Toolbar

FileMaker Plug-In Manual

Dacons

This manual assumes that you have elementary knowledge of FileMaker.

Please send feedback to info@dacons.net

Website: http://www.dacons.net

May 03, 2011

CONTENTS

CONTENTS iii

FUNCTION INDEX v

TOOLBAR FEATURES 6
- Function overview 6
- Possible solutions 6

GETTING STARTED 7
- About this manual 7
- Software requirements 7
- Installing the FileMaker plug-in 8
- Hands-on examples 8
- Multi-user setups 8

TOOLBAR BASICS 9
- How to use Toolbar functions 9
- Function parameters 11
- Required and optional parameters 11
- Result codes 12

CREATE AND DESTROY TOOLBARS 14
- Create Toolbar 14
- Destroy Toolbar 16
- Set Toolbar State 18
- Retrieve Toolbar State 19
- Docking Toolbar 19

TABS 21
- Add Tabs 21
- Remove Tabs 22

BUTTONS 23
- Add Buttons 23
- Remove Buttons 24
- Set Button State 25
- Retrieve Button State 26

ADDITIONAL FUNCTIONS 28
- Retrieve last result code 28

Retrieve plug-in version 29
Register the plug-in 29

FUNCTION INDEX

TBar_CreateToolbar 14
TBar_RemoveToolbar 16
TBar_SetToolbarState 18
TBar_GetToolbarState 19
TBar_Attach 19
TBar_InsertTab 21
TBar_RemoveTab 22
TBar_InsertButton 23
TBar_RemoveButton 24
TBar_SetButtonState 25
TBar_GetButtonState 26
TBar_GetLastResultCode 28
TBar_GetVersion 29
TBar_RegisterSession 29

FUNCTION OVERVIEW
Toolbar Features

Toolbar lets you create your own toolbars that are rendered in a system-native style on Mac OS X and Windows. This is an indispensable plug-in that lets you go beyond the user interface capabilities of FileMaker. Toolbar gives FileMaker solutions a convenient user interface that is today's standard. Each toolbar button triggers a user-defined script. Use custom icons for your toolbar buttons. Toolbars can be docked or floating.

Use Toolbar to enrich your solution with custom toolbars, not seen in FileMaker before.

Function overview

- Create unlimited amount of custom toolbar, that will remain when you change the active layout
- The toolbars are floating and dockable, and can be moved to any location
- Create tabbed and plain toolbars
- Manage any toolbar content and state from a script
- Use custom icons for toolbar buttons
- Give your FileMaker application a user interface context that goes beyond the current layout

Possible solutions

- Create a professional look for your database
- No need to create additional layouts to have multiple buttons in your solution

INTRODUCTION
Getting Started

This chapter describes how to use this manual and provides all information you need to install the plug-in. In addition, you will see how to explore the powerful features of Toolbar using the files that come with this plug-in.

About this manual

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

Chapter 1 (Chapter 1, p. 9) explains how to use Toolbar functions with FileMaker. You will learn about plug-in functions, parameters, and result codes.

Chapter 2 (Chapter 2, p.14) provides all information you need to create, destroy, and alter the current toolbar state.

Chapter 3 (Chapter 3, p.21) provides all information you need to create, remove, and alter the current tabs state.

Chapter 4 (Chapter 4, p. 23) provides all information you need to create, remove, and alter the current buttons state.

Finally, **Chapter 5** (Chapter 5, p. 28) provides information about advanced plug-in functions that let you retrieve result codes, configure the log engine and more.

Software requirements

Toolbar requires FileMaker 7 or later. The FileMaker editions Pro, Advanced/Developer and Runtime are supported. The plug-in is shipped in two different file formats, one for Windows and one for Mac OS X.

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

Installing the FileMaker plug-in

Before installing the Toolbar plug-in, ensure you select the correct version from the download package according to your operating system (Windows or Mac OS X).

Next, ensure that FileMaker is closed. Then copy the Toolbar plug-in for your operating system in the *Extensions* folder which is located in the FileMaker application folder. Now, please launch FileMaker.

Hands-on examples

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

Continue reading this manual to find out how the features shown in the Quick Start file have been implemented in FileMaker using Toolbar plug-in functions.

