

This document contains information that is proprietary to Analog Flavor. The software and documentation are furnished under a license agreement. Use and distribution of the proprietary information is only authorized in accordance with the terms of the license agreement . This document may be copied in whole or in part for internal business purposes only, provided that this entire notice appears in all copies.

This document is for information and instruction purposes. Analog Flavor reserves the right to make changes in specifications and other information contained in this publication without prior notice.

ANALOG FLAVOR AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

ANALOG FLAVOR SHALL NOT BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOST PROFITS) ARISING OUT OF OR RELATED TO THIS PUBLICATION OR THE INFORMATION CONTAINED IN IT, EVEN IF ANALOG FLAVOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Analog Flavor EURL au capital de 5000 Euro
SIREN 814 075 727

3 bis, rue des Engenières, 38360 Sassenage, France.
Telephone: +33 6 28 34 91 82
support@analogflavor.com
www.analogflavor.com

Table of Contents

Chapter 1	BeSpice Wave Widget	5
1.1	Waveform File Examples	6
1.2	Interactive commands	6
1.2.1	Command Return Values	6
1.2.2	Command Result Values	6
1.2.3	The Control Panel	6
1.2.4	The Command Logger	6
1.3	Widget Styles	7
Chapter 2	C/C++ Application Integration	8
2.1	The Header Files	8
2.2	The Access Functions	8
2.3	Registering Callback Functions	9
2.4	Linking the Shared Library	9
2.5	Linking the Static Library	10
Chapter 3	The Windows API Widget	11
3.1	Widget Commands	11
3.2	Integration to a Windows Application	11
3.3	Integration to a WPF Application	12
3.4	Integration to a Qt Application	12
3.5	Integration to a wxWidgets Application	13
3.6	Integration to a Tk Application	13
3.7	Integration to a Tk Application in C	13
3.8	Integration to a Python Script	14
3.8.1	Integration to a PyQt Application	14
3.8.2	Integration to a wxPython Application	14
3.8.3	Integration to a Tk Inter Application	14
Chapter 4	Tk Script Integration	15
4.1	af_bspwave_widget	15
4.1.1	Synopsis	15
4.1.2	Widget Specific Options	15
4.1.3	Widget Commands	16
4.2	How the Tk widget works	16
4.3	Drag and Drop with Tk	16
4.4	Tk Script Example	16
4.4.1	The Communication Tab	17
4.4.2	The Commands Tab	17
4.4.3	The Data Tab	17
4.4.4	The New Plot Tab	18
4.4.5	The Existing Plots Tab	18
4.4.6	The Gui Options Tab	18
4.4.7	The Callbacks Tab	18

4.4.8	The Drag and Drop Tab	18
Chapter 5	Tk C/C++ Application Integration	19
5.1	The Library Access Functions	19
5.1.1	tk_bspwave_create_widget_from_path	19
5.2	Tk Application Example	20
Chapter 6	Widget Specific Commands	22
6.1	Dragging Interactive Commands To BeSpice Wave	22
6.2	Dragging Curve Names From BeSpice Wave	22
Chapter 7	Accessing The Waveform Parser	23
Chapter 8	Extending BeSpice Wave's Waveform Parser	24
Chapter 9	Extending BeSpice Wave With Graphical Plug-ins	25
Chapter 10	Troubleshooting	26
10.1	Limitations	26
10.2	Reporting Errors	26
10.3	License Problems	26

CHAPTER 1 BESPICE WAVE WIDGET

BeSpice Wave is a waveform viewer enhanced and specialized for visualizing and analyzing Spice simulation results. Many different file formats are supported.

BeSpice Wave can be integrated into other applications. Therefore it is distributed as a shared or static library. The integration makes use of BeSpice Wave's interactive commands. This user manual handles the particularities of the widget integration. The interactive commands are documented in BeSpice Wave's user manual.

The following platforms are supported:

- wxWidgets
- Tk applications
- Tk scrips
- TkInter (Python)
- Windows API
- Qt
- Wrappers for the Windows API for WPF, MFC, Qt, wxWidgets, wxPython, PyQt, Tk and TkInter.

The software is distributed in a compressed Analog Flavor archive that can be downloaded from our ftp site. The file name is “analog_flavor.tar.gz” and might have an extension indicating the version or build date. Copy this archive to an appropriate location and decompress it using the shell command “tar xzf analog_flavor.tar.gz”.

