occur. This attribute has a very specific form and should look like `"01/21/2002"`. That is, month, forward-slash, day, forward-slash, four digit year. If you have a date field that you want the event to occur on, you can use something like '`DateToText(dateField)`' and it should output the correct form. If you are using a non-English version of FileMaker, you may have to use something like '`Month(dateField) & "/" & Day(dateField) & "/" & Year(dateField)`'. This attribute is optional; default value is `""`, unless the time attribute is defined, in which case the default value is today's date.

`time` **Attribute** - Defines the time the script event should occur. This attribute has a very specific form and should look like `"14:30:00"`. That is, 24-hour hour, colon, minutes, colon, seconds. If you have a time field that you want the event to occur on, you can use something like '`TimeToText(timeField)`' and it should output the correct form. This attribute is optional; default value is `""`, unless the date attribute is defined, in which case the default value is midnight (`"00:00:00"`).

`interval` **Attribute** - Defines the interval for the script event. This attribute has a very specific form and should look like `"1d 2h 3m 4s"`. That is, a number followed by 'd' (days), 'h' (hours), 'm' (minutes), or 's' (seconds), with multiple measurements separated by a space. This attribute is optional; the default value is `""`.

`repeat` **Attribute** - Defines whether or not the script event should repeat. Used in conjunction with the interval attribute for a repeating script event. Specify either `"yes"` or `"no"`. This attribute is optional; the default value is `"no"`.

`persistent` **Attribute** - Defines whether or not this script event should still occur even if FileMaker Pro is closed and reopened later. Specify either `"yes"` or `"no"`. This attribute is optional; the default value is `"no"`.

Examples:

```
<script db="test.fp5" script="go" />
```

Calls script "go" in database "test.fp5" immediately.

```
<script db="test.fp5" script="check" name="record check" value="3" />
```

Calls script "check" in database "test.fp5" immediately, with user-supplied name "record check" and user-supplied value "3".

```
<script db="datebook.fp5" script="reminder" name="date reminder"
     date="03/05/2002" time="08:30:00" />
```

Calls script "reminder" in database "datebook.fp5" at 8:30 AM on March 5, 2002.

```
<script db="timer.fp5" script="time's up" name="timer" interval="1h 15m" />
```

Calls script "time's up" in database "timer.fp5" in 1 hour and 15 minutes from now.

```
<script db="appoint.fp5" script="check appointments" name="check" interval="30m"
     time="7:00:00" />
```

Calls script "check appointments" in database "appoint.fp5" every 30 minutes starting today.

```
<script db="email.fp5" script="check email" name="check email" interval="5m"
     repeat="yes" persistent="yes" />
```

Calls script "check email" in database "test.fp5" every 5 minutes. Will be saved and resumed when FileMaker Pro is closed and reopened.

<menu> Tag

This tag is the main tag for all menu definitions. You need to define at least the `name` attribute to have a valid `<menu>` tag. This tag can contain <menuitem>, <submenu>, and <insertmenu> tags. This tag has five attributes, one of which is required:

`name` **Attribute** - Defines the name of this menu. You use this name when telling SCRIPTit to display the menu, when deleting the menu, or when inserting the menu into another menu. This attribute is required.

`action` **Attribute** - Defines whether or not this is a new menu or an update to an existing menu. Specify either `"new"` or `"update"`. This attribute is optional; default value is `"new"`.

`defaultdb` **Attribute** - Defines the default database for all the menu items contained in this menu. If you do not specify a database when defining a menu item, this default database will be used. This attribute is optional; default value is "".

`defaultscript` **Attribute** - Defines the default script for all the menu items contained in this menu. If you do not specify a script when defining a menu item, this default script will be used. This attribute is optional; default value is "".

`defaultvalue` **Attribute** - Defines the default value for all the menu items contained in this menu. If you do not specify a value when defining a menu item, this default value will be used. This attribute is optional; default value is "".

`modifierkeys` **Attribute** - Defines the modifier keys that are required in order to show the menu. Specify `"none"` or `"shift"`, `"ctrl"`, `"opt"`, `"cmd"`, or combinations using the plus sign (`"shift+ctrl+cmd"`). If the specified modifier keys are not held down when you call Scrp-ShowMenu, then the menu will not be shown. Instead, the default script will be called in the default database. If you are making a cross-platform solution, you will only want to use the `"ctrl"` and `"shift"` modifier keys because windows does not have the `"opt"` and `"cmd"` keys. This attribute is optional; default value is `"none"`.

Examples:

```
<menu name="test">
```

Defines a menu named "test".

```
<menu name="test" action="update">
```

Updates the menu named "test".

```
<menu name="my menu" defaultdb="mydb.fp5" defaultscript="menu click">
```

Defines a menu named "my menu", whose menu items will use the "menu click" script in the "mydb.fp5" database unless those menu items define their own script and/or database.

```
<menu name="menu" defaultdb="test.fp5" defaultscript="find all"
    modifierkeys="ctrl">
```

Defines a menu named "menu", whose menu items will use the "find all" script in the "test.fp5" database unless those menu items define their own script and/or database. If the Control key is not held down when the Scrp-ShowMenu function is called, the "find all" script in the "test.fp5" database will be called instead of showing the menu.

<submenu> Tag

This tag defines a submenu of the current menu. You can use this tag to create heiarchial menus. You need to define at least the name attribute to have a valud `<submenu>` tag. This tag can contain `<menuitem>`, `<submenu>`, and `<insertmenu>` tags. This tag has seven attributes, one of which is required:

`name` **Attribute** - Defines the name of this submenu. This name appears on the parent menu where the submenu cascades from the parent menu. This attribute is required.

`defaultdb` **Attribute** - Defines the default database for all the menu items contained in this submenu. If you do not specify a database when defining a sub menu item, this default database will be used. This attribute is optional; default value is `""`.

`defaultscript` **Attribute** - Defines the default script for all the menu items contained in this submenu. If you do not specify a script when defining a sub menu item, this default script will be used. This attribute is optional; default value is `""`.

`defaultvalue` **Attribute** - Defines the default value for all the menu items contained in this submenu. If you do not specify a value when defining a submenu item, this default value will be used. This attribute is optional; default value is "".

`enabled` **Attribute** - Defines whether or not this submenu is enabled. Specify either "yes" or "no". If the submenu is disabled, it will appear greyed and the submenu will not appear when the user places their cursor over it. This attribute is optional; default value is "yes".

