User's Guide

For the

Solo Instrument Performance Suite

V1.5.1

6-5-07

Table Of Contents

What's New In Version 1.5

First off, you'll be glad to know that nothing has been done to damage the sound or musicality of either the **Legato** or **Vibrato** scripts. Moreover, your investment of time in crafting custom presets will not be lost. Rather, you will be able to easily transfer all your presets from **V110** (or even V105/V1051) to **V1.5** without the need to manually re-enter any of the parameters, **including custom CC assignments and settings.**

In **V1.5**, the **Solo-Mode Logic** (which is fundamental to the operation of the **SIPS-Legato Script**) has been completely redesigned. The new logic is much more efficient and easier to understand and maintain. And, by taking over many of the functions previously relegated to the buggy and sometimes unpredictable KSP callback triggering and sustain pedal logic, **V1.5** will hopefully avoid many of the previously-observed anomalies. For example, **V1.5** should be useable in any script slot (including slot 1) and should also exhibit a fairly uniform behavior between K2 versions (provided the version you are running supports all the necessary user interface elements). The old logic also required nearly double the polyphony actually needed because of muted notes that were used (to work-around some KSP quirks). The new **Solo-Mode Logic** doesn't require such work-arounds and thus requires less polyphony and, more importantly, doesn't pass on any muted notes to higher script slots.

Prior versions of the **SLS** offered two release modes; **Knob Setting** and **Key-Up/BTime**. In the **Key-Up/BTime** mode, the prior note (the one fading out during the crossfade) would begin its note-end release phase when the corresponding key was lifted or when the **BTime** setting expired. This latter dependence on **BTime** was an artifact of the old Solo-Mode Logic rather than an intended 'feature'. With **V1.5,** the new logic eliminates **BTime's** involvement. Now the two modes simply begin the release phase when the **Knob Setting** percentage of **XTime** has been reached (in the **Knob Setting** mode) or, when you lift the corresponding key (in the **Key-Lift** mode). This latter mode is now independent of the **BTime** setting (as it should have been all along).

Assigning a MIDI CC to control various **SIPS** parameters has been made easier and more flexible. Instead of an Edit box where you dial in the CC# that you want (with 0 meaning none and -1 meaning the Pitch Wheel), **V1.5** provides a drop-down menu of controllers for you to select. These menu choices are annotated with their customary MIDI associations (per the MMA). In addition, there is a 'learn' feature for when you aren't sure which CC# is controlled by some physical slider or knob on your keyboard. Once a CC is assigned to control a parameter in one of the SIPS scripts, prior versions would block further propagation of such assigned CCs. **V1.5 no longer blocks assigned CCs** so you can now assign a single CC to control multiple parameters if you wish (even parameters in two or more separate scripts). **V1.5** of the **SVS** has improved upon the way you control the overall Vibrato Amount. There is now a MIDI-controllable Knob for setting Vibrato Amount as well as a new drop-down Menu that allows you to select **Knob Only**, **Envelope Only**, or **Both**.

Finally, **V1.5** uses the new **ISCS** (Inter-Script Communication System) module for more efficient data and message transfers between scripts including preset import/export. Using the **ISCS** also paves the way for allowing **SIPS** to be more easily chained with other scripts in the future. Initially, **V1.5** includes the code needed to allow **SIPS** to be chained with the **VXF** (Nils Liberg's Velocity Crossfade Script). However, the **ISCS** is a generalized module that should eventually allow **SIPS** to be used with other scripts as well (as soon as the authors involved get together to work out the details).

I encourage you to re-read this entire **User's Guide** in order to get the most from **V1.5**. There have been many small, and sometimes subtle changes made to this manual to bring it current. As a minimum, you should carefully read '**An Introduction To SIPS'** starting on page 5. If you want to get the most musical results from **SIPS**, reading and understanding **all** the material presented in this **User's Guide** is essential.

Acknowledgements

I want to take this opportunity to thank everyone in the K2 community who played some part in the development of these scripts, both the original release and various updates. Many of you offered useful ideas and suggestions and many others offered encouragement. Since I'm not involved in orchestral work, I've had to depend upon others to try the scripts with their libraries and provide feedback as to the musicality and usefulness of the scripts in an orchestral context.

I want to thank Gary Lionelli who provided much useful feedback about the script's musicality and participated in the early development of the human interface for the initial release. I especially want to thank Theo Krueger and Andrew Keresztes for 'jumping in with both feet' in the eleventh hour when Gary's workload made it impossible for him to continue. Theo developed most of the Legato Script presets and Andrew developed the rest. The Vibrato Script presets were developed by Theo Krueger, Andrew Keresztes, and Martin Nadeau. I also want to thank Martin for doing a very meticulous job of proof-reading the orignal User's Guide and finding most of my mistakes (you'll have to find the rest, especially in the revised manuals). Theo, Andrew, and Martin also created some very nice mp3 demos for each preset that they developed. Finally, to more fully showcase SIPS in an orchestral context, Theo and Andrew composed several impressive multitrack orchestrations that are sure to become classics.

I also want to thank Nils Liberg for developing and sharing his KScript Editor with us because it has made writing and maintaining the source code so much easier. I also want to thank both Nils Liberg and Frederick Russ for taking over hosting of the SIPS Download and Demo pages since Theo's web site became history. I also need to thank Nils for his willingness to take over maintenence of SIPS during my on again off again health problems. While my health still continues to be somewhat uncertain, the recent addition of a pacemaker has started to make a positive difference; enough so that it allowed me to tackle generating this **V1.5** update. The Good Lord willing, I might soon be able to take full responsibility for SIPS again. In fact, I'm already starting to think about V2 ;-).

Finally, my heartfelt thanks to all of you on the forums who have posted so many kind remarks and encouraging thoughts. I also appreciate the many emails and PMs expressing your concern for my health situation and of course I especially appreciate your prayers on my behalf. I'm sure that your prayers in no small way were responsible for my being able to provide this new version and so it is my sincere hope that you will enjoy and benefit from this latest update of SIPS.

May God Bless all of you,

Bob

An Introduction to SIPS

The Solo Instrument Performance Suite, **SIPS**, is a collection of K2 Scripts that, when chained, are designed to work together harmoniously to provide a useful number of effects often referred to as performance tools. Initially, **SIPS** has been released with two member scripts — the **SIPS Legato Script** and the **SIPS Vibrato Script** which will be referred to hereafter as the **SLS** and **SVS** respectively. The KSP currently provides for up to 5 chained (cascaded) scripts that can be assigned to each instrument. So, up to 3 more member Scripts may eventually be developed and added to the suite.

SIPS is intended to be used with Solo, monophonic instruments. As such, **SIPS** is designed with a **Solo-Mode Control Logic**, similar to that provided with many synthesizers. This doesn't mean that you can't use **SIPS** with sectional or layered samples (when that might be appropriate), it just means that *ordinarily*, you won't be playing chords. However, for playing an occasional chord, **SIPS** allows you (under MIDI control) to disable the Solo-Mode Logic at any time you wish (see page 15).

