VST-Plugin Unit Test

Version 1.2.0

Welcome

Thank you for downloading this tool. **VST-Plugin Unit Test** is an Extreme Programming unit test for generic VST-Plugins.

In order to get the most out of the **VST-Plugin Unit Test**, please spend a few moments reading this brief manual.

License

The pre-compiled **VST-Plugin Unit Test** has a very simple license:

1. **VST-Plugin Unit Test** is freeware. This means that you are free to use this plugin in any context. Also you are free to share it on a personal base (ie. give it to friends). However, only the entire unaltered archive, including this document, may be shared. Public redistribution is only allowed on request.

2. Copyright of the code and the pre-compiled plug-in remain property of the *Delphi ASIO & VST Project* and namely *Christian-W. Budde*.

3. This plug-in is provided at no cost; therefore the author *Christian-W. Budde* assume no responsibility for any negative effects that may occur to the end user or the equipment used to run the plug-in.

4. Magazine editors are welcome to include the plug-in on cover mount discs or similar media; However, it is mandatory to inform the author *Christian-W. Budde* about this. A copy of the publication is always appreciated, but not expected.

User Interface

Since this application is based on the DUnit framework, the user interface is essentially identical (except a splash screen shown while the plugins are scanned).

Here is a screenshot:

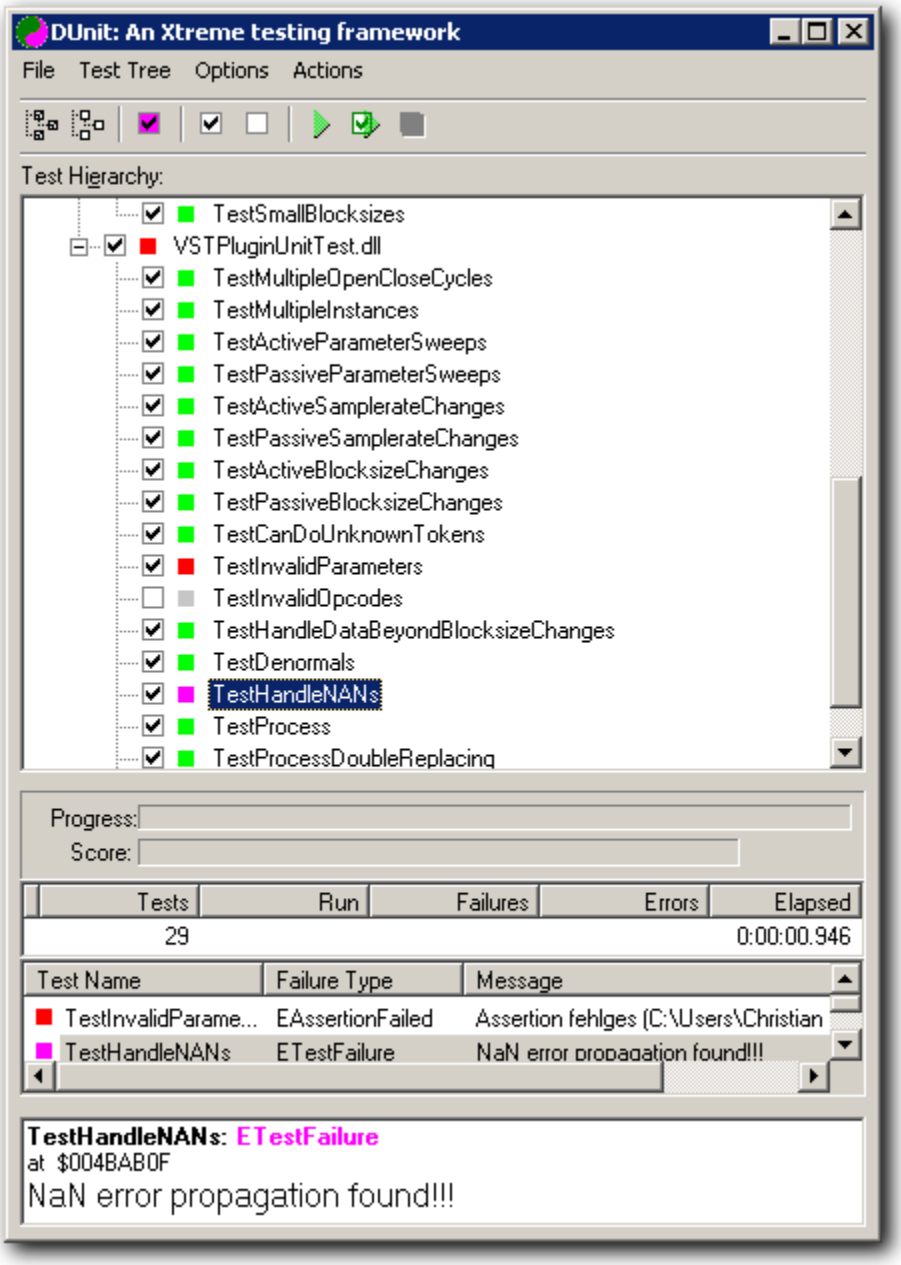

Figure 1: VST-Plugin Unit Test Screenshot with some performed tests. The test in red failed, while the test in pink revealed a problem.

Within the 'Test Hierachy' window different tests can be selected. By right clicking the window it is possible to select/deselect all. Also it is possible to select/deselect only the current or select the failed only. The selection will be stored across the sessions.

After a selection has been made, the tests can be started via the tool bar, the main menu or using [F9].

Each test is performed one by one. If one test fails the other tests will still be performed. Keep in mind that in case of an error it might happen that subsequent tests might fail due to corrupted memory.

All for any test the result is reported by a small coloured square beside the name. Green means the test passed without any remarks. In case of a pink square an internal check failed. If the test crashed it will be shown as a red square. Skipped tests have a grey square.

Below the 'Test Hierachy' two bars can be found. One identifies the progress while the other gives a score for the selected tests. A score of 100% equals all tests passed.

Further below a log window can be found. It contains all problems found (checks and errors). Details of each problem are presented in the window on the bottom. Study the details carefully to spot whether the problem is reasonable.

Also the time elapsed can be seen either for the whole set of tests or for each test individually.

