

## **Tobybear presents: VST Plugin Template and Tutorial for Borland Delphi**

### **1. Introduction**

Hello everyone! My name is Tobias Fleischer alias Tobybear and I develop audio plugins for the VST plugin interface created by Steinberg Media Technologies GmbH (<http://www.steinberg.net>). You may have seen or heard my freeware and shareware audio plugins at <http://www.tobybear.de>.

In this document, together with the template files that should have come with it, I will show you how you can create VST plugins and synths with Borland Delphi compilers (<http://www.borland.com>). At the moment, these plugins are for the Microsoft Windows platforms (Windows 95, 98, ME, 2000, XP) only, as there is no Delphi version for the Mac (yet?).

#### **Disclaimer and Credits**

I am not affiliated with or working for Steinberg, Borland or any other company, and all information was purely written out of private interest and is given at best knowledge to the public domain. I cannot guarantee that everything is 100% right and working on every system under every condition - you use this at your own risk! If you encounter a problem, please contact me (see end of document for contact info).

The original VST SDK for C++ is published/distributed by Steinberg free of charge. Just take a look at their website (<http://www.steinberg.net>), somewhere under "3rd party developers" you should find the relevant links. Even if the SDK is free to download, there is still a license agreement attached to it, which I also have also included in this archive. Please read this Steinberg VST license first, and if you don't agree, don't develop plugins with this SDK :-)

**The latest official Steinberg SDK online documentation as well as further important info can be found on this site: <http://ygrabit.steinberg.de>**

I did not include the page contents with this template because of possible copyright violations and for the sake of up-to-date info, so be sure to check this site out!

The C++ SDK was initially translated to Delphi format by Frederic Vanmol. At his site <http://www.axiworld.be> you can find this translation as well as some converted Steinberg SDK examples. My Delphi VST template is based on the SDK conversion files from Frederic that I extended a bit. Everything you need to start coding plugins in Delphi is included with this download (except the Delphi compiler), so you can start right away.

### **What is Delphi?**

Most compilers on the market are for the C++ language, but the Delphi language (sometimes referred to as Object Pascal) is the successor of Turbo Pascal and equally powerful! In terms of speed, performance and especially RAD (rapid application development), Delphi is in my opinion often even superior to for example Microsoft VC++. Since there is hardly any information about coding DSP/VST stuff in Delphi, I decided to give it a go and publish this tutorial/template. If you still think Delphi is not as capable or fast as C++, take a look at the dozens of plugins on my site, they are all written in Borland Delphi and some even come with full source code.

### **What Delphi version can I use and where can I get it?**

You can use any Delphi version from 3 onward for VST development, but for ease of use, I would strongly recommend Delphi 5 or better. There is a free full version of Delphi 6 (the so called "Personal Edition") available online from Borland, not restricted and free for personal use, look for it on their public FTP server. With the release of the new Delphi 7, there is a free edition as well, but it is distributed only on computer magazine CDs at the moment, at least as far as I know.

Look around, many magazines all over the world have Delphi 7 for the price of the magazine on their cover disks, but maybe you are also able to grab a Delphi PE version from a Borland server. This VST template was developed with Delphi 6 by the way. Most certainly you can also buy Delphi from Borland directly or in your favorite shop. but be sure to check out the correct version, as the price tag usually lays between \$20 and \$4000! For VST development, the simplest version should suffice though.

## 2. Installation

### What software do I need?

- a Delphi compiler (available for free or very cheap these days)
- the VST SDK from Steinberg (Delphi version is already included in the download)
- optionally the DIB components for great graphical interface controls  
(also included in the archive)
- a VST host to test your plugins
- endurance and good nerves :-)

### What is included with this download?

\CommonVST	The VST SDK headers/units
\Demo_DIB	A demo plugin using the DIB components
\Demo_VU	A demo plugin using a VU meter
\Demo_MidiPlugin	A demo for a MIDI-only VST plugin
\Demo_NoGUI	A demo for a filter plugin without GUI
\DIBControls	Pete Morris' excellent DIB controls
\Manual and Tutorial	This document
\Steinberg VST-SDK Documentation	The SDK documentation from Steinberg
\Steinberg VST-SDK License	The VST license from Steinberg
\Template	The actual Delphi VST template

## **How do I install and use the template?**

Good news, no installation procedure is necessary! Just go to the "Template" folder of the unpacked archive and double-click on "VstPlugin.dpr" to launch Delphi with the VST template!

