Simba Documentation

Release 1.0.1

Merlijn Wajer, Raymond van Venetie

September 29, 2014

CONTENTS

Note: Welcome to the Documentation for the Simba project. The documentation is actively being worked on and somewhat incomplete.

This documentation aims to provide information on just about anything there is to know about Simba.

Starting points are *What Is Simba?*, *Getting Started with Simba* and the *Scripting Reference*

Want to help out with the documentation? See *Documentation Documentation*.

Simba is a program used to repeat certain (complicated) tasks. Typically these tasks involve using the mouse and keyboard. Simba is programmable, which means you can design your own logic and steps that Simba will follow, based upon certain input or as a reaction to a certain action.

Simba is created by the SRL community, fully open source and actively maintained.

For community forums see http://villavu.com/

Contents:

CHAPTER

ONE

WHAT IS SIMBA?

Simba is a program used to repeat certain (complicated) tasks. Typically these tasks involve using the mouse and keyboard. Simba is programmable, which means you can design your own logic and steps that Simba will follow, based upon certain input.

This "input" can vary greatly; from data in files to colors and text on the screen.

Simba can:

- Find and read colours on the screen.
- Click or move the mouse to a specific position on the screen. Typically this is the position of a found color or bitmap.
- Read text on the screen and turn it into actual text. (Optical Character Recognition)
- Capture and analyse images on the screen.
- Read and write files.
- Connect to the internet to read websites and post data to them.
- Run pascal programs for you. If you're a bit creative you can have a lot of fun stuff with Simba, you could even make a game in it!

and more. Start now by *Installing simba*.

Currently Simba is still in its beta stages; this includes the documentation. You will probably notice some parts of the documentation aren't finished or are just plain missing.

Simba is being updated almost every day. To see changes as they are added, view https://github.com/MerlijnWajer/Simba.

If you want to know more about Simba, see *Why Simba?*

CHAPTER

TWO

WHY SIMBA?

So why would you use Simba?

For several reasons. Number one is probably the active community combined with the open-mindedness of the Simba developers.

2.1 Free

Simba is, and always will be free. It is free as in Free Beer **and** Freedom.

Simba is 100% free to use. In fact, we welcome you to share it with the rest of the world.

The source code to the program is freely available, under the GPL (v3) license.

Note: The source code allows everyone to see how the program actually works. Most developers choose not to share their source code and therefore decide to withhold knowledge. The Simba developers highly disagree with this mentality and will fight for a world with free knowledge for everyone. "Closed Source" as not sharing source code is called, is holding back innovation and research in the world.

2.2 Extensible

Simba supports both plug-ins in binary format as well as extensions written in the Simba programming language (which is Pascal-based).

This way one can easily extend Simba. See *Writing Simba Extensions*

2.3 Reliable

Simba is actively maintained by a team of knowledgeable programmers.

We as a team do our best to make Simba into a quality product. However this does not guarantee a bug free program. (Bug free programs don't exist)

Should you encounter a bug, please consider sending a bug report so we can try to resolve the issue: *Reporting Bugs*.

2.4 Cross Platform

Simba has been written with freedom in mind and believes Simba should not be limited to one platform, as that limits its users to one platform as well.

Currently the supported platforms are Windows and Linux with attempts being made to port it to OSX (it may run on OSX if you install X11).

Even though we try hard to make Simba bug-free on all platforms, some issues remain. The biggest issue is the interpreter that executes your program. It is an external component and does not play too well on Linux. Hopefully this will be improved upon soon.

There are plans on supporting interpreter for alternative languages such as Python, these would largely resolve the current problems with the PascalScript interpreter on Linux/Mac.

2.5 Fast

The Simba developers have designed and written Simba with speed in mind. It is important to understand, however, that premature optimization usually ends up hurting development by introducing bugs and other forms of instability. Simba attempts to find a middle-ground and has sacrificed some speed for readability and maintainability, but is overall still quite fast.

2.6 Open Minded

Open minded. We appreciate your help, ideas and criticism!

2.7 Debug Friendly

Simba aims to be debug friendly. It will show you where your errors are during compilation, and when you are running scripts Simba will point out where you are passing invalid arguments. The action Simba takes when this happens is configurable. (It can issue a warning and continue, or simply terminate the script)

2.8 Well Documented

Simba is well documented. (You're looking the documentation right now...) There is still a lot left to document though.

CHAPTER

THREE

GETTING STARTED WITH SIMBA

This page will help you install Simba and highlight some basic Simba features.

3.1 Installing simba

Installing Simba is pretty straightforward and will only take a couple of minutes.

3.1.1 Simba installer

Everyone can get Simba at http://wizzup.org/simba/

Go to *Download* and download the Simba installer.

Once it is done downloading, simply run the installer, follow the steps and please make notice of the following note:

Note: Simba will install to C:/ by default; if you want to install it somewhere else, make sure you select a different location! If you are running Windows Vista or Windows 7, then you will probably want to install it to C:/ or another place; as long as writing in the actual directory does not require administrator permissions. C:/ is a safe bet.

You can install Simba to your `Program Files` directory, but you will have to run Simba as administrator in that case.

Simba will probably tell you there is an update available. In this case, an update button will appear. Click it and the Simba Updater will show up. Alternatively you can update Simba using Tools -> Update which will update Simba if there is a new update available.

Note: Updating Simba as soon as an update is released is recommended.

3.1.2 Portable Simba Installation

To get a portable installation at this time is not supported but will follow sooner or later.

3.2 Setting up SRL 4 with Simba

SRL has been deprecated starting January 2012.

3.3 Setting up SRL 5 with Simba

If you were using the *Simba installer* then you can simply enable the `SRL Updater` extension. (Go to View -> Extensions and enable `srl.sex`)

See *Simba Extensions* for more information on (other) Simba extensions.

This is the only supported way. You can simply do a git clone on the srl repository, but if you can do that, then you should be able to set up SRL yourself as well.

3.4 Installing scripts

Scripts can be downloaded with the Script Manager, in Tools -> Script Manager.

Note: The Script Manager is not finished yet.

3.5 Troubleshooting

If you run into problems that are not mentioned here, make sure you look at the *Troubleshooting* page.

CHAPTER

FOUR

TROUBLESHOOTING

This is a general troubleshooting page and will hopefully cover all potential problems you may encounter. Before you read any further, make sure your Simba is up to date. See *Updater in Simba*.

4.1 Where is the DTM Editor?

The DTM Editor a Simba Extension. See *Simba Extensions* for more information on extensions and how to enable the DTM editor.

4.2 Where are my Extensions?

All extensions are in the *Extensions* folder. If this folder is empty, you may need to manually download the Extensions or reinstall Simba. If you have downloaded a recent Simba (0.97.3 or higher), you will also have an Extension Updater Extension (you read that right!), make sure to enable this extension as well.

CHAPTER

FIVE

REPORTING BUGS

Report all bugs at http://bugs.villavu.com/

Make sure you do a quick search for your bug to see if it already exist. If it does, click its link and add any information you can into the comments. If your bug does not exist, follow these steps:

1. Upon opening that page, you should see a “Report Issue” link. Click this.
2. Now, you should select the project to report to. Use the following criteria:
   - If the problem is with the GUI, choose “Simba”.
   - If the problem seems to be in the library, choose “MML”.
   - If the problem is not in one of the above or you know it is an SRL function causing the problem, choose “SRL”.

   If you can not decide, choose “Simba”. The developers can move the ticket as needed.
3. Now, choose the Category that the bug is in. This should be self-explanatory; choose a topic related to what you were doing when the bug occurred. Choose general if you are unsure. Developers can, again, change the ticket if required.
4. Select the reproducibility. If this is a request for a feature, choose “N/A”.
5. Severity should be ranked according to how much the bug affects the program. Crashes will be dealt with first, then blocks, etc. This step goes along with priority, which should be chosen along with the severity.
6. Version, at this time, is not needed for Simba and MML issues. If the issue occurs with an older script, choose “SRL 4 Compatibility”.
7. Now write the summary of the bug. Choose something better than “Does not Work”. If, for example, you were experiencing a crash of the program when trying to use the OCR to scan the character “~” on the 30th of February while you were woodcutting willows with an iron axe, a suitable Summary might be “Crash While Using OCR on Specific Char”.
8. Using the same example for above, a Description should explain the problem while being concise. This field is not the field for technical details, simply describe the problem.
9. Include any technical details such as what exactly you did, etc. Use pictures if possible.
10. Finally, hit submit. It should return successful.

CHAPTER

SIX

DETAILED FEATURE OVERVIEW

Simba has some fancy features; the most basic operations as well as detailed information about settings will be explained here.

6.1 Simba Script Manager

Community-created scripts will be available for download from the Script Manager. The script manager will appear soon.

6.2 Updater in Simba

As soon as there is an update for Simba, an extra icon will appear in the icon bar. If you click it, an update window will appear. Click *Update* and Simba will update itself.

Note: You will have to restart Simba to use the updated version.

Alternatively you can use Tools -> Update to update.

6.3 Simba Extensions

Simba extensions are exactly what the name suggests: Extensions for Simba. These extensions vary from updaters to firewalls and design tools.

By default all extensions are disabled. They reside in the *Extensions* folder and end with `.sex`. To enable or disable Extensions go to Tools > Extensions. There you will find a list of all extensions.

6.3.1 Recommended Simba Extensions

Recommended:

- Extension Updater (Highly recommended)
- SRL Updater (and Installer/Download)
- Associate Files
- Security extension. (Implements a File and Fire wall)

For developers:

- DTM Editor
- Paster Extension
- CRov, Crash Recovery.
- Code Formatter extension.

6.4 Settings Form

As of Simba 0.99, a decent settings manager has been implemented. If you still want to use the old settings manager, use the option *Settings (Expert)* from the menu.

6.5 Simba settings

The Simba settings form can be used to modify the settings. These options vary from how often Simba will check for an update to whether the tray icon is displayed at all times.

Settings can be changed by going to the menu *Tools* menu and the selecting the *Settings* option.

If you at any time have serious problems with Simba not behaving it as you expect it should and you changed something in the settings, you could try closing Simba and deleting `settings.xml`. Simba will create one with default settings if it does not exist.

6.5.1 General Settings

`Settings/General/`

Recent files history

`MaxRecentFiles`

Stores the amount of recent files to remember. The default is *10*.

6.5.2 Extensions

`Settings/Extensions`

Settings related to extensions are found here.

Extension directory

`Path`

This setting contains the directory Simba searches for its extensions.

6.5.3 Plugins

`Settings/Plugins/`

Plugin directory

`Path`

This setting contains the directory Simba searches for its plugins.

6.5.4 Includes

`Settings/Includes/`

Include directory

`Path`

This setting contains the directory Simba searches for its global includes.

6.5.5 Fonts

`Settings/Fonts/`

Automatically load fonts on startup

`LoadOnStartUp`

Default is *True*. If set to *False* Fonts will not be loaded on startup.

Font Directory

`Path`

Contains the directory Simba will look for when searching for fonts.

6.5.6 Updater

`Settings/Updater/`

The `Updater` refers only to the Simba updater, and not to other updaters like the Font updater or other third party updaters.

Automatically check for updates

`CheckForUpdates`

Default is *True*. If set to *True*, Simba will automatically check if there are updates available. If you don't want Simba to check for updates of Simba, set this to *False*.

How often we check for updates

`CheckEveryXMinutes`

Default is *30*. The number here defines how often Simba will check for Simba updates, in minutes.

6.5.7 Code Hints

`Settings/CodeHints/`

Automatically pop up Code Hints

`ShowAutomatically`

If set to *True*, code hints will be shown when appropriate. On *False* code hints will never be shown automatically, but can still be accessed by pressing *Ctrl+Space*.

6.5.8 Code Hints

`Settings/CodeCompletion/`

Automatically pop up Code Hints

`ShowAutomatically`

If set to *True*, code completion will pop up when appropriate. On *False* code will never be completed automatically.

6.5.9 Function list

`Settings/FunctionList/`

Show on Start

`ShowOnStart`

It true is set to *True*, the function list will be visible on startup. *False* requires you to enable it yourself.

6.5.10 Color Picker

`Settings/ColourPicker`

Show colour history on pick?

`ShowHistoryOnPick`

If set to *True*, the Colour History form will be shown every time the user has picked a colour.

6.5.11 Script Tabs

`Settings/Tabs/`

Script opening mode

`OpenScriptInNewTab`

Simba can load Scripts in two different ways. It can open every script in a new tab, or it can override the currently active tab. Setting this to *True* opens scripts in new tabs, this is the default.

Behaviour when closing tabs

`OpenNextOnClose`

Once a tab is closed, Simba can open your most recent tab, or the tab that is next to the closed tab. Setting this to *False* jumps back to your most recently used tab.

6.5.12 Source Editor

`Settings/SourceEditor/`

Source colouring

`LazColors`

The default colour theme is the same as Lazarus' colours. If you prefer another (more familiar?) theme, set this to *False*.

Default script

This contains the path to the default script. You can change this to set your own default script. (Opening a blank tab loads up the default script)

6.5.13 Tray Icon

`Settings/Tray`

Visiblity of the tray icon

`AlwaysVisible`

If you want the tray icon only to be visible when Simba is minimized, set this to *False*.

Note: Here should be a list of the most important Simba settings; what they influence, plus the possible values for them.

6.6 User Interface

6.6.1 Menu items

(Format) Main menu item Submenu item, [shortcut,]description

File Menu

- New, Ctrl+N, Opens a new tab with the default script
- Open, Ctrl+O, Prompted for an existing file to open in a new tab (by default)
- Open recent, Lists up to the last 10 (by default) files that have been open
- Save, Ctrl+S, Save the current script - if it hasn't been saved before or opened, then prompted for a filename
- Save as, Save the current script as a new filename
- Save as Default, Make it so all new files will be the current script.
- Save All, Shift+Ctrl+S, Saves all currently open scripts
- New Tab, Ctrl+T, Opens a new tab with the default script
- **Close Tab, Ctrl+W, Closes current tab, prompting for a save if it hasn't** been already
- **Close all tabs, Closes all tabs, prompting for a save if individual tabs** haven't been already
- Exit, Ctrl+Q, Closes Simba, prompting for a save if individual tabs haven't been already

Edit Menu

- Undo, Ctrl+Z, Changes the script to a previous state where available
- Redo, Ctrl+Y, Changes the script to a later state where available
- Cut, Ctrl+X, Removes the currently selected text and makes it available to paste
- Copy, Ctrl+C, Copies the selected text and makes it available to paste
- Paste, Ctrl+V, Inserts a copy of previously copied/cut text at the current position
- Delete, Delete/Backspace, Removes the currently selected text
- Select All, Ctrl+A, Makes all the text in the open tab selected
- Find, Ctrl+F, Allows user to search for the first occurrence of a text sequence, making it selected, with or without case matching as well as highlighting other occurrences
- Find next, F3, Selects next occurrence of text sequence being searched for
- Replace, Ctrl+R, Opens a find/replace dialogue box, allowing a search string to be specified as well as a replacement string, with options of finding occurrences and replacing a single or all occurrences

Script

- Run, F9, Compile and start executing the script
- Compile, Ctrl+F9, Only compile the script without executing
- Pause, Stop executing the script, but allow it to be resumed from the current place via run
- Stop, F2, Stop executing the script completely without allowing it to be resumed from the current place

View

- Colour History, Toggles display of the colour picker history form
- Debug Image, Toggles display of the debug image "orm"
- Function List, Toggles display of the function list box
- Extensions, Toggles display of the extensions form

Tools

- Fill Function List, (Ctrl+Q, which is also assigned to Exit, which takes precedence), Reloads the functions from the script into the function list box
- Update, Opens the update form
- Settings, Opens the settings form
- Bitmap conversion, Opens a form for converting images to bitmap strings
- Export script as HTML, Prompts for a filename to save HTML file with formatted version of script that has syntax highlighting

Help

- About, Opens about form
- Handbook, Opens up the online information resource for Simba in default browser.
- Report a bug, Opens up the online bugtracker for Simba in default browser
- Check for new SRL, Sees if local version is the latest version of SRL

SRL

(Only exists if the srl.sex/SRL Updater extension is installed and enabled):

- Update SRL, Downloads new version of SRL if one is available
- Automatically update, toggles whether or not to update SRL without asking upon starting Simba/enabling SRL Updater extension

6.6.2 Toolbar buttons

(Format) Function (listed from left to right), [shortcut command,]menu [,description if not present on menu]

- New, Ctrl+N, File
- Open, Ctrl+O, File
- Save, Ctrl+S, File
- Save All, Shift+Ctrl+S, File
- Cut, Ctrl+X, Edit
- Copy, Ctrl+C, Edit
- Paste, Ctrl+V, Edit

- Run, F9, Script
- Pause, Script
- Stop, F2, Script
- Add Tab, Ctrl+T, File
- Close Tab, Ctrl+W, File
- Clear debug, no menu, Removes all lines in the debug box
- Colour picker, no shortcut (accessible from View->Colour History), Freezes screen and allows use to pick a colour while displaying extra information
- Select target, no menu, Allows user to choose an area they would like Simba to perform all actions on (must be held then released over target to select it)
- Reload plugins, no menu, (Currently unavailable) Allows user to reload plugins after updating them without requiring Simba to be restarted
- Minimize to tray, no menu, Removes Simba from visible screen and start bar allowing it to only be accessed via the Simba tray icon
- Toggle console, no menu, Either displays or removes the output console

6.6.3 SyncEdit

Multiple references to an identifier in a section of text can be altered at once using SyncEdit. Select the text in which you wish to alter an identifiers name and press Ctrl+J. The highlighted area should turn from a blue colour to a green colour with identifiers displayed with grey boxes around them, except for the currently selected identifier (which should be in a blue box) and other references to the same identifier (which should be in a purple /pink box). To change to a different identifier, click in a different box. To edit the identifier, simply edit one of the boxes. When finished, press Esc. This can be used to change variable names, procedure/function names (and calls to them), custom type names (and declarations that use them). It can also be used to change references to a type as well as any uses of that type. Trying to edit something not in a box causes SyncEdit to exit, as it would when pressing Esc.

6.6.4 Read Only / External Editor Mode

Simba has a read only mode that can be enabled and disabled per tab. If a tab is read only, it will reload the file the script belongs to every time, and you're obviously not able to edit the script in Simba. This is particularly useful if you're editing from a different editor.

6.6.5 Function list

It can be undocked or moved from one side to the other. To reset it to its default position, restart Simba. As of now, it doesn't remember where it was last. If it doesn't correctly display Script functions/procedures, it may be that there is a problem with the script before the declaration. Items are not currently sorted alphabetically at any level, however Script and Includes items are added in the order they appear in the relevant scripts. If you close it, it can be displayed again from the View menu.

6.6.6 Extensions

The Extensions form is launched from the View menu via the Extensions item. To enable or disable an extensions, select it from the list and tick or untick the box in the bottom left hand corner appropriately (ticked for enabled, unticked for disabled). Listed extensions are, by default, located in the Extensions folder where the Simba executable is. The file extensions .sex is short for Simba EXtension. Extensions focus more on the Simba form or similar, general area where as plugins focus more on the scripting aspect.

For a list of all Simba extensions, have a look here at the *Simba Extensions*.

6.6.7 Colour Picker History

The colour picker history form is launched from the View menu via the Colour History item. It lists all colours that have been picked since launching Simba, unless they have been deleted, as well as information on where they were picked and RGB values. Names can be given to colours, to help relate it to what it represents, by selecting a colour item from the left and changing the top field on the right. It is, by default, displayed after picking a colour using the button on the main Simba toolbar.

6.6.8 Update Form

The update form is launched from Tools menu via the Update item. Informs user if no update is currently available; else it will displaying the form. Gives the option of updating when the current version is not the latest.

Warning: Include a picture?

6.6.9 Settings Form

The Simba settings form is launched from the Tools menu via the Settings item. The main tree consists of 3 sections: one for the functionality of Simba (Settings), one for the appearance of Simba (LastConfig) and one for holding settings relevant to extensions (Extensions).

The following information applies for the main Settings branch and not LastConfig or Extensions.