This will generate a directory “analog_flavor” with the following directory structure:

- **analog_flavor**, the main directory.
 - **bin**, containing the wrappers for the main executables (Linux) or the binary executables (Windows).
 - **documentation**, containing the user manuals and documentation.
 - **examples**, containing some basic examples for the use of Analog Flavor software and examples for the use of the widget.
 - **license**, the location of the license file and executables to start and stop the license server.
 - **platform**, containing the binary executables and libraries for Linux 32 and 64 bit and Windows X86 and X64 (64 bit).

The widget distribution contains examples that can be launched or compiled following the instructions in readme.txt files.

1.1 Waveform File Examples

Example waveform files and corresponding spice netlists can be found at:

```
.../analog_flavor/examples/bspwave_widget/examples/
```

The directory also contains an interactive command file. This file shows how interactive commands can be used to control BeSpice Wave. The commands can also be transmitted over the widget interface.

1.2 Interactive commands

The interactive commands are documented in detail in BeSpice Wave's user manual. Some interactive commands have been optimized for widget integrators.

When using the interactive mode the user has still full control of the tool. That means that curves or data sections can be added or removed, plot windows can be closed or BeSpice Wave might be closed by the user. As a consequence some interactive commands might fail.

1.2.1 Command Return Values

When executing an interactive command, BeSpice Wave returns a code that has been defined in the file “af_return_codes.h”. If the code “af_code_queued” is returned, the command is executed in a thread and when the function returns it is not yet clear if the command will succeed. BeSpice Wave will remain busy until the command has been executed. All further commands will be queued. The status of the widget can be queried by sending the interactive command “get_status”. BeSpice Wave is ready to execute the next command when the returned code changes from “af_code_queued” to either “af_code_ok” or “af_code_error”.

For script interfaces the returned code is converted to a string.

1.2.2 Command Result Values

When executing an interactive command that queries BeSpice Wave, one or several string values are generated in addition to the return code of the interactive command function. These strings have to be retrieved by an appropriate function. If the command result is an integer value, the returned string can be converted accordingly.

For script interfaces the result values are directly converted to a list of strings.

1.2.3 The Control Panel

The control panel allows integrators to test some of the available interactive commands. It can be activated by the interactive command “show_control_panel”. For all provided examples it can also be shown by selecting “Show Control Panel” from the “Dev” menu.

The control panel shows a notebook that allows to execute some interactive commands. The notebook tab “Communication” shows the text command sent to the widgets and the returned answer.

1.2.4 The Command Logger

Once the BeSpice Wave widget has been integrated, it will be controlled by interactive commands.

These commands can be debugged using the command logger. It can be activated by the interactive command “show_command_logger”. For all provided example it can be shown by selecting “Show Command Logger” from the “Dev” menu. All commands sent to BeSpice Wave and their answers are recorded in The shown text editor.

1.3 Widget Styles

When creating a BeSpice Wave widget the “style” flag for the widget can be defined. All constants that can be combined to define the style can be retrieved from the file “bspwave_widget.h”. For script interfaces these constants have to be defined as integer values such as 1 to activate debug mode.

- AF_BSPWAVE_STYLE_DEBUG: activates debug mode. That means that additional messages are logged. The user has control over the appearance of the widget from the tool's configuration menu.
- AF_BSPWAVE_STYLE_USE_TOOLBAR: activates a tool bar.
- AF_BSPWAVE_STYLE_USE_NOTEBOOK: shows all pages in a notebook.
- AF_BSPWAVE_STYLE_AUTO_LOAD_SAVE_CONFIG: automatically loads and saves the tool's default configuration file.

All style flags defining the widget's appearance can also be set over interactive commands.

CHAPTER 2 C/C++ APPLICATION INTEGRATION

BeSpice Wave is written in C/C++. As a consequence the simplest way to integrate the widget is to use the C interface. This requires to include the C header files when compiling and to link to either a shared or a static library. The interface is written in C to make an integration with different compilers possible.