The only thing you must do: go to "Tools - Environment options - Library path" and add the folder "CommonVST" from the extracted archive to the search path.

Then press F9 to build the project and generate a file "VstPlugin.dll" in the current folder - your new VST plugin! Copy this into the "vstplugins" folder of your host software and load it to see if it works. You can also change the default output folder for the current project so that the compiled DLL is automatically created in your preferred folder. It might also be a good idea to copy the template folder to a new folder for each new plugin you want to create. This is what I did to create the included demos, by the way.

## **3. Getting started**

### **The Basics**

Now you have your Delphi compiler in front of you, ready to go, but where to? Before going anywhere, you should first have a basic understanding of what a VST plugin is and how it works.

I have included the original Steinberg VST 2 documentation (as PDF and HTML), but the examples given there are of course in C++ and actually there are lots of things explained in detail that you do not have to care about with my Delphi template. Nevertheless, you should at least look into it once :-)

I try now to give my own explanation of the internal structure of a VST plugin as it is used in my Delphi template:

Although concealed from the outside, you usually have two "threads" (=programs, tasks) running at the same time in one VST plugin: a plugin thread and an editor thread. The plugin thread is the program that implements the actual audio processing algorithm and the parameter management, the editor thread is a program that simply

presents an interface to the user. These "threads" are running independently from each other and can synchronize/communicate/exchange parameters via special functions (we will come to that later).

This has several advantages:

- You can build a plugin without an editor/graphical interface - okay, this is quite uncommon these days, but most hosts have an internal standard parameter display which is used in this case
- The plugin can process audio even if the editor window is closed
- The editor thread can be designed and developed independently and tested as a standalone program by itself before being incorporated into the plugin

### **How do I access the files associated with those two threads?**

In my template, there are two units, `uPlugin.pas` and `uEditor.pas`, which represent the two threads mentioned above. Use the tab controls or press Ctrl-F12 to browse to `uEditor.pas` for example, then press F12 to display the interface. Here you can freely rearrange things, add new components or change properties.

Now go to `uPlugin.pas`: Phew! Lots of complicated stuff inside! But don't worry, you definitely do not need to understand everything at once!

### **Basic configuration**

In `uPlugin.pas` there is a procedure called `APlugin.Create`. This is the constructor of the plugin and contains necessary information for the VST host like how many parameters and audio input/output channels the plugin has.

Let's go through the most important settings:

- There is a set variable `canDos` that defines some of the plugin's "abilities". In the template code (and also in `DVstTemplate.pas`), all possible "canDos" are listed and you can include the ones you need. For further "canDo" information, please check the original Steinberg SDK documentation. Normally, you can leave the settings as they are (`receiveVstEvents` and

`receiveVstMidiEvent`: plugin can receive MIDI events, `sendVstEvents` and `sendVstMidiEvent`: plugin can send MIDI events).

- The next set variable `properties` defines some more properties for the plugin. Again, you can leave this at the default values for most cases and check the SDK for more info about these settings. One thing is important though: if `prIsSynth` is included in the `properties`, your plugin will be recognized as a VSTi (instrument/synth) by the host, else as a normal VST effect plugin.
- The `numInputs` and `numOutputs` variables define the number of audio inputs and outputs
- The `numPrograms` and `numParameters` variables define the number of programs (= presets) and parameters.
- The `UniqueID` identifier is a four-character string that should be unique for every plugin you are developing.

*From the Steinberg SDK:*

"The host use the unique identifier to identify the plug-in, for instance when it is loading effect programs and banks. It has to be different for every plugin. Be inventive; you can be pretty certain that someone already has used 'DLAY'."

- The `EffectName`, `ProductName` and `VendorVersion` strings contain more information about your plugin, the `VendorVersion` variable contains the version number.
- The `initialDelay` variable is useful if your plugin introduces a fixed sample delay and you want the host to compensate for this. Just set this variable to the delay (in samples) that your plugin produces - not all hosts pay attention to this value though. Note that variable delay compensation is not possible in any of the hosts I know, therefore you can't change the delay lateron.