Get started

In order to get the program started correctly you should place it either in any VST-Plugin directory or pass the name of the VST-Plugin to be tested as parameter.

The application works well in combination with debuggers and it can be used for testing during the development.

The tests

Right now there are three types of tests. First the basic interface is tested, second several I/O related things and last but not least some thread processing related issues are tested. In this section the tests are explained in detail

Basic Tests

TestMultipleOpenCloseCycles

For this test the plugin is rapidly opened and closed for 10 times. This should not result in any problem. At least if no second thread is started by the VST-Plugin under test ('VUT')

TestMultipleInstances

For this test 33 instances are created and closed in a different, random order. It does not contain any check, but some plugins have been found that crashed within this test situation.

TestActiveParameterSweeps

For this test all available parameters are stepped through with a step-width of 0.01. According to the specifications the plugin has been opened (Opcode: effOpen) prior to the test.

TestPassiveParameterSweeps

For this test all available parameters are stepped through with a step-width of 0.01. In contrast to the test above, the plugin has not been activated (opened) before the parameter changes take place. Ideally the plugin should ignore this calls to prevent any crashes.

TestActiveSamplerateChanges

The samplerate of the VUT is changed in a range between 1 Hz and 1411200 Hz (exponentially). The plugin is activated. This shouldn't result in an error. However, it might take some time depending on the calculations the plugin performs.

TestPassiveSamplerateChanges

The samplerate of the VUT is changed in a range between 1 Hz and 1411200 Hz (exponentially). In contrast to the test above, the plugin has not been activated (opened). Ideally the plugin should ignore this calls and return immediately.

TestActiveBlocksizeChanges

The block size is changed starting from 0 to 64 in 1 sample steps for small blocks. Medium blocks between 19 and 1216 (in steps of 19) samples and large blocks between 1025 and 65600 (in steps of 1025) samples are tested. No processing takes place. Ideally this test should run without any remarks. Eventually an error could occur due to the block size of 0, but since no processing takes place this is not vitally.

TestPassiveBlocksizeChanges

The same test as above, but without the plugin activated. Again this should run without any remarks.

TestCanDoUnknownTokens

This test performs several queries for non existing CanDos. First an empty string is queried. Second the predefined, customized and long string '2zcrfo3874zrbiwrbgrvsdrgbviwztvi374vöowiurzbi4t7zviw74tvposihfopit' is queried. Last but not least a random artificial string of 1112 characters is queried. The plugin should ignore all these calls.