The Settings section allows toggling of:

- Loading fonts on startup (Fonts->LoadOnStartUp)
- Showing the function list on startup (FunctionList->ShowOnStart)
- Using Lazarus syntax highlighting (SourceEditor->LazColors) [Requires restarting Simba]
- Using CPascal or PascalScript interpreter (Interpreter->UseCPascal)
- Automatically checking for Simba updates (Updater->CheckForUpdates)
- Opening next tab upon closing current (Tabs->OpenNextOnClose) [Unsure on what it exactly does]
- Open new scripts in current or new tab (Tabs->OpenScriptInNewTab) [Does prompt for saving if current script is unsaved]
- Automatically showing code hints (CodeHints->ShowAutomatically)
- Automatically show code completion (CodeCompletion->ShowAutomatically)
- Automatically open colour picker history form after picking a colour (ColourPicker->ShowHistoryOnPick)

The Settings section allows changing the path of:

- Where to load includes from (Includes->Path)
- Where to load fonts from (Fonts->Path)
- Where to check latest fonts version (Fonts->VersionLink)

- Where to download latest fonts from (Fonts->UpdateLink)
- Path to the default script (SourceEditor->DefScriptPath)
- Where to load extensions from (Extensions->Path) [The file extension for Simba extensions can also be changed]
- Where to load plugins from (Plugins->Path)
- Where to load the news from (News->URL)
- Where to check latest Simba version (Updater->RemoteVersionLink)
- Where to download latest Simba from (Updater->RemoteLink)

The Settings section has further settings which do not fall into the above two categories of sorts. These are:

- Version of local fonts (Fonts->Version)
- How often to check for new version of Simba (Updater->CheckEveryXMinutes)
- Number of recent files to be remembered (General->MaxRecentFiles)
- The file extension for Simba Extensions (Extensions->FileExtension)

It is highly suggested that the large majority of settings should not be altered unless you know what you are doing.

6.7 Not Well Known Features

Simba has several features that are relatively unknown. A few will be listed here.

6.7.1 Custom Block Folding

TODO

6.7.2 Timestamped Writeln

Simba can timestamp all your debug.

```
SetScriptProp(SP_WriteTimeStamp, [True]);
WriteLn('Before the wait');
Wait(1000);
Writeln('After the wait');
```

CHAPTER

SEVEN

TUTORIAL

7.1 Introduction to Simba and Programming

7.2 Simba Basics

7.3 Simba Mouse Input

7.4 Simba Keyboard Input

7.5 Simba Colours

7.6 Simba Colour Tolerance Speed

7.7 Simba Bitmaps

7.8 Simba Deformable Template Models (DTM)

7.9 Simba TPointArrays

7.10 Simba Files

CHAPTER

EIGHT

SCRIPTING REFERENCE

Covered in this section are the functions available when using Simba with the default (Pascal) engine.

Note: More chapters need to be written, see *Documentation TODO*

8.1 Mouse and Keyboard

Simba contains several functions to manipulate the mouse and keyboard. Features range from clicking and moving the mouse to faking keypresses.

8.1.1 Types

A few variables are exported for working with the Simba mouse functions.

TClickType, which defines the click type.

```
const
    mouse_Right = 0
    mouse_Left = 1
    mouse_Middle = 2
```

TMousePress, which defines if the mouse button is to be down or up.

```
TMousePress = (mouse_Down, mouse_Up);
```

Keyboard functions often use Virtual Keys. See *virtualkeys* for a complete list.

8.1.2 Mouse Functions

Simba's coordinate system is similar to most computer coordinate systems. Coordinate *(0, 0)* is the top-left part of your selected window (the desktop by default). To get to point *(5, 0)* we move five pixels to the right; to get to *(5, 5)* we move five pixels to the right, and five pixels down.

Recall that the first value is *x*, the second value is *y*: *(x, y)*

MoveMouse

```
procedure MoveMouse(x, y: integer);
```

MoveMouse moves the mouse pointer to the specified x and y coordinates.

The following example will move the mouse to position *(10, 10)*; relative to the selected client. (To get to point (10, 10) visually, recall that (0, 0) is the *top left* part and to get to (10, 10) we move 10 pixels to the right, and ten pixels down.)

```
Program MouseMove;

begin
  MoveMouse(10, 10);
end.
```

GetMousePos

```
procedure GetMousePos(var x, y: integer);
```

GetMousePos returns the current position of the mouse in x and y.

The following example moves the mouse 1 pixel to the right, relative to its current position:

```
Program MouseMoveRelative;

var x, y: integer;

begin
  GetMousePos(x, y);
  MoveMouse(x + 1, y);
end.
```

HoldMouse

```
procedure HoldMouse(x, y: Integer; clickType: TClickType);
```

HoldMouse holds the given mouse button specified by clickType down at the specified *(x, y)* coordinate. If the mouse if not at the given *(x, y)* yet, the mouse position will be set to *(x, y)*.

The following example holds the left mouse button and moves it one pixel to the right relative to its current position.

```
program HoldMouse;

var x, y: integer;

begin
  GetMousePos(x, y);
  HoldMouse(x, y, mouse_Left);
  MoveMouse(x + 1, y);
end.
```

ReleaseMouse

```
procedure ReleaseMouse(x, y: Integer; clickType: TClickType);
```

HoldMouse holds the given mouse button (clickType) down at the specified x, y coordinate. If the mouse if not at the given x, y yet, the mouse position will be set to x, y.

The following example holds the left mouse button and moves it one pixel to the right and releases it to simulate a drag and drop motion.

```
program HoldMouseRelative;

var x, y: integer;

begin
  GetMousePos(x, y);
  HoldMouse(x, y, mouse_Left);
  MoveMouse(x + 1, y);
  GetMousePos(x, y);
  ReleaseMouse(x, y, mouse_Left);
end.
```

ClickMouse

```
procedure ClickMouse(x, y: Integer; clickType: Integer):
```

ClickMouse performs a click with the given mouse button (clickType) at the specified *(x, y)* coordinate. This `click` equals an immediate click, with no wait between holding down and releasing the mouse button. To create a more human-like effect, use the HoldMouse and ReleaseMouse functions.

The following example clicks the right mouse button at a specified point.

```
program ClickMouse;

var x, y: integer;

begin
  ClickMouse(x, y, mouse_Right);
end.
```

8.1.3 Keyboard Functions

Keyboard functions are obviously used to manipulate the keyboard input. It can also be used to read states of specific keys.

KeyDown

```
procedure KeyDown(key: Word);
```

KeyDown sends a request to the Operating System to “fake” an event that causes the keyboard `key` to be “down”.

The following example holds down the “Enter” key. (Note that if you call KeyDown you must call key up afterwards to release the key.)

```
program KeyDown;

begin
```

```
  KeyDown(13);
end.
```

KeyUp

```
procedure KeyUp(key: Word);
```

KeyDown sends a request to the Operating System to "fake" an event that causes the `key` to be "up".

The following example holds down the "Enter" key and release after pausing to simulate a human action.

```
program KeyDownRelative;

begin
  KeyDown(13);
  wait(RandomRange(50, 100));
  KeyUp(13);
end.
```

PressKey

```
procedure PressKey(key: Word);
```

PressKey sends a request to the Operating System to "fake" a key event.

The following example simulates pressing the "Enter" key.

```
program PressKey;

begin
  PressKey(13);
end.
```

IsKeyDown

```
function isKeyDown(key: Word): boolean;
```

Returns *True* if *key* is currently being held.

The following example says "Hello World" if the "F5" key is held down.

```
program PressKey;

begin
  while not(isKeyDown(116)) do
    Wait(50);
  writeln('Hello World');
end.
```

GetKeyCode

```
function GetKeyCode(c: char) : integer;
```

Returns the keycode of character *c*.

SendKeys

```
procedure SendKeys(const s: string; keywait, keymodwait: integer);
```

Type the contents of the string *s*. While typing, hold the keys for *keywait*. SendKeys should work with any keyboard layout on Windows.

Example:

```
SendKeys('Hello, World', 100, 30);
```

Note: If your goal is to randomly wait a small time per key, you should consider something like this:

```
s := 'Hello, World';
for i := 0 to length(s) do
begin
  SendKeys(s[i], 50+Random(51), 30+Random(30));
  Wait(20+Random(30));
end;
```

8.1.4 Keyboard Virtual Keys

- UNKNOWN: 0
- LBUTTON: 1
- RBUTTON: 2
- CANCEL: 3
- MBUTTON: 4
- XBUTTON1: 5
- XBUTTON2: 6
- BACK: 8
- TAB: 9
- CLEAR: 12
- RETURN: 13
- SHIFT: 16
- CONTROL: 17
- MENU: 18
- PAUSE: 19
- CAPITAL: 20
- KANA: 21
- HANGUL: 21
- JUNJA: 23
- FINAL: 24
- HANJA: 25

- KANJI: 25
- ESCAPE: 27
- CONVERT: 28
- NONCONVERT: 29
- ACCEPT: 30
- MODECHANGE: 31
- SPACE: 32
- PRIOR: 33
- NEXT: 34
- END: 35
- HOME: 36
- LEFT: 37
- UP: 38
- RIGHT: 39
- DOWN: 40
- SELECT: 41
- PRINT: 42
- EXECUTE: 43
- SNAPSHOT: 44
- INSERT: 45
- DELETE: 46
- HELP: 47
- 0: 30
- 1: 31
- 2: 32
- 3: 33
- 4: 34
- 5: 35
- 6: 36
- 7: 37
- 8: 38
- 9: 39
- A: 41
- B: 42
- C: 43
- D: 44

- E: 45
- F: 46
- G: 47
- H: 48
- I: 49
- J: 4A
- K: 4B
- L: 4C
- M: 4D
- N: 4E
- O: 4F
- P: 50
- Q: 51
- R: 52
- S: 53
- T: 54
- U: 55
- V: 56
- W: 57
- X: 58
- Y: 59
- Z: 5A
- LWIN: 5B
- RWIN: 5C
- APPS: 5D
- SLEEP: 5F
- NUMPAD0: 96
- NUMPAD1: 97
- NUMPAD2: 98
- NUMPAD3: 99
- NUMPAD4: 100
- NUMPAD5: 101
- NUMPAD6: 102
- NUMPAD7: 103
- NUMPAD8: 104
- NUMPAD9: 105

- MULTIPLY: 106
- ADD: 107
- SEPARATOR: 108
- SUBTRACT: 109
- DECIMAL: 110
- DIVIDE: 111
- F1: 112
- F2: 113
- F3: 114
- F4: 115
- F5: 116
- F6: 117
- F7: 118
- F8: 119
- F9: 120
- F10: 121
- F11: 122
- F12: 123
- F13: 124
- F14: 125
- F15: 126
- F16: 127
- F17: 128
- F18: 129
- F19: 130
- F20: 131
- F21: 132
- F22: 133
- F23: 134
- F24: 135
- NUMLOCK: 90
- SCROLL: 91
- LSHIFT: A0
- RSHIFT: A1
- LCONTROL: A2
- RCONTROL: A3

- LMENU: A4
- RMENU: A5
- BROWSER_BACK: A6
- BROWSER_FORWARD: A7
- BROWSER_REFRESH: A8
- BROWSER_STOP: A9
- BROWSER_SEARCH: AA
- BROWSER_FAVORITES: AB
- BROWSER_HOME: AC
- VOLUME_MUTE: AD
- VOLUME_DOWN: AE
- VOLUME_UP: AF
- MEDIA_NEXT_TRACK: B0
- MEDIA_PREV_TRACK: B1
- MEDIA_STOP: B2
- MEDIA_PLAY_PAUSE: B3
- LAUNCH_MAIL: B4
- LAUNCH_MEDIA_SELECT: B5
- LAUNCH_APP1: B6
- LAUNCH_APP2: B7
- OEM_1: BA
- OEM_PLUS: BB
- OEM_COMMA: BC
- OEM_MINUS: BD
- OEM_PERIOD: BE
- OEM_2: BF
- OEM_3: C0
- OEM_4: DB
- OEM_5: DC
- OEM_6: DD
- OEM_7: DE
- OEM_8: DF
- OEM_102: E2
- PROCESSKEY: E7
- ATTN: F6
- CRSEL: F7

- EXSEL: F8
- EREOF: F9
- PLAY: FA
- ZOOM: FB
- NONAME: FC
- PA1: FD
- OEM_CLEAR: FE
- HIGHESTVALUE: FE
- UNDEFINED: FF

8.2 Colour Finding

Finding colours on the screen is quite simple. Simba offers methods like `FindColor` to locate colours on the screen.

These methods are usually composed out of several (but not always all) components:

- The colour to search for. This is an RGB color.
- An area to search in, defined by *x1*, *y1*, *x2*, *y2*. If any of coordinates are outside the clients bounds; two things can happen depending on your settings:
  - Simba throws an Exception.
  - Simba will resize the bounds to fit the client without notifying you.
- Tolerance applied to the colour matching. With a maximum tolerance all colours are matched.
- Spiral. A spiral defines a point where the search will start from. This is particulary useful if you want the first result near specific coordinates.
- AreaSize. The size the box of colours should be. Usually this is not adjustable.
- A single point in *x*, *y* can be returned, or a set or points called a *TPointArray*.

Note: Other techniques exist, which involve relative point distance from one point to another; these are found in the *Deformable Template Models (DTM)* section.

Note: Although the documentation uses the `English` spelling of `colour`; the code for compatibility sake uses `color`, without the u.

8.2.1 Colour Finding Methods

A list of all colour finding methods in Simba.

SimilarColors

```
function SimilarColors(C1, C2, Tolerance: Integer): Boolean;
```

SimilarColors returns true if the two passed colours are *similar* given the passed *Tolerance*.

GetColor

```
function GetColor(x, y: Integer): Integer;
```

GetColor returns the color on the coordinate *(x, y)*.

Example printing the colour on coordinate (25, 25).

```
program printcolour;

begin
  Writeln('Colour is ' + IntToStr(GetColor(25, 25)))
end.
```

GetColors

```
function GetColors(const Coords : TPointArray) : TIntegerArray;
```

GetColors returns an array of the colours at the given *Coords*.

CountColor

```
function CountColor(Color, xs, ys, xe, ye: Integer): Integer;
```

Returns how many times *Color* occurs in the area defined by (*xs*, *ys*), (*xe*, *ye*)

CountColorTolerance

```
function CountColorTolerance(Color, xs, ys, xe, ye, Tolerance: Integer): Integer;
```

Returns how many times *Color* occurs (within *Tolerance*) in the area defined by (*xs*, *ys*), (*xe*, *ye*)

FindColor

```
function FindColor(var x, y: Integer; col, x1, y1, x2, y2: Integer): Boolean;
```

FindColor returns true if the exact colour given (col) is found in the box defined by *x1*, *y1*, *x2*, *y2*. The point is returned in *x* and *y*. It searches from the top left to the bottom right and will stop after matching a point.

FindColorTolerance

```
function FindColorTolerance(var x, y: Integer; col, x1, y1, x2, y2, tol: Integer): Boolean;
```

FindColorTolerance returns true if a colour within the given tolerance range *tol* of the given colour *col* is found in the box defined by *x1*, *y1*, *x2*, *y2*. Only the first point is returned in *x* and *y*. Whether or not a colour is within the tolerance range is determined by the *Colour tolerance* mode. It searches from the top left to the bottom right and will stop after matching a point.

FindColors

```
function FindColors(var pts: TPointArray; col, x1, y1, x2, y2): Boolean;
```

FindColors returns a list of all points that match the colour *col* in an area defined by *x1*, *y1*, *x2*, *y2*. It returns true if one or more points have been found.

FindColorsTolerance

```
function FindColorsTolerance(var pts: TPointArray; col, x1, y1, x2, y2, tol: Integer): Boolean;
```

FindColorsTolerance returns true if at least one point was found. A point is found if it is within the given tolerance range *tol* of the given colour *col* and inside the box defined by *x1*, *y1*, *x2*, *y2*. Whether or not a color is within the tolerance range is determined by the *Colour tolerance* mode. It searches from the top left to the bottom right and will find all matching points in the area.

FindColorSpiral

```
function FindColorSpiral(var x, y: Integer; color, xs,ys,xe,ye:Integer): Boolean;
```

Same as FindColor, but starts searching from *x*, *y*.

FindColorSpiralTolerance

```
function FindColorSpiralTolerance(var x, y: Integer; color, xs,ys,xe,ye,tolerance:Integer): Boolean
```

Same as FindColorTolerance, but starts searching from *x*, *y*.

FindColorsSpiralTolerance

```
function FindColorsSpiralTolerance(x, y: Integer; var pts: TPointArray; col, x1, y1, x2, y2, tol: Int
```

Same as FindColorsTolerance, but starts searching from *x*, *y*.

Find areas of colours

```
function FindColoredArea(var x, y: Integer; color, xs, ys, xe, ye, MinArea: Integer): Boolean;
```

FindColoredArea finds an area that consists out of *Color* and has a minimal size of *MinArea*. If you want minimal area of 5x5 pixels (25), then set MinArea to 25.

```
function FindColoredAreaTolerance(var x, y : Integer; color, xs, ys, xe, ye, MinArea, Tolerance : Int
```

FindColoredArea finds an area that consists out of Colours that match *Color* with the given *Tolerance* and has a minimal size of *MinArea*. If you want minimal area of 5x5 pixels (25), then set MinArea to 25.

8.2.2 Colour tolerance

Simba contains several algorithms for determining if two colours are equal given a tolerance. There are three algorithms, from fastest to slowest:

- CTS 0: Quick and dirty comparison. Matches if the differences between the three RGB values are <= Tolerance

- CTS 1: RGB comparison that uses the Pythagorean distance in the RGB cube to define tolerance. Matches if the distance <= Tolerance.
- CTS 2: HSL comparison. It has two modifiers that modify the result tolerance, Hue and Saturation. The lower the modifier, the higher tolerance required for a match. They can be set seperately and therefore used to distinguish very specific colours. Some differ a lot in saturation, but very little in hue. Luminance is assigned a somewhat static function, and has no modifier.
- CTS 3: Comparison using CIE L*a*b colour space, uses Pythagorean distance.

Get and Set Colour Tolerance

```
procedure SetColorToleranceSpeed(cts: integer);
```

Set the current colour tolerance speed. Valid values are: 0, 1 and 2. Somewhat improperly named compared to the other CTS functions.

```
SetColorToleranceSpeed(2);
```

And the proper way to get the current tolerance is to use the following function, which returns the current colour tolerance speed:

```
function GetToleranceSpeed: Integer;
```

Example printing the Color Tolerance

```
Writeln(Format('Tolerance Speed = %d', [GetToleranceSpeed]))
```

Get And Set Colour Modifiers

```
procedure SetToleranceSpeed2Modifiers(nHue, nSat: Extended);
```

Set colour speed 2 modifiers.

```
// 42.0 is a very high value, but this doesn't matter as this code is
// only meant to illustrate how to use this function
SetToleranceSpeed2Modifiers(42.0, 0.4)
```

The following function

```
procedure GetToleranceSpeed2Modifiers(var hMod, sMod: Extended);
```

returns colour speed 2 modifiers.

Example getting the modifiers:

```
procedure WriteModifiers;
var
    H, S: Extended;
begin
    GetToleranceSpeed2Modifiers(H, S);
    Writeln(format('H = %f; S = %f', [H, S]));
end;
```

8.3 Colour Conversions

8.3.1 Colour spaces

Explain some colour spaces here.

8.3.2 Colour Conversion Methods

ColorToRGB

```
procedure ColorToRGB(Color: integer; var r, g, b: Integer);
```

RGBtoColor

```
function RGBtoColor(r, g, b: Integer): TColor;
```

ColorToHSL

```
procedure ColorToHSL(Color: Integer; var h, s, l: Extended);
```

HSLToColor

```
function HSLToColor(H, S, L: Extended): TColor;
```

ColorToXYZ

```
procedure ColorToXYZ(Color: Integer; var x, y, z: Extended);
```

XYZToColor

```
function XYZToColor(X, Y, Z: Extended): TColor;
```

RGBToHSL

```
procedure RGBToHSL(R, G, B: Integer; var h, s, l: Extended);
```

HSLtoRGB

```
procedure HSLtoRGB(H, S, L: extended; var R, G ,B: Integer);
```

RGBToXYZ

```
procedure RGBToXYZ(R, G, B: Integer;var x, y ,z: Extended);
```

XYZToRGB

```
procedure XYZToRGB(X, Y, Z: Extended; var R, G, B: Integer);
```

8.4 Working with Files

Files in Simba are all *integers*, internal handles for Simba which on their turn point to operating system files. Functions like CreateFile and OpenFile return a file handle. You should not forget to close these when you no longer need them.