The **Solo-Mode Logic** for **SIPS** is contained within the **SLS**. Thus, the **SLS** must be positioned first in the chain, with the **SVS** following it. While the **SLS** can be loaded and run by itself, the **SVS** *cannot* be run by itself since (for proper operation) it depends upon the **Solo-Mode Control Logic** provided by the **SLS**. Thus, if you disable the **SLS** (or if the **SLS** is bypassed or not even installed), the **SVS** will automatically disable its effect.

A Note of Caution about Chaining Scripts

All the member scripts in **SIPS** have been designed to work together when they are chained in the stipulated order. However, this is not necessarily true of scripts in general. Scripts often interfere with each other, sometimes in strange ways. So generally you should not put any scripts in front of or after **SIPS** member scripts unless you know for sure that they are compatible with **SIPS** (and each other). I'm sure that many of you have experienced these incompatibility problems first hand when trying to combine two or more of your favorite scripts. While there is no simple solution for this problem, a number of techniques have been developed and some progress is being made.

In order to chain scripts, among other things there will usually be a need for the scripts to communicate with each other and to pass data back and forth. The member scripts of **SIPS** are no exception to this. However, the KSP doesn't provide any convenient means of data sharing between scripts so, in the past, script writers have been forced to adapt various ad hoc schemes (usually using MIDI CCs) to provide a crude form of interscript communication. To provide a more general and powerful solution, a new 'importable' module named the **ISCS** (**Inter-Script Communication System**) was developed. This module provides a variety of services to help script writers to make their scripts chainable. **V1.5** of **SIPS** is the first suite of scripts to use the **ISCS** but hopefully others will follow.

Nils and I have been working together to make **SIPS** and his **VXF Script** (Velocity Crossfade Script) chainable. To this end, **V1.5** of **SIPS** includes the code needed to interface with the **VXF** and probably soon after you read this, Nils will have updated his **VXF** with the necessary code to interface with **SIPS**. If this integration of **SIPS** and **VXF** is successful, there may be other joint efforts to make various scripts compatible. To make that process easier and more uniform, we will make the source code for the **ISCS** available to the K2 Scripting community, free of charge. However, full usage of the **ISCS** in a hosting script requires that the host script be assigned a unique **Script ID** code so it can unambiguously be identified by other scripts using the **ISCS**. I guess I'll take on the job of assigning these numbers to script writers but, I'll be counting on each script author to cooperate with the process. The **ISCS** also supports a limited mode which allows a non-registered host script (one with no assigned **SID**) to receive **Omni Messages** and data and to utilize the **ISCS** support for local **Pseudo-Calling**.

Setting the Instrument Range

For things like keyswitches to work properly, it is important that all the scripts in **SIPS** know the normal instrument range. The **Instrument Range** is set with the first script of the suite (currently the **SLS**) using the following procedure. In the upper right-hand corner, click the button labeled **Set Inst Range** and follow the prompts (that appear on the button), ie hit the lowest instrument key followed by the highest instrument key. If you do this correctly, the display box beneath the button will show the keyboard range that you have set. These values will also be broadcast to the remaining scripts in the suite which should now also show the same values for **Low Key** and **High Key**. The easiest way to enter this range data is to use the K2 keyboard display. After clicking **Set Inst Range**, simply click the lowest 'blue' note followed by the highest 'blue' note on the K2 keyboard display. Once you have set the correct instrument range, **SIPS** will pass any notes above or below this range without any processing and thus will not interfere with the normal operation of key switches and such.

Assigning MIDI Controllers

Member scripts of **SIPS** allow many of their parameters to be controlled in real time via MIDI CCs. When a panel parameter can be controlled by a CC, there will be an associated ***assignment-button*** nearby (usually in the bottom row, just underneath the associated panel control). When a CC is assigned to control a parameter, the *assignment-button* will be **lit** and will display the parameter name along with the CC# assigned to control it. For example, when CC#21 is assigned to control **XTime**, the lit button will display '**XTime CC# = 21**'.When no CC is assigned, the button will be dark and will indicate that MIDI control is 'OFF' (ie '**XTime CC -OFF-**').

To change a CC assignment, *double-click* the button to open a drop-down list of MIDI CCs. Scroll through the menu to find the CC you want and click on it. This will close the menu and re-display the assignment-button (with your new CC assignment shown). **CAUTION: When the instrument editor 'wrench' is open (ie not in performance mode), the KSP handles I/O rather sluggishly (especially if the script editor's text window is open). So instead of *double-clicking*, you may need to space the pair of clicks a little in time.** Actually, the **1st click** brings up a drop-down button labeled '**- CC Menu -**' and the **2nd click** opens the menu. The drop-down menu displays the MIDI CC numbers from 1 to 119, along with a short description of their 'customary function' as assigned by the MMA. You are at liberty to assign any of these CCs, regardless of their 'customary' usage, but it is your responsibility to be sure there is no conflict of such assignments within your system.

To assign a CC number, simply click on that menu selection. Alternatively, if you want to assign some physical controller on your keyboard but aren't sure what CC# it represents, simply move the slider or knob and then click on the menu line labeled '**Last CC Moved**'. To de-assign CC control of a parameter, click on the upper menu line labeled '**MIDI CC - OFF -.**

In addition to the standard CCs from 1 to 119, some **SIPS** parameters can also be controlled by the Pitch Wheel. When this is the case, the menu will include a line labeled '**Pitch Wheel**' near the top of the list. The Pitch Wheel, **PW**, is unusual in that it can be moved both down and up from its center position. In general, **SIPS** may use the **PW** in one of two ways. In **uni-polar mode**, the **PW** has the same effect when moved upward as it does when moved downward. In **bi-polar mode**, the **PW** has a positive effect when moved upward and a negative effect when moved downward. Refer to the specific parameter you want to control with the **PW** for how **SIPS** uses it.

Add-On Control

SIPS Member scripts often use a special form of MIDI control known as **add-on control** to provide more precise adjustment of certain parameters. One example of this for the **SLS** is the **XTime** parameter. In addition to the CC assignment button, there is also an edit box just above it (labeled **CC Range**). Any CC assigned to **XTime** will be used to 'add-on' to the **XTime** value set by the knob. The amount of the add-on of course depends on the position of the CC itself, but in addition it's also affected by the setting of **CC Range**. This edit box displays in percent and can be set to any value from 10% to 200%. Perhaps a simple example may help to clarify this.

Suppose you set the **XTime** knob to a value of 350ms and the **CC Range** box to 10%. Now if you assign CC2 to control **XTime**, as you swing CC2 from min to max, the value of **XTime** will swing from 350ms to 385ms. In other words, the CC will add-on 10% of 350ms when CC2 is at max. If you instead set **CC Range** to 100%, CC2 will swing the value of **XTime** from 350ms to 700ms. Similarly, if **CC Range** is set to 200%, CC2 *would* swing the value of **XTime** from 350ms to 1050ms. However, since the max setting allowed for **XTime** is 1000ms, the add-on (combined with the knob setting) cannot exceed that limit. All MIDI add-on controls work in exactly this same way. Most add-on controls allow you to assign the Pitch Wheel as a **uni-polar** control but always refer to the discussion of the specific panel parameter itself to find out how the Pitch Wheel may be used to control it (ie using uni-polar or bi-polar mode).