After these settings, the constructor of the base class is called with:

```
inherited Create(audiomaster, numPrograms, numParameters,  
    TPluginEditorWindow);
```

You can replace "TPluginEditorWindow" with "nil" if you do not want an own GUI editor window and use the host's default representation

In my template, I also added two useful event handlers (more on them later):

```
OnParameterChange := ParameterChanged;  
OnEditorIdle := EditorIdle;
```

### How do I process audio?

There is a procedure `processAudio` at the end of `uPlugin.pas`. This procedure is automatically called whenever there is audio data to be processed. In VST, audio inputs and outputs are always processed in the same call, so basically you must fill all the samples in all output buffers with your data. This data can be generated by an oscillator for example (for a VST synthesizer) or it is a processed version of the input buffer (for a VST effect). You get three parameters in the `processAudio` procedure:

- `sampleframes` tells you how many samples to process (all input and output buffers have the same size during each call to `processAudio`).
- `inputs` and `outputs` are arrays of single arrays containing the input buffers (which should only be read from) and the output buffers (which should only be written to). `inputs[0]` is the first input channel and contains audio data from `inputs[0][0]` to `inputs[0][sampleframes-1]`, `inputs[1]` is the second input channel and contains audio data from `inputs[1][0]` to `inputs[1][sampleframes-1]`. `outputs[0]` is the first output channel (from `outputs[0][0]` to `outputs[0][sampleframes-1]`), `outputs[1]` is the second audio output channel (`outputs[1][0]` to `outputs[1][sampleframes-1]`) and so on...

## What about parameters?

Naturally, most algorithms only make sense if there is some way to influence parameters like the cutoff frequency of a filter or the panorama position.

You define the number of parameters in the constructor of the plugin with the `numParameters` variable. In the regular VST SDK, you can only have parameters with a floating point range of 0 to 1 (normalized floating point representation). In my template you can additionally use an extended interface that allows mapping this to any range, adding parameter smoothing, exponential and logarithmic mappings and proper names and units.

More on my extensions later, let's look at the normal 0..1 parameters first:

The parameters need to be initialized to default values, which is done in the `APlugin.initializeParameters` procedure in `uPlugin.pas`.

Just call `setParameter(index, val)` for each parameter, where `index` is the number of parameter you want to change (in the range of 0 .. `numParameters - 1`) and `val` is the value for it (in the range 0 .. 1). You can also call `setParameter` later on, whenever you need to change the value of a parameter.

If you need to query the current value of a parameter, just call `getParameter(index)` to get the value of parameter `index`. For a filter, you might for example call this function for parameter 0 and 1 at the beginning of each `processAudio` call to determine the current values for cutoff and resonance.

As we will see later, a parameter is often changed by the user (when he is moving a slider on the GUI for example). It is therefore often necessary to be notified whenever a parameter changes to a new value. The `OnParameterChange` event does just this, and you can write an event handler to react whenever parameter `index` has changed. In my template, I already connected this event to an event handler called `ParameterChanged`.

Now on to my own parameter interface extensions!

In `APlugin.initializeParameters` you can also define some basic properties for parameters by accessing and changing the various fields of the `ParameterProperties` record.



Here is an example:

```
ParameterProperties[0].name := 'Volume';
ParameterProperties[0].min := -90;
ParameterProperties[0].max := 0;
ParameterProperties[0].units := 'dB(fs)';
ParameterProperties[0].curve := ctExponential;
ParameterProperties[0].curveFactor := 1000;
ParameterProperties[0].smoothingFactor := 0.1;
```

The settings above change the properties of parameter 0 in the following way:

- `name` is the name of the parameter, or rather what it controls.
- `min` and `max` define new upper and lower limits (instead of 0 .. 1). Here I used a range of -90 to 0. If you don't define these values, the default 0..1 mapping is used.
- `units` is a string defining what units are used for expressing the parameter. This only affects the graphical representation, eg. here 0..-90 dB(fs)
- `curve` describes the mapping to use, available are `ctLinear` (default), `ctExponential` and `ctLogarithmic`. You can further influence the form of the curve by changing the `curveFactor`.
- Finally, the `smoothingFactor` controls how fast a parameter reacts to sudden value changes. A value of 1 (default) means the new value is applied immediately, while smaller values (0..1) make the parameter approach the new value smoothly over a certain time (the smaller the factor, the longer it takes to reach the new value).