2.1 The Header Files

The headers for using BeSpice Wave widget consists of several files:

- “af_return_codes.h”. This file defines return codes used by all Analog Flavor interfaces.
- “bspwave_widget.h”. This file structures and constants common to all C/C++ BeSpice Wave Widget interfaces.
- The 3rd file depends on the used interface and will be specific to wxWidgets, Tk, etc. For Qt it is the file “bspwave_qt_widget.h” for Windows it is “bspwave_win_widget.h”.
- “af_wp_types.h”. This file defines data types that are used to access BeSpice Wave's waveform parser. The types also allow to exchange data between a plug-in defined by an integrator and BeSpice Wave.
- “af_wp_plugin.h” and “bspwave_plugin.h”. These header files needs to be included to implement a non-graphical plug-in for waveform formats that are not natively supported by BeSpice Wave. The interface can be used in the same way as illustrated in the Waveform Parser API Documentation.
- The interface defined by “af_wp_plugin.h” and “bspwave_plugin.h” can also be used to access the waveform parser from an integrating applications. The interface can be used in the same way as illustrated in the Waveform Parser API Documentation.
- BeSpice Wave can be extended by graphical plug-ins. An additional header file such as “bspwave_qt_plugin.h” for Qt applications define the API for all graphical extensions. The graphical extension does not necessarily have to be define any graphical elements. It can entirely rely on the built-in widgets.

This chapter deals with the common aspects that of all C/C++ integrations. The interface specific aspects will be handled separately.

2.2 The Access Functions

BeSpice Wave is entirely controlled by the following functions:

- A function used to create a BeSpice Wave widget. This function depends on the used interface and returns a pointer to a window or NULL if it fails.
- A function sending interactive commands to BeSpice Wave. These functions allow to open files,

retrieve information on the parsed files, show curves and control the appearance and the behavior of the tool. A more detailed information on interactive commands can be found in BeSpice Wave's user guide.

- A function retrieving the results of interactive commands from BeSpice Wave. The returned value is a NULL terminated list of C strings. The strings remain valid only until the next interactive command sent. Integer return values are printed to strings.
- A function allowing to register callback functions. These functions allow to trigger actions in the main application from events occurring in BeSpice Wave. The registering function is specific to the interface but the registered function prototypes are identical to all interfaces.

All functions are thread-safe. However on most platforms the widget should be created in the main thread.

2.3 Registering Callback Functions

Callback functions are registered using an interface-specific function and the structure “bspwave_widget.h” commonly defined in the header file “bspwave_widget.h”.

The structure defines the following fields :

- The object pointer “callback_object”. This object will be passed as first argument to all callback functions.
- The function pointer “close_button_activated”. If this function is registered, a close button will be added to the widget's tool bar. When the button is hit, this function will be called.
- The function pointer “curve_highlighted”. If this function is registered, it will be called each time a curve is highlighted. The flag “flag_clear” will be set if highlighting is cleared on all previously highlighted curves. The flag “from_browser” will be set if the curve was selected in the browser window.
- The function pointer “curve_unhighlighted”. If this function is registered, it will be called each time a curve is un-highlighted in BeSpice Wave. The flags will be set like for “curve_highlighted”.
- The function pointer “curve_activated”. If this function is registered, it will be called each time the user activates a curve by double-clicking on it. The flags will be set like for “curve_highlighted”.
- The function pointer “logger_message”. If this function is registered, it will be called each time BeSpice Wave emits an error message. The severity can range from 1 (information message) to 6 (debug message).
- The function pointer “status_bar_message”. If this function is registered, it is called each time BeSpice Wave prints a message to the status bar. The message might include a value indicating the progress of an action. This value ranges from 0 to 100.

2.4 Linking the Shared Library

The advantage of linking the shared library is that the tool size will be smaller if the library is used in several tools. In addition the library can be updated without re-compiling the entire tool. However on

Linux this might require to adjust the `LD_LIBRARY_PATH`. Dynamically loaded libraries might be incompatible and copying the tool from one location to another might cause problems.

2.5 Linking the Static Library

Linking the static library avoids all compatibility issues. However updating BeSpice Waves requires to recompile the entire application.

The size of the static library will be considerably reduced when linking the entire application.

CHAPTER 3 THE WINDOWS API WIDGET

The BeSpice Wave Widget has been ported to the native Windows API. This widget can be used in many different environments on Windows. We provide wrappers and examples for a use in a C# WPF, VBS WPF, MFC, Qt, Tk and wxWidgets applications and from the script languages such Tk and Python (TkInter, wxPython and PyQt). Python makes use of the “ctypes” module to access the underlying C library.

The examples have been conceived for an integration on X86 (32 bit Windows). The build instructions might have to be adjusted for X64 (64 bit Windows application).