8.4.1 CreateFile

```
function CreateFile(const Path: string): Integer;
```

Create a file with *Path*. Raturns -1 on failure, otherwise returns the handle to the file.

8.4.2 OpenFile

```
function OpenFile(const Path: string; Shared: Boolean): Integer;
```

Opens file for reading. Opens shared if *Shared* is true. Returns -1 on failure, otherwise returns the handle to the file.

8.4.3 RewriteFile

```
function RewriteFile(const Path: string; Shared: Boolean): Integer;
```

Opens file for rewriting. (File is cleared on open) Opens shared if *Shared* is true. Returns -1 on failure, otherwise returns the handle to the file.

8.4.4 AppendFile

```
function AppendFile(const Path: string): Integer;
```

Opens file for writing (appending). Returns -1 on failure, otherwise returns the handle to the file.

8.4.5 CloseFile

```
procedure CloseFile(FileNum: Integer);
```

Close the file defined by *FileNum*. Never forget to close your files!

8.4.6 DeleteFile

```
function DeleteFile(const Filename: string): Boolean;
```

Delete the file with name *Filename*. Returns true on success.

8.4.7 EndOfFile

```
function EndOfFile(FileNum: Integer): Boolean;
```

Returns true if the end of the file has been reached.

8.4.8 FileSize

```
function FileSize(FileNum: Integer): LongInt;
```

Returns the file size in characters.

8.4.9 ReadFileString

```
function ReadFileString(FileNum: Integer; var s: string; x: Integer):
Boolean;
```

Read *x* characters into string *s* from file *FileNum*. Returns true if the number of characters read equals *x*.

8.4.10 WriteFileString

```
function WriteFileString(FileNum: Integer; s: string): Boolean;
```

Writes *s* to file *FileNum*. Returns false on failure.

8.4.11 SetFileCharPointer

```
function SetFileCharPointer(FileNum, cChars, Origin: Integer): Integer;
```

Seek through the file. Set the cursor to *cChars* from *Origin*.

Origin can be any of these:

```
{ File seek origins }
FsFromBeginning = 0;
FsFromCurrent   = 1;
FsFromEnd       = 2;
```

8.4.12 FilePointerPos

```
function FilePointerPos(FileNum: Integer): Integer;
```

Returns the position of the *cursur* in the file. (What character # you are at)

8.4.13 DirectoryExists

```
function DirectoryExists(const DirectoryName : string ) : Boolean;
```

Returns true if the directory exists.

8.4.14 CreateDirectory

```
function CreateDirectory(const DirectoryName : string) : boolean;
```

Creates a directory. Returns true on success.

8.4.15 FileExists

```
function FileExists (const FileName : string ) : Boolean;
```

Returns true if the file exists.

8.4.16 ForceDirectories

```
function ForceDirectories(const dir : string) : boolean;
```

Creates multiple *nested* directories. Returns true on success.

8.4.17 GetFiles

```
function GetFiles(const Path, Ext : string) : TStringArray;
```

Returns the files in the directory defined by *Path* with extension *Ext*. You can also set Ext as '*' to return all files in Path.

8.4.18 GetDirectories

```
function GetDirectories(const path : string) : TStringArray;
```

Returns the directories in *path*.

8.4.19 WriteINI

```
procedure WriteINI(const Section, KeyName, NewString, FileName: string);
```

The following example writes to a specific Line in a Specified INI File.

```
program WriteINIExample;

Var Section, Keyname, NewString, FileName: String;

begin
  Section := 'What subsection in the INI it is being Writen.';
  KeyName := 'Space in the specified Subsection in which you string is writen.';
  NewString := 'What your Writing into the INI file.';
  FileName := 'The Name of the INI File you are writing too.';
  WriteINI(Section, KeyName, NewString, ScriptPath + FileName);
end.
```

Note: ScriptPath will Automatically point the file finding to the same folder the script is saved in.

Note: This procedure can be used in conjunction with ReadINI to saved Player Data for the next time a script is run.

8.4.20 ReadINI

```
function ReadINI(const Section, KeyName, FileName: string): string;
```

The following example writes to a specific Line in a Specified INI File then Reads that line and prints it's findings.

```
program WriteINIReference;

Var Section, Keyname, NewString, FileName: String;

begin
  Section := 'What subsection in the INI it is being Writen.';
  KeyName := 'Space in the specified Subsection in which you string is writen.';
  NewString := 'What your Writing into the INI file.';
  FileName := 'The Name of the INI File you are writing too.';
  WriteINI(Section, KeyName, NewString, ScriptPath + FileName);
  Writeln(ReadINI(Section, KeyName, ScriptPath + FileName);
end.
```

8.4.21 DeleteINI

```
procedure DeleteINI(const Section, KeyName, FileName: string);
```

The following example deletes the specific line inside the specified INI file.

```
program DeleteINIExample;

begin
  DeleteINI('Section', Key, File);
end;
```

8.4.22 ExtractFileExt

```
function ExtractFileExt(const FileName: string): string;');
```

Returns the file extension from file *Filename*.

8.4.23 DeleteDirectory

```
function DeleteDirectory(const Dir: String; const Empty: Boolean): Boolean;
```

Deletes the directory *dir*, if Empty is true will delete the directorys contents else will not.

8.5 OCR

Simba has OCR functions (Optical Text Recognition); these are used to *read* text from an image.

It also has a wide variation of text *finding* functions. Both will be covered in this article.

8.5.1 Fonts

Fonts are an essential part of the text finding and identifying. Now follows a brief explanation of the Font related functions in Simba.

All fonts have an unique identifier in the shape of a `string`. If you create or load a new font, they will need to be given a name.

LoadSystemFont

```
function LoadSystemFont(const SysFont: TFont; const FontName: string):
boolean;
```

This font loads a previously created Font with the name specified by *FontName*.

Example:

```
program new;
var
  Font : TFont;
begin
  Font := TFont.Create;
  Font.Name := 'Courier New';
  Font.Size := 10;
  Font.Style := [];
  LoadSystemFont(Font,'test');
  DisplayDebugImgWindow(0,0);
  DisplayDebugImgWindow(150,50);
  DrawBitmapDebugImg(BitmapFromText('BMP[0] has not been freed','test'));
  Font.free;
end.
```

LoadFont

```
function LoadFont(const FontName: string; shadow: boolean): boolean;
```

Load the font specific by the *FontName*. The font has to recide in the Fonts directory.

As of Simba version 993 and up, LoadFont also accepts paths outside the Fonts directory. In this case *FontName* is thus an entire path; and the internal font name is simply the name of the last folder in the path.

FreeFont

```
function FreeFont(const FontName: string): boolean;
```

This function is used to free a font identified by *FontName*.

BitmapFromText

```
function BitmapFromText(const text, font: String): integer;
```

This function creates a bitmap from a string *text* with the given *font*. For an explanation on how to use and work with Bitmaps, please refer to *Bitmaps*.

TPAFromText

```
function TPAFromText(const text, font: String;var w,h : integer): TPointArray;
```

This function creates a TPA from a string *text* with the given *font*. For an explanation on how to use and work with TPAs, please refer to *scriptref-tpointarray*.

TPAFromTextWrap

```
procedure TPAFromTextWrap(const text, font: String;var w,h : integer;var TPA : TPointArray);
```

A wrapper function for the previously mentioned function. Required to work arounds bugs in the interpreter.

MaskFromText

```
function MaskFromText(const text, font: String): TMask;
```

This function creates a Mask from a string *text* with the given *font*. For an explanation on how to use and work with TPAs, please refer to *scriptref-masks*.

8.5.2 Reading Text

rs_GetUpText

```
function rs_GetUpText: string;
```

This function is a function specific to RuneScape(tm); it reads the text in the upper left corner into a string.

How these functions actually work can be found here: *Uptext*.

rs_GetUpTextAt

```
function rs_GetUpTextAt(x, y : integer): string;
```

This function is a function specific to RuneScape(tm); it reads the text at the specified position in (*x*, *y*) into a string.

rs_GetUpTextAtEx

```
function rs_GetUpTextAt(x, y : integer; shadow: boolean; fontname: string): string;
```

This function is a function specific to RuneScape(tm); it reads the text at the specified position in (*x*, *y*) into a string, optionally using the shadows of the font. Fontname specifies the name of the font to use; with the other functions this defaults to “UpChars”.

GetTextAt

```
function GetTextAt(const atX, atY, minvspacing, maxvspacing, hspacing,color, tol, len: integer;const
```

A general function for reading text. Reads text at (*atX*, *atY*) with a minimal vertical spacing of *minvspacing* and a maximal vertical spacing of *maxvspacing*, the text colour should match the colour *color* with the given tolerance *Tolerance*; the length of the text is specified with *len*. Finally, the font to use for the identifying is specified with the fontname *font*.

GetTextAtEx

```
function GetTextAtEx(const xs,ys,xe,ye, minvspacing, maxvspacing, hspacing,color, tol: integer;const
```

A general function for reading text. Reads text in the rectangle defined by (*xs*, *ys*), (*xe*, *ye*) with a minimal vertical spacing of *minvspacing* and a maximal vertical spacing of *maxvspacing*, the text colour should match the colour *color* with the given tolerance *Tolerance*; the length of the text is specified with *len*. Finally, the font to use for the identifying is specified with the fontname *font*.

GetTextATPA

```
function GetTextATPA(const ATPA : T2DPointArray; const maxvspacing : integer; const font : string): s
```

Similar to GetTextAt but reads the text from a ATPA rather than the client.

GetTextAtExWrap

```
function GetTextAtExWrap(const xs,ys,xe,ye, minvspacing, maxvspacing, hspacing,color, tol: integer;co
```

A wrapper function for the previously mentioned function. Required to work arounds bugs in the interpreter.

8.5.3 Modifying the Uptext filter

rs_ResetUpTextFilter

Reset the colours for the colour-filter to default. See rs_SetUpTextFilter for an example.

rs_SetUpTextFilter

```
procedure rs_SetUpTextFilter(filter: TOCRFilterDataArray);
```

Defines the colours that the colour-filter will look for.

Example:

```
program UpTextFilter;

{ Some constants for the OCR - taken directly from Simba }
const
    { Very rough limits for R, G, B }
    ocr_Limit_High = 190;
    ocr_Limit_Med = 130;
    ocr_Limit_Low = 65;


    { 'base' Colours of the Uptext }

    { White }
    ocr_White = 16777215;

    { Level < Your Level }
    ocr_Green = 65280;

    { Level > Your Level }
    ocr_Red = 255;

    { Interact or Level = Your Level }
    ocr_Yellow = 65535;

    { Object }
    ocr_Blue = 16776960;

    { Item }
    ocr_ItemC = 16744447;

    { Item }
    ocr_ItemC2 = ocr_Red or ocr_Green;

    { Shadow }
    ocr_Purple = 8388736;

const
    OF_LN = 256;
    OF_HN = -1;


{ Helper function to easily load a struct }
function load0(r_low,r_high,g_low,g_high,b_low,b_high,set_col: integer;
    is_text_color: boolean): tocrfilterdata;
begin
  result.r_low := r_low;
  result.r_high := r_high;
  result.g_low := g_low;
  result.g_high := g_high;
  result.b_low := b_low;
  result.b_high := b_high;
  result.set_col := set_col;
  result._type := 0;
  result.is_text_color:= is_text_color;
end;
```

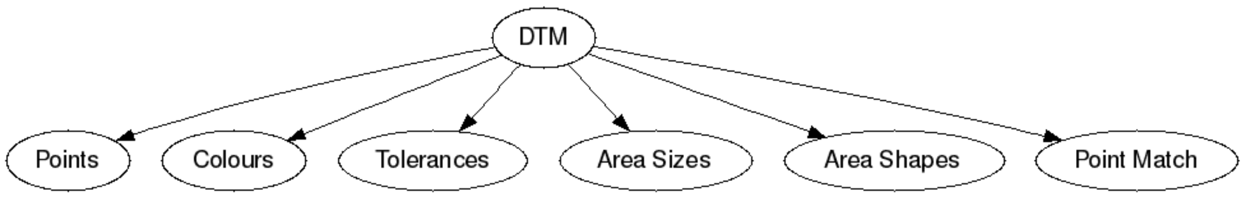
```
{
Load our own ''filter data''. This particular set doesn't contain the item
colours - those are replaced with extra (effectively nill as they already
exist) green colours.
}
rocedure foo;
var filterdata: TOCRFilterDataArray;
begin
  setlength(filterdata, 9);

  filterdata[0] := load0(65, OF_HN, OF_LN, 190, OF_LN, 190, ocr_Blue, True); // blue
  filterdata[1] := load0(65, OF_HN, OF_LN, 190, 65, OF_HN, ocr_Green, True); // green

  // ''False'' item
  filterdata[2] := load0(65, OF_HN, OF_LN, 190, 65, OF_HN, ocr_Green, True); // green

  { This is the real one }
  //filterdata[2] := load0(OF_LN, 190, 220, 100, 127, 40, ocr_ItemC, True); // itemC

  filterdata[3] := load0(OF_LN, 190, OF_LN, 190, 65, OF_HN, ocr_Yellow, True); // yellow
  filterdata[4] := load0(OF_LN, 190, 65, OF_HN, 65, OF_HN, ocr_Red, True); // red
  filterdata[5] := load0(OF_LN, 190, OF_LN, 65, 65, OF_HN, ocr_Red, True); // red 2
  filterdata[6] := load0(190 + 10, 130, OF_LN, 65 - 10, 20, OF_HN, ocr_Green, True); // green 2

  // ''False'' item 2
  filterdata[7] := load0(65, OF_HN, OF_LN, 190, 65, OF_HN, ocr_Green, True);

  { This is the real one }
  //filterdata[7] := load0(190, 140, 210, 150, 200, 160, ocr_ItemC2, True); // item2, temp item_c
  filterdata[8] := load0(65, OF_HN, 65, OF_HN, 65, OF_HN, ocr_Purple, False); // shadow

  rs_SetUpTextFilter(filterdata);
end;


var
  bmp: integer;
begin
  bmp := LoadBitmap('uptext.png');
  SetTargetBitmap(bmp);

  writeln( rs_GetUpTextAt(0, 0) );

  foo;

  writeln( rs_GetUpTextAt(0, 0) );

  rs_ResetUpTextFilter;

  writeln( rs_GetUpTextAt(0, 0) );

  SetDesktopAsClient;
  FreeBitmap(bmp);
end.
```

8.6 Deformable Template Models (DTM)

A DTM is in my view just a relatively simple way of defining a relationship between several points. Each of these points have a relative offset to each other, and may different in colour, tolerance, area size and shape. A DTM generally consists out of one *Main Point*, and several *Sub Points*.

Finding functions for DTM include the usual parameters. For explanation on these, see *Colour Finding*.

The structure of a DTM looks like this:



Where each point in a DTM has a colour, tolerance, area size and area shape entity. The main point's *point* is typically `(0, 0)`, and all the *subpoint* points are relative to the main point.

8.6.1 Example of a simple DTM

If one was to create his own DTM, he would first have to think of a useful DTM structure.

Say:

```
MainPoint = (123, 456)
SubPoint_1 = (122, 460)
SubPoint_2 = (120, 450)
```

Then we could create the following MDTM structure:

```
// Give dtm.p a length of three.
// Mainpoint
dtm.p[0] = Point(123, 456);

// Subpoints
dtm.p[1] = Point(122, 460)
dtm.p[2] = Point(120, 450)
```

Note that we do not include other variables, such as colour, tolerance, area size and area shape; they are of no importance in this example.

However, this code is not very clear about the DTM's points. Better would be to write:

```
// Give dtm.p a length of three.
// Mainpoint
dtm.p[0] = Point(0, 0);

// Subpoints
dtm.p[1] = Point(-1, 4)  // 122 - 123 = -1, 460 - 456 = 4
dtm.p[2] = Point(-3, -6) // 120 - 123 = -3, 450 - 456 = -6
```

As you can see it is perfectly valid to use negative points.

8.6.2 Colour and Tolerance

The colour value of a point in a DTM is just a RGB integer value. Black = 0, Red = 255, White = 16777215, et cetera.

The value tolerance decides if a colour is similar enough to the given colour; if this is the case, we say that the colours *matched*.

With no Area Size and Area Shape specified we say that a DTM matches if for each point in the DTM, the colour at the relative point matches the colour in dtm with the given tolerance.

8.6.3 Area Size and Shape

Area Size and Shape add that nifty extra functionality to DTM's. *Area Size* defines the area that should all match the colour with the given tolerance. *Area Shape* is currently not implemented, mainly because we haven't found a good use for area shapes.

8.6.4 Loading a DTM from a string

It is also possible to load a DTM from a *zipped* string. The details of the algorithm will not be explained here. (Have a look at dtm.pas if you're interested)

8.6.5 MDTM and TDTM

One may know DTM's as a different type:

```
TDTMPointDef = record
  x, y, Color, Tolerance, AreaSize, AreaShape: integer;
end;

TDTMPointDefArray = Array Of TDTMPointDef;

TDTM = record
  MainPoint: TDTMPointDef;
  SubPoints: TDTMPointDefArray;
end;
```

The MML provides the two functions *MDTMtoTDTM* and *TDTMtoMDTM* to directly convert between the two types.

8.6.6 Main Point and AreaSize / Shape

The main point's area size and shape are not used in the current implementation. It wouldn't be that hard to add them, however.

8.6.7 DTMFromString

```
function DTMFromString(const DTMString: String): Integer;
```

Load a DTM from a string generated by the DTM Editor.

8.6.8 SetDTMName

```
procedure SetDTMName(DTM : integer;const name : string);
```

Assign the DTM a name. Very useful for debugging purposes as it allows the programmers to find out what DTMs are not being freed.

8.6.9 FreeDTM

```
procedure FreeDTM(DTM: Integer);
```

Free a DTM identified by *DTM*.

8.6.10 FindDTM

```
function FindDTM(DTM: Integer; var x, y: Integer;
xs, ys, xe, ye: Integer): Boolean;
```

FindDTM is the most basic DTM finding function. It takes a box to search in, defined by *x1, y1, x2, y2*; and if the DTM is found, it will set *x* and *y* to the coordinate the DTM was found at and it will also return true. Else, it returns false. Once a DTM is found, it will stop searching. In other words; it always returns the first found DTM.

8.6.11 FindDTMs

```
function FindDTMs(DTM: Integer; var p: TPointArray;
xs, ys, xe, ye: Integer): Boolean;
```

FindDTMs is like FindDTM, but it returns an array of *x* and *y*, as the *TPointArray* type.

8.6.12 FindDTMRotatedSE

```
function FindDTMRotatedSE(DTM: Integer; var x, y: Integer;
xs, ys, xe, ye: Integer; sAngle, eAngle, aStep: Extended;
var aFound: Extended): Boolean;
```

FindDTMRotatedSE behaves like FindDTM. Only, it will rotate the DTM between sAngle and eAngle by aStep each time. It will also return the angle which the DTM was found at. Start rotating at StartAngle.

8.6.13 FindDTMRotatedAlternating

```
function FindDTMRotatedAlternating(DTM: Integer; var x, y: Integer;
xs, ys, xe, ye: Integer;
sAngle, eAngle, aStep: Extended; var aFound: Extended): Boolean;
```

FindDTMRotatedAlternating behaves like FindDTM. Only, it will rotate the DTM between sAngle and eAngle by aStep each time. It will also return the angle which the DTM was found at. Starts at (sAngle + eAngle) / 2 degrees and alternates between - and + aStep to search for the DTM. It finds the *closest* math to (aAngle+eAngle) / 2.

8.6.14 FindDTMsRotatedSE

```
function FindDTMsRotatedSE(DTM: Integer; var Points: TPointArray;
xs, ys, xe, ye: Integer; sAngle, eAngle, aStep: Extended;
var aFound: T2DExtendedArray) : Boolean;
```

FindDTMsRotatedSE behaves like FindRotatedDTMSE, but finds all DTM occurances. Since one point can be found on several angles, aFound is a 2d array.