To use this new parameter interface, you have to call the procedures `setParameterEx` and `getParameterEx` instead. They work the same as their `setParameter` and `getParameter` counterparts, except that you can now use the extended features. So you can set the volume to -35 dB by simply calling `setParameterEx(0, -35)`.

The regular function do also still work (albeit still within the 0..1 range) so it is no problem to use them together.

For example, these two calls achieve the same thing for the properties defined in the example above:

```
setParameter(0, 1);  
setParameterEx(0, 0);
```

The first method sets parameter 0 to maximum ("1" equals 100%). The second method sets parameter 0 to a value of 0 dB(fs).

### **How can the user control the parameters?**

Well, now it's getting interesting...

First of all, any parameter you define will automatically be accessible as an automation target from the host side, so you can for example draw curves for the cutoff frequency in Steinberg Cubase SX.

But of course, the most common use of parameters is that they are controlled by some components on the plugin GUI like sliders, dials, knobs, buttons, menus, etc.

Therefore, you will need to open the GUI part of the template by pressing Ctrl-F12 and choosing `uEditor.pas` to display. Press F12 to show the GUI form and add any control you like. You can use the standard VCL controls or any other component. I would recommend Pete Morris' excellent DIBControls components that are also included with this download (more on them later). In the template, I just used some standard scrollbars.

The most important thing to do is to connect the control with the parameter of the plugin's audio thread in such a way that

- the control always represents the current state of the parameter
- changing the control automatically results in an update of the parameter

There is of course the danger of a deadlock (or endless loop): changing the scrollbar results in updating the parameter which results in updating the scrollbar which results in updating the parameter... I used the `manually` variable in the editor to prevent this deadlock. Basically, I just call `effect.setParameter(index, value)` or `effect.setParameterAutomated(index, value)` to update the parameter with the new value whenever the user moves the scrollbar handle (check

`TPluginEditorWindow.par0Change` function). The `setParameterAutomated` procedure is different to `setParameter` in that it also provides automation data for the host which is generally a good idea, so this is the function to prefer.

I also put a timer on the GUI to update the scrollbar display whenever the parameter changes in the plugin thread. This is for example the case when a new preset or bank is loaded. In the timer, we check the current parameter value (with `effect.getParameter`) and compare it to the state of the scrollbar. If it is different, the scrollbar is updated.

Please note that I convert the scrollbar value (range: 0..100) to the normalized representation (range: 0..1) and vice-versa because I use the regular `setParameter` and `getParameter` functions for this example.

### **What is `OnEditorIdle` used for?**

There is another way to exchange data between the plugin and editor thread: the `OnEditorIdle` event. This event happens whenever the editor window is open and the host/plugin is able to safely update the GUI. In my template, I already added an event handler `APlugin.editorIdle`, where you can access both the editor form and the plugin parameters at the same time. This is useful if you want to update a control in the editor to reflect for example the state of a variable inside the algorithm (like a LED light to go on if the signal in the algorithm during the last processing block was clipped). In the `Demo_VU` project that is also included with this download, I do for example calculate and store the average audio energy in the `processAudio` function and in `APlugin.editorIdle`, I display a VU meter according to this value.

Keep in mind that `editorIdle` might be called very frequently. It depends on the host, but it can be called several times per processing block. If your calculations don't need to happen that often, then you should create some sort of flag so that your idle routine will know whether to do what it's supposed to do or not. I did choose the timer method for updating the editor GUI display because it works on a time base I can define (the timer interval), while the frequency of the `editorIdle` calls can differ from host to host

## 4. MIDI control

### How and where is MIDI data handled in the plugin?

Okay, actually there is no real "MIDI" data, but generally "events" in VST terms. This can be video, audio, MIDI, trigger, sysex, control events..., although currently only MIDI type of events make sense. The event handler can be found in the `processMIDI` procedure of my template. Just set/update the appropriate parameters based upon the MIDI messages you received and you have MIDI control for your plugin in an instant!

### Can an effect plugin receive MIDI input?

Yes, this is part of the VST 2.0 specifications and it works quite well, though not every hosts might support it.

### Can I make a plugin that is both instrument and effect?

Basically no, but at least with hosts that can send MIDI to effect plugins you could just make your plugin an effect plugin, then it will show up as an insert effect and you could route your MIDI signals to it. In most hosts, you will lose the possibility to have multiple audio outputs then though.