`groupid` **Attribute** - Defines a user-supplied group identifier for this submenu. This is used in conjunction with the `modifierkeys` attribute to selectively show alternative menu items depending on which modifer keys are held down. See the Full Menu Examples section for examples that use this attribute. This attribute is optional; default value is "".

`modifierkeys` **Attribute** - Defines the modifier keys that are required in order to show this submenu. Specify "none" or "shift", "ctrl", "opt", "cmd", or combinations using the plus sign ("shift+ctrl+cmd"). If the specified modifier keys are not held down when this menu is shown, this submenu will not be shown. If you are making a cross-platform solution, you will only want to use the "ctrl" and "shift" modifier keys because windows does not have the "opt" and "cmd" keys. This attribute is optional; default value is "none".

`style` **Attribute** - Defines the font style of this submenu. Specify "none" or "bold", "italic", "underline", or combinations using the plus sign ("bold+italic"). This attribute is optional; default value is "none".

Examples:

```
<submenu name="Go to Layout">
```

Defines a submenu named "Go to Layout".

```
<submenu name="Go to Layout" defaultdb="main.fp5"
    defaultscript="Change Layout">
```

Defines a submenu named "Go to Layout", whose menu items will use the "Change Layout" script in the "main.fp5" database unless those menu items define their own script and/or database.

```
<submenu name="Special" enabled="no" style="italic">
```

Defines a submenu named "Special" that is disabled and whose name is displayed using the italic font.

```
<submenu name="Hidden" modiferkeys="shift+ctrl" groupid="hidden">
```

Defines a submenu named "Hidden" that is only displayed if the Shift and Control keys are held down when showing the menu.

<menuitem> Tag

This tag is used to define a single item on the menu or submenu. You need to at least define the name attribute to have a valid <menuitem> tag. This is an empty tag, so you can either use a forward-slash before the closing angled bracket ("/>") or follow the tag immediately with a closing </menuitem> tag. This tag has nine attributes, one of which is required:

name **Attribute** - Defines the name of the menu item. This name appears on the menu. If the name is defined as "-", a separator will be placed at this location in the menu instead of a normal menu item. If you are defining a menu separator, only the groupid and modifierkeys attributes will be used; all others will be ignored. The name of the chosen menu item can be retrieved using the Scrp-LastMenuChoice function. This attribute is required.

db **Attribute** - Defines the database that contains the script for this menu item. If this is not defined, the default database from the parent submenu or menu will be used. This attribute is optional; the default value is "".

script **Attribute** - Defines the script to be called when the user selects this menu item. If this is not defined, the default script from the parent submenu or menu will be used. This attribute is optional; the default value is "".

value **Attribute** - Defines a user-supplied value for this menu item. This value can be retrieved using the Scrp-LastMenuValue function. This attribute is optional; the default value is "".

enabled **Attribute** - Defines whether or not this menu item is enabled. Specify either "yes" or "no". If the menu item is disabled, it will appear greyed and the user will not be able to select it. This attribute is optional; the default value is "yes".

groupid **Attribute** - Defines a user-supplied group identifier for this menu item. This is used in conjunction with the modifierkeys attribute to selectively show alternative menu items depending on which modifer keys are held down. See the Full Menu Examples section for examples that use this attribute. This attribute is optional; default value is "".

modifierkeys **Attribute** - Defines the modifier keys that are required in order to show this menu item. Specify "none" or "shift", "ctrl", "opt", "cmd", or combinations using the plus sign ("shift+ctrl+cmd"). If the specified modifier keys are not held down when this menu is shown, this menu item will not be shown. If you are making a cross-platform solution, you will only want to use the "ctrl" and "shift" modifier keys because windows does not have the "opt" and "cmd" keys. This attribute is optional; default value is "none".

style **Attribute** - Defines the font style of this menu item. Specify "none" or "bold", "italic", "underline", or combinations using the plus sign ("bold+italic"). This attribute is optional; default value is "none".

mark **Attribute** - Defines the mark for this menu item. Specify "none", "check", "bullet", or "diamond". The mark is displayed to the left of the menu item. This attribute is optional; default value is "none".

Examples:

```
<menuitem name="Click Me" />
```

Defines a menu item with the caption "Click Me" that uses the default database and default script of the parent menu.

```
<menuitem name="Click Me" script="menu click" value="item4" />
```

Defines a menu item with the caption "Click Me" that runs the "menu click" script in the default database and returns "item4" when the Scrp-LastMenuValue function is used.

```
<menuitem name="Hidden" style="bold" mark="bullet"
    modifierkeys="shift+opt+ctrl+cmd" db="test.fp5" script="hidden"/>
```

Defines a menu item with the caption "Hidden" that calls the "hidden" script in the "test.fp5" database. This item is displayed in the bold font with a bullet mark on the left, and is only displayed if the Shift, Option, Control, and Command keys are held down.

<insertmenu> Tag

This tag allows you to insert any menu into another menu when that menu is displayed. The menu you are inserting does not have to be defined when you define the parent menu (the one you are inserting into), but it does need to be defined by the first time you display the parent menu. The items from the inserted menu are inserted directly into the parent menu. If you want to insert the menu as a sub menu, you must wrap the `<insertmenu>` tag in <submenu> and </submenu> tags. If none of the menu items from the menu you are inserting have databases and/or scripts defined, then the parent menu's default database and default script will be used. You need to at least define the `name` attribute to have a valid `<insertmenu>` tag. This is an empty tag, so you can either use a forward-slash before the closing angled bracket ("`/>`") or follow the tag immediately with a closing `</insertmenu>` tag. This tag has three attributes, one of which is required:

`name` **Attribute** - Defines the name of the menu you want to insert. When you show the parent menu, this name must equal a name of another menu you have defined. This attribute is required.

`groupid` **Attribute** - Defines a user-supplied group identifier for this inserted menu. This is used in conjunction with the `modifierkeys` attribute to selectively show alternative menu items depending on which modifer keys are held down. See the Full Menu Examples section for examples that use this attribute. This attribute is optional; default value is "".

`modifierkeys` **Attribute** - Defines the modifier keys that are required in order to insert this menu. Specify `"none"` or `"shift"`, `"ctrl"`, `"opt"`, `"cmd"`, or combinations using the plus sign (`"shift+ctrl+cmd"`). If the specified modifier keys are not held down when the parent menu is shown, this menu will not be inserted. If you are making a cross-platform solution, you will only want to use the `"ctrl"` and `"shift"` modifier keys because windows does not have the `"opt"` and `"cmd"` keys. This attribute is optional; default value is `"none"`.

Examples:

`<insertmenu name="Layouts" />`

Inserts a menu named "Layouts" into the current menu.