8.6.15 FindDTMsRotatedAlternating

```
function FindDTMsRotatedAlternating(DTM: Integer;
var Points: TPointArray; xs, ys, xe, ye: Integer; sAngle, eAngle, aStep:
Extended; var aFound: T2DExtendedArray) : Boolean;
```

FindDTMsRotatedAlternating behaves like FindRotatedDTMAlternating, but finds all DTM occurances. Since one point can be found on several angles, aFound is a 2d array.

8.6.16 AddMDTM

```
function AddMDTM(const d: TMDTM): Integer;
```

8.6.17 AddDTM

```
function AddDTM(const d: TMDTM): Integer;
```

Load a TMDTM structure as DTM in Simba's system. (After it is loaded you can use it in FindDTM, etc)

8.6.18 AddSDTM

```
function AddSDTM(const d: TSDTM): Integer;
```

Load a TSDTM structure as DTM in Simba's system. (After it is loaded you can use it in FindDTM, etc)

8.6.19 GetDTM

```
function GetDTM(index: Integer) : TMDTM
```

Returns the TMDTM of the given DTM index.

8.6.20 SDTMToMDTM

```
function SDTMToMDTM(Const DTM: TSDTM): TMDTM;
```

Convert a SDTM to MDTM.

8.6.21 PrintDTM

```
procedure PrintDTM(const DTM : TMDTM);
```

Print the DTM contents.

8.6.22 MDTMToSDTM

```
function MDTMToSDTM(Const DTM: TMDTM): TSDTM;
```

Convert a MDTM to SDTM.

8.6.23 CreateDTMPoint

```
function CreateDTMPoint(x,y,c,t,asz : integer; bp : boolean) : TMDTMPoint;
```

Create a DTM point.

8.7 Target Window Functions

8.7.1 Freeze

```
function Freeze: boolean;
```

If you call Freeze, the data that is *currently* in the client is stored into memory. Simba will then target this memory for all further finding operations; until *Unfreeze* is called. This can dramatically increase speed if you don't care if the image doesn't change. It can be even more important if you don't *want* the image to change; if you want to analyze a specific frame.

Use like:

```
Freeze;

if findcolors(...) then
  ...

Unfreeze;
```

Make sure you never forget to call Unfreeze!

8.7.2 Unfreeze

```
function Unfreeze: boolean;
```

Unfreeze the client data and restore the original client. See *Freeze* for more details.

8.7.3 GetClientDimensions

```
procedure GetClientDimensions(var w, h:integer);
```

Return the size of the client in *w* and *h*. If it fails, it returns -1 for both h and w.

8.7.4 GetClientPosition

```
procedure GetClientPosition(var left, top:integer);
```

Return the position of the client in *left* and *top*. May return negative values.

8.7.5 SetTargetBitmap

```
function SetTargetBitmap(Bitmap : integer): integer;
```

Set a bitmap as target / client. (The bitmap must be loaded by Simba) This can be combined with the SetPersistent-MemoryBitmap feature to achieve the same effect as SetTargetArray.

8.7.6 SetTargetArray

```
function SetTargetArray(P: Integer; w, h: integer): integer;
```

Set a target array as client data. This is generally not something you'd want to call yourself. It is mainly included for external components to allow Simba to efficiently target its memory. See the SMART source on how to do this.

8.7.7 SetEIOSTarget

```
function SetEIOSTarget(name: string; initargs: Variant): integer;
```

8.7.8 SetImageTarget

```
procedure SetImageTarget(idx: integer);
```

Set the Image target defined by index *idx* as active target. An Image target controls what data Simba performs colour (and bitmap, dtm, etc) searches on.

Both SetTargetBitmap, and SetTargetArray return a target index. Alternatively you can get the index of the current target with GetImageTarget.

8.7.9 SetKeyMouseTarget

```
procedure SetKeyMouseTarget(idx: integer);
```

Set the KeyMouse target defined by index *idx* as active target. A KeyMouse target controls how Simba moves the mouse cursor and emulates the keyboard.

8.7.10 GetImageTarget

```
function GetImageTarget: integer;
```

Returns the current Image target.

8.7.11 GetKeyMouseTarget

```
function GetKeyMouseTarget: integer;
```

Returns the current KeyMouse target.

8.7.12 ExportImageTarget

```
function ExportImageTarget : TTarget_Exported;
```

8.7.13 ExportKeyMouseTarget

```
function ExportKeyMouseTarget : TTarget_Exported;
```

8.7.14 FreeTarget

```
procedure FreeTarget(idx: integer);
```

Free a previously loaded target.

This procedure does not free the data associated with the target as in the case of SetTargetBitmap or SetTargetArray.

8.7.15 SetDesktopAsClient

```
procedure SetDesktopAsClient;
```

Set the default desktop as client.

8.7.16 ActivateClient

```
procedure ActivateClient;
```

Set the current target as active for key input.

8.7.17 IsTargetValid

```
function IsTargetValid: boolean;
```

Returns true if the current target is valid.

8.8 Finding a specific window

8.8.1 GetProcesses

```
function GetProcesses: TSysProcArr;
```

Returns processes with the title of their window, the handle of the window, the process id and their width and height.

With TSysProc being defined as:

```
TSysProc = record
    Title: string;
    Handle: integer;
    Pid: integer;
    Width, Height: integer;
end;
```

Example usage:

```
function FindAndSetTarget(TitlePrefix: String; SetAsTarget: Boolean): Boolean;
var
  T: TSysProcArr;
  I: Integer;
begin
  T:= GetProcesses();
  for I := 0 to high(T) do
    if StartsWith(TitlePrefix, T[i].Title) then
    begin
      Result := True;
      if SetAsTarget then
      begin
        SetTarget(T[i]);
        ActivateClient;
      end;
    end;
end;
```

8.9 Client Area

Client Areas were introduced to cope with clients which have a normal coordinate system, but a variable base for this coordinate system.

More specifically: client areas allow you to transparently add a certain X and Y to all the mouse and image (finding, dtm, etc) functions.

Support for Mouse and Image targets have been separated. This is required for targets that only support say, an Image target.

In this case you do not want to accidentally touch or reset the Mouse target area.

Setting an area multiple times is the same as resetting it and then setting the area. Multiple calls to *SetClientArea will not result in nested areas.

8.9.1 MouseSetClientArea

```
function MouseSetClientArea(x1, y1, x2, y2: integer): boolean;
```

Define a new Client Area for all Mouse operations on this Mouse target.

8.9.2 MouseResetClientArea

```
procedure MouseResetClientArea;
```

Reset the Client Area for the Mouse Target.

8.9.3 ImageSetClientArea

```
function ImageSetClientArea(x1, y1, x2, y2: integer): boolean;
```

Define a new Client Area for all Image operations on this Image Target.

8.9.4 ImageResetClientArea

```
procedure ImageResetClientArea;
```

Reset the Client Area for the Image Target.

8.10 Internet Functions

The internet functions in Simba range from HTTP functions and low-level socket functions. With the HTTP functions you can scrape the web, send form data to web pages.

With the socket functions you can implement virtually any network procotol: there have been several IRC bots.

8.10.1 HTTP Functions

OpenWebPage

```
procedure OpenWebPage(const url : string);
```

OpenWebPage opens the given web page (url) with your default browser.

Example:

```
OpenWebPage('http://villavu.com');
```

GetPage

```
function GetPage(const url : string): string;
```

GetPage returns a string of HTML from the given web page.

InitializeHTTPClient

```
function InitializeHTTPClient(HandleCookies: Boolean): Integer;
```

InitializeHTTPClient creates a new client and assigns it an ID. You use this for all the other web functions that require a client.

InitializeHTTPClientWrap

```
function InitializeHTTPClientWrap(HandleCookies: Boolean): Integer;
```

This should probably not be documented.

FreeHTTPClient

```
procedure FreeHTTPClient(Client: Integer);
```

Free the HTTP client returned by *InitializeHTTPClient*.

GetHTTPPage

```
function GetHTTPPage(Client: Integer;const URL: string): string;
```

GetHTTPPage is just like GetPage, except you can choose which client to get the HTTP code from.

SetHTTPUserAgent

```
procedure SetHTTPUserAgent(Client: Integer;const Agent: string);
```

SetHTTPUserAgent allows you to change the agent string of a client.

PostHTTPPage

```
function PostHTTPPage(Client: Integer;const Url,PostData: string): string;
```

PostHTTPPage requests to post data (PostData) on the web page (Url) of the client (Client).

PostHTTPPageEx

```
function PostHTTPPageEx(Client: Integer;const Url: string): string;
```

PostHTTPPageEx is just like PostHTTPPage but uses predefined post data added by ddPostVariable and cleared by ClearPostData.

ClearPostData

```
procedure ClearPostData(Client: Integer);
```

ClearPostData clears the post data added to the web page (Client). Used with PostHTTPPageEx.

AddPostVariable

```
procedure AddPostVariable(Client: Integer;const VarName, VarValue: string);
```

AddPostVariable adds a post variable to the web page (Client). Used with PostHTTPPageEx.

GetRawHeaders

```
function GetRawHeaders(Client: Integer): string;
```

GetRawHeaders returns a string of headers from the specified client.

8.10.2 Socket Functions

Simba's Socket Functions. Examples required; if you have one, please let u know.

CreateSocket

```
function CreateSocket: integer;
```

CreateSocket creates a new socket and assigns it an ID.

FreeSocket

```
procedure FreeSocket(Index: integer);
```

FreeSocket frees the socket with the ID (Index) assigned to it upon creation.

ConnectSocket

```
procedure ConnectSocket(Client: integer; IP, Port: string);
```

ConnectSocket connects the socket to an IP and port on the specified client (Client).

BindSocket

```
procedure BindSocket(Client: integer; IP, Port: string);
```

BindSocket binds a connected socket to an IP and port on the specified client (Client).

ListenSocket

```
procedure ListenSocket(Client: integer);
```

ListenSocket allows for a client socket to accept connections.

AcceptSocket

```
function AcceptSocket(Client: integer): integer;
```

AcceptSocket accepts pending connection requests to a client socket.

CloseSocket

```
procedure CloseSocket(Client: integer);
```

CloseSocket closes connections to a client socket.

RecvSocket

```
function RecvSocket(Client: integer): string;
```

RecvSocket method reads all data waiting for read.

RecvSocketStr

```
function RecvSocketStr(Client: integer): string;
```

Method waits until data string is received. This string is terminated by CR-LF characters. The resulting string is returned without this termination (CR-LF)

RecvSocketEx

```
function RecvSocketEx(Client, Length: integer): string;
```

RecvSocketEx returns received data of a specified length from a bound socket as a string.

SendSocket

```
procedure SendSocket(Client: integer; Data: string);
```

SendSocket sends a string of data to a bound client socket.

SetTimeout

```
procedure SetTimeout(Client, Time: integer);
```

SetTimeout sets a maximum amount of time for a bound client socket to wait for data from another socket. Time is in *milliseconds*.

SocketInfo

```
procedure SocketInfo(Client: integer; out IP, Port: string);
```

SocketInfo sets where a bound socket will be sending data to (out IP, out Port).

8.10.3 Generic functions

SetProxy

```
procedure SetProxy(Client : Integer; pHost, pPort : String);');
```

SetProxy configures a proxy with the given client (Client) proxy host (pHost) and port (pPort).

8.11 Bitmaps

A bitmap in Simba is simply a two dimensional *field of colours*. These colours can all be the same, or they can be of different colours. Simba features functions to create, manipulate and search for bitmaps.

The bitmaps are - just as files - represented as integer in Simba (they point to a list of bitmaps, and the value of the integer is the position in the list). So typically, when referring to bitmaps in Simba, you simply represent them as an integer:

```
var bmp, x, y: integer;
begin
    bmp := CreateBitmap(10, 10); // Create a bitmap of size (10, 10)
    if FindBitmapIn(bmp, x, y, 0, 0, 300, 300) then
      writeln('Found it!');
    FreeBitmap(bmp); // Don't forget to free it when we are done.
end;
```

Note that the previous example doesn't make a lot of sense as the bitmap has only been created and not filled with any colours, they are as of yet, undefined. You can also create bitmaps from screenshots and load them when your script starts using the *BitmapFromString* function, or simple store them as files and load them using the *LoadBitmap* function.

8.11.1 Word of caution on bitmap creation

Bitmaps in Simba are internally all instances of *TMufasaBitmap*. Scripts should generally access bitmaps using their *handle*: an integer. All functions referenced here either require a bitmap *handle* or return one.

If you want to gain more access over a specific bitmap, see the *GetMufasaBitmap* function. It is highly unrecommended to create bitmaps like this, because Simba will not free them automatically for you. (There's no problem doing it like this if you only want to perform operations on it and then free it again)

```
var bmp: TMufasaBitmap;
begin
    bmp := TMufasBitmap.Create;
end;
```

Because there is no way to get a *handle* to this bitmap; as it will not be managed by Simba internally. (All Bitmaps created by *CreateBitmap* are managed by Simba, if you don't know what this means: you generally want Simba to manage the bitmaps)

If you still want access to the TMufasaBitmap, use *GetMufasaBitmap*, described below.

8.11.2 GetMufasaBitmap

```
function GetMufasaBitmap(bmp : integer) : TMufasaBitmap;
```

Returns the *TMufasaBitmap* for the given bitmap. They both reference the same bitmap. TMufasaBitmap is a more advanced interface to bitmaps in Simba. There is no way to get a *internal* (integer) reference to a bitmap if you create it with TMufasaBitmap.Create; so the recommended way is to use *CreateBitmap* to get the integer reference/handle and then call this function to get the class reference.

```
var bmp: TMufasaBitmap;
    bmph: integer;
begin;
    bmph := CreateBitmap(100, 100);
    bmp := GetMufasaBitmap(bmph);

    bmp.SetSize(150,150); // also changes bmph, as they are the same bitmap.
end;
```

8.11.3 CreateBitmapString

```
function CreateBitmapString(bmp : integer) : string;
```

Creates a string for the given bitmap, you can use this to save a bitmap for later use, for example loading it again using *BitmapFromString*.

8.11.4 CreateBitmap

```
function CreateBitmap(w,h :integer) : integer;
```

Create a bitmap with width *h* and height *h*. Returns the reference to the created bitmap.

8.11.5 FreeBitmap

```
procedure FreeBitmap(Bmp : integer);
```

Free the bitmap. You should do this when you no longer need the bitmap. Be careful when working with bitmaps: not freeing it when you no longer need it leads to memory leaks, which will eventually make your script crash. (Unless you stop it in time, in which case Simba will free the bitmaps for you)

8.11.6 SaveBitmap

```
procedure SaveBitmap(Bmp : integer; path : string);
```

Save the given bitmap to the specified path.

8.11.7 BitmapFromString

```
function BitmapFromString(Width,Height : integer; Data : string): integer;
```

Load a bitmap from the given string. This command is usually generated with the Bitmap to String feature in Simba.

8.11.8 LoadBitmap

```
function LoadBitmap(Path : string) : integer;
```

Load a bitmap from a path to a file. Formats known to work are BMP and PNG images.

8.11.9 SetBitmapSize

```
procedure SetBitmapSize(Bmp,NewW,NewH : integer);
```

Change the size of the bitmap. Previous data will be preserved (if possible), so enlarging the bitmap won't destroy the old data, but shrinking it will inevitably destroy some data. (Everything that falls out of the new bounds)

8.11.10 GetBitmapSize

```
procedure GetBitmapSize(Bmp : integer; var BmpW,BmpH : integer);
```

Returns the size of the bitmap in *BmpW*, *BmpH*.

8.11.11 SetPersistentMemoryBitmap

```
procedure SetPersistentMemoryBitmap(bmp: integer; mem: PtrUInt; awidth, aheight: integer);
```

Set the internal bitmap *bmp* data pointer to *mem*. Size is described by *awidth* and *aheight*. *mem* should be a pointer of four bytes.

Byte order: BGRA (Blue, Green, Red, Alpha).

Previous data in the bitmap will be lost.

8.11.12 ResetPersistentMemoryBitmap

```
procedure ResetPersistentMemoryBitmap(bmp: Integer);
```

Undo the effect of SetPersistentMemoryBitmap. Data in the bitmap before the SetPersistentMemoryBitmap will not be restored.

8.11.13 StretchBitmapResize

```
procedure StretchBitmapResize(Bmp,NewW,NewH : integer);
```

8.11.14 CreateMirroredBitmap

```
function CreateMirroredBitmap(Bmp : integer) : integer;
```

8.11.15 CreateMirroredBitmapEx

```
function CreateMirroredBitmapEx(Bmp : integer; MirrorStyle : TBmpMirrorStyle) : integer;
```

8.11.16 FastSetPixel

```
procedure FastSetPixel(bmp,x,y : integer; Color : TColor);
```

Set the pixel on the bitmap at position x, y to *color*.

8.11.17 FastSetPixels

```
procedure FastSetPixels(bmp : integer; TPA : TPointArray; Colors : TIntegerArray);
```

Set the pixels on the bitmap at position TPA[index] to Colors[index].

8.11.18 FastGetPixel

```
function FastGetPixel(bmp, x,y : integer) : TColor;
```

Return the colour of pixel on the bitmap, position specified by x, y.

8.11.19 FastGetPixels

```
function FastGetPixels(Bmp : integer; TPA : TPointArray) : TIntegerArray;
```

Return an array of the colours on the bitmap; positions specified by *TPA*.

8.11.20 GetBitmapAreaColors

```
function GetBitmapAreaColors(bmp,xs, ys, xe, ye: Integer): T2DIntegerArray;
```

Returns all the colours in the area defined by (*xs*, *xy*, *xe*, *ye*) on the bitmap in a two dimensions integer array.

8.11.21 FastDrawClear

```
procedure FastDrawClear(bmp : integer; Color : TColor);
```

Draw *Color* on every pixel on the bitmap.

8.11.22 FastDrawTransparent

```
procedure FastDrawTransparent(x, y: Integer; SourceBitmap, TargetBitmap: Integer);
```

8.11.23 SetTransparentColor

```
procedure SetTransparentColor(bmp : integer; Color : TColor);
```

8.11.24 GetTransparentColor

```
function GetTransparentColor(bmp: integer) : TColor;
```

8.11.25 FastReplaceColor

```
procedure FastReplaceColor(Bmp : integer; OldColor,NewColor : TColor);
```

8.11.26 CopyClientToBitmap

```
procedure CopyClientToBitmap(bmp, xs, ys, xe, ye: Integer);
```

Copy client area *xs, ys, xe, ye* to specified bitmap.

8.11.27 BitmapFromClient

```
function BitmapFromClient(const xs, ys, xe, ye: Integer): Integer;
```

Create a bitmap from the client. Area specified by *xs, ye, xe, ye*.

8.11.28 SetBitmapName

```
procedure SetBitmapName(Bmp : integer; name : string);
```

Assign a name to the bitmap. Mainly for debugging purposes. (It will write the name of the bitmap if it hasn't been freed.)

```
program new;

var bmp: integer;
begin
  bmp := CreateBitmap(10, 10);
  SetBitmapName(bmp, 'We will not free this bitmap');
end.
// Simba will print what bitmap has not been freed (along with his long
// name)
```

8.11.29 FindBitmap

```
function FindBitmap(bitmap: integer; var x, y: Integer): Boolean;
```

Searches for the Bitmap *bmp* on the entire client. Returns true if found. If found, *x, y* specifies the position where the bitmap was found.

8.11.30 FindBitmapIn

```
function FindBitmapIn(bitmap: integer; var x, y: Integer;  xs, ys, xe, ye: Integer): Boolean;
```

Searches for the Bitmap *bmp* on the client in the area defined by *xs,ys,xe,ye*. Returns true if found. If found, *x, y* specifies the position where the bitmap was found.

8.11.31 FindBitmapToleranceIn

```
function FindBitmapToleranceIn(bitmap: integer; var x, y: Integer; xs, ys, xe, ye: Integer; toleranc
```

Searches for the Bitmap *bmp* on the client in the area defined by *xs,ys,xe,ye*. Tolerance defines the tolerance per pixel when matching bitmaps. See *Colour tolerance* for more information on tolerance. Returns true if found. If found, *x, y* specifies the position where the bitmap was found.

8.11.32 FindBitmapSpiral

```
function FindBitmapSpiral(bitmap: Integer; var x, y: Integer; xs, ys, xe, ye: Integer): Boolean;
```

Searches for the Bitmap *bmp* on the client in the area defined by *xs,ys,xe,ye*. Returns true if found. If found, *x, y* specifies the position where the bitmap was found. Search starts from a point defined by *x, y*.