Another alternative: compile two versions of your plugin, one as a synth, one as an effect. This is done by including/excluding `prIsSynth` in the plugin properties, in the constructor of your plugin. Of course it sounds a bit strange to have two versions of your plugin which only differ in this small point, but that is the most commonly used method, even by major companies like Native Instruments.

### Can a plugin/synth send MIDI data to the host?

Yes, it can, although it is highly dependent upon the host what happend with this data and where it is sent to! In my template, I defined some high-level functions for this:

```
procedure MIDI_Out(b1, b2, b3, b4, offset);  
procedure MIDI_CC(ch, num, val, offset);  
procedure MIDI_ChannelAftertouch(ch, val, offset);  
procedure MIDI_NoteOff(ch, note, val, offset);
```

```
procedure MIDI_NoteOn(ch, note, val, offset);  
procedure MIDI_PitchBend(ch, val, offset);  
procedure MIDI_PitchBend2(ch, x1, x2, offset);  
procedure MIDI_PolyAftertouch(ch, note, val, offset);  
procedure MIDI_ProgramChange(ch, val, offset);
```

MIDI sending works best in hosts that allow routing of the MIDI input channels (including the internal event bus), so far it works limitedly with Steinberg Cubase 5.x and really good with Cubase SX and Plogue Bidule, maybe also others.

### **Can I code a MIDI-only plugin?**

With the current version of the VST standard, you can theoretically code MIDI-only plugins by setting the number of audio in and out ports to zero and only use the `processMIDI()` function for MIDI processing. But not all hosts support such a non-audio/MIDI-only plugin, it works for example in Plogue Bidule but not in Steinberg Cubase.

The best solution at the moment is to define a regular audio effect plugin (eg 2-ch audio in, 2-ch audio out) that can receive and send MIDI. The audio part is simply ignored, either by letting the signal pass through or by muting it, the MIDI is processed as described above. You can then add this plugin as a channel insert on any audio channel for example in Cubase and route the MIDI ins and outs as you like.

The MIDI plugins supplied by Steinberg in Cubase SX (Step Designer, Arpatche, ...) are NOT coded with the VST SDK, but with another SDK that has recently been made public.

### **How do I receive the current tempo info from the host?**

The template already contains a variable `tempo` that contains the current bpm (beats per minute) value. This value is automatically synchronized if the host tempo changes.

### **How do I convert MIDI notes to frequency and vice versa?**

I have written a Delphi library for several of those conversions. It is available on my site <http://www.tobybear.de>, just look in the "Developers" section for the "Tunings" section.

### **Can I use different tunings/scales for my synths?**

This is a topic very often neglected by developers! The possibility to change the scale (ie what MIDI note corresponds to what audio/oscillator frequency) greatly enhances a synths feature list and is often requested by users. I converted Mark Henning's "Tuningmap" class (<http://www.anamark.de>) to Delphi and added a converter for files created with the widely used "Scala" program. Just download the file from my site, then you will be able to read and use Anamark, VAZ and Scala scale files from your synthesizer plugin!

## 5. DIBControls

### What are the DIBControls?

Included with this release are the great DIB freeware graphic control components for Delphi 5-8 by Peter Morris. In my opinion, these controls are one of the fastest and best controls ever for GUI design: transparency, masks, realtime movements, rotation, zooming, animations, sliders, dials, knobs, buttons, windows, almost everything is possible and really easy to do once you figured it out. You can get the latest release of DIB, some demos and templates as well as some good DIB tutorial AVI videos from his site <http://www.droopyeyes.com>, but the most up-to-date version should be included with this document (file "dibcontrols.zip" in the directory "DIBControls").

### How do I install the DIB graphic components?

This is quite easy and should normally work without problems!

Note: it is not mandatory to install those components, only if you want to use them for your GUIs, but I seriously suggest you should do so.

1. Create a folder on your disk named "DIB" or similar (in this example we will use "C:\Delphi\DIB"), then extract all of the files from the archive "dibcontrols.zip" into that folder (with subdirectories).

2. In Windows-Explorer, double-click on the appropriate DPK install package for your Delphi version, ie "D6DIBInstall.dpk": Delphi should open with the package editor. Now click "Compile" and then "Install" in the package window. After the install process you should get a message telling you what new components were installed.