`<insertmenu name="Hidden" modifierkeys="shift+cmd" />`

Inserts a menu named "Hidden" into the current menu if the Shift and Command keys are held down.

<updatemenuitem> Tag

This tag allows you to update the attributes of any menu item on any menu you have previously defined. The `<updatemenuitem>` tag is useful for enabling a previously disabled menu item (or vice versa), placing a mark next to an item, or changing any other attribute assocated with a menu item. This tag is used by itself like the <script> tag and does not need to appear within <menu> or <submenu> tags. This tag is an almost exact replica of the <menuitem> tag, except that instead of specifying just the name of the menu item, you must specify the "path" to the menu item so that SCRIPTit knows which item to update. Only those attributes that you define will replace the original menu item's attributes. You need to at least define the `name` attribute to have a valid `<updatemenuitem>` tag. This is an empty tag, so you can either use a forward-slash before the closing angled bracket ("`/>`") or follow the tag immediately with a closing `</updatemenuitem>` tag. This tag has nine attributes, one of which is required:

`name` **Attribute** - Defines the "path" to the menu item that you are updating. The "path" is made up of the menu name, followed by any submenu names, followed by the actual name of the menu item, all separated by the forward-slash, and looks like `"menu name/submenu name/menu item name"`. This attribute is required.

`db` **Attribute** - Defines the database that contains the script for this menu item. If this is not defined, the default database from the parent submenu or menu will be used. This attribute is optional; if the attribute is missing, the original database attribute will be unchanged; if the attribute exists, but is set to `""`, the original database attribute will be cleared.

`script` **Attribute** - Defines the script to be called when the user selects this menu item. If this is not defined, the default script from the parent submenu or menu will be used. This attribute is optional; if the attribute is missing, the original script attribute will be unchanged; if the attribute exists, but is set to `""`, the original script attribute will be cleared.

`value` **Attribute** - Defines a user-supplied value for this menu item. This value can be retrieved using the Scrp-LastMenuValue function. This attribute is optional; if the attribute is missing, the original value attribute will be unchanged; if the attribute exists, but is set to `""`, the original value attribute will be cleared.

`enabled` **Attribute** - Defines whether or not this menu item is enabled. Specify either `"yes"` or `"no"`. If the menu item is disabled, it will appear greyed and the user will not be able to select it. This attribute is optional; if the attribute is missing, the original enabled attribute will be unchanged.

groupid **Attribute** - Defines a user-supplied group identifier for this menu item. This is used in conjunction with the modifierkeys attribute to selectively show alternative menu items depending on which modifer keys are held down. See the Full Menu Examples section for examples that use this attribute. This attribute is optional; if the attribute is missing, the original groupid attribute will be unchanged; if the attribute exists, but is set to "", the original groupid attribute will be cleared.

modifierkeys **Attribute** - Defines the modifier keys that are required in order to show this menu item. Specify "none" or "shift", "ctrl", "opt", "cmd", or combinations using the plus sign ("shift+ctrl+cmd"). If the specified modifier keys are not held down when this menu is shown, this menu item will not be shown. If you are making a cross-platform solution, you will only want to use the "ctrl" and "shift" modifier keys because windows does not have the "opt" and "cmd" keys. This attribute is optional; if the attribute is missing, the original modiferkeys attribute will be unchanged; if the attribute is "none", the original modiferkeys attribute will be cleared.

style **Attribute** - Defines the font style of this menu item. Specify "none" or "bold", "italic", "underline", or combinations using the plus sign ("bold+italic"). This attribute is optional; if the attribute is missing, the original style attribute will be unchanged; if the attribute is "none", the original style attribute will be cleared.

mark **Attribute** - Defines the mark for this menu item. Specify "none", "check", "bullet", or "diamond". The mark is displayed to the left of the menu item. This attribute is optional; if the attribute is missing, the original mark attribute will be unchanged; if the attribute is "none", the original mark attribute will be cleared.

Examples:

```
<updatemenuitem name="my menu/Click Me" enabled="no" />
```

Updates the "Click Me" menu item in the menu named "my menu" to be disabled.

```
<updatemenuitem name="my menu/Go to Layout/Main" mark="check" />
```

Updates the "Main" menu item in the submenu "Go to Layout" of the menu named "my menu" to be checked.

```
<updatemenuitem name="test menu/Item4" db="" script="" />
```

Clears the database and script attributes of the "Item4" menu item in the menu named "test menu".

<insertmenuitem> Tag

This tag allows you to insert a new menu item into any menu you have previously defined. The <insertmenuitem> tag is useful for creating menus based on records in a portal. This tag is used by itself like the <script> tag and does not need to appear within <menu> or <submenu> tags. This tag is an almost exact replica of the <menuitem> tag, except that you must specify the "path" to the menu item you want to insert before so that SCRIPTit knows where to insert the new item. If you want to insert a menu item as the last item on the menu, use "__LAST__" as the item name. You need to at least define the name and before attributes to have a valid <insertmenuitem> tag. This is an empty tag, so you can either use a forward-slash before the closing angled bracket ("/>") or follow the tag immediately with a closing </insertmenuitem> tag. This tag has eleven attributes, two of which are required:

name **Attribute** - Defines the name of the menu item. This name appears on the menu. If the name is defined as "-", a separator will be placed at this location in the menu instead of a normal menu item. If you are defining a menu separator, only the groupid and modifierkeys attributes will be used; all others will be ignored. The name of the chosen menu item can be retrieved using the Scrp-LastMenuChoice function. This attribute is required.

before **Attribute** - Defines the "path" to the menu item that you are inserting before. The "path" is made up of the menu name, followed by any submenu names, followed by the actual name of the menu item, all separated with the forward-slash, and looks like "menu name/submenu name/menu item name". If inserting as the last item, use "__LAST__" as the menu item like "menu name/submenu name/__LAST__". This attribute is required.

autocreatemenu **Attribute** - Defines whether or not SCRIPTit should auto create the menu or any submenus specified in the "path" from the before attribute if they do not already exist. In other words, if you specify "my menu/Layouts/__LAST__" as the before attribute, but the "Layouts" submenu does not exist, SCRIPTit will create it. Specify either "yes" or "no". This attribute is optional; the default value is "no".

db **Attribute** - Defines the database that contains the script for this menu item. If this is not defined, the default database from the parent submenu or menu will be used. This attribute is optional; the default value is "".

script **Attribute** - Defines the script to be called when the user selects this menu item. If this is not defined, the default script from the parent submenu or menu will be used. This attribute is optional; the default value is "".

value **Attribute** - Defines a user-supplied value for this menu item. This value can be retrieved using the Scrp-LastMenuValue function. This attribute is optional; the default value is "".

enabled **Attribute** - Defines whether or not this menu item is enabled. Specify either "yes" or "no". If the menu item is disabled, it will appear greyed and the user will not be able to select it. This attribute is optional; the default value is "yes".

groupid **Attribute** - Defines a user-supplied group identifier for this menu item. This is used in conjunction with the modifierkeys attribute to selectively show alternative menu items depending on which modifer keys are held down. See the Full Menu Examples section for examples that use this attribute. This attribute is optional; default value is "".