8.11.33 FindBitmapsSpiralTolerance

```
function FindBitmapsSpiralTolerance(bitmap: integer; x, y: Integer; var Points : TPointArray; xs, ys,
```

Searches for the Bitmap *bmp* on the client in the area defined by *xs,ys,xe,ye*. Tolerance defines the tolerance per pixel when matching bitmaps. See *Colour tolerance* for more information on tolerance. Search starts from a point defined by *x, y*. Returns true if found. If found, each point in *TPA* specifies a match.

8.11.34 FindBitmapSpiralTolerance

```
function FindBitmapSpiralTolerance(bitmap: integer; var x, y: Integer; xs, ys, xe, ye,tolerance : int
```

Searches for the Bitmap *bmp* on the client in the area defined by *xs,ys,xe,ye*. Tolerance defines the tolerance per pixel when matching bitmaps. See *Colour tolerance* for more information on tolerance. Search starts from a point defined by *x, y*. Returns true if found. If found, *x, y* specifies the position where the bitmap was found.

8.11.35 RotateBitmap

```
function RotateBitmap(bitmap: Integer; angle: Extended): Integer;
```

8.11.36 DesaturateBitmap

```
function DesaturateBitmap(Bitmap : integer) : integer;
```

8.11.37 InvertBitmap

```
procedure InvertBitmap(Bitmap : integer);
```

8.11.38 CopyBitmap

```
function CopyBitmap(Bitmap:  integer) : integer)
```

Creates a copy of the *Bitmap*. Returns the bitmap copy.

8.11.39 GreyScaleBitmap

```
function GreyScaleBitmap(bitmap : integer) : integer
```

Creates a copy of the bitmap, greyscaled.

8.11.40 BrightnessBitmap

```
function BrightnessBitmap(Bitmap,br : integer) : integer;
```

Changes the brightness of a bitmap, intensity defined by *br*. Returns a new bitmap with the brightness applied.

If you instead want to apply brightness to the current bitmap, see *Applying a filter on the current bitmap*

8.11.41 ContrastBitmap

```
function ContrastBitmap(bitmap : integer; co : extended) : integer;
```

Changes the constrast of a bitmap, returns a new bitmap with the contrast applied.

8.11.42 PosterizeBitmap

```
function PosterizeBitmap(Bitmap : integer; po : integer) : integer;
```

Posterizes a bitmap, intensity defined by *po*; returns a new bitmap with the posterisation applied.

Applying a filter on the current bitmap

```
var b: integer;
begin
    // Dummy bitmap. You'll want something that's not just a blank bitmap.
    B:=CreateBitmap(100,100);

    // Apply the filter (Posterize in this case) without making a copy.
    GetMufasaBitmap(b).Posterize(GetMufasaBitmap(b), 10);

    // Always free your bitmaps when you no longer use them. :)
    FreeBitmap(b);
end.
```

8.11.43 CreateMaskFromBitmap

```
function CreateMaskFromBitmap(Bitmap : integer) : TMask;
```

8.11.44 FindMaskTolerance

```
function FindMaskTolerance(const mask: TMask; var x, y: Integer; xs,ys, xe, ye: Integer; Tolerance, C
```

8.11.45 FindBitmapMaskTolerance

```
function FindBitmapMaskTolerance(mask: Integer; var x, y: Integer; xs, ys, xe, ye: Integer; Tolerance
```

8.11.46 FindDeformedBitmapToleranceIn

```
function FindDeformedBitmapToleranceIn(bitmap: integer; var x,y: Integer; xs, ys, xe, ye: Integer; to
```

8.11.47 DrawTPABitmap

```
procedure DrawTPABitmap(bitmap: integer; TPA: TPointArray; Color: integer);
```

Draws a TPointArray on a bitmap. Each point in the TPointArray is *painted* on the bitmap by setting the pixel on the bitmap (position defined by tpa point) to *color*.

8.11.48 DrawATPABitmap

```
procedure DrawATPABitmap(bitmap: integer; ATPA: T2DPointArray);
```

Draws a Array of TPointArray on a bitmap. Each point in the TPointArray is *painted* on the bitmap by setting the pixel on the bitmap (position defined by tpa point) to a color. Colors differ per TPointArray (group).

8.11.49 DrawATPABitmapEx

```
procedure DrawATPABitmapEx(bitmap: integer; ATPA: T2DPointArray; Colors: TIntegerArray);
```

Draws a Array of TPointArray on a bitmap. Each point in the TPointArray is *painted* on the bitmap by setting the pixel on the bitmap (position defined by tpa point) to a color. Colors are defined by *Colors*.

8.11.50 DrawBitmap

```
procedure DrawBitmap(Bmp: Integer; Dest: TCanvas; x, y: Integer);
```

Draw the bitmap to a TCanvas.

8.11.51 RectangleBitmap

```
procedure RectangleBitmap(bitmap : integer; const box : TBox; Color : TColor);
```

8.11.52 FloodFillBitmap

```
procedure FloodFillBitmap(bitmap : integer; const StartPoint : TPoint; const SearchCol,ReplaceCol : T
```

8.11.53 CalculatePixelShift

```
function CalculatePixelShift(Bmp1,Bmp2 : Integer; CompareBox : TBox) : integer;
```

8.11.54 CalculatePixelTolerance

```
function CalculatePixelTolerance(Bmp1,Bmp2 : Integer; CompareBox : TBox; CTS : integer) : extended;'
```

8.11.55 CropBitmap

```
procedure CropBitmap(const bmp: integer; const xs, ys, xe, ye: integer);
```

Crops the bitmap, removes all points outside (xs, ys, xe, ye).

```
var
  bmp: integer;
begin;
  bmp := BitmapFromClient(0, 0, 50, 50);
  CropBitmap(bmp, 10, 10, 40, 40);
end;
```

8.11.56 GetColorsBitmap

```
function GetColorsBitmap(const bmp: integer): TIntegerArray;
```

Returns a TIntegerArray of all colours on the bitmap, the result will be sorted row by row.

```
var
  bmp, i: integer;
  arr: TIntegerArray;
begin;
  bmp := BitmapFromClient(0, 0, 10, 10);
  arr := GetColorsBitmap(bmp);

  for i := 0 to high(arr) do
    writeln(intToStr(arr[i]));
end;
```

8.11.57 BitmapToMatrix

```
function BitmapToMatrix(const bmp: integer): T2DIntegerArray;
```

Returns a two dimensions integer array of all the colors on the bitmap.

```
var
 bmp, x, y: integer;
 matrix: T2DIntegerArray;
begin
  bmp := BitmapFromClient(0, 0, 10, 10);
  matrix := BitmapToMatrix(bmp);

  for y := 0 to 10 do
    for x := 0 to 10 do
      writeln(matrix[y][x]);
end;
```

8.11.58 DrawMatrixBitmap

```
procedure DrawMatrixBitmap(const bmp: integer; const matrix: T2DIntegerArray);
```

Draws a matrix onto the bitmap.

```
DrawMatrixBitmap(bmp, matrix);
```

8.11.59 ThresholdAdaptiveBitmap

```
procedure ThresholdAdaptiveBitmap(const bmp: integer; Alpha, Beta: Byte; Invert: Boolean; Method: TBr
```

Applys a ThresholdAdaptive filter onto the bitmap. Vaild TBmpThreshMethods are (TM_Mean, TM_MinMax);

```
ThresholdAdaptiveBitmap(bmp, 0, 255, false, TM_Mean, 0);
```

8.11.60 ThresholdAdaptiveMatrix

```
procedure ThresholdAdaptiveMatrix(var Matrix: T2DIntegerArray; Alpha, Beta: Byte; Invert: Boolean; Me
```

Applys a ThresholdAdaptive filter onto a image matrix (created by BitmapToMatrix). Vaild TBmpThreshMethods are (TM_Mean, TM_MinMax);

```
ThresholdAdaptiveBitmap(matrix, 0, 255, false, TM_Mean, 0);
```

8.11.61 ResizeBilinearMatrix

```
procedure ResizeBilinearMatrix(var Matrix: T2DIntegerArray; NewW, NewH: Integer);
```

Uses the Bilinear resize method to resize a image matrix (created by BitmapToMatrix) to width *newW* and height *newH*.

```
ResizeBilinearMatrix(matrix, 500, 500);
```

8.12 SQLite Functions

Simba has support for reading and manipulating SQLite databases. Open connections are all represented by integers in scripts. The integers point to an index in an internal array of pointers which is managed by Simba. sqlite_open and function sqlite_open_v2 return an integer that you use for most other functions. This page documents only the functions, and not SQLite or the SQL language. After opening a connection, you should use sqlite_close on it when you are no longer using it. If, however, for some reason you forget, Simba will free all unfreed connections automatically.

8.12.1 sqlite_open

```
function sqlite_open(filename : string) : integer;
```

Will try to open a connection to the database file specified and returns an index to the internal array. If the database file does not exist, it will attempt to create it. This will return -1 if no connection could be established.

Example:

```
DB := sqlite_open('test.db'); // DB would be an integer that you pass as the index parameter to the
```

8.12.2 sqlite_open_v2

```
function sqlite_open_v2(filename : string; flags : integer) : integer;
```

Does the same as sqlite_open however you can provide flags to use when opening.

Open flags:

```
const
  SQLITE_OPEN_READONLY = 1;
  SQLITE_OPEN_READWRITE = 2;
  SQLITE_OPEN_CREATE = 4;
  SQLITE_OPEN_URI = 40;
  SQLITE_OPEN_NOMUTEX = 8000;
  SQLITE_OPEN_FULLMUTEX = 10000;
  SQLITE_OPEN_SHAREDCACHE = 20000;
  SQLITE_OPEN_PRIVATECACHE = 40000;
```

Example:

```
DB := sqlite_open_v2('test.db', SQLITE_OPEN_READONLY); // Open as read only, will not create file.
DB := sqlite_open_v2('test.db', SQLITE_OPEN_READWRITE or SQLITE_OPEN_CREATE); // Open as read write,
```

8.12.3 sqlite_version

```
function sqlite_version() : string;
```

Returns the version of the loaded SQLite library expressed as a string (x.y.z).

```
Writeln(sqlite_version()); // Outputs 3.7.10 for me
```

8.12.4 sqlite_version_num

```
function sqlite_version_num() : integer;
```

Returns the version of the loaded SQLite library expressed as an integer (x * 1000000 + y * 1000 + x).

Example:

```
Writeln(sqlite_version_num()); // Outputs 3007010 for me
```

8.12.5 sqlite_query

```
function sqlite_query(index : integer; sql : string) : boolean;
```

Attempts to execute a query on the database handle specified by index. Returns true if SQLITE_OK is returned by SQLite. If it returns false, it is useful to see what sqlite_errMsg outputs.

Example:

```
sqlite_query(DB, 'CREATE TABLE test (id INTEGER PRIMARY KEY AUTOINCREMENT, name VARCHAR(50) UNIQUE NO
sqlite_query(DB, 'INSERT INTO test (name) VALUES (''Sex'');');
```

8.12.6 sqlite_queryValue

```
function sqlite_queryValue(index : integer; sql : string; out results : T2DStringArray) : boolean;
```

Attempts to execute a query on the database handle specified by index. Return true if SQLITE_OK is returned by SQLite. This will also save the resulting rows in the Results variable provided. The first array will be an array containing column names. If it returns false, it is useful to see what sqlite_errMsg outputs.

Example:

```
sqlite_queryValue(DB, 'SELECT * FROM test;', Results);
Writeln(Results); // Should output [['id', 'name'], ['1', 'Sex']]
```

8.12.7 sqlite_queryResult

```
function sqlite_queryResult(index : integer; sql : string; var error : boolean) : T2DStringArray;
```

Attempts to execute a query on the database handle specified by index. The resulting rows are returned. If an error occurred during the query, the error boolean will be set to true. Otherwise, it will be false.

Example:

```
Results := sqlite_queryResult(DB, 'SELECT * FROM test;', error);
if error then
  [...] // do your error handling here...
Writeln(Results); // Should output [['id', 'name'], ['1', 'Sex']]
```

8.12.8 sqlite_escape

```
function sqlite_escape(s : string): string;
```

Sanitizes a string for input into the database by replacing apostrophes with anothe apostrophe. It will return the escaped string.

Example:

```
Writeln(sqlite_escape('foo '' or 1=1')); // Outputs foo '' or 1=1. Note that it looks as I inputted
```

8.12.9 sqlite_close

```
procedure sqlite_close(index : integer);
```

Closes the database handle specified by index (removing file locks, etc.). Don't forget to use this when you're done working on a database!

Example:

```
DB := sqlite_open('test.db');
// [...]
sqlite_close(DB);
```

8.12.10 sqlite_errMsg

```
function sqlite_errMsg(index : integer) : string;
```

Returns the error message returned by the last SQLite library call. You must provide an index to a database handle. If no error has occurred, this will return 'not an error'.

Example:

```
sqlite_query(DB, 'asdfghjkl');
Writeln(sqlite_errmsg(DB)); // near "asdfghjkl": syntax error
```

8.12.11 sqlite_errCode

```
function sqlite_errCode(index : integer) : integer;
```

Returns the result code returned by the last SQLite library call. You must provide an index to a database handle. If no error has occurred, this will return SQLITE_OK.

Result codes:

```
const
  SQLITE_OK = 0; // Successful result
  SQLITE_ERROR = 1; // SQL error or missing database
  SQLITE_INTERNAL = 2; // Internal logic error in SQLite
  SQLITE_PERM = 3; // Access permission denied
  SQLITE_ABORT = 4; // Callback routine requested an abort
  SQLITE_BUSY = 5; // The database file is locked
  SQLITE_LOCKED = 6; // A table in the database is locked
  SQLITE_NOMEM = 7; // A malloc() failed
  SQLITE_READONLY = 8; // Attempt to write a readonly database
  SQLITE_INTERRUPT = 9; // Operation terminated by sqlite3_interrupt()
  SQLITE_IOERR = 10; // Some kind of disk I/O error occurred
  SQLITE_CORRUPT = 11; // The database disk image is malformed
  SQLITE_NOTFOUND := 12; // Unknown opcode in sqlite3_file_control()
  SQLITE_FULL := 13; // Insertion failed because database is full
  SQLITE_CANTOPEN := 14; // Unable to open the database file
  SQLITE_PROTOCOL = 15; // Database lock protocol error
  SQLITE_EMPTY = 16; // Database is empty
  SQLITE_SCHEMA = 17; // The database schema changed
  SQLITE_TOOBIG = 18; // String or BLOB exceeds size limit
  SQLITE_CONSTRAINT = 19; // Abort due to constraint violation
  SQLITE_MISMATCH = 20; // Data type mismatch
  SQLITE_MISUSE = 21; // Library used incorrectly
  SQLITE_NOLFS = 22; // Uses OS features not supported on host
  SQLITE_AUTH = 23; // Authorization denied
  SQLITE_FORMAT = 24; // Auxiliary database format error
  SQLITE_RANGE = 25; // 2nd parameter to sqlite3_bind out of range
  SQLITE_NOTADB = 26; // File opened that is not a database file
  SQLITE_ROW = 27; // sqlite3_step() has another row ready
  SQLITE_DONE = 28; // sqlite3_step() has finished executing
```

Example:

```
sqlite_query(DB, 'asdfghjkl');
Writeln(sqlite_errmsg(DB)); // 1 (SQLITE_ERROR)
```

8.13 String Functions

8.13.1 Capitalize

```
function Capitalize(str : string) : string;
```

8.13.2 CompressString

```
function CompressString(const Str : string) : string;
```

8.13.3 DecompressString

```
function DecompressString(const Compressed : string) : string;
```

8.13.4 Base64Encode

```
function Base64Encode(const str : string) : string;
```

8.13.5 Base64Decode

```
function Base64Decode(const str : string) : string;
```

8.13.6 Format

```
function Format(const fmt : string;const args : array of const) : string;
```

8.13.7 ToStr

```
function ToStr(x) : string;
```

8.13.8 StringMatch

```
function StringMatch(checkCompare, goalCompare: string): extended;
```

Uses levenshtein distance to work out the match % of the two strings.

8.13.9 Between

```
function Between(s1, s2, str: string): string;
```

8.13.10 MultiBetween

```
function MultiBetween(str, s1, s2: string): TStringArray;
```

Splits a string into an array of strings by giving it an begin and an end tag. Useful for data reading

8.13.11 IntToStr

```
function IntToStr(value: Integer): String;
```

8.13.12 FloatToStr

```
function FloatToStr(value: Extended): String;
```

8.13.13 BoolToStr

```
function BoolToStr(value: Boolean): String;
```

8.13.14 StrToInt

```
function StrToInt(value: String): Integer;
```

8.13.15 StrToIntDef

```
function StrToIntDef(value: String; default: Integer): Integer;
```

8.13.16 StrToFloat

```
function StrToFloat(value: String): Extended;
```

8.13.17 StrToFloatDef

```
function StrToFloatDef(value: String; default: Extended): Extended;
```

8.13.18 StrToBool

```
function StrToBool(value: String): Boolean;
```

8.13.19 StrToBoolDef

```
function StrToBoolDef(value: String; default: Boolean): Boolean;
```

8.13.20 ExtractFromStr

```
function ExtractFromStr( Str : string; Extract : StrExtr) : string;
```

8.13.21 Replace

```
function Replace(Text, FindStr, ReplaceStr: string; Flags: TReplaceFlags): string;
```

8.13.22 ReplaceWrap

```
function ReplaceWrap(Text, FindStr, ReplaceStr: string; Flags: TReplaceFlags): string;
```

8.13.23 Implode

```
function Implode(Glue: string; Pieces: TStringArray): string;
```

8.13.24 Explode

```
function Explode(del, str: string): TStringArray;
```

8.13.25 ExplodeWrap

```
procedure ExplodeWrap(del, str: string; var res : TStringArray);
```

8.13.26 Padl

```
function Padl(s: String; i: longInt): String;
```

8.13.27 Padz

```
function Padz(s: String; i: longInt): String;
```

8.13.28 Padr

```
function Padr(s: String; i: longInt): String;
```

8.13.29 ExecRegExpr

```
function ExecRegExpr( const RegExpr, InputStr : String) : boolean;
```

8.13.30 SplitRegExpr

```
procedure SplitRegExpr( const RegExpr, InputStr : String; Pieces : TStrings);
```

8.13.31 ReplaceRegExpr

```
function ReplaceRegExpr( const RegExpr, InputStr, ReplaceStr : String; UseSubstitution : boolean) : S
```

8.13.32 PosEx

```
function PosEx(needle, haystack: String; offset: integer): integer;
```

8.13.33 IsArrInStr

```
function IsArrInStr(strArr: TStringArray; s: string): boolean;
```

Returns true if any of the indexes in TStringArray StrArr is found in string s

8.14 TPointArray Functions

8.14.1 Quicksort

```
procedure Quicksort(var Arr : TIntegerArray);
```

Sorts a TIntegerArray using the Quicksort algorithm

8.14.2 tSwap

```
procedure tSwap(var a, b: TPoint);
```

Swaps the values of a and b around

The following example switches point P with point P2. You can run the example for a demonstration. It will print before the swap and after the swap to illustrate the change.

```
program tSwapExample;

var P, P2: Tpoint;

begin
  P := Point(111, 111);
  P2 := Point(222, 222);
  writeln('P := ' + tostr(P));
  writeln('P2 := ' + tostr(P2));
  tSwap(P, P2);
  writeln('P := ' + tostr(P));
  writeln('P2 := ' + tostr(P2));
end.
```

8.14.3 tpaSwap

```
procedure tpaSwap(var a, b: TPointArray);
```

Swaps the values of a and b around

8.14.4 SwapE

```
procedure SwapE(var a, b: Extended);
```

Swaps the values of a and b around

8.14.5 RAaSTPAEx

```
procedure RAaSTPAEx(var a: TPointArray; const w, h: Integer);
```

Leaves one point per box with side lengths W and H to the TPA

8.14.6 RAaSTPA

```
procedure RAaSTPA(var a: TPointArray; const Dist: Integer);
```

Leaves one point per box with the side length Dist

8.14.7 NearbyPointInArrayEx

```
function NearbyPointInArrayEx(const P: TPoint; w, h:Integer;const  a: TPointArray): Boolean;
```