Sometimes it is desireable to have one CC control several parameters (within one or several scripts) but if this isn't what you want, be careful that you don't accidentally make such duplicate assignments. Script members of **SIPS** do not check for nor disallow duplicate CC assignments either within an individual script, the suite of scripts, or with K2 itself. Therefore, it is up to you to assign CCs such that there are no conflicts. **CAUTION: Prior versions of SIPS used to block MIDI CCs from further propagation once assigned to control a SIPS parameter. To be more flexible, V1.5 no longer blocks assigned CCs from continuing on through the script chain and thus to K2 itself.** Therefore if you are using an instrument that also is configured to use one or more of the CCs you assign to control **SIPS** parameters, you will now have to disable those parameters in the instrument (unless you want them to also be affected when you move the CC). Alternatively, you can choose a different CC (not used by K2) or, if you have a spare script slot, you can use my CC Blocker Script and set its slider switches to block any of the assigned CCs from reaching K2.

SIPS Preset System

Each **SIPS** member script, has a **Preset** drop-down menu in the upper-left corner of its control panel. This menu contains a number of **Built-In Presets** as well as space for you to store up to 20 **User Presets** of your own custom design. The preset menu also contains a number of action **'commands'** which will be discussed as we go along. The **Built-In Presets** are usually named for the type of instrument or instrument family that the preset was designed to be used with. These presets were designed and installed by the developers of **SIPS**. However, since there is a good deal of variance from one sample library to another, as well as marked differences in the sound you may be trying to achieve, you should consider the **Built-In Presets** merely as 'starting-point templates' that you will want to customize for your particular library and application. Because of this, built-in presets will not generally contain values for every control-panel parameter. For example, the built-in presets do not contain values for Instrument Range, assigned MIDI controllers, etc. Rather, built-in presets contain a general subset of the control panel parameters that are referred to as the **'Key'** parameters (see page 13). The remaining panel parameters are referred to as the **'Extended'** parameters.

To recall a **Built-In Preset**, just click on the **Preset Menu** drop-down button and then click on the desired **Preset Name** in the menu. The menu will then close and the **Preset's Name** will appear on the button. The **Key** parameter values for the selected preset will instantly be copied to the control panel. The **extended** panel parameters however **will be left unchanged**. Once you recall a preset, you can tweak the values and/or add MIDI control, etc to obtain the desired effect. If your customized panel is intended to be used only with one particular instrument, you can simply re-save the instrument. The scripts, along with the customized panel, will then be saved with it. The next time you load the instrument, it will be recalled just as you left it (including the Extended parameters). However, if you then recall another preset, your customized panel will be lost and can be restored only by reloading the re-saved instrument.

User Presets

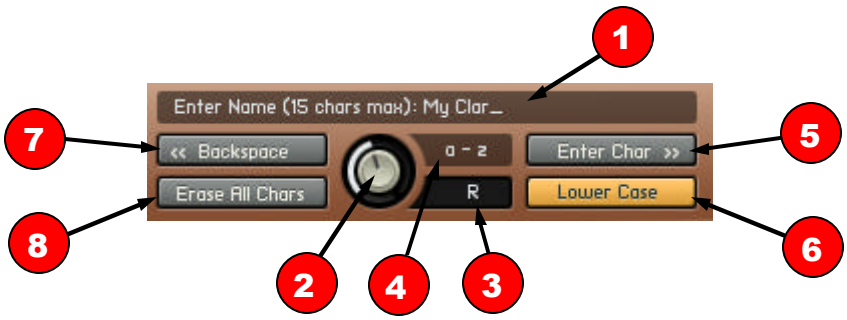
If you develop one or more customized presets or you would like to keep the control panel settings for several instruments together in one place, you can save your control-panel settings as **User Presets**. To save a **User Preset**, first set the control panel the way you want it and then open the **Preset Menu** and click on the – **Save As** – *command*. You'll find this command between the **Built-In** and **User Preset** lists. After clicking on **Save As**, the menu will close and the button will read **- Save As -**. Now, open the menu again and click on one of the user preset locations (initially named **My Preset #1**, **My Preset #2**, etc). When you do this, the menu will close again with the user preset name on the button. The panel settings are now stored in the **User Preset** and this preset can be recalled just like any of the **Built-In Presets**. However, before you end your K2 session, **you must re-save the script** first, either as a **.nkp** file or with the instrument as a **.nki**. file. If you don't resave the script, your new **User Preset** will not be there the next time you load the script.

While **User Presets** can be recalled to the control panel just like any of the built-in presets, there is one notable difference. Starting with **V110** of **SIPS**, **User Presets** store **all** the panel parameters (including the **extended** parameters) whereas the built-in presets store only the **key** parameters. Thus, as already stated, when you recall a **built-in preset**, the **extended** panel parameters **are unaffected** and retain whatever values they were set to last. However, when you recall a **User Preset** (that you have saved in **V110** or higher), the **extended** panel parameters **are affected** (they will be set to the values they had when the preset was saved).

There are several things you should note about the **Save As** operation. First, the **Save As** command is only active until you select the following **User Preset** (the save destination). If you want to save another preset you will have to click the **Save As** command again. Second, if the **User Preset** slot you save to already has a preset (previously saved there), **it will be overwritten** and replaced by the new preset. Finally, if you click **Save As** and then change your mind, simply click on any **Built-In Preset** (or any other menu command) to cancel the pending **Save As**. As originally supplied, all the **User Presets** are **'empty'**. If you attempt to recall a **User Preset** that's empty, the control panel will remain unchanged and K2's status line will read '**Preset Empty**'.

Naming User Presets

Starting with **V110** of **SIPS**, you can rename any **User Preset** (even an empty one). To rename a **User Preset**, open the Preset Menu and click on the **'Rename'** command (located just after all the **User Presets**. This will open the **Name Edit Panel** (as shown below) in the lower right-hand corner of the main control panel. .



Name Edit Panel

When renaming a preset, the new name is entered in the **Name Edit Window (1)** character by character as selected by the **Alpha Knob (2).** As the **Alpha Knob** is rotated (by dragging up and down over it), you can choose one of 42 chars spread over 4 zones. The selected character is always displayed in the knob's **Data Window (3)** while the **Active Zone (4)** is displayed just above the **Data Window**.

Starting from minimum (full CCW position), the four zones are: **Space, Alphabet, Digits,** and **Symbol Set**. The **Space Zone** only has one character (the blank character). The **Alphabet Zone** contains the letters **A - Z**. The **Digits Zone** contains the numerals **0 - 9** and the **Symbol Set Zone** currently contains the five symbols **# < > + -.** It's very easy to find the desired character by dragging rapidly at first while watching the **Active Zone** display until you get to the appropriate zone. Then slow your dragging speed as you approach the desired character in the data window. When you are very near you can hold down the Shift key as you drag for more precision and it will be easy to 'zero in' on the desired character. With just a little practice, you can easily get to any desired character quickly.