3.1 Widget Commands

The windows API is defined in the C header file “bspwave_win_widget.h”. The following functions are available:

- **win_bspwave_create_widget**. This function creates a new BeSpice Wave widget. It is safe to call it from a thread. If “NULL” is passed as parent, a top level window is created.
- **win_bspwave_process_command_line**. This function processes the command line arguments such as “--socket <ip> <port>” in an identical way to the main BeSpice Wave application.
- **win_bspwave_execute_command**. This function executes an interactive command. The function returns a code like explained above.
- **win_bspwave_get_command_result** and **win_bspwave_get_command_result_line**. These commands allow to retrieve the results of queries submitted to BeSpice Wave. The first command returns an array of NULL-terminated string. The second command returns a single NULL-terminated string or NULL if the passed index is larger that the number of available result strings.
- The access functions **win_bspwave_show_widget**, **win_bspwave_reparent_widget** and **win_bspwave_move_widget** allow to show or hide the widget, control the widgets size or to reparent it. These commands are especially useful when used from Script.
- The function **win_bspwave_register_callback** allows to register callback functions. They are called by BeSpice Wave when certain actions are triggered and allow the integrator to perform corresponding actions.
- The function **win_bspwave_create_and_run_main** creates a top level BeSpice Wave window with menu bar and runs an event loop. If the argument “flag_run_in_thread” is 1, BeSpice Wave executes in a thread and immediately returns from the function. Otherwise it returns when the window is closed by the user. This represents the simplest way to integrate a BeSpice Wave window to your application.

3.2 Integration to a Windows Application

Examples for the Integration of the BeSpice Wave widget to a native Windows application is given in

the directory

```
.../analog_flavor/examples/bspwave_widget/examples/win_test
```

The examples implemented in the file `mswBspWaveMainTestApp.c` is the simplest possible example for the use of the BeSpice Wave widget library. It simply creates a top level window and runs an event loop.

The example in `mswBspWaveWidgetTestApp.c` creates a top level window and a BeSpice Wave widget as child. A simple menu allows to perform simple actions such as opening a new file or showing the control panel or the command logger.

The `readme.txt` file gives some hints how the examples can be compiled using `cmake`. The makefile generator `cmake` can be downloaded from <http://www.cmake.org>.

3.3 Integration to a WPF Application

By adding a wrapper the Windows API BeSpice Wave widget, it can easily be integrated into a C# or Visual Basic WPF application. Some example files are shown in

```
.../analog_flavor/examples/bspwave_widget/examples/wpf_win_test
```

Integrating to a WPF project requires a wrapper based on `HwndHost` and written in C++ CLR. It is implemented in the files `cliWinBspWaveWrapper.cpp` and `winBspWaveWrapperRuntime.cpp`. These files can be found in your installation tree. The basic steps for integrating BeSpice Wave to your WPF project are the following:

Create a C++ CLR project that you can add to the same solution. In that CLR project you need to compile the files `cliWinBspWaveWrapper.cpp` and `winBspWaveWrapperRuntime.cpp`.

In your C# or VBS project:

- Create a class inheriting from `cliWinBspWaveWrapper` which is implemented in the C++ CLR part of your solution. This class can handle all callbacks emitted by BeSpice Wave in virtual functions.
- Insert a border widget as placeholder window in your project's xaml file.
- Handle the window loaded event by instantiating the BeSpice Wave Wrapper and showing it as child of the placeholder widget.

The example files and the `readme.txt` in

```
.../analog_flavor/examples/bspwave_widget/examples/wpf_win_test
```

give some additional hints.

3.4 Integration to a Qt Application

By adding a simple wrapper the Windows API BeSpice Wave widget, it can easily be integrated into a Windows Qt application. BeSpice Wave is also available as Qt widget but using the Windows widget

allows to create an integration that does not depend on a particular Qt version and does not have to be compiled with a specific compiler. The examples in

`.../analog_flavor/examples/bspwave_widget/examples/qt_win_test`

defines a Qt wrapper for the BeSpice Wave widget. It is implemented in `qtWinBspWaveWrapper.h` and `qtWinBspWaveWrapper.cpp`. The wrapper is used in a simple widget example and in a docked widget example. The wrapper might evolve and should be used by integrators.

Build instructions using `qmake` are given in the file `readme.txt`. They might have to be adjusted to your Qt installation.

3.5 Integration to a wxWidgets Application

Like for Qt, the Windows API BeSpice Wave is wrapped by a C++ class to make it integrable into any Windows wxWidgets application. BeSpice Wave is also available as wx widget but using the Windows widget allows to create an integration that does not depend on a particular wxWidgets version and does not have to be compiled with a specific compiler. The examples in

`.../analog_flavor/examples/bspwave_widget/examples/wx_win_test`

defines a wxWidgets wrapper for the BeSpice Wave widget. It is implemented in `wxWinBspWaveWrapper.h` and `wxWinBspWaveWrapper.cpp`. The wrapper is used in a simple widget example. The wrapper might evolve and should be used by integrators.