Returns true if the point P is near a point in the TPA a with the

8.14.8 NearbyPointInArray

```
function NearbyPointInArray(const P: TPoint; Dist:Integer;const  a: TPointArray): Boolean;
```

Returns true if the point P is near a point in the TPA a with the

8.14.9 QuickTPASort

```
procedure QuickTPASort(var A: TIntegerArray; var B: TPointArray; iLo, iHi: Integer; SortUp: Boolean)
```

8.14.10 QuickATPASort

```
procedure QuickATPASort(var A: TIntegerArray; var B: T2DPointArray; iLo, iHi: Integer; SortUp: Boolea
```

8.14.11 SortTPAFrom

```
procedure SortTPAFrom(var a: TPointArray; const From: TPoint);
```

Sorts the TPA a from the TPoint From

8.14.12 SortATPAFrom

```
procedure SortATPAFrom(var a: T2DPointArray; const From: TPoint);
```

Loops though each index of the T2DPointArray sorting each tpa from the TPoint

8.14.13 SortATPAFromFirstPoint

```
procedure SortATPAFromFirstPoint(var a: T2DPointArray; const From: TPoint);
```

Sorts the T2DPointArray from the TPoint, based from the first point in each TPointArray

8.14.14 SortATPAFromMidPoint

```
procedure SortATPAFromMidPoint(var a: T2DPointArray; const From: TPoint);
```

Sorts the T2DPointArray from the TPoint, based from the middle point in each TPointArray

8.14.15 SortATPAFromFirstPointX

```
procedure SortATPAFromFirstPointX(var a: T2DPointArray; const From: TPoint);
```

Sorts the T2DPointArray from the TPoint, based from the first point X value in each TPointArray

8.14.16 SortATPAFromFirstPointY

```
procedure SortATPAFromFirstPointY(var a: T2DPointArray; const From: TPoint);
```

Sorts the T2DPointArray from the TPoint, based from the first point Y value in each TPointArray

8.14.17 InvertTPA

```
procedure InvertTPA(var a: TPointArray);
```

Reverses the TPA

8.14.18 InvertATPA

```
procedure InvertATPA(var a: T2DPointArray);
```

Reverses the T2dPointArray

8.14.19 MiddleTPAEx

```
function MiddleTPAEx(const TPA: TPointArray; var x, y: Integer): Boolean;
```

Stores the middle point from the TPA in x and y

The following example will find the Colors and make a TPA then give you the x and y coordinates for the middle of the TPA.

```
program MiddleTPAExExample;

var
  TPA: TPointArray;
  x, y: Integer;

begin
  findcolors(TPA, 205, 0, 0, 100, 100);
  MiddleTPAEx(TPA, x, y);
  Mouse(x, y, 0, 0, 1);
end.
```

8.14.20 MiddleTPA

```
function MiddleTPA(const tpa: TPointArray): TPoint;
```

Returns the middle TPA in the result

The following example like the previous one gives you the coordinates for the Middle of the TPA, but it returns it with the result being a TPoint.

```
program MiddleTPAExample;

var
  TPA: TPointArray;
  P: TPoint;

begin
  findcolors(TPA, 205, 0, 0, 100, 100);
  P := MiddleTPAEx(TPA);
  Mouse(P.x, P.y, 0, 0, 1);
end.
```

8.14.21 MedianTPAEx

```
procedure MedianTPAEx(var tpa: TPointArray; out x, y: integer);
```

Finds the point closest to the middle of the TPointArray, returns the TPoint in parameter x and y.

8.14.22 MedianTPA

```
function MedianTPA(var tpa: TPointArray): TPoint;
```

Returns the point closest to the middle of the TPointArray.

8.14.23 SortATPASize

```
procedure SortATPASize(var a: T2DPointArray; const BigFirst: Boolean);
```

Sorts the T2dPointArray from largest to smallest if BigFirst is true or smallest to largest if BigFirst is false

The following Example Sorts the ATPA from largest to smallest.

```
program SortATPASizeExample;

var
  TPA: TPointArray;
  P: TPoint;

begin
  findcolors(TPA, 205, 0, 0, 100, 100);
  ATPA := TPAtoATPA(TPA, 10);
  SortATPASize(ATPA, true);
end.
```

8.14.24 SortATPAFromSize

```
procedure SortATPAFromSize(var a: T2DPointArray; const Size: Integer; CloseFirst: Boolean);
```

Sorts the T2DPointArray from Size by the closest first if CloseFirst is true

8.14.25 FilterTPAsBetween

```
procedure FilterTPAsBetween(var atpa: T2DPointArray; const minLength, maxLength: integer);
```

Loops though each index of the T2DPointArray, removing the TPointArrays if their length falls beetween minLength and MaxLength.

8.14.26 InIntArrayEx

```
function InIntArrayEx(const a: TIntegerArray; var Where: Integer; const Number: Integer): Boolean;
```

Returns true if Number was found in the TIntegerArray a and returns its location in Where

8.14.27 InIntArray

```
function InIntArray(const a: TIntegerArray; Number: Integer): Boolean;
```

Returns true if Number is found in the TintegerArray a

8.14.28 ClearSameIntegers

```
procedure ClearSameIntegers(var a: TIntegerArray);
```

Deletes the indexes in the TintegerArray a which are duplicated

8.14.29 ClearSameIntegersAndTPA

```
procedure ClearSameIntegersAndTPA(var a: TIntegerArray; var p: TPointArray);
```

Deletes the indexes in the TIntegerArray a and TPointArray p which are duplicated

8.14.30 SplitTPAEx

```
function SplitTPAEx(const arr: TPointArray; w, h: Integer): T2DPointArray;
```

Splits the points with max X and Y distances W and H to their own TPointArrays;

8.14.31 SplitTPA

```
function SplitTPA(const arr: TPointArray; Dist: Integer): T2DPointArray;
```

Splits the points with max distance Dist to their own TPointArrays

8.14.32 ClusterTPAEx

```
function ClusterTPAEx(const TPA: TPointArray; width, height: Integer): T2DPointArray;
```

Splits the points to their own TPointArrays if they fall outside of ‘width’ and ‘height’ bounds. An alternative to SplitTPAEx, will be extremely fast compared to SplitTPAEx with a width/height less than 100.

8.14.33 ClusterTPA

```
function ClusterTPA(const TPA: TPointArray; dist: Extended): T2DPointArray;
```

Splits the points with max distance ‘dist’ to their own TPointArrays. An alternative to SplitTPA, will be extremely fast compared to SplitTPA with a distance less than 100.

8.14.34 FloodFillTPA

```
function FloodFillTPA(const TPA : TPointArray) : T2DPointArray;
```

8.14.35 FilterPointsPie

```
procedure FilterPointsPie(var Points: TPointArray; const SD, ED, MinR, MaxR: Extended; Mx, My: Intege
```

Removes the points that are in the TPointArray Points that are not within the the degrees SD (Strat Degrees) and
ED (End Degrees) and the radius’ MinR (Min Radius) and MaxR (Max Radius) from the origin Mx and My

8.14.36 FilterPointsLine

```
procedure FilterPointsLine(var Points: TPointArray; Radial: Extended; Radius, MX, MY: Integer);
```

Returns the result in the TPointArray Points. Returns the points from the TPointArray Points that are on the line Radial from the center mx, my that is with the radius Radius

8.14.37 FilterPointsDist

```
procedure FilterPointsDist(var Points: TPointArray; const MinDist, MaxDist: Extended; Mx, My: Intege
```

Removes the points from the TPointArray Points that are not within the radius MinDist (Min Distance) and MaxDist
from the origin Mx and My

8.14.38 FilterPointsBox

```
procedure FilterPointsBox(var points: TPointArray; x1, y1, x2, y2: integer);
```

Removes the points from the TPointArray that are not within the bounds of the box.

8.14.39 GetATPABounds

```
function GetATPABounds(const ATPA: T2DPointArray): TBox;
```

Returns the boundaries of the T2DPointArray ATPA as a TBox

8.14.40 GetTPABounds

```
function GetTPABounds(const TPA: TPointArray): TBox;
```

Returns the boundaries of the TPointArray TPA as a TBox

8.14.41 FindTPAinTPA

```
function FindTPAinTPA(const SearchTPA, TotalTPA: TPointArray; var Matches: TPointArray): Boolean;
```

Looks for the TPoints from SearchTPA inside TotalTPA and stores the matches inside the TPointArray Matches

8.14.42 GetSamePointsATPA

```
function GetSamePointsATPA(const  ATPA : T2DPointArray; var Matches : TPointArray) : boolean;
```

Finds duplicate Points inside the T2DPointArray ATPA and stores the results inside the TPointArray Matches

8.14.43 FindTextTPAinTPA

```
function FindTextTPAinTPA(Height : integer;const  SearchTPA, TotalTPA: TPointArray; var Matches: TPoi
```

Looks for the TPoints from SearchTPA inside TotalTPA with a maximum y distance of Height and stores the matches inside the TPointArray Matches

8.14.44 SortCircleWise

```
procedure SortCircleWise(var tpa: TPointArray; const cx, cy, StartDegree: Integer; SortUp, ClockWise
```

Sorts the TPointArray tpa from the point cx, cy if Sortup is true. Starting at StartDegree going clockwise if Clockwise is True

8.14.45 LinearSort

```
procedure LinearSort(var tpa: TPointArray; cx, cy, sd: Integer; SortUp: Boolean);
```

Sorts the TPointArray tpa from cx, cy if Sortup is true on the degree angle sd

8.14.46 RotatePoint

```
function RotatePoint(Const p: TPoint; angle, mx, my: Extended): TPoint;
```

Rotates the TPoint p around the center mx, my with the angle

8.14.47 ChangeDistPT

```
function ChangeDistPT(const PT : TPoint; mx,my : integer; newdist : extended) : TPoint;
```

Returns a TPoint with the distance newdist from the point mx, my based on the position of the TPoint TP

8.14.48 ChangeDistTPA

```
function ChangeDistTPA(var TPA : TPointArray; mx,my : integer; newdist : extended) : boolean;
```

Returns the result in the TPointArray TPA with the distance newdist from mx, my based on the current position TPA

8.14.49 FindGapsTPA

```
function FindGapsTPA(const TPA: TPointArray; MinPixels: Integer): T2DPointArray;
```

Finds the possible gaps in the TPointArray TPA and results the gaps as a T2DPointArray. Considers as a gap if the gap length is >= MinPixels

8.14.50 RemoveDistTPointArray

```
function RemoveDistTPointArray(x, y, dist: Integer;const  ThePoints: TPointArray; RemoveHigher: Boole
```

Finds the possible gaps in the TPointArray TPA and removes the gaps. Considers as a gap if the gap length is >= MinPixels

8.14.51 CombineTPA

```
function CombineTPA(const Ar1, Ar2: TPointArray): TPointArray;
```

Attaches the TPointArray Ar2 onto the end of Ar1 and returns it as the result

8.14.52 ReArrangeandShortenArrayEx

```
function ReArrangeandShortenArrayEx(const a: TPointArray; w, h: Integer): TPointArray;
```

Results the TPointArray a with one point per box with side lengths W and H left

8.14.53 ReArrangeandShortenArray

```
function ReArrangeandShortenArray(const a: TPointArray; Dist: Integer): TPointArray;
```

Results the TPointArray a with one point per box with side length Dist left

8.14.54 TPAtoATPAEx

```
function TPAtoATPAEx(const TPA: TPointArray; w, h: Integer): T2DPointArray;
```

Splits the TPA to boxes with sidelengths W and H and results them as a T2DPointArray

8.14.55 TPAtoATPA

```
function TPAtoATPA(const TPA: TPointArray; Dist: Integer): T2DPointArray;
```

Splits the TPA to boxes with sidelength Dist and results them as a T2DPointArray

8.14.56 CombineIntArray

```
function CombineIntArray(const Ar1, Ar2: TIntegerArray): TIntegerArray;
```

Attaches the TIntegerArray Ar2 onto the end of Ar1 and returns it as the result

8.14.57 MergeATPA

```
function MergeATPA(const ATPA : T2DPointArray)  : TPointArray;
```

Combines all the TPointArrays from the T2DPointArray ATPA into the result

8.14.58 AppendTPA

```
procedure AppendTPA(var TPA: TPointArray; const ToAppend: TPointArray);
```

Attaches the TPointArray ToAppend onto the end of TPA

8.14.59 TPAFromLine

```
function TPAFromLine(const x1, y1, x2, y2: Integer): TPointArray;
```

Returns a TPointArray of a line specified by the end points x1,y1 and x2,y2.

8.14.60 EdgeFromBox

```
function EdgeFromBox(const Box: TBox): TPointArray;
```

Creates a TPointArray from the edge of the TBox box

8.14.61 TPAFromBox

```
function TPAFromBox(const Box : TBox) : TPointArray;
```

Create a TPointArray from the top left and the bottom right of the TBox Box

8.14.62 TPAFromEllipse

```
function TPAFromEllipse(const CX, CY, XRadius, YRadius : Integer) : TPointArray;
```

Creates and returns a TPointArray of the outline of a ellipse

8.14.63 TPAFromCircle

```
function TPAFromCircle(const CX, CY, Radius : Integer) : TPointArray;
```

Creates and returns a TPointArray of a circle, around the center point (CX, CY), with the size determined by Radius

8.14.64 TPAFromPolygon

```
function TPAFromPolygon(const shape: TPointArray) : TPointArray;
```

Returns polygon as a TPointArray from a shape, which can be working either as an array of main points OR border points. note: The order of the points are important.

```
program TPAFromPolygonExample;

var
  tpa, shape: TPointArray;
  bmp: integer;

begin
```

```
  tpa := [point(70,90), point(185,90), point(185,116), point(70,116),
          point(70,140), point(35,105), point(70,70)];

  shape := TPAFromPolygon(tpa);

  bmp := createBitmap(230, 200);
  drawTPABitmap(bmp, shape, 255);

  displayDebugImgWindow(230, 200);
  drawBitmapDebugImg(bmp);
end.
```

8.14.65 FillEllipse

```
procedure FillEllipse(var a: TPointArray);
```

Fills a ellipse, suggested to be used with TPAFromEllipse or TPAFromCircle

8.14.66 RotatePoints

```
function RotatePoints(Const P: TPointArray; A, cx, cy: Extended): TPointArray ;
```

Rotates the TPointArray P around the center cx, cy with the angle a

8.14.67 FindTPAEdges

```
function FindTPAEdges(const p: TPointArray): TPointArray;
```

Returns a TPointArray of the edge points of the TPointArray p

8.14.68 ClearTPAFromTPA

```
function ClearTPAFromTPA(const arP, ClearPoints: TPointArray): TPointArray;
```

Removes the points in TPointArray ClearPoints from arP

8.14.69 ReturnPointsNotInTPA

```
function ReturnPointsNotInTPA(Const TotalTPA: TPointArray; const Box: TBox): TPointArray;
```

All the points from the TPointArray TotalTPA that are not in the TBox Box are returned in the TPointArray Res

8.14.70 PointInTPA

```
function PointInTPA(p: TPoint;const  arP: TPointArray): Boolean;
```

Returns true if the TPoint p is found in the TPointArray arP

8.14.71 ClearDoubleTPA

```
procedure ClearDoubleTPA(var TPA: TPointArray);
```

Deletes duplicate TPAs in the TPointArray TPA

8.14.72 TPACountSort

```
procedure TPACountSort(Var TPA: TPointArray;const max: TPoint;Const SortOnX : Boolean);
```

8.14.73 TPACountSortBase

```
procedure TPACountSortBase(Var TPA: TPointArray;const maxx, base: TPoint; const SortOnX : Boolean);
```

8.14.74 InvertTIA

```
procedure InvertTIA(var tI: TIntegerArray);
```

Reverses the TIntegerArray tI

8.14.75 SumIntegerArray

```
function SumIntegerArray(const Ints : TIntegerArray): Integer;
```

Retuns the sum of all the integers in the TIntegerArray Ints

8.14.76 AverageTIA

```
function AverageTIA(const tI: TIntegerArray): Integer;
```

Gives an average of the sum of the integers in the TIntegerArray tI

8.14.77 AverageExtended

```
function AverageExtended(const tE: TExtendedArray): Extended;
```

Gives an average of the sum of the extendeds in the TExtendedArray tI

8.14.78 SplitTPAExWrap

```
procedure SplitTPAExWrap(const arr: TPointArray; w, h: Integer; var res : T2DPointArray);
```

Splits the points with max X and Y distances W and H to their and returns the result in the T2DPointArray Res

8.14.79 SplitTPAWrap

```
procedure SplitTPAWrap(const arr: TPointArray; Dist: Integer; var res: T2DPointArray);
```

Splits the points with max distance Dist to their own TPointArrays and returns the result in the T2DPointArray Res

8.14.80 FindGapsTPAWrap

```
procedure FindGapsTPAWrap(const TPA: TPointArray; MinPixels: Integer; var Res : T2DPointArray);
```

Finds the possible gaps in the TPointArray TPA and the result is returned in the T2DPointArray Res. Considers as a gap if the gap length is >= MinPixels

8.14.81 RemoveDistTPointArrayWrap

```
procedure RemoveDistTPointArrayWrap(x, y, dist: Integer;const  ThePoints: TPointArray; RemoveHigher:
```

Finds the possible gaps in the TPointArray TPA and removes the gaps. Considers as a gap if the gap length is >= MinPixels and returns the result in the TPointArray Res

8.14.82 CombineTPAWrap

```
procedure CombineTPAWrap(const Ar1, Ar2: TPointArray; var Res :  TPointArray);
```

Attaches the TPointArray Ar2 onto the end of Ar1 and returns the result in the TPointArray Res

8.14.83 ReArrangeandShortenArrayExWrap

```
procedure ReArrangeandShortenArrayExWrap(const a: TPointArray; w, h: Integer; var Res :  TPointArray
```

Results the TPointArray a with one point per box with side lengths W and H left and puts the result in Res

8.14.84 ReArrangeandShortenArrayWrap

```
procedure ReArrangeandShortenArrayWrap(const a: TPointArray; Dist: Integer; var Res :  TPointArray);
```

Results the TPointArray a with one point per box with side length Dist left and puts the result in Res

8.14.85 TPAtoATPAExWrap

```
procedure TPAtoATPAExWrap(const TPA: TPointArray; w, h: Integer; var Res :  T2DPointArray);
```

Splits the TPA to boxes with sidelengths W and H and results them as a T2DPointArray in Res

8.14.86 TPAtoATPAWrap

```
procedure TPAtoATPAWrap(const TPA: TPointArray; Dist: Integer; var Res :  T2DPointArray);
```

Splits the TPA to boxes with sidelength Dist and results them as a T2DPointArray in Res

8.14.87 CombineIntArrayWrap

```
procedure CombineIntArrayWrap(const Ar1, Ar2: TIntegerArray; var Res :  TIntegerArray);
```

Attaches the TIntegerArray Ar2 onto the end of Ar1 and returns it in the TIntegerArray Res

8.14.88 ReturnPointsNotInTPAWrap

```
procedure ReturnPointsNotInTPAWrap(Const TotalTPA: TPointArray; const Box: TBox; var Res :  TPointArr
```

All the points from the TPointArray TotalTPA that are not in the TBox Box are returned in the TPointArray Res

8.14.89 MergeATPAWrap

```
procedure MergeATPAWrap(const ATPA : T2DPointArray; var Res: TPointArray);
```

Combines all the TPointArrays from the T2DPointArray ATPA into the TPointArray Res

8.14.90 TPAFromBoxWrap

```
procedure TPAFromBoxWrap(const Box : TBox; var Res : TPointArray);
```

Create a TPointArray from the top left and the bottom right of the TBox Box and returns the result in Res

8.14.91 RotatePointsWrap

```
procedure RotatePointsWrap(Const P: TPointArray; A, cx, cy: Extended; var Res :  TPointArray);
```

Rotates the TPointArray P around the center cx, cy with the angle a and returns the result in Res

8.14.92 FindTPAEdgesWrap

```
procedure FindTPAEdgesWrap(const p: TPointArray; var Res :  TPointArray);
```

Returns a TPointArray of the edge points of the TPointArray p and returns the result in the TPointArray Res

8.14.93 ClearTPAFromTPAWrap

```
procedure ClearTPAFromTPAWrap(const arP, ClearPoints: TPointArray;  var Res :  TPointArray);
```

Removes the points in TPointArray ClearPoints from arP and returns the results in Res

8.14.94 SameTPA

```
function SameTPA(const aTPA, bTPA: TPointArray): Boolean;
```

Returns true if the TPointArray aTPA is the same as bTPA

8.14.95 TPAInATPA

```
function TPAInATPA(const TPA: TPointArray;const  InATPA: T2DPointArray; var Index: LongInt): Boolean
```

Returns true if the TPointArray TPA is found in the T2DPointArray InATPA and stores the index in Index

8.14.96 OffsetTPA

```
procedure OffsetTPA(var TPA : TPointArray; const Offset : TPoint);
```

Offsets all the TPAs int the TPointArray TPA but the TPoint Offset

8.14.97 OffsetATPA

```
procedure OffsetATPA(var ATPA : T2DPointArray; const Offset : TPoint);
```

Offsets all the TPAs int the T2DPointArray ATPA but the TPoint Offset

8.14.98 CopyTPA

```
function CopyTPA(const TPA : TPointArray) : TPointArray;
```

Returns the TPointArray TPA

8.14.99 CopyATPA

```
function CopyATPA(const ATPA : T2DPointArray) : T2DPointArray;
```

Returns the T2DPointArray ATPA

8.14.100 PartitionTPA

```
function PartitionTPA(const TPA: TPointArray; BoxWidth, BoxHeight: integer): T2DPointArray;
```

Partitions the TPA in boxes of BoxWidth and BoxHeight.