`modifierkeys` **Attribute** - Defines the modifier keys that are required in order to show this menu item. Specify `"none"` or `"shift"`, `"ctrl"`, `"opt"`, `"cmd"`, or combinations using the plus sign (`"shift+ctrl+cmd"`). If the specified modifier keys are not held down when this menu is shown, this menu item will not be shown. If you are making a cross-platform solution, you will only want to use the `"ctrl"` and `"shift"` modifier keys because windows does not have the `"opt"` and `"cmd"` keys. This attribute is optional; default value is `"none"`.

`style` **Attribute** - Defines the font style of this menu item. Specify `"none"` or `"bold"`, `"italic"`, `"underline"`, or combinations using the plus sign (`"bold+italic"`). This attribute is optional; default value is `"none"`.

`mark` **Attribute** - Defines the mark for this menu item. Specify `"none"`, `"check"`, `"bullet"`, or `"diamond"`. The mark is displayed to the left of the menu item. This attribute is optional; default value is `"none"`.

Examples:

```
<insertmenuitem name="Click Me" before="my menu/Testing"
    script="Menu Item Click" />
```

Inserts a new menu item named "Click Me" before the menu item named "Testing" on the menu named "my menu".

```
<insertmenuitem name="Plants" before="my menu/See Also/__LAST__"
    autoupdatemenu="yes" />
```

Inserts a new menu item named "Plants" as the last item on the "See Also" submenu of the menu named "my menu". If the "See Also" submenu does not exist, SCRIPTit will create it.

<deletemenuitem> Tag

This tag allows you to delete a menu item from any menu you have previously defined. This tag is used by itself like the `<script>` tag and does not need to appear within `<menu>` or `<submenu>` tags. You need to at least define the `name` attribute to have a valid `<deletemenuitem>` tag. This is an empty tag, so you can either use a forward-slash before the closing angled bracket ("`/>`") or follow the tag immediately with a closing `</deletemenuitem>` tag. This tag has one attribute, which is required:

`name` Attribute - Defines the "path" to the menu item that you are deleteing. The "path" is made up of the menu name, followed by any submenu names, followed by the actual name of the menu item, all separated with the forward-slash, and looks like `"menu name/submenu name/menu item name"`. This attribute is required.

Examples:

```
<deletemenuitem name="my menu/Click Me" />
```

Deletes the menu item named "Click Me" from the menu named "my menu".

```
<deletemenuitem name="my menu/See Also/Plants" />
```

Deletes the menu item named "Plants" from the sub menu named "See Also" of the menu named "my menu".

Full Menu Examples

This first example just creates a generic menu to give you an idea of how it works and what the end result looks like. Each of the menu items rely on the default database and default script of the menu.

```
<menu name="menu1" defaultdb="test.fp5" defaultscript="menu click">
  <menuitem name="Item 1" />
  <menuitem name="Item 2" />
  <menuitem name="-" /> <!-- this produces a separator in the menu -->
  <submenu name="Sub Menu">
    <menuitem name="Sub Item 1" />
    <menuitem name="Sub Item 2" />
  </submenu>
</menu>
```

When you use the Scrp-ShowMenu function to show "menu1", a menu will popup that has 4 items on it: "Item 1", "Item 2", a separator, and "Sub Menu". When you hover the mouse over the Sub Menu item, the submenu will pop open that has the "Sub Item 1" and "Sub Item 2" items in it (See Figure 3). Clicking on any of the items, even the submenu items, will result in the "menu click" script in the "test.fp5" database being called. If the "Sub Menu" submenu item defined the defaultdb and defaultscript attributes, then when clicking on the submenu items, those defaultdb and defaultscript attributes would be used.

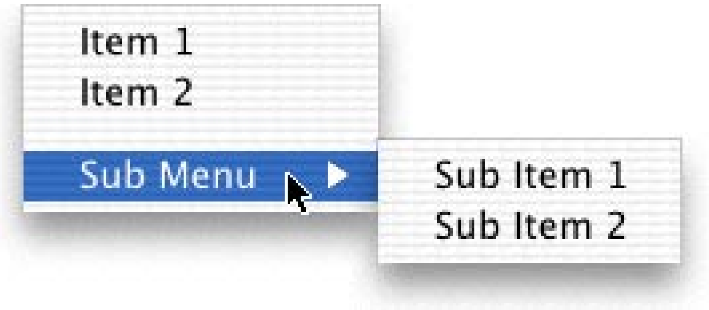


Figure 3. First Example Menu

This next example shows how to hide menu items unless certain modifier keys are held down when the menu is shown.

```
<menu name="menu1" defaultdb="test.fp5" defaultscript="menu click">
  <menuitem name="Item 1" />
  <menuitem name="Item 2" />
  <menuitem name="-" modifierkeys="ctrl+shift" />
  <menuitem name="Hidden Item 1" modifierkeys="ctrl+shift" />
  <menuitem name="Hidden Item 2" modifierkeys="ctrl+shift" />
</menu>
```

When this menu is shown, and no modifier keys are held down, the menu will just pop up with the "Item 1" and "Item 2" menu items (See Figure 4). However, if the Control and Shift keys are held down when the menu is shown, you will also see the separator, the "Hidden Item 1" and the "Hidden Item 2" menu items (See Figure 5).