Multi-user setups

To use Toolbar functions in multi-user network solutions, the plug-in needs to be installed on every computer that uses its functions. Due to the software architecture of FileMaker Server, plug-ins cannot operate on the server side.

CHAPTER 1
Toolbar Basics

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

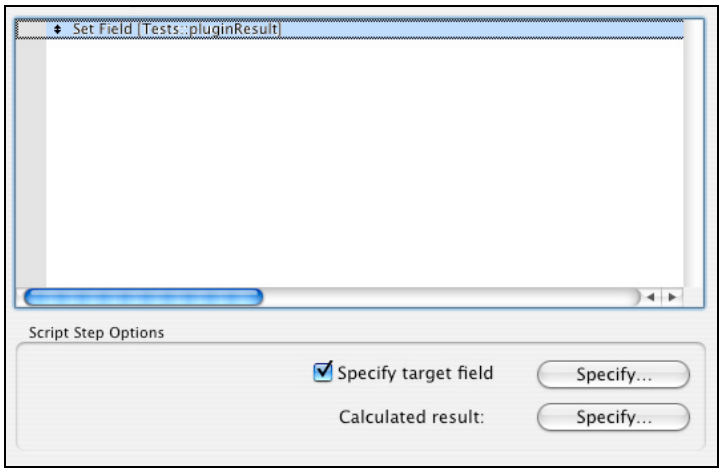
How to use Toolbar functions

Since Toolbar is a FileMaker plug-in, so-called *external functions* are used to trigger the plug-in from a FileMaker database solution. They are called external functions because these functions are provided by a plug-in and thus are not part of the actual FileMaker application. In order to use the external functions provided by a plug-in it must be installed and enabled in the FileMaker application preferences.

To access external functions provided by a plug-in the FileMaker calculation editor is used. The calculation editor is provided by the FileMaker editions Pro and Advanced (Developer in version 7). FileMaker shows the calculation editor dialog window whenever a calculation has to be defined (e.g. for calculated fields, validation calculations etc). In most cases you will invoke Toolbar functions from a script. The easiest bridge between scripts and the calculation editor that invokes plug-in functions is a script step called *Set Field*. Add a *Set Field* script step to your script to start using Toolbar.

Next, specify the target field which will contain the result of the plug-in operation. There are two types of Toolbar results. Some Toolbar functions return content (such as the current status of a button) which can be stored directly in a FileMaker field. Other functions do not return content. Instead, result codes are provided by these functions that tell you if an operation succeeded or failed for a certain reason. Click the *Specify* button provided by the *Set Field* script step in ScriptMaker and define the target field of your plug-in command depending on the type of result. Refer to the functions reference (starting on p.14) for further details about the results returned by each plug-in function.

In the following figure a global field called *pluginResult* (defined in the table *Tests*) has been specified as result field. In most cases, you will choose a global field as result field if the plug-in function invoked does not return content but only a result code. If content is returned by a function choose an appropriate non-global field instead to store results in a database record.

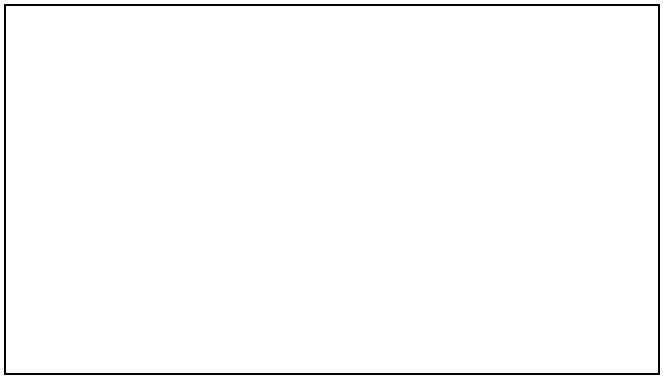


Set Field command in ScriptMaker

If you are working with FileMaker 8 or later you will prefer using script variables over global fields to store non-content plug-in results. Thus, use the script step *Set Variable* instead of *Set Field* for plug-in functions that do not return content.

Click the *Specify* button (the second *Specify* button when using *Set Field*) to open the calculation editor.