Build instructions using `cmake` are given in the file `readme.txt`. They might have to be adjusted to your wxWidgets installation.

3.6 Integration to a Tk Application

The Windows BeSpice Wave widget can be integrated to a Tk Script application on Windows. Therefore the Tk module defined by the shared library

`.../analog_flavor/platform/windows/X86/af_bspwave_widget_tk_win.dll`

must be loaded and used. The module defines the command `af_win_bspwave_widget` that can be used to create a Tk window wrapping the BeSpice Wave widget. The tk script

`.../analog_flavor/examples/bspwave_widget/examples/tk_win_test/example_1_tk_win_bspwave_widget.tcl`

implements a simple Tk application making use of the Tk module and shows how to use the created window to send interactive commands.

3.7 Integration to a Tk Application in C

The above Tk module can also be integrated into a Tk application implemented in C. The directory

`.../analog_flavor/examples/bspwave_widget/examples/tk_win_test_cpp`

contains an example implementing a Tk application in C/C++. The source code for building the library “`af_bspwave_widget_tk_win.dll`” is also included. The directory also contains a `readme.txt` file with

build instructions for cmake and the required CMakeLists.txt file.

3.8 Integration to a Python Script

On Windows there are several possibilities to integrate BeSpice Wave. Your Analog Flavor distribution contains examples for the integration of a simple top window and for the integration into PyQt, wxPython and TkInter. They make use of the “ctypes” Python module. This avoids to have to handle several differences between Python versions. All examples can be found in the directory

.../analog_flavor/examples/bspwave_widget/examples/python_win_test.

The examples are based on the base modules **bspwave_win_interface.py**, **bspwave_qt_interface.py** and **bspwave_wx_interface.py**. These modules might evolve and should be used for all integrations.

The simplest way to integrate BeSpice Wave is to simply create a top window like in example **example_1_py_bspwave_top_window.py**. As BeSpice Wave runs in a thread, the Python interpreter is not blocked.

The Python integrations can't make use of callback functionality.

3.8.1 Integration to a PyQt Application

The file **bspwave_qt_interface.py** implements wrapper widget for integrating BeSpice Wave to a PyQt parent window. The example **example_3_pyqt_bspwave_widget.py** shows how to use the wrapper widget in a simple qt interface. The example **example_4_pyqt_bspwave_dock_widget.py** illustrates the use in a docked widget.

3.8.2 Integration to a wxPython Application

The file **bspwave_wx_interface.py** implements wrapper widget for integrating BeSpice Wave to a wxPython parent window. The example **example_2_wxpython_bspwave_widget.py** shows how to use the wrapper widget.

3.8.3 Integration to a Tk Inter Application

If your Python installation has been configured for TkInter, the Tk module created above can be used from Python code. After having imported the module from **af_bspwave_widget_tk_win.dll** it can be used as shown in example **example_5_tk_inter_bspwave_widget.py**.

CHAPTER 4 TK SCRIPT INTEGRATION

BeSpice Wave Widget can easily be integrated to a Tk script.

Open a Tcl/Tk interpreter window. After having loaded the Tk module using the command:

```
> load ../analog_flavor/platform/Linux/64/libaf_bspwave_widget_tk_8_4.so
```

on Linux or

```
> load ../analog_flavor/platform/windowsX86/laf_bspwave_widget_tk_8_4.dll
```

on Windows. Now the command “af_bspwave_widget” that creates a BeSpice Wave widget is available. The tcl commands

```
> af_bspwave_widget .bspwave -style 71 -width 600 -height 300
```

```
> pack .bspwave -fill both -expand 1
```

will create a BeSpice Wave widget and show it in the main Tk window.

Remark: The file path of the load command might have to be adjusted to your platform.

Remark: The version number 8_4 indicates that the widget can be used for all Tk versions from version 8.4.

4.1 af_bspwave_widget

The command “af_bspwave_widget” creates a BeSpice Wave widget.

4.1.1 Synopsis

af_bspwave_widget pathName ?options?