TestInvalidOpcodes

This test is one of the most evil tests. It mauls the plugin with several invalid opcodes. However, for all these opcodes and combinations it's possible to find a solution that does not lead to a crash of the plugin. Here's a list of the dispatcher calls (if not specified the arguments are zero!):

```
VstDispatch(effSetEditKnobMode, 0, 3);
VstDispatch(effSetViewPosition, -249824962529, -300013512459);
VstDispatch(effSetSpeakerArrangement);
VstDispatch(effOfflineNotify);
VstDispatch(effOfflinePrepare);
VstDispatch(effOfflineRun);
VstDispatch(effSetSampleRate, 0, 0, nil, 0);
VstDispatch(effSetSampleRate, 0, 0, nil, -44100);
VstDispatch(effShellGetNextPlugin);
```

Furthermore all not yet specified opcodes above 128 to 2000 are called. They should be ignored completely to avoid problems with future hosts. All other calls should at least not lead to a crash.

TestInvalidParameters

Within this test non existing parameters are queried. First the parameter just above the last parameter is queried and then a random integer parameter. Next a value of 10 (outside the range!) is set to parameter 0, 'numParam' and a random integer parameter. Furthermore the parameter name, label, display and properties are queried for invalid parameters.

TestString2Parameter

Within this test the optional string to parameter opcode is tested. The test will pick a random value for any parameter, store its display and label text and, change the parameter to a different value and test whether passing the stored display and label text will lead to the initial parameter value (if this conversion is supported).

I/O Tests

The I/O tests include calls for data processing such as ProcessReplacing, Process and ProcessDoubleReplacing.

TestHandleDataBeyondBlocksizeChanges

A block size of typically 909 samples is set, but processing of more samples takes place. According to the specifications this is invalid, but developers might want to ensure that this doesn't lead to crashes. Strategies to avoid crashes are either to ignore the block size or limit the number of processed samples to block size samples.

TestDenormals

This test forces the appearance of denormals if filters are present in the VUT. All output samples are checked for denormal values. No crash should occur, but it is likely that denormal values can be found.

TestHandleNANs

The VUT is feed with a single NaN value (followed by zeroes). Although this scenario is invalid it should not result in a crash (as the plugin might be used for real time processing). The perfect solution to handle this situation

is setting the NaN directly to zero or any other reasonable value. This way no successive plugin will be affected nor error propagation will take place.

TestProcess

In some hosts (Logic 5.5) the Process() function is still used [rather than the ProcessReplacing()]. However, since it is deprecated in VST 2.4 this will lead to a crash (if not catched). The first call to this function passes nil pointers but also 0 sample frames. No processing or crash should happen. A second test will process 'block size' samples of noise.

TestProcessDoubleReplacing

In the VST 2.4 specification the ProcessDoubleReplacing() is introduced. Special flags and switches ensure this is used rather than the ProcessReplacing() function. However, this test ignores the flags and calls the ProcessDoubleReplacing() right away. The perfect solution by the VUT would be to ignore the call. Especially the first call as it is without pointers to any I/O buffers, but also with 0 sample frames.

TestSmallBlocksizes

Very small block sizes are used for processing. This should not make any trouble.

Threaded I/O Tests

Usually the data processing is handled by another thread, while the GUI operates independent and can change settings while processing takes place (without synchronization). Several tests can be performed to verify the VUT can handle critical situations.

TestRandomParameterChangesWhileProcessing

In this test random values are assigned to parameters while the plugin is processing data. At least 81920 are processed in chunks of 8192 samples. This should not lead to any crash. However no checks are performed regarding invalid states (such as NaNs) yet.

TestSamplerateChangesWhileProcessing

In this test the samplerate is changed randomly (in a range of 10 kHz to 100 kHz) while processing takes place. This should not lead to a crash.

Again no checks are performed to ensure the I/O stream is valid (e.g. no NaNs).

TestProcessCallWhileProcessing

A 'zero data' (nil pointer, 0 sample frames) call to the Process() function is done while processing takes place. Again no crash should occur.

Feedback / Bug Reports

I am always eager to hear feedback or have bugs reported. The easiest way is to send me a mail to: Christian@aixcoustic.com

Furthermore feel free to download the source code, that can be found in the Delphi ASIO & VST Project at sourceforge.net.

Version History

1.0.0	First release!
1.1.0	Misc. bugfixes and several new tests
1.2.0	Fixed host emulation bug, new tests

Credits

- Programming: Christian W. Budde
- Additional Framework Programming: Tobias Fleischer, Maik Menz
- DUnit Framework: http://dunit.sourceforge.net/
- Special Thanks: Swen Müller, Duncan Parsons, Laurent de Soras
- Documentation based on a template by Greg Pettit