In the top right corner of the calculation editor dialog you find a drop-down menu called *View*. Switch to the section *External functions*. All Toolbar functions are listed in the section *Toolbar*. If the Toolbar functions do not appear in the list, the plug-in is not installed correctly or it is disabled in the FileMaker application preferences. In this case leave the calculation editor and check that Toolbar is installed as described earlier and that it is enabled in the FileMaker application preferences.



Toolbar functions in the Calculation Editor

Double-click on the Toolbar function in the list which you would like to use. The function is copied to your calculation including a *Quick Reference* text that describes the meaning of the selected function and its parameters. To use the function, parameter values have to be specified.

Function parameters

Most Toolbar functions provide parameters that specify details of a function call. Many of these functions have several parameters. After copying a function to the text box of the calculation editor, FileMaker shows the name of the plug-in function that has been selected and a text label for each parameter of the function.

The following example shows the Toolbar function `TBar_Attach` that has been added to the calculation editor. This function attaches the toolbar to the top of the window.

```
TBar_Attach (tbarName {; action } )
```

After copying the function to your calculation, you need to define a value for each parameter. This can be *hard-coded* values (i.e. a specific toolbar object name for the `tbarName` parameter) enclosed in quotation marks. As in every calculation you may also specify *dynamic* values represented by database field names or script variables that hold the information to be passed to the plug-in as parameter value.

Required and optional parameters

Most functions have some parameters that are required and some that are optional. When invoking a Toolbar function you have to pass a value for all required parameters. Optional parameters can be skipped. If no value is passed to the plug-in for an optional parameter the default value for this parameter will be used.

Required parameters are listed in the beginning and optional parameters are listed at the end of a function call. This allows you to omit optional parameters that you would not like to use in your function call.

After copying a function to the calculation editor you will see optional parameters enclosed in curly braces `{}`. To skip an optional parameter use an empty text value represented by two quotation marks in the text editor `""`. Optional parameters at the end of a function call can be skipped completely. After specifying parameter values you need to remove any curly braces from the function call.

In the following example the function `TBar_SetButtonState` is invoked. A Toolbar name must be specified because the parameter `tbarName` is required. However, the size of the icon to be returned is *not* specified. Since the parameter `size` is optional and there is no further parameter value required for this function call, it can be omitted completely:

```
TBar_SetButtonState ( "MyToolbar"; ""; "1"; ""; "toggle" )
```

The plug-in will choose the default value for the parameter `availability` (remain unchanged).

Result codes

As mentioned earlier, Toolbar functions can be separated into two groups. Content functions return values that are stored in database fields (using *Set Field* script step). If a content function fails, an empty result is returned. To find out *why* a content function failed invoke the function `TBar_GetLastResultCode`. It returns a result code and a short description of the error. The second group of Toolbar functions is called non-content functions. Non-content functions return a result code directly. However, you may also invoke `TBar_GetLastResultCode` to check the result code of non-content functions.

You can evaluate result codes in your script and perform certain activities (e.g. show an error message to the user). To check in ScriptMaker if the last Toolbar operation failed, use the following calculation in an if condition:

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

The following conditions checks for a specific error:

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

A list of all result codes and their meanings follows on the next page.

Result Code	Description
000	(OK) Function completed successfully, no errors occurred
-006	(Log file error) An error occurred setting up / writing the log file
-007	(Invalid registration information) Enter registration exactly as supplied
-008	(Incorrect option) One of the parameters passed in external function has incorrect value
-009	(Name already used) The name is already used by other Toolbar
-010	(Toolbar not found) Toolbar with this name was not found
-013	(Index out of bounds) Tab or button with this index does not exist
-014	(Only for tabbed) This function is accessible for only tabbed toolbars
-015	(Bad icon) Bad icon format
-071	(Internal error) Contact the Dacons Support at http://www.dacons.net/support
-099	(Unknown error) Unknown system error

CHAPTER 2
Create and Destroy Toolbars

Toolbar plug-in enables you to create an unlimited amount of custom toolbars. After toolbar is created, it can be hidden, shown, and destroyed. This chapter explains how to create, destroy, show and hide toolbars, accordingly.