3. In Delphi, go to "Tools - Environment options - Library path" and add the following directories (you may of course have different directory names):

"C:\Delphi\DIB"

"C:\Delphi\DIB\OpenSource"

"C:\Delphi\DIB\PictureFormats"

### **Where is the DIB component documentation?**

On Pete Morris' website <http://www.droopyeyes.com> you will find some demo applications and AVI tutorials (plus the needed codec to play them) that shows how to add them to your project. The principle behind it is very simple, but there is unfortunately no manual. So, if you have found out something cool to do with the DIB components, don't hesitate to share it with the world.

### **How can I reduce the DLL filesize when using the DIB components?**

You might have noticed that the resulting compiled DLL with DIB support is reasonably larger in filesize than the one without DIB components, but there is an easy solution: once you have a project ready for release click on the "DIBsettings1" icon on the template form in `uEditor.pas` and choose "DIB LZH" as DIBcompressor in the properties tab of that component, then save the form.

This saving will take longer than usual, but the filesize will normally be even smaller than the regular Delphi form. If you have lots of elements and DIBs on your form, it is a good idea to turn compression off during design of the form layout and follow the above procedures only as a final step to compress the release version of a plugin.

Using 16 or 256 colored bitmaps instead of 24-bit does of course also reduce the amount needed for storing the resources.

Generally, you can also try to compress the resulting DLL with an executable compressor, filesize will get even smaller and you won't notice the on-the-fly-unpacking, although you have to make sure that the exepacker you used works on all operating systems.



## 6. Some Frequently Asked Questions

### What is this Pentium 4 denormal bug?

A lot of audio programs and plugins currently have problems with the Pentium 4: occasionally high CPU spikes (sometimes over 500%) occur, even when there is no audio processing taking place, sometimes crashing the whole system because of system overload! This affects many shareware plugins, but also the big companies, Steinberg Grand and Waldorf Attack for example. The reason behind it is an Intel "bug" (or probably "feature" in their terms): the P4 goes into a special "super-precise" mode called the "denormalisation" mode once floating point numbers get very small. This is normal and almost every other CPU dealing with floating point numbers does this too.

Unfortunately, calculations in this mode take up much more of the CPU processing power and are much slower, so for realtime applications a CPU should not spend too much time in this "denormal mode".

There is no sure way to switch off this behaviour, at least none as far as I know. The problem existed on the Pentium 3 too, and there were workarounds like adding some "quiet" noise to the signal to keep the floating point numbers always above the threshold. On the Pentium 4 this threshold is much higher though: from my experiments it changed from about  $1.0e-25$  to  $1.0e-24$ , that is a factor of 10! Which does of course mean that the program stays 10 times longer in this "denormal" mode, causing some programs which run smoothly on the P3 to stutter or even crash on a Pentium 4!

There seems to be a new generation of P4s that allegedly do not have this pronounced denormal effect (still worse than the performance of an AMD though), but don't forget: your algorithm should *\*always\** be checked for denormalisation problems, as it is a general problem with floating point numbers on almost any processor. There are excellent resources about this topic, it would probably be a good idea to go to <http://www.musicdsp.org> and read the various papers there.

### How can I avoid the denormal problem in my plugins?

The best and quickest solution I found was to add the constant  $1.0e-24$  to every code location where many denormals might occur – this does not change your audio data much and keeps the numbers above the threshold. The critical locations are usually feedback loops in filters, phasers, delays, ... because once a denormal is inside this algorithm, it is handed "down the line" for some time, each time causing heavy calculations. For your convenience, I already added a denormal constant to the template and automatically denormalize audio input and output by default. Just don't forget to add this constant (it is called `kDenorm`) to your code lines if you make new algorithms, else some people trying out your plugin might complain :-)

Example: `delay[dptr] := delay[dptr] * 0.88 + kDenorm;`

### Why shouldn't I use the `trunc()` function?

`trunc()` is a standard Delphi function that converts the floating point number given as the argument into an integer. A VST plugin using this function will however most likely crash the host or produce a long list of error boxes. Why is that?