The following example will partition a TPA in boxes of 10 width, 10 height and debug it.

```
program PartitionTPA_Test;
 var
   tpa: TPointArray;
   atpa: T2DPointArray;
   canvas: integer;
begin
  tpa := TPAFromEllipse(50, 50, 33, 45);
  FillEllipse(tpa);
  atpa := PartitionTPA(tpa, 10, 10);

  // debugging the result
  canvas := CreateBitmap(100, 100);
  DrawATPABitmap(canvas, atpa);
  ClearDebugImg();
  DisplayDebugImgWindow(100, 100);
  DrawBitmapDebugImg(canvas);
  FreeBitmap(canvas);
end.
```

8.15 Multimedia Functions

8.15.1 Sound Functions

PlaySound

```
procedure PlaySound(Sound : string);
```

PlaySound plays the sound file with the path *Sound*.

Supported formats: *.wav*. (Possibly others, if someone has time to figures this out please let us know)

Example:

```
PlaySound('C:\roar.wav');
```

StopSound

```
procedure StopSound;
```

StopSound stops all currently playing sounds.

CHAPTER

NINE

MML REFERENCE

The Mufasa Macro Library is the core library of Simba. It is used not just to provide scripts with the required functionality, but also used to pick colours and select windows with Simba itself. The MML can run without any user interface.

There currently is an effort to create a standalone library of the MML; called *libMML*. This way any application can just load the library and use the MML functionality.

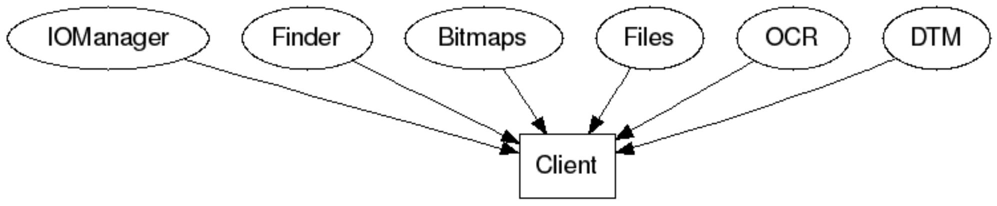
The MML is split up in “Core” classes and “Addon” classes.

Note: This section needs to explain more on the core/addon differences, and provide some more general info about the mml.

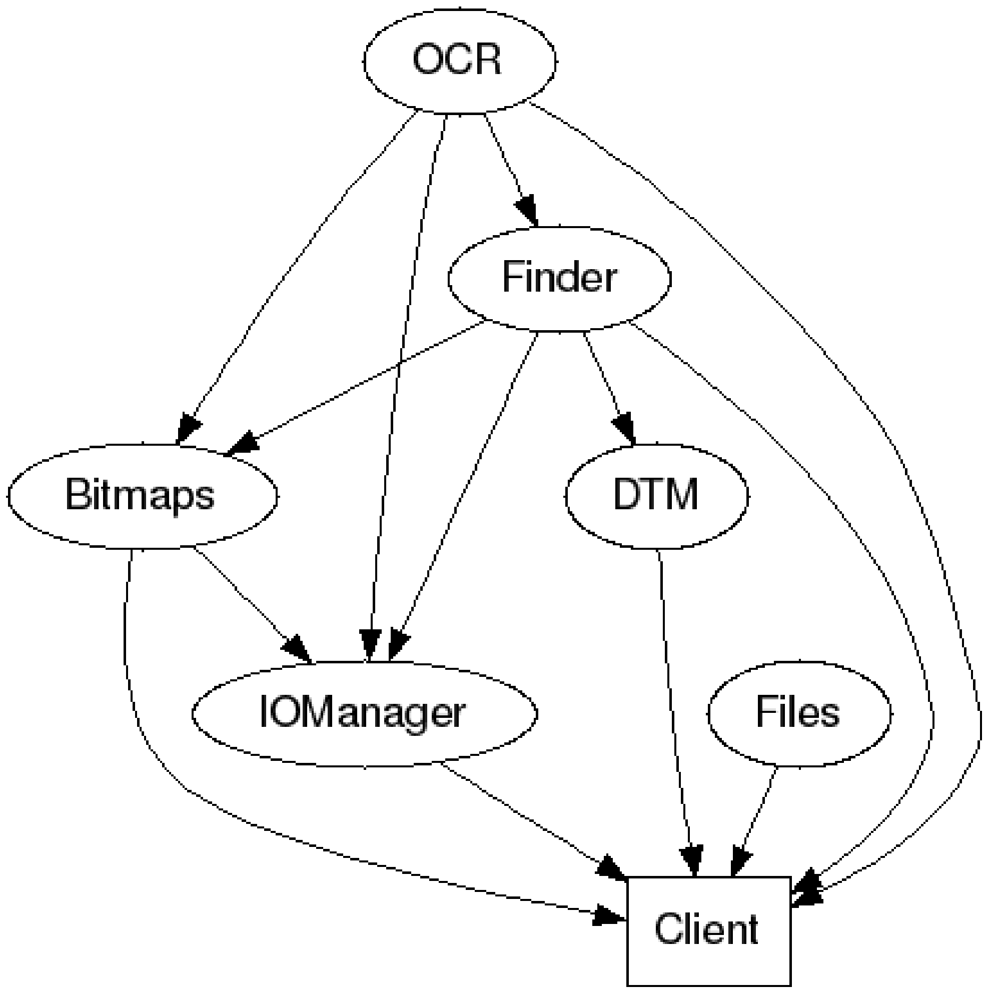
9.1 Client Class

The `TClient` class is the class that glues all other MML classes together into one usable class. Internally, quite some MML classes require other MML classes, and they access these other classes through their “parent” `TClient` class.

An image tells more than a thousands words:



And the class dependency graph: (An arrow indicates a dependency)



The client class does not do much else except creating the classes when it is created and destroying the classes when it is being destroyed.

Properties:

- IOManager
- MFiles
- MFinder
- MBitmaps
- MDTMs
- MOCR
- WriteLnProc

9.1.1 TClient.WriteLn

```
procedure TClient.WriteLn(s: string);
```

9.1.2 TClient.Create

```
constructor TClient.Create(const plugin_dir: string = ''; const UseIOManager : TIOManager = nil);
```

9.1.3 TClient.Destroy

```
destructor TClient.Destroy;
```

9.2 IOManager Class

Note: This page is still WIP. It only covers the function of the IOManager class vaguely. In reality, the IOManager unit contains quite some classes, each with a (slightly) different function. There are not yet documented. (Perhaps BenLand100 can do this?)

The IOManager class manages the core functionality for retreiving Window data, such as the actual pixel data and the position and dimension of a window.

The main purpose is to form a cross platform class to retrieve window information, press and poll mouse buttons and to press and poll keyboard keys.

The IOManager is the only class that should use platform (or operating system) specific calls; this is all abstracted by the IOManager class.

To achieve this, several abstract classes are defined by the IOManager class. Every operating system (or window system) needs it's own implementation of the `TWindow_Abstract` class. We wrote one for both Linux and Windows.

9.2.1 Linux Specific Implementation Details

The Linux specific implementation of the IOManager is written around the Linux X11 Windowing System. The X11 windowing system can run simultaneously to other X11 Window Systems, and each is defined by a `Display`.

A `Window` on a `Display` is represented as an unsigned integer.

To retreive data that is on a window, we use the XGetImage function. This function returns an XImage which contains the raw data. Simba then returns the pointer to this data. The image has to be freed afterwards; and this is done in the `FreeReturnData` procedure. Alternatively we could copy the image data to another place and then free the XImage, but this would result in additional overhead.

Most mouse related functionality is implemented with XQueryPointer, XWarpPointer and XTestFakeButtonEvent. (Requires XTest Extension)

All the keyboard functions use XTest as well. It currently lacks the functionality to test if a key on the keyboard is down.

9.2.2 Windows Specific Implementation Details

9.2.3 Offscreen image capturing

On Linux, offscreen image capturing is possible when compositing is turned on. With compiz, this is turned on by default. Other windows managers like Metacity and KWin can turn this on. When enabled, Simba can capture images from windows that are below others. Minimized does not work. It is also possible to turn on Compositing for specific X11 Windows with an api call, but this is currently not implemented.

The status on Windows is unknown.

9.2.4 Silent Input

So what is Silent Input? We define Silent Input as methods to manipulate the user's mouse and keyboard, without visually using them. So what does this mean?

This basically means that you will still be able to use your mouse while the MML is performing mouse operations on your targetted window/client.

However, silent input is very hard to implement, and often hardly supported by host operating systems. Often silent mouse or keyboard input is simply ignored. So in general it is advised to stick to non silent input.

9.3 TMFinder Class

9.4 TMufasaBitmap Class

9.5 TMOCR Class

The TMOCR class uses the powerful `ocrutil` unit to create some default but useful functions that can be used to create and identify text. It also contains some functions used in special cases to filter noise. Specifically, these are all the `Filter*` functions.

9.5.1 InitTOCR

```
function TMOCR.InitTOCR(const path: string): boolean;
```

InitTOCR loads all fonts in path We don't do this in the constructor because we may not yet have the path.

9.5.2 FilterUpTextByColour

```
procedure TMOCR.FilterUpTextByColour(bmp: TMufasaBitmap);
```

9.5.3 FilterUpTextByCharacteristics

```
procedure TMOCR.FilterUpTextByCharacteristics(bmp: TMufasaBitmap; w,h: integer);
```

9.5.4 FilterShadowBitmap

```
procedure TMOCR.FilterShadowBitmap(bmp: TMufasaBitmap);
```

Remove anything but the shadows on the bitmap (Shadow = clPurple)

9.5.5 FilterCharsBitmap

```
procedure TMOCR.FilterCharsBitmap(bmp: TMufasaBitmap);
```

Remove all but uptext colours clWhite,clGreen, etc.

This assumes that the bitmap only consists of colour 0, and the other constants founds above the functions

9.5.6 getTextPointsIn

```
function TMOCR.getTextPointsIn(sx, sy, w, h: Integer; shadow: boolean;
                               var _chars, _shadows: T2DPointArray): Boolean;
```

This uses the two filters, and performs a split on the bitmap. A split per character, that is. So we can more easily identify it.

TODO:

- Remove more noise after we have split, it should be possible to identify noise; weird positions or boxes compared to the rest, etc.
- Split each colours seperately, and combine only later, after removing noise.

9.5.7 GetUpTextAtEx

```
function TMOCR.GetUpTextAtEx(atX, atY: integer; shadow: boolean): string;
```

GetUpTextAtEx will identify each character, and also keep track of the previous chars' final *x* bounds. If the difference between the .x2 of the previous character and the .x1 of the current character is bigger than 5, then there was a space between them. (Add ' ' to result)

9.5.8 GetUpTextAt

```
function TMOCR.GetUpTextAt(atX, atY: integer; shadow: boolean): string;
```

Retreives the (special) uptext.

9.5.9 GetTextATPA

```
function TMOCR.GetTextATPA(const ATPA : T2DPointArray;const maxvspacing : integer; font: string): str
```

Returns the text defined by the ATPA. Each TPA represents one character, approximately.

9.5.10 GetTextAt

```
function TMOCR.GetTextAt(xs, ys, xe,ye, minvspacing, maxvspacing, hspacing,
                         color, tol: integer; font: string): string;
```

General text-finding function.

9.5.11 GetTextAt (2)

```
function TMOCR.GetTextAt(atX, atY, minvspacing, maxvspacing, hspacing,
                         color, tol, len: integer; font: string): string;
```

General text-finding function. Different parameters than other GetTextAt.

9.5.12 TextToFontTPA

```
function TMOCR.TextToFontTPA(Text, font: String; out w, h: integer): TPointArray;
```

Returns a TPA of a specific *Text* of the specified *Font*.

9.5.13 TextToFontBitmap

```
function TMOCR.TextToFontBitmap(Text, font: String): TMufasaBitmap;
```

Returns a Bitmap of the specified *Text* of the specified *Font*.

9.6 Uptext

To read the UpText, the TMOCR class applies several filters on the client data before performing the actual OCR. We will take a look at the two filters first.

9.6.1 Filter 1: The Colour Filter

We first filter the raw client image with a very rough and tolerant colour comparison / check. We first convert the colour to RGB, and if it falls into the following defined ranges, it may be part of the uptext. We also get the possible shadows.

We will iterate over each pixel in the bitmap, and if it matches any of the *rules* for the colour; we will set it to a constant colour which represents this colour (and corresponding rule). Usually the *base* colour. If it doesn't match any of the rules, it will be painted black. We won't just check for colours, but also for differences between specific R, G, B values. For example, if the colour is white; R, G and B should all lie very close to each other. (That's what makes a colour white.)

The tolerance for getting the pixels is quite large. The reasons for the high tolerance is because the uptext colour vary quite a lot. They're also transparent and vary thus per background. We will store/match shadow as well; we need it later on in filter 2.

To my knowledge this algorithm doesn't remove any *valid* points. It does not remove *all* invalid points either; but that is simply not possible based purely on the colour. (If someone has a good idea, let me know)

In code:

```
for y := 0 to bmp.Height - 1 do
  for x := 0 to bmp.Width - 1 do
  begin
    colortorgb(bmp.fastgetpixel(x,y),r,g,b);

    if (r < ocr_Limit_Low) and (g < ocr_Limit_Low) and
        (b < ocr_Limit_Low) then
    begin
```

```
      bmp.FastSetPixel(x,y, ocr_Purple);
      continue;
    end;

    // Black if no match
    bmp.fastsetpixel(x,y,0);
  end;
```

9.6.2 Filter 2: The Characteristics Filter

This second filter is easy to understand but also very powerful:

- It removes *all* false shadow pixels.
- It removes uptext pixels that can't be uptext according to specific rules. These rules are specifically designed so that it will never throw away proper points.

It also performs another filter right at the start, but we'll disregard that filter for now.

Removing shadow points is trivial if one understands the following insight.

If there some pixel is shadow on *x, y*, then it's neighbour *x+1, y+1* may not be a shadow pixel. A shadow is always only one pixel *thick*.

With this in mind, we can easily define an algorithm which removes all false shadow pixels. In code:

```
{
    The tricky part of the algorithm is that it starts at the bottom,
    removing shadow point x,y if x-1,y-1 is also shadow. This is
    more efficient than the obvious way. (It is also easier to implement)
}

for y := bmp.Height - 1 downto 1 do
  for x := bmp.Width - 1 downto 1 do
  begin
    // Is it shadow?
    if bmp.fastgetpixel(x,y) <> clPurple then
      continue;
    // Is the point at x-1,y-1 shadow? If it is
    // then x, y cannot be shadow.
    if bmp.fastgetpixel(x,y) = bmp.fastgetpixel(x-1,y-1) then
    begin
      bmp.fastsetpixel(x,y,clSilver);
      continue;
    end;
    if bmp.fastgetpixel(x-1,y-1) = 0 then
      bmp.fastsetpixel(x,y,clSilver);
  end;
```

We are now left with only proper shadow pixels. Now it is time to filter out false Uptext pixels.

Realize:

- If *x, y* is uptext, then *x+1, y+1* must be either uptext or shadow.

In code:

```
for y := bmp.Height - 2 downto 0 do
  for x := bmp.Width - 2 downto 0 do
  begin
```

```
    if bmp.fastgetpixel(x,y) = clPurple then
      continue;
    if bmp.fastgetpixel(x,y) = clBlack then
      continue;

    // Is the other pixel also uptext?
    // NOTE THAT IT ALSO HAS TO BE THE SAME COLOUR
    // UPTEXT IN THIS CASE.
    // I'm still not sure if this is a good idea or not.
    // Perhaps it should match *any* uptext colour.
    if (bmp.fastgetpixel(x,y) = bmp.fastgetpixel(x+1,y+1) ) then
      continue;

    // If it isn't shadow (and not the same colour uptext, see above)
    // then it is not uptext.
    if bmp.fastgetpixel(x+1,y+1) <> clPurple then
    begin
      bmp.fastsetpixel(x,y,clOlive);
      continue;
    end;

   // If we make it to here, it means the pixel is part of the uptext.
  end;
```

9.6.3 Identifying characters

Note: This part of the documentation is a bit vague and incomplete.

To actually identify the text we split it up into single character and then pass each character to the OCR engine.

In the function *getTextPointsIn* we will use both the filters mentioned above. After these have been applied, we will make a bitmap that only contains the shadows as well as a bitmap that only contains the uptext chars (not the shadows)

Now it is a good idea to count the occurances of all colours (on the character bitmap); we will also use this later on. To split the characters we use the well known *splittpaex* function.

We will then sort the points for in each character TPA, as this makes makes looping over them and comparing distances easier. We will also calculate the bounding box of each characters TPA.

Note: Some more hackery is then used to seperate the characters and find spaces; but isn't yet documented here.

Normal OCR

Note: To do :-) A large part is already explained above. Most of the other OCR functions are simply used for plain identifying and have no filtering tasks.

9.7 libMML

libMML is an effort to bring the entire MML to a single standalone library.

MML usage will appear here soon.

CHAPTER

TEN

SIMBA REFERENCE

All Simba documentation should be in here.

10.1 Script Manager

Simba script manager. =}

10.2 Writing Simba Extensions

Simba extensions are scripts written for the interpreter that can be embedded into Simba. Purposes vary from updaters to editors.

10.2.1 How they work

Extensions are event based. This means you don't have some `loop` in your program that never ends and does all the logic for you. When a system is event based, you implement some functions and those are called on a certain event.

10.2.2 Extension core hooks

Simba offers several core hooks: init, free, attach and detach. These are used to initialize, show, hide and free your extension. GetName and GetVersion are called to retreive the name and version of an extension.

init

Called when the Extension is initialized.

```
procedure init;
begin;
    Writeln('Initialize your extension here.');
end;
```

If you want to add a button to the menu, do it now. From the SRL updater extension:

```
procedure Init;
begin;
  MainMenuItem := TMenuItem.Create(Simba_MainMenu);
  MainMenuItem.Caption := 'SRL';
  Simba_MainMenu.Items.Add(MainMenuItem);

  MenuCheck := TMenuItem.Create(MainMenuItem);
  MenuCheck.Caption := 'Check for new SRL';
  MenuCheck.OnClick := @OnSRLCheckClick;
  MainMenuItem.Add(MenuCheck);

  MenuUpdate := TMenuItem.Create(MainMenuItem);
  MenuUpdate.Caption := 'Update SRL';
  MenuUpdate.OnClick := @OnSRLUpdateClick;
  MainMenuItem.Add(MenuUpdate);

  AutoUpdate := TMenuItem.Create(MainMenuItem);
  AutoUpdate.Caption := 'Automatically update';
  AutoUpdate.OnClick := @SetAutoUpdate;
  AutoUpdate.Checked := LowerCase(Settings.GetKeyValueDef('AutoUpdate',
                                  'True')) = 'true';
  MainMenuItem.Add(AutoUpdate);

  Timer := TTimer.Create(Simba);
  Timer.Interval := 5000;
  Timer.OnTimer := @OnUpdateTimer;
  Timer.Enabled :=AutoUpdate.Checked;

  started := True;
end;
```

free

Called upon freeing the extension. Usually this means that Simba is closed.

```
procedure free;
begin
    if started then
        writeln('Free() was called');
end;
```

From the SRL updater extension:

```
procedure Free;
begin
  if (started) then
    Timer.Enabled := False;
    { Freeing the components is not needed, as they will be freed
       upon the closure of Simba. }
end;
```

attach

Called when your extension has been enabled.