A basic toolbar construction procedure consists of several steps. First of all the `"TBar_CreateToolbar"` function should be used, to define the general parameters for the new toolbar. On this stage the toolbar does not include any buttons, and therefore can't be displayed. The next step is to add the necessary tabs, in case if the toolbar has a `"tabbed"` style, using the `"TBar_InsertTab"` function. When the tabs are ready (or in case if the toolbar does not have any tabs, and therefore uses an `"untabbed"` style) it is time to and insert the necessary buttons, using the `"TBar_InsertButton"` function. As soon as the toolbar has at least one button, the plug-in will render the toolbar, in case if the new toolbar has a `"visible"` initial state set by appropriate `"visibility"` option of the `"TBar_CreateToolbar"` function. Otherwise, the plug-in will not display the toolbar, until the visibility option is changed to `"visible"` using the `"TBar_SetToolbarState"` function.

Create Toolbar

The function **`TBar_CreateToolbar`** creates a toolbar, based on the parameters specified.

In case if a "`tabbed`" toolbar is created, and `"showTabs"` parameter is set to `"show"`, the tabs will be shown as soon as the toolbar is created; in case if `"showTabs"` parameter is set to `"hide"`, the toolbar still behaves as a "tabbed" toolbar (this regards to all supplemental function calls, such as `"TBar_InsertButton"` and `"TBar_InsertTab"`), but the actual tabs area is not shown, so the toolbar looks like `"untabbed"`. This allows having a toolbar, that contains certain number of toolbars "inside", easy switchable using `"TBar_SetToolbarState"` function.

In case if "untabbed" toolbar is created, the toolbar will have no tab links, and any tab-related functions will fail (such as "TBar_InsertTab", "TBar_SetToolbarState", etc).

The plug-in does not limit the maximum size of the icons that can be applied for toolbar tabs or buttons, however recommended sizes are 32x32, 48x48, 64x64, 96x96, and 128x128.

In case if supplied picture size will be different from the initial icon size that was set when creating a toolbar, the plug-in will automatically adjust the supplied picture to the necessary dimensions.

Syntax:

```
TBar_CreateToolbar ( tbarName; buttonLabelPosition; tabbed {;
tabLabelPosition; buttonIconWidth; buttonIconHeight; tabIconWidth;
tabIconHeight; visibility; showTabs; showCloseButton;
closeButtonAction; dockable; buttonsAlign; filename; cacheScripts}
)
```

Parameters:

`tbarName` – The name of the Toolbar object to be created. This name is used with the other function calls, in order to specify the object name to be addressed.

`buttonLabelPosition` – Position of the text labels (if any), relatively to the button icons. Parameter values can be "top", "bottom", "left", "right", "textOnly", and "iconsOnly".

`tabbed` – Specify, if the toolbar should contain tabs. Parameter values can be "tabbed" and "untabbed".

`tabLabelPosition` – Position of the text labels (if any), relatively to the tab icons. Parameter values can be "top", "bottom", "left", "right" (default), "textOnly", and "iconsOnly".

`buttonIconWidth` – The width of the toolbar button icons in pixels. Default value is "32".

`buttonIconHeight` – The height of the toolbar button icons in pixels. Default value is "32".

`tabIconWidth` – The width of the toolbar tab icons in pixels. Default value is "32".

`tabIconHeight` – The height of the toolbar tab icons in pixels. Default value is "32".

`visibility` – Initial state of the toolbar. Parameter values can be "visible" (default) and "invisible".

`showTabs` – Show tabs (if any). This parameter will be ignored for `"untabbed"` style toolbar. Parameter values can be `"show"` (default) and `"hide"`.

`showCloseButton` – Show "Close" button. Use this option to specify if the toolbar should have a "Close" button. Parameter values can be "`showClose`" (default) and "`hideClose`".

`closeButtonAction` – Use this option to specify if the "Close" button should close the toolbar or just hide it. Parameter values can be "`hideToolbar`" (default) and "`closeToolbar`".

`dockable` – Use this option to specify if the toolbar should automatically dock, when moved to the common toolbars area. Parameter values can be "`dockable`" (default) and "`floating`".

`buttonsAlign` – Use this option to specify the buttons align in a multi-line toolbars. Parameter values can be "`left`" (default), "`right`", "`center`", and "`justify`".