4.1.2 Widget Specific Options

The widget can be created with the following options:

- **-style** <style> this option can activate a debug option and define if a notebook and a tool bar are used. The integer value for “style” can be computed as explained above. If “71” is used, the debug mode, the notebook and the toolbar are activated. The widget is also instructed to reload a previously saved configuration.
- **-width** <width> defines the width of the widget. The widget's size is adjusted if it is used in a grid or if is packed.
- **-height** <heigh> defines the height of the widget.

4.1.3 Widget Commands

The widget accepts the following commands:

- pathName **execute_command** <command>. This command sends an interactive command the BeSpice Wave widget.

The commands allow to read files, show curves etc. The command execution returns a string value. The integer return values from the c function is converted to a string value such as “command succeeded”, “command failed” and “command queued”.

The interactive commands sent to BeSpice Wave are documented in BeSpice Wave's user guide.

4.2 How the Tk widget works

The BeSpice Wave widget mainly uses a widget written for the Windows API or a widget using the X11 interface for Linux. The widget connects to the TK event loop by registering a generic event handler using “Tk_CreateGenericHandler” and processes all X11 events concerning it's X11 windows. After having been processed, the events are removed from the event loop. They are invisible to all other event handlers.

The widget also processes events concerning drag and drop for the application's top window. However these events are not removed from the event loop.

In addition to this BeSpice Wave registers an error handler. This error handler will catch all X11 errors that occurred in BeSpice Wave. All other errors will be processed by the TK error handler.

4.3 Drag and Drop with Tk

The BeSpice Wave widget can accept drops in text format. However this might not be automatic on Linux. BeSpice Wave uses the XDND protocol which is the standard protocol. See <http://www.newplanetsoftware.com/xdnd/> for more information on the used protocol.

However some applications require that the property “XdndAware” must be set on the top window of the application using drag and drop. As BeSpice Wave is not intended to be the the top window of the application, this must be done by the integrating window.

There are two possibilities how this can be achieved:

- if the application uses TkDND the top window property “XdndAware “ is set as soon as a dorp target is registered.
- Bespice Wave can be instructed to be used as “XdndProxy”. This can be done by the interactive command “use_as_drop_proxy 1” . The command “use_as_drop_proxy 0” removes the “XdndProxy” property.

The integrating application should always call “use_as_drop_proxy 1” if it does not used TkDND.

4.4 Tk Script Example

An example script documenting the Tk script integration is available in the installation tree at

.../analog_flavor/examples/bspwave_widget/Tk/test/bspwave_example.tcl

The example requires Tk 8.5 or later. The tab “Tkndnd” requires the installation of Tkndnd. The example creates a main window with a BeSpice Wave widget and a Tk notebook.

The Tk notebook has several tabs. Each of them handles a different aspect of the communication between Tk script and BeSpice Wave.

4.4.1 The Communication Tab

This tab contains a text entry widget that logs the entire communication process. All interactive commands sent to BeSpice Wave and the returned values are shown here. A button allows to clear the text.

4.4.2 The Commands Tab

This tab contains several buttons. Each button sends a command to BeSpice Wave. The sent command is logged in the “Communication” tab.

Some such as “open_file” return the status “command queued” as the command is executed in a thread and BeSpice Wave does not wait for the result before returning from the function call. The current status can be obtained by calling “get_status”. The result of the previous command is returned the first time “get_status” returns something else than “command queued”.

When used in widget mode, BeSpice Wave does not automatically load and save the user's configuration such as recent files, used fonts etc. Instead this has to be done by the interactive commands “read_configuration_file” and “save_configuration_file”. This allows to specify a file different from the default file.

The command “execute_command_file” allows to execute several commands at once by reading from a text file. This might be important for testing and reporting problems.

4.4.3 The Data Tab

This tab allows to retrieve information on the loaded curve data. The data is organized in sections. Each file generates at least one section but it might generate more than one section. This can occur if a simulator performs multiple simulations for changing parameter values or analysis types.

This tab shows how to read all section names. When selecting a section, all curves available in this section are read. A particular curve can be selected and plotted to a plot with a given name. This operation allows to use wildcards. The plot name “*” makes sure that the curve is plotted. If necessary a new plot is created.

4.4.4 The New Plot Tab

This tab allows to create a new plot. The plot type can be selected. Several plot types allow to split the current page instead of creating a new one. Therefore “create new tab” has to be unchecked.

4.4.5 The Existing Plots Tab

This tab shows how to retrieve the names of all plots of all available plot pages. It allows to switch to existing plots and to clear all curves shown in an individual plot.