When you have selected the first character that you want to enter, click on the **Enter Char (5)** button and the selected character will be entered into the **Name Edit Window (1)** and the blinking cursor will move to the right; ready for you to select and enter the next character.

When selecting an alphabetic character, the **Data Window (3)** always displays the upper case form (because lower case letters are less readable). However, if the **Lower Case (6)** button is **active**, when you click **Enter Char (5)**, the corresponding lower-case character will be entered. This is emphasized by the zone display **(4)** of **'a - z'** when the **Lower Case (6)** button is active (instead of **'A - Z'** when the **Lower Case** button is inactive).

If you change your mind about a character you have entered (or you have used the wrong case), you can click on the **Backspace (7)** button to delete the character to the left of the cursor so you can re-enter it. You can also use multiple presses of the **Backspace** button and each click will delete another character until you reach the left margin of the name-entry field. If you want to delete the entire name string that you have entered and start over, you can simply click the **Erase All Chars (8)** button.

You will most often want to start your name with an upper case letter and then continue with lower-case letters after that. Therefore, when you first open the **Name Edit Panel** (or whenever you **Backspace** to the left margin), the **Lower Case** button will automatically be **deactivated** and after you enter the first character, the **Lower Case** button will automatically be **activated** for you. If this turns out to not be the correct choice for your situation, you can simply override it by manually clicking the **Lower Case** button to set it the way you want it.

When you have entered all the characters of your new name, simply open the **Preset Menu** and click on the **User Preset** name you want to replace. **Unfortunately, your new name will not actually appear in the menu immediately because K2 will not allow a script to update a menu entry anywhere but in the initialization callback, or ICB**. The easiest way to force the **ICB** to run without having to save and reload the script, is to open the KSP text editor and click the **'Apply'** button. Once this is done, you will see your new preset name in the **Preset Menu**. However, before you end your K2 session, you will still have to resave the script (either as a **.nkp** or with the instrument as a **.nki** file). Otherwise, the next time you load the script your new names (as well as any other changes you make) will be lost. If you are going to rename several presets, you need not hit **'Apply'** nor resave the script until you have made all the changes (unless you need to see the changes right away in the **Preset Menu**).

Updating Your Custom Presets to V1.5

You may have saved one or more **.nkp** files with custom **User Presets** that you spent a fair amount of time developing. Or, you may have saved one or more Instruments with customized control panel settings and/or a set of custom **User Presets**. Thus, when you update to **V1.5**, you'll most likely want to transfer all your custom presets and control panel settings from your prior version of **SIPS** (preferably without having to painstakenly write down all the settings on paper and then re-enter them all over again into the new version). **.nki** and **.nkp** files with custom presets can easily be updated to the current version of **SIPS** by using the **Auto-Import** feature built into **SIPS. Auto-Import** will allow you to transfer all your custom control panels and **User Presets** from any prior version of **SIPS** (including **V105**, **V1051**, and **V110**) with a minimum of effort (amounting to little more than loading and then resaving each **.nkp** or **.nki** once you get the 'hang of it').

Using the new **Auto-Import** feature requires that you know how to place a copy of the **V1.5**, **K2-Ready** source code into your computer's clipboard. If you are unfamiliar with how to perform such operations, before proceding with this discussion, you should carefully read the entire section titled **Installing a 3rd Party Script** starting on page 34 (paying special attention to **Placing the K2-Ready Source in the Clipboard** on page 36).

In general there are two kinds of files that you will want to upgrade, **.nkp** files and **.nki** files. **.nkp** files contain only a single script (either the **SLS** or the **SVS**). **.nkp** files will have a control panel with a full set of parameters (**key** and **extended**) and possibly a set of up to 20 **User Presets.** On the other hand, **.nki** files contain both an instrument, and possibly both the **SLS** and the **SVS** that were saved with the instrument. For such **.nki** files, both the **SLS** and **SVS** will have a full control panel saved with them and in addition the scripts may also contain up to 20 **User Presets** each. For **V105** (and **V1051**), **User Presets** contained only **key** parameters (just like the built-in presets). But, starting with **V110** (which of course includes **V1.5**), **User Presets** contain the full set of parameters (both **key** and **extended**).

To upgrade an earlier version of a **.nkp** file for the **SLS**, proceed as follows. Place a copy of the **V1.5 K2-Source** code for the **SLS** into the clipboard (or have it standing-by in **Note Pad** or the **NL Editor**). Launch K2 and load some suitable instrument. Then open the **Script Editor** and use the **Script** button to load the **.nkp** file you want to upgrade. Next, open the KSP Editor's text window to expose the source code for the old version of the **SLS**. Now, with a copy of the **V1.5** source in the clipboard, click on the KSP text window (to give it the focus if need be) and then hit **ctl-A, ctl-V**. This will 'replace' the **old** source code with the **V1.5** source code from the clipboard. Now click the **'Apply'** button and the **V1.5** control panel should replace the old version's panel. Next, name the script by double-clicking the title box and typing the desired name followed by the enter key. You can now close the text window and save the updated script as a **.nkp** (with a different file name if you wish to preserve the earlier version). The new **.nkp** is now updated to **V1.5** but it also contains all the control panel settings and **User Presets** from the old **.nkp**. In the same way, you can update any **.nkp** file for the **SVS** except that you must load the clipboard with the **K2-Source** code for **V1.5** of the **SVS** (instead of **V1.5** of the **SLS**).

The process for updating your **.nki** files is very similar to updating a **.nkp** file except that it may involve both a **SLS** and a **SVS** update. Therefore, you will need to arrange to alternately place the source for the **SLS** and the **SVS** into the clipboard during the update procedure. This is fairly easy to do since recent versions of the **NL Editor** allow loading multiple source files. So, simply load the main source files (for **V1.5** of the **SLS** and **SVS**) into the NL Editor and when you need the **SLS** in the clipboard, just click the **SLS** file tab and hit **F5**. Similarly, when you need the **SVS** in the clipboard, click the **SVS** file tab and hit **F5**.

Now, launch K2 and load the **.nki** file you want to update, open the script editor, select the **SLS** script tab and then open the text edit window to expose the source code for the **old** version of the **SLS**. Next, with the **V1.5** source code for the **SLS** in the clipboard use **ctl-A, ctl-V** to paste the **V1.5** source into the KSP Editor ('replacing' the **old SLS** source). Now, click the **'Apply'** button to update the **SLS**. Next, place the **V1.5 SVS** source code in the clipboard and then select the **SVS** script tab in the KSP (in order to expose the **old** source code for the **SVS**). Then, use **ctl-A, ctl-V** to 'replace' the old source with the **V1.5** source for the **SVS**. Now, click on the **'Apply'** button to update the **SVS**. You can now rename the two updated script slots by double-clicking their title boxes.