Well, something for the techies: for realtime floating point operations (eg OpenGL graphics or audio plugins), the control word of the 8087 floating point unit of the processor has to be set so that no floating point exceptions (similar to "internal error messages") may occur. This is automatically done at the initialisation of the `DAudioEffect.pas` unit by calling `Set8087CW(Default8087CW or $3F);`

However, there are few commands like `trunc()` that reset this control word back to normal, causing your plugin to produce error message after error message!

A simple solution for this behaviour: use my `f_trunc()` function instead (included in `DDspStuff.pas`).

If you find other functions that reset the control word, please tell me, so I can add them here!

### **How can I code a polyphonic synth?**

Well, try to code a monophonic version first :-)

Basically you need the architecture of a monophonic version with all its components (eg. oscillator, filter, envelopes, ...) for each voice you want to have. Now this would be a good idea to dive into object-oriented programming! Define your own class with all the needed components for one voice, then create as much objects from this class as voices are needed in the constructor of your plugin. Okay, it is not that easy in all cases, as you will probably find out when you try it yourself, but it is a good starting point. As always, some experience IS needed to get the thing working properly.

### **Can I also code DirectX plugins with this SDK?**

No, you can't. Sorry, but just like the original Steinberg SDK, this is for VST plugins only. If you want to code for DirectX, you might check out <http://www.thedirectxfiles.com>, you will find the relevant DirectX SDK for C++ there. If you prefer Delphi (and don't want to do header translations) you might look at <http://www.cloneensemble.com>, there is a Delphi DirectX plugin SDK there, but I don't know how up-to-date it is.

### **Who can I ask for help if I am stuck?**

It might often happen that you are stuck with a particular problem. Now where to adress to?

A good idea might be the DSP & VST coding forum on <http://www.musicdsp.org> (I am also a forum moderator there) - feel free to ask anything related to audio programming there, people will most likely be able to help you out or at least point you in the right direction.

Another good way would be the mailing lists: You can subscribe to the MusicDSP mailing list on the site above for DSP related stuff (not necessarily VST specific!), Steinberg offers a VST developer mailing list, look on <http://www.steinberg.net> for the subscription link.

If you are really stuck with coding/compiling your plugin, you might also consider dropping me a mail ([tobybear@web.de](mailto:tobybear@web.de)), although I can't promise that I will answer immediately :-)

### **Do I have to pay anything to anyone if I want to make/sell VST plugins?**

Well, haven't you read the SDK license agreement? :-) :-)

To put it short: No, you do not have to pay anything as long as you are not violating any copyright laws (ie you may not use copyrighted samples and pictures, but also algorithms can be protected by patents/laws!)

What you have to do is give credit to Steinberg in the about box and the manual of your plugin: *"VST is a trademark of Steinberg Media Technologies GmbH"*

If you have built a plugin based on my template, I would strongly suggest also adding the following lines somewhere in your about screen and manual:

*"VST SDK Delphi translation by Frederic Vanmol ([www.axiworld.be](http://www.axiworld.be))"*

*"VST Template code by Tobias Fleischer/Tobybear ([www.tobybear.de](http://www.tobybear.de))"*

### **How many parameters can my plugin have?**

Practically unlimited. I have successfully tried plugins with several thousand single parameters and did not encounter problems, except for older versions of Emagic Logic.

### **Is it possible to debug my plugins in realtime using breakpoints?**

That depends on your host and debugger. You should use hosts that load quickly and can access plugins comfortably, as booting up the whole Cubase or Logic system take quite some time on most computers. Nevertheless a check with Cubase (demo or full version) should always be made before releasing your plugin as it is the initial and most wide-spread VST host application.

### **How can I set the DLL output directory in Delphi?**

Select "Project - Options - Directories" to set the output directory to the "vstplugins" folder of your VST host - that way you won't have to copy the DLL manually after each recompile.

### **How can I start my sequencer automatically from Delphi?**

If you press F9 in Delphi to compile your plugin, you will normally get a message telling you that there is no host application defined. Go to "Run - Parameters" (in some Delphi versions this is called "Start - Parameters") to set your sequencer/testing program as the host application. Now each time you recompile, this program is automatically launched. I recommend my MiniHost program for testing, as you can also give the DLL to load as a parameter.

### **My plugin does not show up at all!**

If it does not show up at all, make sure you've copied it into the correct "vstplugins" directory of your host (check host settings/preferences). Most times there are several of these directories on the system and if it's not in the right one, the host won't find your plugin. Also make sure you have a unique ID for your plugin.