4.4.6 The Gui Options Tab

This tab shows how the appearance of the widget can be changed by using interactive commands. The user doesn't have access to this functionality. The different options allow to optimize BeSpice Wave for the space available in the parent application. It also allows to restrict user access to BeSpice Wave's functionality.

4.4.7 The Callbacks Tab

This “Callback” tab allows to register callback functions. These functions can be either functions defined by a script or C functions. Here no particular functionality is implemented for the callbacks. The example simply prints a message indicating that a particular event occurred in BeSpice Wave.

4.4.8 The Drag and Drop Tab

This tab shows how drag and drop can be used to communicate with BeSpice Wave. The above text widget serves a drag source. Instead of sending an interactive command over the usual interface functions, it can be included in a xml statement and be dropped to BeSpice Wave.

The second text widget serves as drop target. Curves dragged from BeSpice Wave and dropped to this widget are converted to a xml string indicating the concerned curves. The curve names might be converted to a “*” if a complete section is concerned.

CHAPTER 5 TK C/C++ APPLICATION INTEGRATION

In a C/C++ application the BeSpice Wave widget can be integrated without using the interpreter. Instead the C functions corresponding to Tk commands can be called directly.

Therefore the C function headers in

```
.../analog_flavor/examples/bspwave_widget/tk/include
```

must be included. On Linux the shared library

```
.../analog_flavor/platform/Linux/64/libaf_bspwave_widget_tk_8_4.so
```

or the static library

```
.../analog_flavor/platform/Linux/64/libaf_bspwave_widget_tk_8_4_static.a
```

must be linked. On windows the shared library

```
.../analog_flavor/bin/libaf_bspwave_widget_tk_8_4.dll
```

must be linked.

Remark: The file path might have to be adjusted to your platform.

Remark: the version number 8_4 indicates that the widget can be used for all Tk versions from version 8.4.

5.1 The Library Access Functions

The headers for using BeSpice Wave widget consists of 3 files:

- “af_return_codes.h”. This file defines return codes used by all Analog Flavor interfaces.
- “bspwave_widget.h”. This file defines structures common to all C/C++ BeSpice Wave Widget interfaces.
- “bspwave_tk_widget.h”. This file defines the Tk specific access functions.

All C/C++ integrations have common aspects that have been detailed [above](#). The function specific to the Tk integration is the function used to create the Tk widget.

5.1.1 tk_bspwave_create_widget_from_path

Synopsis:

```
Tk_Window tk_bspwave_create_widget_from_path(Tcl_Interp *interp, Tk_Window
main_win, const char *window_path, int x, int y, unsigned int width, unsigned int
height, int style_flags, int argc, char **argv)
```

Arguments:

```
Tcl_Interp *interp          the used interpreter
```

Tk_Window main_win	the parent window
const char *window_path	the hierarchical window path
int x, y	the initial position of the widget in the parent window
unsigned int width, height	the initial size of the widget
int style_flags	style flags
int argc	reserved, should be set to 0
char **argv	reserved, should be set to 0

The style flags can be combined from the values

```
AF_BSPWAVE_STYLE_DEBUG  
AF_BSPWAVE_STYLE_USE_TOOLBAR  
AF_BSPWAVE_STYLE_USE_NOTEBOOK
```

defined in “bspwave_Tk_widget.h”.

5.2 Tk Application Example

Several example applications have been included to the distribution. The source code is located at

```
.../analog_flavor/examples/bspwave_widget/tk_test/tk_bspwave_widget_test_x.cpp
```

It can be compiled by adapting and running the build script “build.sh”. This script will build executables all C/C++ examples.

The Tk script example is more complete as it contains a Tk interface showing how interactive commands can be used from Tk. Therefore the script example might be more appropriate to understand the communication process over interactive commands.

test_bspwave_tk_widget_1.c

This example is a pure C example. The other examples use C++ for convenience. The BeSpice Wave Widget is created using the C API function “tk_bspwave_create_widget_from_path”.

test_bspwave_tk_widget_2.cpp

This example calls the Tcl script “bspwave_example.tcl” from a C program . This is a convenient way to debug C and Tcl/Tk code.

test_bspwave_tk_widget_3.cpp

This example is similar to the first one. The difference is that the Tcl command “af_bspwave_widget” is called to create the BeSpice Wave Widget.

test_bspwave_tk_widget_4.cpp

This example uses the tkBspWaveWrapper class to create a BeSpice Wave Widget. This class has some methods that facilitate the use of interactive commands. It also defines some virtual functions that are used as callback functions for some events that might occur in BeSpice Wave.