You can now resave the instrument **.nki** file (using a new name if you want to preserve the original version of the **.nki**). The new **.nki** will now contain **V1.5** scripts with the control panel settings and **User Presets** of the old instrument file. If you have a lot of Instruments to update, instead of updating as just outlined, you may want to first load a bunch of instruments into the K2 rack and then go through them one by one in a 2-pass operation. First put **V1.5** of the **SLS** in the clipboard and update only the **SLS** script for each instrument. This way all the loaded instruments can be 'half-updated' without changing the contents of the clipboard. After all the instruments have the **SLS** update, load **V1.5** of the **SVS** into the clipboard and then redo each instrument, this time just updating the **SVS** script. Finally, resave all the (now fully updated) instruments.

Auto-Import Problems

When you 'replace' the source code for a **SIPS** member script and then hit **'Apply',** the code block that executes tries to verify that what you are trying to do is valid. To do this, the **Auto-Import** algorithm checks certain attributes of the 'old' source code and the 'new' source code. These attributes are then compared to determine if the import will be valid. Valid upgrades can be made from any earlier version of **SIPS** (such as **V105**, **V1051**, or **V110**), however you must also be sure that the old and new scripts belong to the same member. In other words you can't import from the **SVS** to the **SLS** or vice versa. If **V1.5** is the importing script and the 'old' script is any older or newer **SIPS** script, all illegal combinations **are detectable** and will be reported by **V1.5** (an error message box appears and indicates that an **Auto-Import** error has occured.

If you try to downgrade from **V1.5** or **V110 to V105** the error won't be reported because **Auto-Import** wasn't implemented until **V110** (so V105 doesn't perform it). This downgrade operation should be harmless, but it doesn't accomplish anything either. In general, the **Auto-Import** feature is only intended for upward mobility of user presets, not downward. For example if the next version of **SIPS** turns out to be **V200**, it will be possible to upgrade **V1.5** to **V200** but not the other way.

Now if you try to update to **V1.5** from some other script which is not part of the **SIPS** family, the problem **may or may not be detected** and, the outcome may or may not be OK. To understand this situation it will be helpful if you read page 35 on Source Files and the new K2.1 double-buffering of persistent variables. Generally, if the 'old' script has no persistent variables named the same as **V1.5** of **SIPS**, trying to update should be relatively harmless. However, if one or more persistent variables in the 'old' script should happen to have the same name as some of the persistent variables in **SIPS**, trouble may ensue. If the name match happens to occur in certain key areas, the **Auto-Import** logic may detect it and warn you but chances are that such will not be the case.

Therefore, to be on the safe side, whenever you are updating to **V1.5** of the **SLS** or **SVS**, be absolutely sure that the 'old' source file is a **SIPS** file. If you do this, the results will either be harmless or you will be warned with a suitable error message. On the other hand, if you want to install **V1.5** of **SIPS** from a source file and you don't want to transfer your custom presets, use the **'fresh-install'** procedure outlined on page 35 (ie before pasting the **V1.5** source into the KSP editor, simply load the **Empty** preset). This will clear K2's persistence buffers and you will get a 'fresh' install (whereby all settings will be initialized to their default values). A fresh install like this is the same as loading the original **.nkp** file for **V1.5** (the one included with the original **SIPS** package download).

Warning About Compact Compiler Output

Nils Liberg's KScript Editor has a developer option called **'Compact Compiler Output'** that reduces the overall size of the K2-Ready Source code. However, when compiling **SIPS** source code, **DO NOT use the Compact Compiler mode** unless you are installing **SIPS** as a 'fresh install' (ie you have no User Presets from prior versions that you wish to carry forward). If you use Compact mode, **Auto Import will not be able to successfully transfer your presets from prior versions of SIPS**.

In versions of the **NL Editor** prior to **V1.22.4**, Compact mode did not always result in identifiers being compressed to the same string of characters and this could mess up **K2's persistent variable** scheme (which is used for preset migration). Starting with **V1.22.4** of the **NL Editor**, Nils has standardized his symbol compaction algorithm with the *intent* of not changing it in future releases of the editor. Therefore, if you are starting a 'fresh' install of **SIPS**, you might want to consider using Compact mode. However, if you do, you will have to use Compact mode for all future updates of **SIPS** and if the current compaction function is ever changed, you may not be able to migrate your presets forward to future versions of **SIPS**. Note also that the K2-Source code of the **.nkp** files that are supplied with **SIPS** are *not* in Compact mode so if you want to use Compact mode you must 'fresh' install the NL Source code (see page 35) rather than starting with the **.nkp** files.

Extending User Presets

User Presets in **V105** only contain the **key** parameters and do not include the **extended** parameters. For **V110** on up however, whenever you save a **User Preset**, **all panel parameters** are saved (including the **extended** parameters). So, after an **Auto-Import** from **V105**, **User Presets** do **not** contain the **extended** parmeters (since they didn't exist in **V105**) and until you **resave** them with **V1.5**, these extended parameters are set to a special 'token' value to indicate that they are **undefined**. Thus when you recall a **User Preset** imported from **V105**, the **extended** parameters of the control panel are left unchanged (just as they would have been in **V105**).

However, after updating to **V1.5** you can update your imported **User Presets** (if you wish) to include the **extended** parameters. To do this, all you have to do is the following. First, recall an imported **User Preset** to the control panel. Then, set the **extended** parameters on the control panel to the desired values and finally, use the **'Save As'** command to resave the **User Preset**. The next time you recall t**his User Preset, all panel parameters will be recalled as desired.**

Recall of Instrument Range

Included in the **extended** parameters of a panel is the **Instrument Range, IR**. If you recall an **SVS User Preset** that has a different **IR** from the current **SLS**, the **IR**s could become 'out of step'. This kind of situation is rare, but if it occurs, all you have to do is reset the **IR** from the **SLS**. You can also avoid this situation entirely if you always recall the **SVS** preset first and then recall the mating **SLS** preset. When an **SLS User Preset** is recalled, the **IR** is broadcast to the remaining **SIPS** member scripts so they will all be brought into step with the **SLS**.

Import/Export of User Presets

When **V105** of **SIPS** was written, it was before double-buffering of persistent variables had been added to K2. Therefore an **Import/Export, I/E** feature was added to **V105** of **SIPS** to assist in the process of transferring presets when updating. Since this function is now provided by the **Auto-Import** feature added to **SIPS** (starting with V110), we no longer need the **I/E** function for its original purpose. However, **V1.5** retains a slimmed-down version of the **I/E** function because it facilitates the creation of **User Preset Libraries** by allowing you to re-group and/or re-arrange panel settings from several different **.nkp** or **.nki** files. Since **User Presets** now 'remember' **all the panel parameters** and can be named, this ability could prove to be very useful.

For **V110** and up, the **I/E** function is intended to be used only to transfer **User Presets** between scripts of the same exact kind (both member type and preset format class). Moreover, the **Export** command only transfers one **User Preset** at a time. There is no longer any **bulk-export** of all **User Presets** (as there was in **V105**) because that is most easily done via the new **Auto-Import** feature. To transfer **User Presets** (one at a time) from one script to another, position the receiving script in the next slot to the right of the sending script. Open the **Preset Menu** of the receiving script and click on the **'Import'** command. The **Import** command is usually the last menu item. The menu will close and the button will display **Import** indicating that the script is ready to receive **User Presets**.