`fileName` – Specify the name of a file using this parameter, if you would like to attach the toolbar to a specific solution. Leave this parameter empty to display the toolbar with any file.

`cacheScripts` – Use this option, if you would like the plug-in to cache scripts for your toolbar. This option may speed up the scripts execution, but if the user will change the order or amount of scripts within the database, the cache has to be updated using the `TBar_UpdateCache` function. Values can be "`cache`" and "`noCache`" (default)

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

Example:
In the following example a simple text-only (no icons) toolbar with no tabs is created:

```
TBar_CreateToolbar ( "My Toolbar"; "right"; "untabbed" )
```

For this example, the following result should be returned:

```
000 (OK)
```

Destroy Toolbar

The function **`TBar_RemoveToolbar`** destroys a toolbar, and any associated data.

Syntax:

```
TBar_RemoveToolbar ( tbarName )
```

Parameters:
`tbarName` – The name of the Toolbar object to be deleted.

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

Example:
The following example destroys a toolbar named "My Toolbar":

```
TBar_RemoveToolbar ( "My Toolbar" )
```

For this example, the following result should be returned:

```
000 (OK)
```

Update Toolbar Cache

The function **`TBar_UpdateCache`** updates the cache of the database scripts. Use this function only if you have used the "`cacheScripts`" parameter for the "`TBar_CreateToolbar`" function call.

Syntax:
`TBar_UpdateCache ( tbarName )`

Parameters:
`tbarName` – The name of the Toolbar object to be updated.

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

Example:
The following example updates a toolbar cache for the toolbar object named "My Toolbar":

```
TBar_UpdateCache ( "My Toolbar" )
```

For this example, the following result should be returned:

```
000 (OK)
```

Set Toolbar State

The function **`TBar_SetToolbarState`** updates current state of the toolbar, such as visibility, active tab, and labels positioning.

In case if Toolbar was created using the `"showTabs"` parameter set to `"hide"`, use this function to switch the active tab.

Syntax:
```
TBar_SetToolbarState ( tbarName {; visibility; showTabs; activeTab; buttonLabelPosition; tabLabelPosition } )
```

Parameters:
`tbarName` – The name of the Toolbar object to be altered.

`visibility` – The new state of the Toolbar object. Parameter values can be `"visible"`, `"invisible"`, and `"toggle"` (change the current state to the opposite). If this parameter is empty, the sate of the toolbar will remain unchanged.

`showTabs` – Show tabs (if any). This parameter will be ignored for "`untabbed`" style toolbar. Parameter values can be `"show"` and `"hide"`. If this parameter is empty, visibility sate of the toolbar tabs will remain unchanged.

`activeTab` – Set active tab. In case if this parameter is empty, active tab is not changed.

`buttonLabelPosition` Position of the text labels (if any), relatively to the button icons. Parameter values can be `"top"`, `"bottom"`, `"left"`, and `"right"`.

`tabLabelPosition` – Position of the text labels (if any), relatively to the tab icons. Parameter values can be `"top"`, `"bottom"`, `"left"`, and `"right"`.

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

Example:
In the following example we shell hide the toolbar, named "My Toolbar":

```
TBar_SetToolbarState ( "My Toolbar"; "invisible" )
```

For this example, the following result should be returned:

```
000 (OK)
```

Retrieve Toolbar State

The function **`TBar_GetToolbarState`** retrieves the current state of the toolbar, such as visibility, active tab, and labels positioning.

Syntax:
`TBar_GetToolbarState ( tbarName; attributeType )`

Parameters:
`tbarName` – The name of the Toolbar object to be queried.

`attributeType` – The state type of the Toolbar object to be retrieved. Parameter values can be `"visibility"`, `"tabsStyle"`, `"showTabs"`, `"activeTab"`, `"buttonLabelPosition"`, `"tabLabelPosition"`, `"docking"`, `"width"`, `"height"`, `"horizontalPosition"`, and `"verticalPosition"`.

Result:
Returns a corresponding Toolbar parameter value. In case if function output is empty, please use the "`TBar_GetLastResultCode`" function, to obtain the last result code.