CHAPTER 6 WIDGET SPECIFIC COMMANDS

The interactive commands sent to BeSpice Wave are documented in BeSpice Wave's user guide. Some interactive commands are only available for the widget integration. These commands are documented here.

6.1 *Dragging Interactive Commands To BeSpice Wave*

Interactive commands can be sent to BeSpice Wave by calling the corresponding C functions. Another possibility consist in using the drag and drop mechanism. A command or a set of commands can be encapsulated into a short xml string such as:

```
<?xml version="1.0" ?>
<af_bspwave_command>
add_plot "new analog plot" analog 1
</af_bspwave_command>
```

If dropped to the BeSpice Wave widget, the command “add_plot” will be executed. The Tk Script example makes use of the module “Tkndnd” to demonstrate this mechanism.

6.2 *Dragging Curve Names From BeSpice Wave*

BeSpice Wave makes use of the drag and drop mechanism to export curves as comma separated values. However when integrated it also allows to drag and drop an xml file defining curve names. Therefore an option must be activated. After having sent the command

```
set_option use_curve_names_for_internal_dnd 1
```

to BeSpice Wave, drag and drop within the tool yields a text string like the following:

```
<?xml version="1.0" ?>
<af_bspwave_curve_list>
<curve section="fourbitadder.spi3" name="v(7)" />
</af_bspwave_curve_list>
```

This allows to implement interactions based on curve names instead on curve values.

CHAPTER 7 ACCESSING THE WAVEFORM PARSER

BeSpice Wave's waveform parser can be accessed directly by any integrating application. The interface is a reduced version of the one described in the Waveform Parser API Documentation. This documentation is available at `.../documentation/waveform_parser_library.html` in your software distribution package. The header file that has to be included depends on the used platform. It is `"bspwave_qt_parser.h"` for Qt.

It takes the pointer to the generated widget as first argument. Some functions from the Waveform Parser API have not been ported. It is not possible to open a new file. However the missing functions are available as interactive commands.

The Waveform parser API must be called from the main thread. Otherwise an error is returned.

CHAPTER 8 EXTENDING BESPICE WAVE'S WAVEFORM PARSER

BeSpice Wave's waveform parser can be extended to waveform formats that are not natively supported. This mechanism has been described in the Waveform Parser API Documentation. This documentation is available at [../documentation/waveform_parser_library.html](#) in your software distribution package. The plug-in has to be registered and unregistered using the functions defined in “bspwave_qt_parser.h” or equivalent.

BeSpice Wave attempts to load all shared libraries located in the same directory as it's binary as plug-ins. This concerns all libraries called “libaf_plugin_....so”.

CHAPTER 9 EXTENDING BESPICE WAVE WITH GRAPHICAL PLUG-INS

BeSpice Wave can be extended by writing graphical plug-ins. These plug-ins do not necessarily have to define any graphical elements themselves. To do so, include “bspwave_qt_plugin.h” or equivalent.

The graphical plug-in can also make use of the waveform parser. The mechanism has been described in the Waveform Parser API Documentation. This documentation is available at [.../documentation/waveform_parser_library.html](#) in your software distribution package. All functions and objects it can use for this purpose are passed to it in the function “register_waveform_parser_functions”.

BeSpice Wave attempts to load all shared libraries located in the same directory as it's binary as plug-ins. This concerns all libraries called “libaf_plugin_....so”.

CHAPTER 10 TROUBLESHOOTING

This section handles some issues that might occur when integrating BeSpice Wave as a widget.

10.1 Limitations

The BeSpice Wave widget has some limitations over the complete tool.

- The Tcl/Tk widget requires a Tcl/Tk version with thread support to run correctly.

10.2 Reporting Errors

When reporting errors following some recommendations might simplify support.

If the problem occurred for a specific waveform file, please include this file to your request.

If the problem concerns the graphical user interface, please include a screen shot.

If the problem occurred while executing interactive commands it might be possible to reproduce it by generating a command file with the corresponding commands.

10.3 License Problems

Running BeSpice Wave requires a license file. This file should be included in your software package. However when moving the libraries or linking the static libraries the license file might no longer be found. If BeSpice Wave does not find the license file “license.txt” or “af_license.txt” please copy it to the location of the linked shared library or to the location of the linked application. Alternatively it can be copied to a parent directory or to a directory named “license” in a parent directory. However BeSpice Wave only searches up to 5 parent directories.