Next, open the Preset Menu of the sending script and click on the **Export** command. The menu will close and the button will display **Export**. Now, open the menu again and click on the **User Preset** you want to export. If the operation is successful, the Importing script will display the message **'Import Done'** in the K2 status area. The exported **User Preset** is now available in the receiving script (including the preset's name) but you will have to resave the receiving script if you want to retain the new preset the next time you load the script. Also, just as when you are renaming a preset, the imported preset's name will not appear in the **Preset Menu** until you either click on the **'Apply'** button or you save and reload the script (see page 9). After you have exported a preset, the sending script will remain in the **Export** mode in case you want to send another preset (the receiving script will also remain in the Import mode). To send another preset, simply open the **Preset Menu** again and click on the next **User Preset** that you want to send.

When you have finished sending presets you can end the **Export** mode by selecting any built in preset (or by clicking on any other 'command'). Note that imported presets are postioned the same as in the sending script (ie if you export the **5th User Preset**, it will be imported as the **5th User Preset**). Also note that if you try to export an empty preset, **nothing will be sent** and the K2 status area will display the message **'Preset Empty'**. Or, if you try to do an **I/E** operation between two non-compatible scripts, the data will not be transferred and the **V1.5** receiving script will display the message **'Preset Format Mismatch'** in the K2 status area.

If you want to transfer a control panel, first save the panel to one of the **User Presets** using the '**Save As**' command, then **Export** that **User Preset.** With this general idea, you can use the **I/E** facility to load up to 20 instruments (one by one) and 'collect' their control panels as **User Presets** into one single **.nkp** file. In addition, while the **Export** *command* always transfers a **User Preset** to the same location in the import script as it was in the export script, you **can rearrange the order** in the new script (at least with some degree of lattitude) because, until the importing script is completely filled, you can always move a preset by first recalling it to the panel and then using **Save As** to write it to another user location.

About the Built-In Presets

The current version of the **SLS** contains a total of 17 built-in presets and the **SVS** contains 10. The 'fresh-install' default presets are **Clarinet 1** for the **SLS** and **Basic Setup** for the **SVS**. **Basic Setup** makes a good starting-point for constructing new Vibrato presets. The remaining Built-In presets were developed by **Theo Krueger, Andrew Keresztes**, and **Martin Nadeau**. Their presets can be identified by their initials in the right margin of each preset in the menu. Thus, the legend **TK#**, **AK#**, and **MN#** are used for **Theo's**, **Andrew's**, and **Martin's** presets, respectively, with the numbers after their initials used to tie the presets to their corresponding **mp3** demos. Each demo is prefixed with a corresponding tag.

Not all of the control panel parameters are stored with the **Built-In** presets. **SLS Built-In** presets only contain the following 10 parameters: **XTime**, **AtkFade**, **NodeVol**, **BTime**, **Slope**, **Bend**, **RlsFade**, **RlsMode**, **Offset**, and **OfstMode**. Whereas **SVS** built-in presets contain the following 16 parameters: **Depth**, **Width**, **Speed**, **Pk-Dwell**, **Width-Var**, **Depth-Var**, **Speed-Drift**, **Speed-Per**, **Wave-Drift**, **Wave-Per**, **Onset**, **Rise**, **Decay**, **Sustain, Env-Menu,**.and **Vib-Amt.** This doesn't mean that the remaining parameters (the **extended** parameters) are unimportant, just that they tend to be more Instrument-specific and/or dependent on individual preferences. For this reason, **Extended parameters ARE stored with your custom User Presets**.

Again, it must be emphasized that the **Built-In** presets should only be viewed as a reasonable starting point for several reasons. First of all, many of the presets were developed with a certain amount of generality in mind. As an example, **Theo** describes his presets this way: **"These presets are a little bit 'contained', meaning that I didn't overuse the bending or try to make them overly-expressive. I did that considering the various libraries and trying to make settings that would work well for most. Despite that, I hope they serve as a good guideline for people to adjust them to any library"**. Theo's preset **demos** (mp3s) were basically done using just the raw preset (without CC riding) so that you can hear what the basic preset itself sounds like. However, to get the most from **SIPS**, you will want to customize these presets on an instrument by instrument basis and, you will want to add **MIDI** control to enable you to add your own personal touch of realism to the sound. It's not practical to include **MIDI CC** assignments with the built-in presets because everyone has a different set of favorites and will tend to use CCs in a somewhat different way. Furthermore, just assigning CCs doesn't 'play them' for you. Yet, **assigning CCs and 'riding them' during the performance is almost essential to producing a really convincing effect**.

If you use the built-in presets as is, along with a 'set it and forget it approach', you surely won't get the most from **SIPS**. While the effect can still be surprisingly good, don't expect miracles. On the other hand, if you are willing to add some **MIDI** controls and put some effort into using them musically, the results can be **extremely convincing**. The artful use of CCs can be likened to viewing **SIPS** as you would a musical instrument. If you want a quality, musical sound, don't expect the scripts to 'play themselves'. Andrew did his preset demos (mp3s) with CC riding added and I think you will find his demos are quite convincing. Here's what Andrew said about how he used **SIPS** to make his demos: **"I use SIPS as an instrument.— it rarely stays static. I'm always riding the CCs. So as time goes on, I'll get better at it. Plus, for strings (for me), it's imperative to have varying gliss times and amounts"**.

Be sure you study the rest of this **User's Guide** so that your future knob twiddling can be carried out with some solid technical knowledge backing it up. You will especially want to read the sections titled '**Guidelines for Making Legato Settings'** and '**Guidelines For Making Vibrato Setttings'**. These sections offer a solid recipe for 'cooking up' presets. In addition, I asked Theo and Andrew to write about their experience with **SIPS** and to share with us their **tips and techniques for creating presets** and using **SIPS** under fire. So, be sure that you read what these **SIPS Pioneers** have to say, you'll find their narratives starting on page 32 of this **User's Guide**.

SIPS Legato Script

Introduction

Simulating a legato effect with a Script is all about connecting notes, or more specifically, controlling note transistions. When you play two overlapping notes, the Legato Script must 'retire' the old note and 'establish' the new note. But the old note can't just be ended and the new note started or it won't sound like legato playing. For a brief period of time, called the 'transistion' period, components of both the old and the new notes are present.

The two major functions peformed by a Legato Script are Crossfading and Bending. The Crossfading function controls the relative volume of the old and new notes while the Bending function warps the pitches of the two notes during the transistion period. The objective is to do all this in such a way that the transistion from the old to the new note is done smoothly and *sounds*, for all practical purposes, just as it would when done with a real instrument by a real player. This turns out to be rather 'a tall order' because there are many conflicting requirements and the current KSP tool kit is rather limited. However, the **SIPS Legato Script**, when *properly set up* is capable of providing some extremely realistic and convincing legato sounds. Of course to *properly set up* the **SLS**, it is important that you understand what all of the knobs and such do. While a number of presets are provided, these should be viewed only as starting points for your further customization on an instrument by instrument basis. This is especially important if you are expecting the **SLS** to work with a wide variety and quality of sampled instruments.