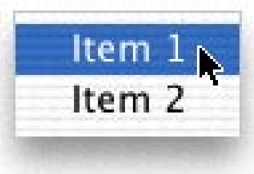


Figure 4. Menu with no Modifier Keys

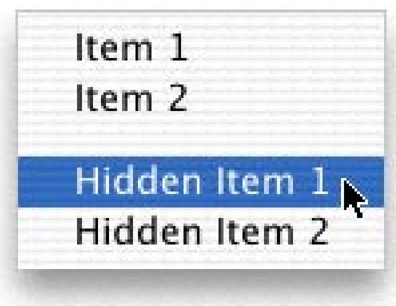


Figure 5. Menu with Control and Shift held down

This next example shows how to use the modifierkeys and groupid attributes to selectively show items depending on which modifer keys were held down when the menu was first shown.

```
<menu name="menu1" defaultdb="test.fp5" defaultscript="menu click">
  <menuitem name="Item 1" />
  <menuitem name="Item 2" />
  <menuitem name="Item 3 (ctrl)" modifierkeys="ctrl" groupid="item3" />
  <submenu name="Item 3 (shift)" modifierkeys="shift" groupid="item3">
    <menuitem name="Sub Item" />
  </submenu>
  <insertmenu name="menu2" modifierkeys="ctrl+shift" groupid="item3" />
  <menuitem name="Item 3" groupid="item3" />
</menu>
<menu name="menu2">
  <menuitem name="Menu 2 Item (ctrl+shift)" />
</menu>
```

When the menu named "menu1" is shown, you will see the items "Item 1" and "Item 2" all the time. However, for "Item 3" it will depend on which modifier keys were held down when the menu was shown. If no modifier keys are held down, then "Item 3" will show (See Figure 6). If the Control key is held down, the "Item 3 (ctrl)" menu item will show. If the Shift key is held down, then a submenu with the name "Item 3 (shift)" will be shown (See Figure 7). If the Shift and Control keys are held down, then the menu named "menu2" will be inserted, which will place the item "Menu 2 Item (ctrl+shift)" in the menu. For this to work, all four of those alternative items must have the same group identifier; in this case "item3". If you want a grouped alternative item to show when there are no modifierkeys, it must be the last item in the group. This is why the normal "Item 3" with no modifier keys is the last item of the group.

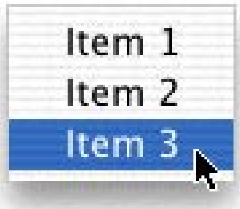


Figure 6. Menu with no Modifier Keys

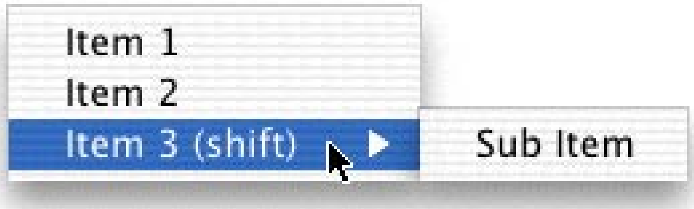


Figure 7. Menu with Shift held down.

This example shows off different variations of the styles and marks available (See Figure 8). One of the items is also disabled.

```
<menu name="menu1" defaultdb="test.fp5" defaultscript="menu click">
  <menuitem name="Item 1" style="bold" />
  <menuitem name="Item 2" mark="check" enabled="no" />
  <menuitem name="Item 3" style="bold+underline" mark="diamond" />
  <menuitem name="Item 4" style="italic" mark="bullet" />
</menu>
```

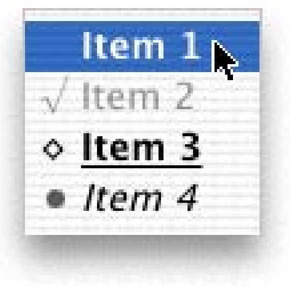


Figure 8. Menu with styles

This next example shows how an inserted menu will use the default database and/or default script of the parent menu it was inserted into.

```
<menu name="menu1" defaultdb="test.fp5" defaultscript="Menu 1 Click">
  <insertmenu name="menu3" />
</menu>
<menu name="menu2" defaultdb="test.fp5" defaultscript="Menu 2 Click">
  <insertmenu name="menu3" />
</menu>
<menu name="menu3">
  <menuitem name="Item 1" />
  <menuitem name="Item 2" />
  <menuitem name="Item 3" script="Item 3 Click" />
</menu>
```

When the menu named "menu1" is shown, and the user clicks on "Item 1" or "Item 2", the "Menu 1 Click" script in the "test.fp5" database will be called. However, when the menu named "menu2" is shown, and the user clicks on "Item 1" or "Item 2", the "Menu 2 Click" script in the "test.fp5" database will be called. If either the "menu1" or "menu2" menus are shown, and the user clicks on "Item3", then the "Item 3 Click" script in the "test.fp5" database will be called.

Now that you have seen some examples of the kinds of menus that you can create with SCRIPTit, let's look at some practical examples. Let's say you have a value list of items and you want to make it to where when a user picks an item from the list, it calls a script to make sure they can select that item, or to do some other task. Here is a quick and eazy way to transform a value list into a SCRIPTit menu:

```
Set Field [SCRIPTit Result, External("Scrp-DefineXML",

    "<menu name='Products' action='update' defaultdb='" &
    Status(CurrentFileName) & "' defaultscript='Product Menu Click'>¶" &

    "<menuitem name='" & Substitute( ValueListItems(
    Status(CurrentFileName), "Products"), "¶", "' />¶<menuitem name='") &
    "' />¶" &

    "</menu>") & "¶" &

    External("Scrp-ShowMenu", "Products")]
```

This example uses one of FileMaker's very useful, but not widely used, Design calculation functions, the `ValueListItems` function. That function will give you a return separated list of items from the value list you specify. The `Subsitute` function is then used to replace all those returns (the paragraph marks) with the appropriate XML markup. If the "Products" value list contained these items:

Apple
Banana
Pear

Then the above calculation would transform that into this XML Markup:

```
<menu name='Products' action='update' defaultdb='test.fp5'
defaultscript='Product Menu Click'>
<menuitem name='Apple' />
<menuitem name='Banana' />
<menuitem name='Pear' />
</menu>
```

This is then sent to the Scrp-DefineXML function to create the menu, and then the Scrp-ShowMenu function is used to bring up the menu. If the user selects any of the items, it will run the "Product Menu Click" script to do any checking you may want to do based on the item chosen. The "Product Menu Click" script can use the Scrp-LastMenuChoice function to retrieve which item was actually chosen.

This value list example can be quite powerful if you stop and think about all the kinds of value lists you can have. Value lists are not just lists that have manually typed in values. You can create value lists based on records from another database. You can create value lists from relationships where you only see the related values. You can even use value lists from other files. So, turning a value list into a menu can be very useful.