Example:
In the following example we shell retrieve an active tab index, from the toolbar, named "My Toolbar":

```
TBar_GetToolbarState ( "My Toolbar"; "activeTab" )
```

Below please find an example of function call result:

```
1
```

Docking Toolbar

The function **`TBar_Attach`** attaches and detaches the Toolbar to the default toolbars location (to the top of the active window on Windows, and top of the screen on Mac OS X).

Syntax:
`TBar_Attach ( tbarName {; action } )`

Parameters:
`tbarName` – The name of the Toolbar object to be affected.

`action` – The new Toolbar state to be applied. Parameter values can be `"attach"`, `"detach"`, and `"toggle"`, (default), - these options will operate relative to the top window (or screen) position. Alternatively the toolbar can be attached to the other non-standard positions, using the following parameter values: `"attachBottom"`, `"attachLeft"`, `"attachRight"`, `"toggleBottom"`, `"toggleLeft"`, and `"toggleRight"`.

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

Example:
In the following example we shell dock the toolbar, named "My Toolbar":

```
TBar_Attach ( "My Toolbar"; "attach" )
```

Below please find an example of function call result:

```
000 (OK)
```

CHAPTER 3
Tabs

Toolbar provides functions that enable you to add and remove the toolbar tabs dynamically, depending of your current toolbar object settings.

Add Tabs

The function **`TBar_InsertTab`** adds a new tab to the toolbar, in case if toolbar was created with a "`tabbed`" style. In case if the toolbar has an "`untabbed`" style, the function will return an error. In case if a "`tabbed`" toolbar is set to hide the tabs, new tab will not be shown or become active. In order to activate this tab, the user should use the "`TBar_SetToolbarState`" function.

Syntax:

```
TBar_InsertTab ( tbarName {; label; bitmap; tooltip; posIndex;
script; scriptParameter; fileReference } )
```

Parameters:

`tbarName` – The name of the Toolbar object to be modified.

`label` – The text label of the new tab. This parameter is optional, and the value does not have to be unique.

`bitmap` – Bitmap to be used as a tab icon. The plug-in supports any image formats, supported by FileMaker. Image transparency will be automatically recognized (PNG and GIF formats only). Please notice, that in case if supplied image size is different from the dimensions set by the "`TBar_CreateToolbar`" function, the plug-in will automatically adjust the image size accordingly.

`tooltip` – The text label of the new tab. This parameter is optional, and the value does not have to be unique.

`posIndex` – Position of the new tab. In case if the value of "`posIndex`" is greater then the amount of tabs available in the toolbar, or equals "`-1`", the new tab will be added to the end of the list, and will receive appropriate index.

`script` – The script to be triggered on tab press event. Use this parameter if you want the plug-in to trigger also a script, when the tab becomes active switched.

`scriptParameter` – Script parameter.

`fileReference` – File, that contains the script to be executed. In order to obtain the current file name the following FileMaker native function call is recommended: Get (FileName)

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

Example:
In the following example we shell add a new tab, named "The First Tab", to the toolbar named "My Toolbar":

```
TBar_InsertTab ( "My Toolbar"; "The First Tab" )
```

For this example, the following result should be returned:

```
000 (OK)
```

Remove Tabs

The function **`TBar_RemoveTab`** removed a tab from the toolbar. All of the buttons that belong to this tab are automatically removed. In case if the toolbar has an "`untabbed`" style, the function will return an error.

Syntax:
```
TBar_RemoveTab ( tbarName; posIndex )
```

Parameters:
`tbarName` – The name of the Toolbar object to be modified.

`posIndex` – Tab index to be removed.

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

Example:
In the following example we shell remove a tab with index "`2`", from the toolbar named "My Toolbar":

```
TBar_RemoveTab ( "My Toolbar"; "2" )
```

For this example, the following result should be returned:

```
000 (OK)
```

CHAPTER 4
Buttons

Toolbar provides functions that enable you to add and remove the toolbar buttons dynamically, depending of your current toolbar object needs.

Add Buttons

The function **`TBar_InsertButton`** adds a new button to the toolbar, according to the parameters specified. Also this function is used to insert a "`separator`" and "`line-break`" items, where "`separator`" will add a horizontal separation between items, and "`line-break`" will make the next items move to the next line.