Playing Legato

To play a phrase legato, the **SLS** requires that all the 'inside' notes of the phrase overlap. The first note of a new phrase is sensed by the script based on the fact that no other notes are still sounding (ie no keys are still held down and the sustain pedal is off). The **SLS** plays the first note of a phrase normally (that is with its normal attack). If this first note ends before another note starts, the script will interpret the 2nd note as the start of another new phrase. Such notes are not affected in any way by the script. However, if the first note is still sounding when the 2nd note starts, the notes are considered overlapping **and the Legato Effect will be generated**. Similarly with the 2nd and 3rd notes, the 4th and 5th notes and so on. When the last note played ends before another note is played, the script considers it to be the 'end of the current phrase'. Unless you are using the special '**Key-Lift**' release mode, the amount of time overlap is not important to the script.

Use of the Sustain Pedal

Normally, you tell the script that you want the legato effect for a pair of notes by not releasing the first note's key until you have pressed the next note's key. However, the script produces the legato effect for any pair of notes that overlap. So if you depress the sustain pedal during the first note, then even if you release the key before striking the next key, if the sustain pedal is down when the second key hits, the script will 'see' the notes as overlapping and add the legato effect. Unlike prior versions of the **SLS**, **V1.5** processes the sustain pedal with its own logic (K2's normal handling of the pedal is disabled). Thus, if you hold the sustain pedal down while playing a phrase, there will be no excessive 'buildup' of polyphony because the **SLS** fades out the prior note when each new note starts. In fact, the polyphony would never exceed two if it weren't for the fact that some instruments have a long release tail. Because of this, if you play a fast legato passage, polyphony can exceed two for brief periods when multiple note tails may overlap (but, this will happen whether or not you use the sustain pedal). So, you can optionally play legato either by overlapping the keys or by using the sustain pedal (whichever is more convenient). However, if your phrase contains two or more notes in a row that are the same pitch, you can't actually overlap the keys so for this case, you will need to use the sustain pedal to legato-connect the notes. Note also, that when using the special **Key-Lift** release mode with the sustain pedal, both the key **and** the pedal must be released to shorten the fade-out note.

Playing Chords

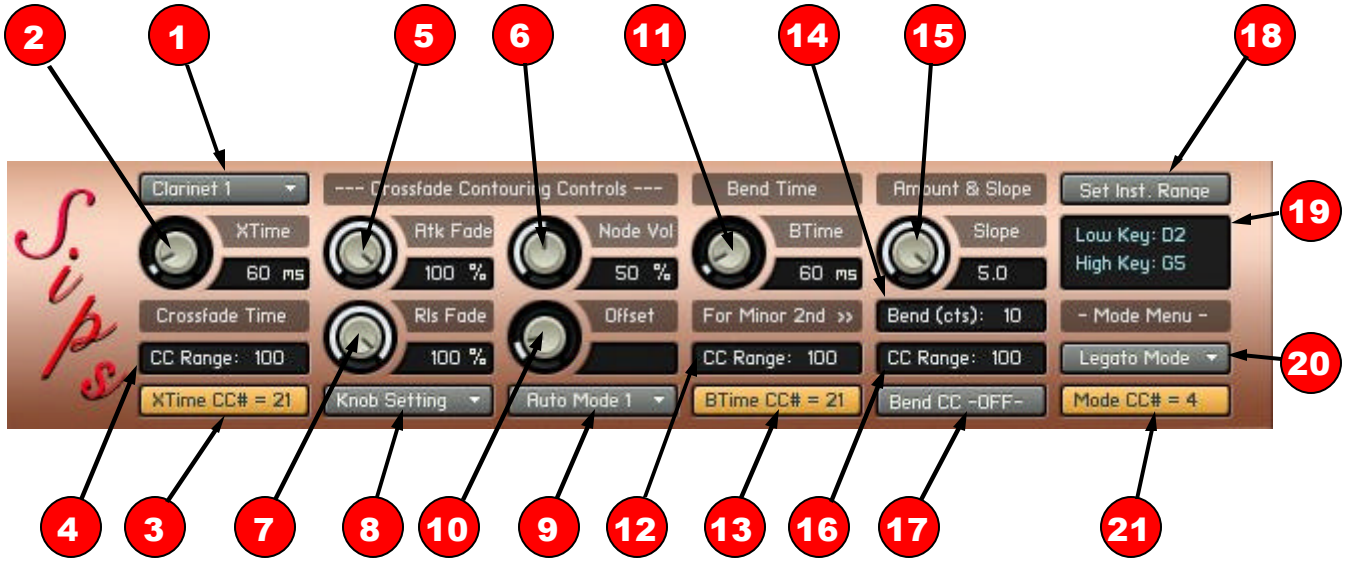
Prior versions of the **SLS** allowed the playing of chords in **Solo** and **Bypass** modes but, there were several unforseen problems with its implementation that made it awkward to use in actual practice. In **V1.5,** the MIDI-controllable **SLS Mode Menu** function has been completely redesigned to provide a much more musician-friendly operation.

For **V1.5**, the **Mode Menu** choices are **Legato Mode**, **Solo Mode**, and **SIPS OFF**. Of course **Legato Mode** is the primary mode of the script**. Solo Mode** (as in prior versions of **SIPS**) allows for playing monophonic lines where each new note played forces the prior note to its release phase (but provides no crossfading or bending like **Legato Mode** does). However, **unlike** prior versions, **Solo Mode** in **V1.5**, does **not** allow playing chords (with or without the sustain pedal depressed). Rather, the sustain pedal merely insures that the individual notes will be connected (as it does for the **Legato Mode**).

However, the **SIPS OFF** mode completely removes the legato and solo-mode effects of the script and thus allows normal playing, **including the playing of chords if desired**. And, since the **Mode Menu** can be assigned to a MIDI CC, you can easily switch back and forth between **Legato** (or Solo) **Mode** and **SIPS OFF** in real time as you play. However, unlike prior versions of the **SLS**, the transitions and overlaps provided by **V1.5** are much more musical. For example, if you are playing a legato phrase and hold the last note of the phrase as you switch from **Legato Mode** to **SIPS OFF**, subsequent notes will play normally (without the legato or solo-mode effect) but, the last held note will carry over until released. Similarly, if you play some passage in the **SIPS OFF** mode and hold the last chord played as you switch back to **Legato Mode**, subsequent notes will play legato but the last held notes from the **SIPS OFF** mode will continue to sound until released.

For lack of a better name, we might call this feature **'Musical Carryover'**. Basically it simply means that the last note or chord sounding in the prior mode can be overlapped into the next mode. This can be done either by holding the key (or keys) down during the mode change, or by using the sustain pedal to accomplish the same thing. Hopefully, the new logic used to implement this feature will provide for a musically-satisfying and a more intuitive behaviour for real-time mode changes.

The Legato Script Control Panel



(1) **Preset Selector** drop-down menu. Selecting an Instrument or Instrument Class from this menu will set certain key parameters to a good starting point for you to further customize by ear. For a list of key parameters recalled by the built-in **SLS** presets, see page 13. For complete information on **User Presets** and other features of the **SIPS** Preset System, see page 8.