This next example shows how to make a menu of layouts and when the user chooses one of the items, it will switch to that layout. This example contains two main scripts and several scripts named "Go to Layout XXXX" where "XXXX" is the name of the layout. The first script contains a calculation that looks almost exactly like the value list example above, except that it uses one of FileMaker's other Design calculation functions, the LayoutNames function:

```
Set Field [SCRIPTit Result, External("Scrp-DefineXML",

    "<menu name='Layouts' action='update' defaultdb='" &
    Status(CurrentFileName) & "' defaultscript='Layout Menu Click'>¶" &

    "<menuitem name='" & Substitute( LayoutNames( Status(CurrentFileName)),
    "¶", "' />¶<menu item name='") & "' />¶" &

    "</menu>") & "¶" &

    External("Scrp-ShowMenu", "Layouts")]
```

The "Layout Menu Click" script contains this:

```
Set Field [SCRIPTit Result, External("Scrp-DefineXML",
    "<script db='" & Status(CurrentFileName) & "' script='Go to Layout " &
    External("Scrp-LastMenuChoice", "") & "' />")]
```

When the first script is called, the user is presented with a menu containing all the layout names. When the user selects a menu item, the "Layout Menu Click" script is run. That script then turns around and sets up an immediate script event that calls a script called "Go to Layout XXXX" where "XXXX" is the name of the layout. It gets the name of the layout selected by the user by using the Scrp-LastMenuChoice function.

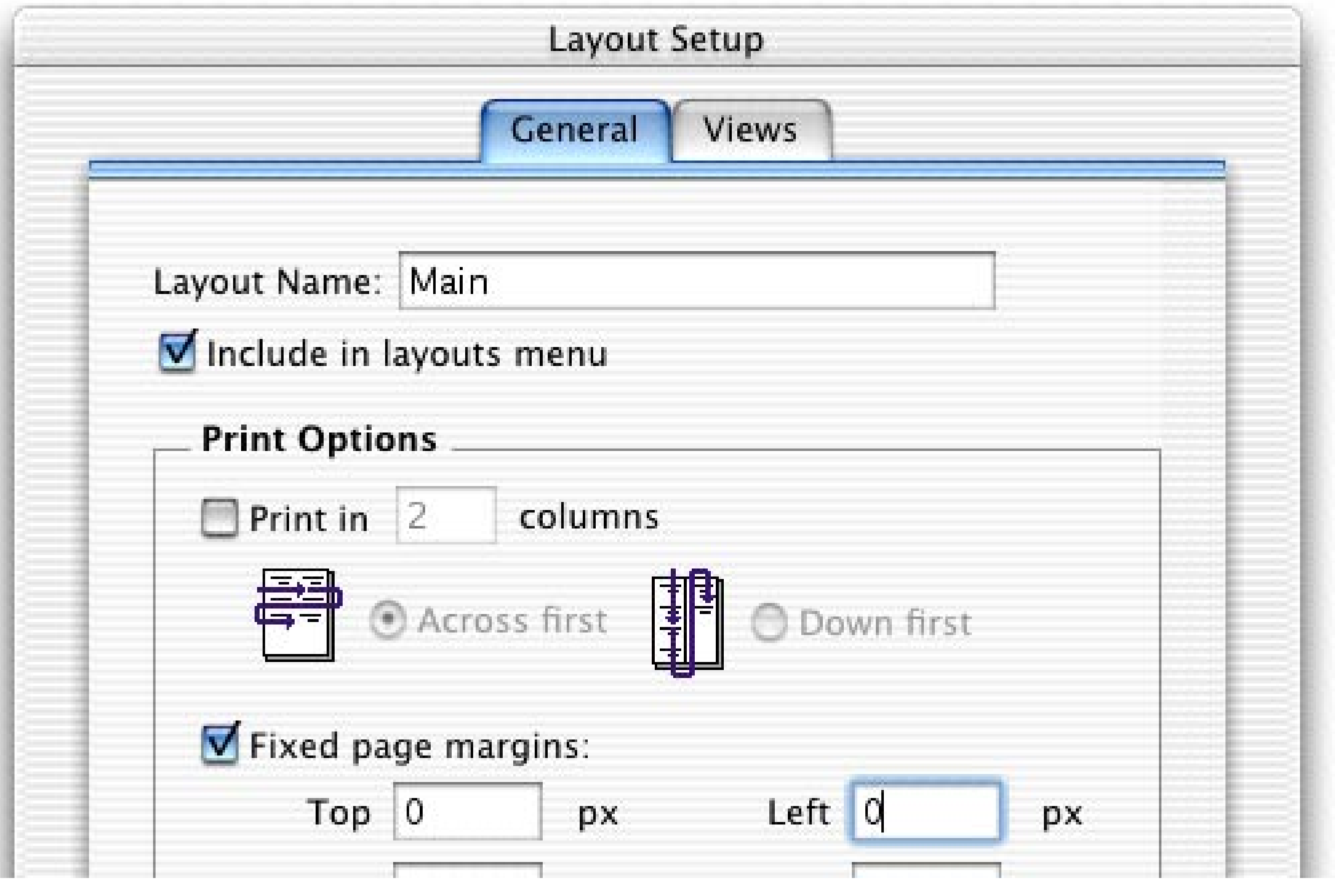
This next example will show you how to create a menu from items in a portal. This takes a little more work, but it can be highly flexible. Here is the script:

```
Set Field [SCRIPTit Result, External("Scrp-DefineXML",
    "<menu name='Products' action='update' defaultdb='" &
    Status(CurrentFileName) & "' defaultscript='Product Menu Click'>¶" &
    "</menu>")]
Go to Field [Products::Name]
Go to Portal Row [First]
Loop
    Set Field [SCRIPTit Result, External("Scrp-DefineXML",
        "<insertmenuitem before='Products/__LAST__' name='" &
        Products::Name & "' value='" & Products::ID & "' />")]
    Go to Portal Row [Exit after last, Next]
End Loop
Set Field [SCRIPTit Result, External("Scrp-ShowMenu", "Products")]
```

This example uses the <insertmenuitem> tag to insert each of the items from the portal into the empty menu created in the first Set Field. Whenever the user chooses one of the items in the menu, the "Product Menu Click" script in the current database will be run.

Creating menus from portals can be very flexible. You can have custom sort orders for the items (unlike alphabetically in a value list). You can create menus based on the current found set, to only show certain items. You can use values from the database to selective disable certain items, or bold other items. You can use the value attribute to give your menu items a different value than what is shown on the menu (something that cannot be done with value lists). The only thing that this example requires is that the portal be on the current layout when the script is run. You could freeze the window, switch to a hidden layout that contains the portal, and switch back to the current layout before showing the menu, though, and that would solve that small problem. This example could also be adapted to create a menu from all the records in the current database using the Go To Record/Request/Page script step.