Syntax:

```
TBar_InsertButton ( tbarName; type {; label; bitmap; tooltip;
script; scriptParameter; fileReference; availability; state;
tabIndex; posIndex; radioId } )
```

Parameters:

`tbarName` – The name of the Toolbar object, to add the button to.

`type` – The type of the button to be added. Parameter values can be "`button`", "`check-button`", "`separator`", and "`line-break`".

`label` – Button text label.

`bitmap` – Bitmap to be used as a button icon. The plug-in supports any image formats, supported by FileMaker. Image transparency will be automatically recognized (PNG and GIF formats only). Please notice, that in case if supplied image size is different from the dimensions set by the "`TBar_CreateToolbar`" function, the plug-in will automatically adjust the image size accordingly.

`tooltip` – The tooltip of the new button.

`script` – The script to be triggered on button press event.

`scriptParameter` – Script parameter.

`fileReference` – File, that contains the script to be executed. In order to obtain the current file name the following FileMaker native function call is recommended: Get (FileName)

`availability` – Button availability state. Parameter values can be "`enabled`" (default), and "`disabled`".

`state` – Button check state. Parameter values can be "`check-on`", and "`check-off`" (default).

`tabIndex` – Tab index for the new button. In case If the toolbar has an "`untabbed`" style, this parameter should be "`1`" or empty, otherwise the function will return an error. If the toolbar has a "`tabbed`" style, and this parameter is empty, the button will be added to the Tab with greatest index (to the last tab).

`posIndex` – Position of the new button. Index "`-1`" will add the button to the end.

`radioId` – Radio button group Id. This Id should be used with "`check-button`" buttons types. Index "`-1`" means "no group". In case if button state is "`check-off`", this parameter will be ignored.

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

Example:
In the following example we shell add a new button, named "My Button", to the toolbar named "My Toolbar". The button shell have a tooltip "My Tooltip", and shell start a script "My Script":

```
TBar_InsertButton ( "My Toolbar"; "button"; "My Button"; ""; "My
Tooltip"; "My Script"; Get (FileName) )
```

For this example, the following result should be returned:

```
000 (OK)
```

Remove Buttons

The function **`TBar_RemoveButton`** removes a button from the toolbar.

Syntax:

```
TBar_RemoveButton ( tbarName; tabIndex; posIndex )
```

Parameters:
`tbarName` – The name of the Toolbar object to be altered.

`tabIndex` – In case if the toolbar has an "`untabbed`" style, this parameter should be "`1`" or empty, otherwise the function will return an error. In case if the toolbar has a "`tabbed`" style, this parameter should contain a valid tab index, otherwise the function will return an error.

`posIndex` – Position of the button to be removed.

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

Example:
In the following example we shell remove a button, with index "`1`", from the toolbar named "My Toolbar":

```
TBar_RemoveButton ( "My Toolbar"; "1"; "1 )
```

For this example, the following result should be returned:

```
000 (OK)
```

Set Button State

The function **`TBar_SetButtonState`** updates current state of the button, such as availability, and state.

Syntax:

```
TBar_SetButtonState ( tbarName; tabIndex; posIndex {;
availability; state } )
```

Parameters:
`tbarName` – The name of the Toolbar object to be altered.

`tabIndex` – In case if the toolbar has an "`untabbed`" style, this parameter should be empty, otherwise the function will return an error.

In case if the toolbar has a "`tabbed`" style, this parameter should contain a valid tab index, otherwise the function will return an error.

`posIndex` – Position of the button to be affected.

availability – Button availability state. Parameter values can be "enabled", "disabled", and "toggle" (change the current state to the opposite). If this parameter is empty, the availability of the button will remain unchanged.

state – Button check state. Parameter values can be "check-on", "check-off", and "toggle" (change the current state to the opposite). If this parameter is empty, the sate of the button will remain unchanged.

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

Example:
In the following example we shell change the button availability to an opposite. We will alter the first button, on the first tab of the "My Toolbar" toolbar.

```
TBar_SetButtonState ( "My Toolbar"; "1"; "1"; "toggle" )
```

For this example, the following result should be returned:

```
000 (OK)
```

Retrieve Button State

The function **TBar_GetButtonState** returns a button availability or check state. In case if check state is requested for "check-button", function result value can be "check-on" or "check-off". For other button styles the function will always return "check-off" value. In case if the specified button does not exist, the function will return an empty result.