### **My plugin crashes only some hosts, while working fine in others! What now?**

As I already said further above, there is no VST host SDK, therefore a lot of the VST hosting stuff of a sequencer was probably coded by guessing and is usually done very, very differently in the various major hosts available. That means your plugin might run in one host, but not in the other - which is not good for the end-user as he expects any VST plugin to run on any VST host. Therefore you should very thoroughly check your plugin under various hosts and try to pin down where the problem occurs! Often it is not the host but the plugin that has done something wrong, so don't automatically blame others!

Most problems or crashes (especially with memory violations) happen in the constructor of the plugin, so look through it carefully, comment out some lines and add messageboxes for testing purposes until you find what is causing this strange behaviour.

If you have found a problem specific to one host and it is definitely the host's fault, please inform the company/vendor of that host of this problem. As an intermediate solution, you might add a fix that is only active under a certain host:

### **What else do I need (to know) to start developing plugins?**

Please be aware that audio plugin development is a very complex issue. You should be good in your programming language (C++/Delphi), know what pointers and classes are and how to inherit and overwrite objects and functions, else you won't get far. It is not so complicated, but you need this basic knowledge before going anywhere. Good books for object oriented languages are available and a good point to start if you know that this is your weak area.

Furthermore, if you want to code/implement your own audio algorithms, you should have a profound knowledge in higher level mathematics and DSP (digital signal processing). This can be a very complicated and technical issue, but if you don't know how a lowpass filter works mathematically, how are you going to implement one? FFT, FIR/IIR filters, poles and complex numbers, shaping functions, signal theory, etc. should be at least a bit familiar to you. Again, there are excellent resources available, especially for beginners. I am not saying that it is impossible to do plugins without this knowledge, but these things will most certainly pop up very soon when developing audio plugins.

### **Are there any good sites on the net concerning DSP programming/learning?**

Yes, of varying complexity and difficulty. I compiled a list of selected sites dealing with this topic, with a description what each site is about. You can find it on my site (<http://www.tobybear.de>) under "DSP Stuff". A very good starting point with some excellent VST and algorithm stuff are the MusicDSP and VST source code archives: <http://www.musicdsp.org> and <http://www.urs-h.com/vstsource>.

### **I just finished my first VST plugin, what now?**

Is it working stable? Have you tested it under several hosts and different operating systems? It should at least be tested with demo versions of several hosts, for example Cubase (Steinberg), Wavelab (Steinberg), FL Studio (In-Line Software) and Orion (Synapse Software). If your plugin works fine in all of them, chances are high it will work in most other hosts.

If you think your plugin is worth releasing, post a note on some frequented VST forum like K-V-R (<http://www.kvraudio.com/>) and give a valid email adress and/or website for feedback/bug reports.

Before uploading your plugin to a free web space provider, please bear in mind that many people will try to download it upon reading your post, so your traffic limit might very soon be exceed (trust me, I know what I am talking about :-) ).

Finally, listen closely to the feedback you get from other people. More important than implementing newly requested features is to take care that your program is running smoothly and error-free. Show presence in the various online plugin discussion boards and constantly try to improve your coding skills as well as your plugins, then people will adore you :-)

## 7. Credits

Original C++ VST SDK by Steinberg (<http://www.steinberg.net>)

Original VST SDK Delphi translation by Frederic Vanmol (<http://www.axiworld.be>)

VST Template code by Tobias Fleischer/Tobybear (<http://www.tobybear.de>)

Please give proper credits to all in your about box and manual!

The template has been successfully tested with Borland Delphi 5 Enterprise and Delphi 6 Personal Edition.

Get the latest version at my site: <http://www.tobybear.de>

### Contact information

Tobybear Productions:

Tobias Fleischer

Prisnitzweg 7

D-82538 Geretsried

Germany

e-mail: [tobybear@web.de](mailto:tobybear@web.de)

web: <http://www.tobybear.de>

### Copyright information

© 2001-2005 Tobybear Productions. All Rights Reserved.

**VST is a trademark of Steinberg Media Technologies GmbH.**

**All other product names and any trademarks mentioned are used for identification purposes only and are copyrights of their respective holders.**