This final example shows how to use the Scrp-ShowMenu function to display your popup menu in a specific place. If you are wanting to display the menu in a specific place, you are most likely wanting to line it up with some object on your layout, like a button. This is very easy to do. Switch to Layout mode and then go to the Layout Setup dialog. (You can bring up that dialog by selecting "Layout Setup..." from the "Mode" menu in FMP 4 or the "Layouts" menu in FMP 5.) Next, check the "Fixed page margins" checkbox and change the Top and Left margins to 0 (See Figure 9). Finally, click the OK button.



Layout Name: Main

☑ Include in layouts menu

☐ Print in 2 columns

● Across first ○ Down first

☑ Fixed page margins:

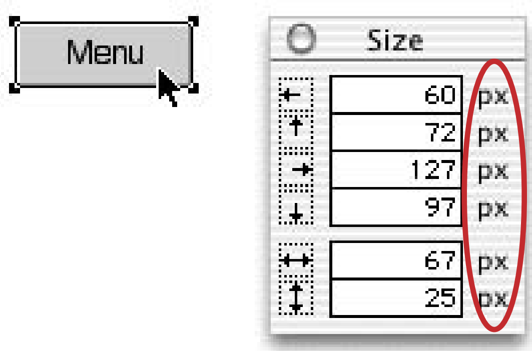
Top 0 px Left 0 px

Figure 9. Layout Setup dialog

Now, select the object on the layout that you want to line the menu up with and look at the Size pallete (See Figure 10). (In FMP 4, choose "Size" from the "Show" menu. In FMP 5, choose "Object Size" from the "Show" menu.) Click on the measurement indicator until it reads "px" (See Figure 10, circled in Red). If you want to line the menu up with the left and bottom edges of the button, then just write down those two numbers. Finally, modify your Scrp-ShowMenu function to look like this:

```
External("Scrp-ShowMenu", "name=MyMenu, x=60, y=97")
```

When you click on the button to show the menu, it will pop up right below the button (See Figure 11).



60 px
72 px
127 px
97 px

67 px
25 px

Figure 10. The Size palette

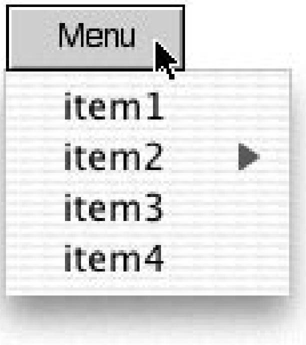


Figure 11. The Result

SCRIPTit External Function Reference

Available Functions

The following is a list of all available functions and a complete description of each. For general information on how the `External()` functions work and how you can use them in your calculations, please see the How to use Plug-ins PDF Document distributed with this Document.

Scrp-Version

This function returns the current version of SCRIPTit when the parameter string is empty (""). You can also use this function to test whether the plug-in is installed. If you call this function and nothing is returned, then the plug-in is either not installed, or not enabled.

Example:

```
External("Scrp-Version", "")
```

Scrp-Register

This function allows you to register your copy of SCRIPTit. This function takes your Registration First Name, Registration Last Name, and Registration Number all separated by the pipe character ("|"). ("|" is created by typing shift-backslash.)

Example:

```
External("Scrp-Register", "First Name|Last name|Registration Number")
```

Scrp-EnableScripts

This function is used to enable the actual calling of scripts. If scripts are enabled, then when it comes time for SCRIPTit to call a script in your database, it will call it. If scripts are disabled, then SCRIPTit will not call the script, but will remove the script event from the internal script queue. If your database has calculated fields that use SCRIPTit, and you need to modify those calculated fields, it is a good idea to call Scrp-DisableScripts before modifying the calculated field. Then when you save your modifications and FileMaker Pro recalculates all of the calculated fields, SCRIPTit will not call a script for each record. The parameter should be the empty string ("").

Example:

```
External("Scrp-EnableScripts", "")
```

See Also:

Scrp-DisableScripts

Scrp-DisableScripts

This function is used to disable the actual calling of scripts. If scripts are disabled, then SCRIPTit will not call the script on its scheduled time, but will remove the script event from the internal script queue. If your database has calculated fields that use SCRIPTit, and you need to modify those calculated fields, it is a good idea to call `Scrp-DisableScripts` before modifying the calculated field. Then when you save your modifications and FileMaker Pro recalculates all of the calculated fields, SCRIPTit will not call a script for each record. The parameter should be the empty string ("").

Example:

```
External("Scrp-DisableScripts", "")
```

See Also:

Scrp-EnableScripts

Scrp-PopupMenuErrors

This function enables or disables Popup dialogs that display menu errors. If a menu item does not have a database or script associated with it, and none of its parent items do, then when the user chooses that menu item, nothing will happen. If `PopupMenuErrors` has been turned on, then when the user chooses that menu item, an error dialog will pop up indicating the error. This function is useful when troubleshooting your menus and determining why a script is not being called for any specific menu item. To turn on `PopupMenuErrors`, use the parameter `"Yes"` or `"On"`. To turn off `PopupMenuErrors`, use the parameter `"No"` or `"Off"`.

Examples:

```
External("Scrp-PopupMenuErrors", "Yes")
External("Scrp-PopupMenuErrors", "Off")
```

Scrp-DefineXML

This is the main script and menu definition function. This function takes your XML script and menu definitions, parses them, and sets up scripts to be called and menus to be displayed. For a complete understanding of this function, please see the SCRIPTit XML Definitions section earlier in this documentation. This function does not require that you use the root <SCRIPTit> and </SCRIPTit> tags unless you are using an internal DTD for entity definitions. If you do not use one of the root tags, you cannot use the other. In other words, both the opening <SCRIPTit> and closing </SCRIPTit> tags must be there, or both of the tags must not be there. You cannot have just the opening or just the closing tag.

Examples:

```
External("Scrp-DefineXML", "<script db='" & Status(CurrentFileName) & "'
        script='validate' />")
External("Scrp-DefineXML", XML_Menus)
```

See Also:

Scrp-DefineXMLFile

Scrp-DefineXMLFile

This function is similar to the Scrp-DefineXML function above, except that instead of defining the actual XML text, you sepecify a file on the hard drive containing the XML text. The XML file you specify must be a "well-formed" XML document. Unlike the Scrp-DefineXML function above, the XML file you specify must contain the root <SCRIPTit> and </SCRIPTit> tags.