Syntax:

```
TBar_GetButtonState ( tbarName; tabIndex; posIndex;
attributeType )
```

Parameters:
tbarName – The name of the Toolbar object to be altered.

tabIndex – In case if the toolbar has an "untabbed" style, this parameter should be empty, otherwise the function will return an error.

In case if the toolbar has a "tabbed" style, this parameter should contain a valid tab index, otherwise the function will return an error.

posIndex – Position of the button to be affected.

`attributeType` – Button attribute type to be queried. Parameter values can be "`availability`" (default), and "`state`".

Result:
Returns a corresponding button attribute value. In case if function output is empty, please use the "`TBar_GetLastResultCode`" function, to obtain the last result code.

Example:
In the following example we shell obtain the first button availability, on the first tab of the "My Toolbar" toolbar.

```
TBar_GetButtonState ( "My Toolbar"; "1"; "1"; "availability" )
```

Below please find an example of function call result:

```
enabled
```

CHAPTER 5
Additional Functions

Toolbar provides additional plug-in functions that are described in this chapter. These functions enable you to retrieve result codes, check the exact version number of the plug-in installed and unlock the trial version using with a registration name and a registration code from script so that no end-user interaction is required to unlock a trial version.

Retrieve last result code

The function **`TBar_GetLastResultCode`** returns the result code of the last Toolbar function that has been invoked. Please refer to p. 13 for a table of all result codes.

Syntax:
`TBar_GetLastResultCode`

Parameter:
No parameter needed.

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

Result:
The result code for the last plug-in operation is returned together with a short description. If the result returned is empty no plug-in function been invoked prior to `TBar_GetLastResultCode`.

Example:
The function `TBar_GetButtonState` current button state (attribute value). It does not return error codes. If an error occurs, this function returns an empty result. In the following example, the specified button cannot be found.

```
TBar_GetButtonState ( "My Toolbar"; "1"; "8"; "availability" )
```

Since the button cannot be found an empty result is returned. To find out why the function `TBar_GetAttribute` failed the function `TBar_GetLastResultCode` is invoked.

```
TBar_GetLastResultCode
```

In this example case it returns:

```
-013 (Index out of bounds)
```

Retrieve plug-in version

The function **`TBar_GetVersion`** returns the version of the installed Toolbar plug-in. Use this function to check if the plug-in is installed when your database solution starts (start-up script). The version information provided by this function can also be used for the AutoUpdate plug-in which pushes updated versions of other plug-ins to all FileMaker network clients automatically. Review the FileMaker documentation for more information about the AutoUpdate plug-in.

Syntax:
`TBar_GetVersion ({ control })`

Parameters:
`control` – This parameter is optional. It can be used to customize the content returned by the plug-in function. Leave this parameter empty to retrieve all of the following items. To retrieve only a specific version information item, set this parameter to one of the following values: `"version"` returns the exact version of the plug-in installed. In most cases you will pass this value to the plug-in to retrieve the version number. `"product"` returns the name of the product. `"platform"` returns the operating system the plug-in is running on and `"copyright"` returns copyright information of the plug-in.

Result:
This function returns the plug-in version description including all items specified by the `control` parameter.

Register the plug-in

To remove all trial limitations from the plug-in it has to be registered using the registration data you receive from Dacons after purchasing a Toolbar license. The plug-in can be registered manually using the preferences dialog (Go to: FileMaker Application Preferences ► Plug-Ins ► Toolbar).

To avoid manual plug-in registration when shipping your database solution to a client or distribute a FileMaker Runtime application with Toolbar you can also register the plug-in from the start-up script of your solution by invoking the function **`TBar_RegisterSession`** with your registration data. This function has to be invoked ***prior to any other function***. Note

that registration data from script will not be stored so registration from script has to be invoked every time a solution starts.

Syntax:
`TBar_RegisterSession ( username ; userCode )`

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

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

Result:
This function returns a result code which tells you if the operation succeeded or if errors occurred. Please contact the Dacons Support at http://www.dacons.net/support if you registration code is rejected for any reason.