From the SRL updater extension:

```
procedure Attach;
begin;
  Writeln('From now on, you shall be alerted as to when your SRL is'+
            +'out of date!');
  MainMenuItem.Visible := True;
  Timer.Enabled := AutoUpdate.Checked;
end;
```

detach

Called when your extension has been disabled. (This is not the same as freeing)

```
Procedure Detach;
begin
  Timer.Enabled := False;
  MainMenuItem.Visible := False;
end;
```

GetName

Called when Simba requests the name of your extension.

```
function GetName : string;
begin;
  result := 'SRL Updater';
end;
```

GetVersion

Called when Simba requests the version of the extension.

```
function GetVersion : string;
begin;
  result := '1.0';
end;
```

More extension hooks

The following hooks are called upon if the event that the name describes has happened.

```
EventHooks: Array [0..9] of TEventHook =
(       (HookName : 'onColourPick'     ; ArgumentCount : 3),
    (HookName : 'onOpenFile'       ; ArgumentCount : 2),
    (HookName : 'onWriteFile'      ; ArgumentCount : 2),
        (HookName : 'onOpenConnection'; ArgumentCount : 2),
        (HookName : 'onScriptStart'    ; ArgumentCount : 2),
        (HookName : 'onScriptCompile'  ; ArgumentCount : 1),
    (HookName : 'onScriptExecute'  ; ArgumentCount : 1),
    (HookName : 'onScriptPause'    ; ArgumentCount : 1),
    (HookName : 'onScriptStop'     ; ArgumentCount : 1),
    (HookName : 'onScriptOpen'     ; ArgumentCount : 1));
```

For the exact arguments to all these functions, refer to *Example code*.

onOpenConnection

onWriteFile

onOpenFile

onColourPick

onScriptStart

onScriptOpen

This is a *function*, and should return the script, plus any modifications.

10.2.3 Special Cases

Multiple extensions hooking the same event

So what happens when multiple extensions hook onto the same event/hook?

The behaviour is currently defined, but prone to change in the near future. Currently all extensions are called in the order they were loaded.

The behaviour will probably change to something like the following:

> In the order they were loaded, call any available extensions. The first extension to return non-zero terminates the calling loop.

10.2.4 Pitfalls

Extensions can be very dangerous in the sense that they run on the main thread of Simba; it is very easy to crash Simba or cause it to hang. There is no way to prevent this, so make sure to check what you're doing before you try your own (or someone else's) extension.

10.2.5 Example code

Example code for most hooks:

```
program new;

procedure init;
begin;
  Writeln('init your extension here');
end;
procedure onOpenConnection(var url : string; var Cont : boolean);
begin
  Writeln('Opening url: '  + url);
  Writeln('We shall allow this.. For now!! Gna Gna!');
  Cont := True;
end;
procedure onWriteFile(var FileName : string; var Cont : boolean);
begin
  Writeln('So.. You want to write to file: ' + FileName);
  Writeln('Well for this time only!');
  Cont := True;
end;

procedure onOpenFile(var FileName : string; var Cont : boolean);
begin
  Writeln('So you want to open this file: ' + filename);
  Writeln('Well I don''t care much, lets see what the other hooks think!');
  //Not set Cont as we don't care, while other hooks might
end;

procedure onColourPick(const Colour,x,y : integer);
begin
  Writeln('So you''ve picked a color, huh!?');
  Writeln(inttostr(colour) + ' attuh (' + inttostr(x) +',' + inttostr(y) + ')');
end;

function onScriptStart(var Script : string; var Cont : boolean): String;
begin
  Writeln('So you want to compile the following script!!');
  Writeln(script);
  Writeln('lets allow that for now ;)');
  Cont := True;
  Result := Script;
end;

function onScriptOpen(var Script: String): String;
begin
  Result := Script + 'THIS IS AFTER THE .END, HOPEFULLY');
end;

procedure free;
begin
  Writeln('Free your extension here');
end;

procedure Attach;
begin;
  Writeln('Your extension has been enabled, do stuff here');
end;

Procedure Detach;
begin
  Writeln('Your extension has ben disabled, do stuff here');
end;

//Called to retrieve the name of your extension
function GetName : string;
begin;
  result := 'Leet Extension';
end;

//Called to retrieve the version of your extension
function GetVersion : string;
begin;
  result := '0.001';
end;
begin
end.
```

Note: If you need more examples, you can always look at the Extensions in the Simba git repository.

10.3 Writing Simba Plugins

10.3.1 Plugin overview

Plugins for Simba can be written in virtually every language. (Make sure to read Caveats though)

10.3.2 Simba Plugin ABI

First of all, it is important to know in what way you should represent your functions to Simba. Using the newest ABI is recommended. Currently, all three ABIs are supported; but older ones may be deprecated in subsequent releases.

Plugins with ABI < 2 are not exported to Lape.

Simba ABI Version 0

This is the first Simba plugin ABI. Any plugin that does not export GetPluginABIVersion is treated as this ABI.

In this ABI GetTypeInfo passes the arguments as a native pascal **String** rather than a **PChar**. This is error prone; and versions > 0 use **PChar**.

The calling convention for all functions is expected to be **stdcall**.

Simba ABI Version 1

In this ABI GetTypeInfo passes the arguments as a **PChar**.

The calling convention for all functions (except GetPluginABIVersion) is expected to be **stdcall**.

Simba ABI Version 2

The calling convention for all functions is expected to be **cdecl** on 32 bit platforms; on platforms where **cdecl** does not exist, the native calling convention is expected.

10.3.3 Functions

GetPluginABIVersion

This function should return the ABI version of the plugin. The calling convention for this function is **always cdecl** on 32 bit and the native calling convention if **cdecl** does not exist.

SetPluginMemManager

```
procedure SetPluginMemManager(MemMgr : TMemoryManager);
```

This function should be implemented when one is writing a plugin in Free Pascal. Using the Simba memory manager it is possible to easily pass arrays and other managed structures from the script to the plugin. It is however wise to also store the old memory manager of your plugin and restore it when *OnDetach* is called.

For other languages, this function doesn't really help - however you should realise that it is not possible to pass pascal strings and arrays directly to the script in a safe manner. Doing so will require some hacks (it is possible), but might also lead to memory leaks if implemented wrong or used wrong in scripts.

Warning: Simba does not guarantee that this function will be called only once; make sure you don't overwrite your old memory manager when this function is called twice.

OnAttach

```
procedure OnAttach(info: Pointer);
```

This method is called when the plugin is loaded. Currently *info* will always be zero, but it might be used to pass information in the future.

OnDetach

```
procedure OnDetach();
```

This method is called just before the plugin is beeing freed. *If you changed your memory manager to Simba's; you should now revert to your old one.*

GetFunctionCount

```
function GetFunctionCount: integer;
```

The first function, *GetFunctionCount* returns how many functions are to be imported.

GetFunctionInfo

```
function GetFunctionInfo(x: Integer; var ProcAddr: Pointer; var ProcDef: PChar): integer;
```

Simba will then call *GetFunctionInfo* and *GetFunctionCallingConv* **N** amount of times (where **N** is the result of *GetFunctionCount*) with *x* increased by one every time. Obviously, each function must be mapped to a specific value of *x*.

For *GetFunctionInfo*, the value of *ProcAddr* should be set to the address of the procedure to be called; and *ProcDef* should contain the definition (in Pascal types) of the function.

GetFunctionCallingConv

Warning: This function is deprecated as of ABI >= 2

```
function GetFunctionCallingConv(x: integer): integer; stdcall;
```

GetFunctionCallingConv returns the calling convention for the specific function. Currently, the only two support conventions are *stdcall* (0) and *register* (1).

GetTypeCount

GetTypeInfo

10.3.4 Exporting functions to scripts

To let Simba know what functions you want to export to a script, your plugin needs to implement the following functions: GetFunctionCount and GetFunctionInfo. Refer to their sections

10.3.5 Exporting types to scripts

Warning: TODO

10.3.6 TTarget_Exported

Warning: TODO

10.3.7 Caveats

If you're writing a plugin in a language other than Free Pascal, you'll not be able to share arrays and strings with Simba in an easy manner. (It is possible to "craft" pascal-type strings and arrays)

Pascal Arrays

Say we have an array of *foo* called *bar*. *bar[0]* holds the first element of the array. *bar* - Sizeof(Pointer) contains the length of the array, and *bar* - Sizeof(Pointer) * 2 contains the reference count of the array. If you want to share an array with Simba, make sure the reference is count is high enough so that Simba/Free Pascal won't try to free it for you.

Pascal Strings

Warning: I believe pascal strings are very similar to pascal arrays, but I am not completely sure.

Sharing Arrays and Strings with a FPC Plugin

To share arrays and strings in a nice way with a FPC plugin, you need to create a function called SetPluginMemManager as shown above and make sure it is exported properly. Simba will try to call this function when loading the plugin and will pass the plugin its own memory manager. Use FPC's *SetMemoryManager* to change your own memory manager to Simba's memory manager. When *OnDetach* is called, make sure you reset your memory manager. See SetPluginMemManager.

10.3.8 Sample FPC Plugin

```
{ Example based upon the SPS Plugin }
library project1;

{$mode objfpc}{$H+}

{$macro on}
{$define callconv:=
    {$IFDEF WINDOWS}{$IFDEF CPU32}cdecl;{$ELSE}{$ENDIF}{$ENDIF}
    {$IFDEF LINUX}{$IFDEF CPU32}cdecl;{$ELSE}{$ENDIF}{$ENDIF}
}

uses
  classes, sysutils, math
  { you can add units after this };

var
  OldMemoryManager: TMemoryManager;
  memisset: Boolean = False;


  type T3DIntegerArray = array of array of array of integer;

function HelloPlugin(s: String): String; callconv
begin
  result := s;
end;

function GetPluginABIVersion: Integer; callconv export;
begin
  Result := 2;
end;

procedure SetPluginMemManager(MemMgr : TMemoryManager); callconv export;
begin
  if memisset then
    exit;
  GetMemoryManager(OldMemoryManager);
  SetMemoryManager(MemMgr);
  memisset := true;
end;

procedure OnDetach; callconv export;
begin
  SetMemoryManager(OldMemoryManager);
end;

function GetTypeCount(): Integer; callconv export;
```

```
begin
  Result := 1;
end;

function GetTypeInfo(x: Integer; var sType, sTypeDef: PChar): integer; callconv export;
begin
  case x of
    0: begin
        StrPCopy(sType, 'T3DIntegerArray');
        StrPCopy(sTypeDef, 'array of array of array of integer;');
       end;

    else
      x := -1;
  end;

  Result := x;
end;

function GetFunctionCount(): Integer; callconv export;
begin
  Result := 1;
end;

function GetFunctionInfo(x: Integer; var ProcAddr: Pointer; var ProcDef: PChar): Integer; callconv ex
begin
  case x of
    0:
      begin
        ProcAddr := @HelloPlugin;
        StrPCopy(ProcDef, 'function HelloPlugin(s: String): String;');
      end;

    else
      x := -1;
  end;

  Result := x;
end;

exports GetPluginABIVersion;
exports SetPluginMemManager;
exports GetTypeCount;
exports GetTypeInfo;
exports GetFunctionCount;
exports GetFunctionInfo;
exports OnDetach;

begin
end.
```

CHAPTER

ELEVEN

DOCUMENTATION DOCUMENTATION

This page is the documentation for the documentation.

It is very important to know this by heart when you are writing documentation for Simba.

11.1 Documentation system

The documentation system we use is sphinx. The link to sphinx can be found at the bottom of the page.

Note: It is important to note that there is also a database SQL fulltext engine called Sphinx, but this is not the project we use. We use the Sphinx "documentation system" (http://sphinx.pocoo.org/)

11.2 Building the documentation

In the future, the online documentation will be refreshed every hour. However, if you want to build the documentation yourself, you should install `python-sphinx`.

Move to the `Simba/doc/sphinx` directory and run `make all`. This will place an HTML version of the documentation in `_build/html`.

Note: The build instructions are for Linux only. If you want to build the doc on Windows, you are on your own. The sphinx resource site is probably a good place to start.

11.3 Writing documentation

Sphinx uses the reStructuredText markup language. It is not a hard language, but looking through the quickstart is a good idea: http://docutils.sourceforge.net/docs/user/rst/quickstart.html

As stated above, the markup language is not the hard part about writing documentation; the hard part is simply coming up with good content suited for the documentation. Be sure to check *Documentation TODO*

11.3.1 Directory structure

So you have written a new piece of documentation? Great!

Now we just need to know where to place it. If you have simply extended a file, then there should be no worries as to where to place your new text. However if you are writing a new chapter, then placing the file in the correct directory is something we'd like you to consider.

If you write a chapter for the `Simba Reference` or `Scripting Reference` or `Features` part of the documentation, place it in the `simbaref`, `scriptref` or `features` folder repectively. Any other files can be put directly in the root of the sphinx folder. (The same place as `index.rst`)

11.3.2 Capitalisation

The titles of all major sections have all words capitalized. (The ones with ===) The minor sections and subsections (—) and (~~~) have only the first word and Simba specific words (like Simba itself) capitalized.

Try to stick to the Python documentation standards. (http://docs.python.org/using/index.html)

11.4 References

Sphinx has references, most of the .rst files contain labels and references.

A label looks something like this:

```
.. _<X>-<Y>:
```

Where X is either the name of file or folder; and Y is the name of the file/function if X is a folder. For referring to specific functions for example, use:

```
.. _scriptref-movemouse:
```

To define a label for the MoveMouse function. Labels are always placed right above section/chapter declarations.

CHAPTER

TWELVE

DOCUMENTATION TODO

- *Wizzup* - Script manager (non technical).
- *Wizzup* - Script manager (technical).
- Extend "Getting Started". Include downloading scripts from the manager.
- think of good chapters for the complete tutorial. (it should teach basic stuff, not document all features. script reference is for that purpose)
- write a lot more chapters for simba references. There's plenty to document. It may be useful to check http://wizzup.org/simba/article/4
- Features -> Perhaps (interactive) images?
- **Expand "Troubleshooting"**
    - And its subsection.
- **Expand "Feature Overview"**
    - And its subs. There's like nothing in them, those are the type of pages I was talking about. Combine them under feature overview?
- Write lots of examples.
    - How to use file functions.
    - How to use DTM functions, etc.
- Add a section with examples of stuff that is often done / used.

CHAPTER

THIRTEEN

LIBMML

13.1 libMML

libMML is short for the library to the Mufasa Macro Library.

13.1.1 Why libMML?

One of the first questions that rise are probably *why* libMML and *what* is libMML exactly?

libMML is a loadable C-like library that provides most of the MML functionality. For the ones that do not know, MML is the core component for Simba that does all the *computational* and *algorithmic* work. See *MML Reference* for more information.

To summarize, the MML covers:

- Targetting specific windows and getting the window information such as their bitmaps and dimensions.
- Controlling the mouse and keyboard.
- Finding colours, bitmaps and dtms.
- Text recognition (OCR)
- Accessing files and sockets in an unified manner

Hopefully the *what* part of the question has mainly been covered by now. If you're literate in computer science the *why* question has also been answered as soon as it was mentioned that is was a loadable library - which is also implied by its name. Exporting the MML into a loadable library allows virtually any program to load it and just use all the MML functionality.

13.1.2 Design (issues)

libMML itself should not be too complex. It should simply translate the OOP MML to a non-OOP C-type library and convert datatypes when required (see below as to why). libMML is basically just a codebase that calls MML functions and passes the result along in a slightly different format. In simple cases such as MoveMouse the integers are simply passed; since there's do not differ, but in the case of arrays of any type we have to copy the arrays to a C format - at least until MML internally will no longer use Free Pascal (managed) arrays.

As previously mentioned, libMML is a *C*-type library; this is mentioned explicitly because MML is written in Free Pascal (Object Pascal) which has quite a few different datatypes. Strings are typically not compatible, and arrays are managed in Pascal whereas they are not in C which makes it hard to just *pass* the array along. One of the problems we have to cope with when writing libMML is converting datatypes to C-compatible datatypes. C-compatible datatypes are supported by most programming languages and thus the best way to go when making a universal MML library.

13.1.3 libMML use cases

Theoretically libMML can be loaded by any programming language; but typically each programming languages has it's own kind of programming practices and thus write - again - their own wrapper around libMML. This is what is being done with *pyMML*, the python libMML wrapper. It is still as much in development as libMML is, but the functionality exposed by libMML is succesfully used.

As of writing the pyMML usage looks like this, the passing around of a client may be removed in a later stage, or at least have it's behaviour changed.

```
DLL = MMLCore('../libmml.so')

client = DLL.dll.create_client()
print 'Python Client: %d' % client
if client in (0, 1):
    raise Exception('Could create a client');

c = Color(DLL, client)


ret = c.find((0, 0, 100, 100), 0)
print ret

ret = c.find_all((0, 0, 100, 100), 0, tol=100)
print ret

m = Mouse(DLL, client)

print m[(Mouse.Pos, Mouse.Left, Mouse.Right)]
m[(Mouse.Pos, Mouse.Right)] = ((300,300), True)

print m.getButtonStates()
sleep(0.5)
m.setPos((200,200))

sleep(2)
print 'Done'

m[(Mouse.Left, Mouse.Right, Mouse.Middle)] = [False for x in range(3)]
for v in zip((Mouse.Left, Mouse.Right), m[(Mouse.Left, Mouse.Right)]):
    print v
print m.getPos()

del DLL
```

CHAPTER

FOURTEEN

PYTHON MML

14.1 PyMML Mouse

This is the PyMML Mouse class.

class `mouse.`**`Mouse`**(*MC*, *cli*)

The MML Mouse object communicates directly with libmml, but wraps it around a nice and easy to use layer.

It will allow several ways to set mouse positions and buttons. __getitem__ and __setitem__ are also implemented, so one can access mouse buttons states and positions with [].

`getButtonStates`()

Get the current states of the mouse buttons.

`getPos`()

Get the current mouse position as a tuple.

`setButtonState`(*button*, *downup*)

Set the states of the mouse buttons.

`setPos`(*pos*)

Set the mouse position to the tuple _pos_.

14.2 PyMML Color

This is the PyMML Color class.

class `color.`**`Color`**(*MC*, *cli*)

The Color class.

`count_color`(*count*, *col*, *box*, *tol=0*)

Counts color col in box with tol. Yields integer of count.

`find`(*box*, *color*, *tol=0*)

find a color in a box, with a specific tolerance. returns a tuple of the x, y value of a matched color. None if no color was found.

`find_all`(*box*, *color*, *tol=0*)

find all colors in a box, with a specific tolerance. returned are all the matching points

`find_area`(*col*, *box*, *min_a*, *tol=0*)

Finds a colored area in box with min area min_a with a specific tolerance.

Yields a tuple of x, y values of found area.

find_spiral(*col*, *box*, *tol=0*)

Find a color in a box, searching in the direction of a spiral, with a specific tolerance.

Yields a tuple of x, y values of found color.

get(*pt*)
Gets color at pt[0], pt[1]. Yields integer.

get_tolerance_speed()
Gets CTS. Yields CTS.

get_tolerance_speed_2_modifiers()
Gets CTS2 modifiers. Yields tuple of hue and sat mods.

set_tolerance_speed(*ncts=0*)
Sets CTS to ncts.

set_tolerance_speed_2_modifiers(*hue=0*, *sat=0*)
Sets CTS2 modifiers with hue, sat.

similar_colors(*col1*, *col2*, *tol=0*)
Compares col1 and col2 with tol. Yields boolean

CHAPTER

FIFTEEN

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

INDEX