Examples:

```
External("Scrp-DefineXMLFile", "c:\xmldocs\my_menu.xml")
External("Scrp-DefineXMLFile", "Macintosh HD:xmldocs:my_menu.xml")
External("Scrp-DefineXMLFile", External("Scrp-GetPathToDB", Status(CurrentFileName))
        & "my_menu.xml")
```

See Also:

Scrp-DefineXML
Scrp-GetPathToDB
Scrp-GetPathToFM

Scrp-CancelScript

Use this function to cancel a scheduled script event. You can cancel any script event that has been scheduled for a specific date and/or time, or any script event that is set to run on an interval. Specify the script by name as the parameter.

Example:

```
External("Scrp-CancelScript", "My Script Event")
```

See Also:

Scrp-ListScripts

Scrp-ListScripts

Use this function to view a list of all script events in the internal script queue. If you need to cancel a script, you can use this function to see the names of all the current script events. The parameter should be the empty string ("").

Example:

```
External("Scrp-ListScripts", "")
```

See Also:

Scrp-CancelScript

Scrp-LastScriptName

This function will return the user-defined name of the last script that was called. If you have more than one event calling the same script in your database, you can use this function to distinguish between which script event is calling the script. The parameter should be the empty string ("").

Example:

```
External("Scrp-LastScriptName", "")
```

See Also:

Scrp-LastScriptValue

Scrp-LastScriptValue

This function will return the user-defined value of the last script that was called. When defining a script to be called, you can specify any data you want, such as a record id. When SCRIPTit calls the script you defined, you can use this function in the script that is called to retrieve that data back out and act on it accordingly. The parameter should be the empty string ("").

Example:

```
External("Scrp-LastScriptValue", "")
```

See Also:

Scrp-LastScriptName

Scrp-ShowMenu

Use this function to show a previously defined menu. You can have the menu popup at the current mouse coordinates, or you can specify where the popup menu should appear. If you want the menu to popup at the current mouse coordinates, just specify the name of the menu (ie. "My Menu"). If you want the menu to popup up at specific coordinates on your layout, you specify the values with a comma separated, name=value list. For the name of the menu, specify "name=" followed by the menu name. For the X (vertical, column) pixel coordinate, specify "x=" followed by the X pixel coordinate. For the Y (horizontal, row) pixel coordnate, specify "y=" followed by the Y pixel coordinate. An example: "name=My Menu, x=100, y=200". In all likelihood, if you want to specify coordinates, you will be wanting to line the menu up with some object on your layout. To know the coordinates for the menu to popup at, switch to Layout mode, change the Top and Left margins in the Layout Setup dialog to 0, select the object on the layout, and use the Size palette to get the pixel coordinates. The coordinates must be in pixels, so when looking at the Size palette, click on the measurement indicator to toggle between inches (in), centimeters (cm), and pixels (px). (See the Full Menu Examples section earlier in this document for a better explanation of this including screen shots.)

Examples:

```
External("Scrp-ShowMenu", "My Menu")
External("Scrp-ShowMenu", "name=My Menu, x=100, y=200")
```

Scrp-DeleteMenu

This function will delete a previously defined menu. Specify the name of the menu to delete as the parameter.

Example:

```
External("Scrp-DeleteMenu", "My Menu")
```

Scrp-LastMenuName

This function will return the name of the menu that contains the menu item that the user last selected. If you have more than one menu set up to call the same script in your database, you can use this function to determine which menu was actually used. The parameter should be the empty string ("").

Example:

```
External("Scrp-LastMenuName", "")
```

See Also:

Scrp-LastMenuChoice
Scrp-LastMenuValue

Scrp-LastMenuChoice

This function will return the name or caption of the last menu item the user selected. If you have more than one menu item calling the same script in your database, you can use this function to determine which menu item was actually selected. The parameter should be the empty string ("").

Example:

```
External("Scrp-LastMenuChoice", "")
```

See Also:

Scrp-LastMenuChoice
Scrp-LastMenuValue

Scrp-LastMenuValue

This function will return the user-defined value of the last selected menu item. When defining your menu items, you can specify any data you want, and then that menu item is selected, you can use this function in the script that is called to retrieve that data. The parameter should be the empty string ("").

Example:

```
External("Scrp-LastMenuValue", "")
```

See Also:

Scrp-LastMenuName
Scrp-LastMenuChoice

Scrp-GetPathToDB

You can use this function to return the complete path to the folder containing an open database that resides on the local hard drive. Specify the name of the database as the parameter. (You can use the `Status(CurrentFileName)` function to get the name of the database.)

Examples:

```
External("Scrp-GetPathToDB", "My_Database.fp3")
External("Scrp-GetPathToDB", Status(CurrentFileName))
```

See Also:

Scrp-GetPathToFM

Scrp-GetPathToFM

You can use this function to return the complete path to the folder containing the FileMaker Pro application. The parameter should be the empty string ("").

Example:

```
External("Scrp-GetPathToFM", "")
```

See Also:

Scrp-GetPathToDB

Understanding Error Responses

Every function in SCRIPTit returns a response indicating the success or failure of that function. If the function is successful, it will return a response indicating that it set the value you were trying to set, or completed the task that needed to be completed. If however, the function is not successful, it will return an Error Response. This Error Response is in the form of:

```
ERROR: <Function Name>: <Error Description>
```

For example, if you forgot to specify the FileName of the XML file you are trying to load with the Scrp-DefineXMLFile function, then that function will return the following Error Response:

```
ERROR: DefineXMLFile: You must specify a FileName.
```

Error responses always start with the word "ERROR" in all caps, followed by a colon, followed by the function that had the error, followed by a colon, followed by the actual error that occurred. You can use the various FileMaker Pro Text Functions to extract the different parts of the Error Response for your own use. For instance, if you want to know if the response you just got back was an ERROR, you can use the `LeftWords` function to return the first word and see if it is an error.

Example:

```
Set Field [SCRIPTit Result, External("Scrp-DefineXML", XML_Menu_Defs)]
If [LeftWords(SCRIPTit Result, 1) = "ERROR"]
        <inform the user there was an error in the XML and exit>
End If
```

Credits

Programming, documentation, and example databases by Jake Traynham (jake@comm-unity.net)
Concept, web design, and example databases by Jesse Traynham (jesse@comm-unity.net)

Contact Information

Email: SCRIPTit@comm-unity.net
SCRIPTit Website: http://scriptit.cnsplug-ins.com/
Main Website: http://www.cnsplug-ins.com/
Phone: 817-560-4226
Fax: 817-244-0340

You can write us at:

Comm-Unity Networking Systems
8652 Camp Bowie West
Fort Worth, Texas 76116