(2) **XTime** knob. Sets the total **crossfade** time for note transitions in milli-seconds. If **RlsFade (7)** is set to less than 100%, **XTime** is still the time for the **fade-in** of the new note but the **fade-out** time of the old note will be less (specifically **RlsFade * XTime**)..

(3) Double-click this *assignment-button* to select a MIDI CC for add-on control of **XTime**. **NOTE:** If the **Pitch Wheel** is assigned, it will be used in the **uni-polar** mode (see page 6).

(4) **CC Range** edit box. Sets the percentage of **XTime** that can be added by an assigned CC **(3)**.

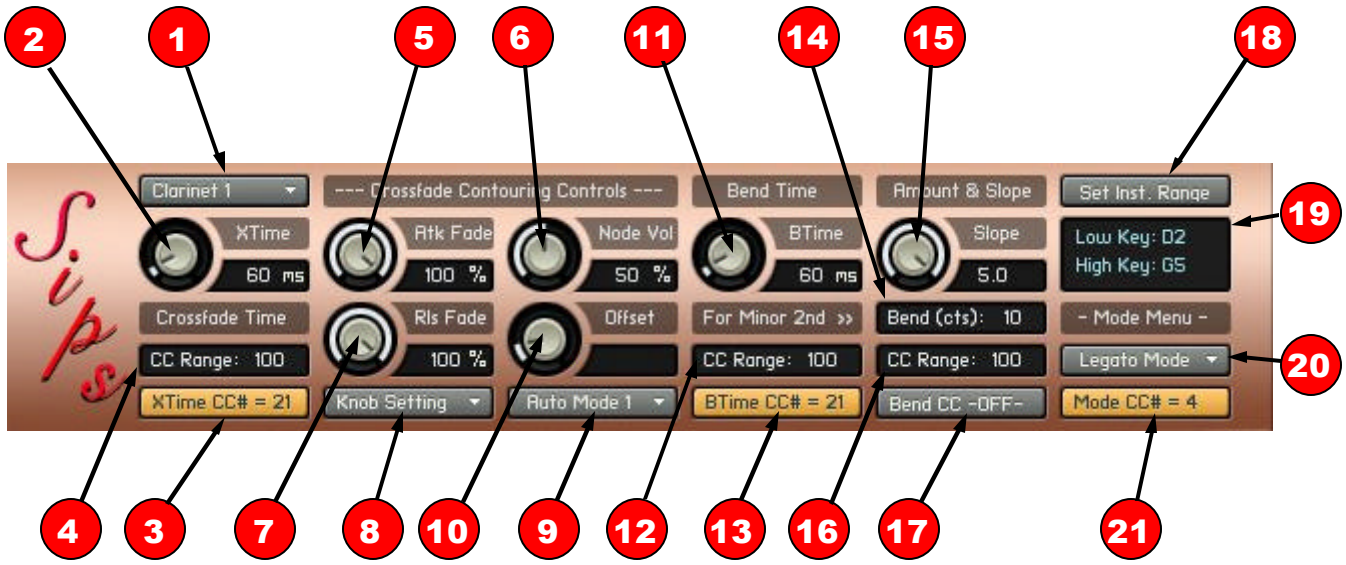
(5) **AtkFade** knob. Sets the percentage of **XTime** that the first segment of a 2-segment fade-in contour uses to rise to **NodeVol**. **NOTE:** If **AtkFade** is set to 100%, **NodeVol (6)** is ignored and a single-segment, linear fade-in will be used over the full **XTime** period.

(6) **NodeVol** knob. Sets the percentage of **Vmax** attained by the first segment of the fade-in contour.

(7) **RlsFade** knob. When the release-mode menu **(8)** is set to **'Knob Setting'**, this knob sets the fraction of **XTime** for the **pre-release** fade-out of the prior note. If **RlsFade** is set to 100%, the prior note fades-out over the full **XTime** interval using a single segment contour.

(8) **Release-Mode Menu** drop-down button. When **Knob Setting** is selected, the **RlsFade** knob **(7)** setting determines the fraction of **XTime** used for **pre-release** fade-out. When **Key-Lift** is selected, the **RlsFade** knob is ignored and the **pre-release** fade-out time is controlled by your keyboard legato note overlap time (see page 21).

(9) **Offset-Mode Menu** drop-down button. Use to select how the sample-start offset time (for 'inside' notes) is determined: **'Auto Mode 1** uses time since the start of the prior note. **'Auto Mode 2'** uses time since the start of the current legato phrase. **'Manual Mode'** uses the fixed time set with the **Offset** knob. And, finally, **'Manual +Rand'** uses a randomly varying offset with the knob setting as the minimum. The knob offset add-on will be a randomly selected multiple of 50ms over the range from 0 to 400ms but the same offset will never be used twice in a row.



(10) **Offset** knob. Sets a fixed sample-start offset time when **(9)** is set to **'Manual Mode'** or sets the minimum, base offset time when **(9)** is set to **'Manual +Rand'** mode. The **Offset** Knob's setting is ignored when **(9)** is set to **Auto Mode 1** or **Auto Mode 2**.

(11) **BTime** knob. Sets the amount of time (starting when **XTime** begins) over which bending occurs.

(12) **CC Range** edit box. Sets the percentage of **BTime** that can be added by an assigned CC **(13)**.

(13) Double-click this *assignment-button* to select a MIDI CC for add-on control of **BTime**. **NOTE:** If the **Pitch Wheel** is assigned, it will be used in the **uni-polar** mode (see page 6).

(14) **Bend** edit box. Sets the amount of bend (in cents) when the played interval is a minor 2nd.

(15) **Slope** knob. Sets the factor by which **Bend** increases for a played interval of one-octave.

(16) **CC Range** edit box. Sets the percentage of **Bend** that can be added by an assigned CC **(17)**.

(17) Double-click this *assignment-button* to select a MIDI CC for add-on control of **Bend.** **NOTE:** If the **Pitch Wheel** is assigned, it will be used in the **uni-polar** mode (see page 6).

(18) **Set Inst. Range** button. Accepts the next two notes played on a MIDI keyboard (or on the K2 keyboard) as the lowest and highest keys for the instrument. These keys are then displayed in **(19)**. **Note:** Setting the Instrument Range for the **SLS** will also set the Instrument Range for all the following **SIPS** member scripts.

(19) **Instrument Range Box.** Displays the Low/High Key of the range set for the Instrument.

(20) **SLS Mode Menu** drop-down button. When **Legato Mode** is selected, the legato effect is enabled. When **Solo Mode** is selected, no legato crossfading or bending is performed, but each overlapping note played will terminate the prior note. When **SIPS OFF** is selected, the script has no effect on the MIDI stream other than to provide sustain pedal control logic and 'musical carryover' (see page 15). This mode can be used to play chords if desired (albeit without the legato effect).

(21) **Mode CC** edit box. Allows a MIDI CC to be assigned to select the **SLS Mode**. When a valid CC is assigned, the 3 modes are selected by the CC's value as follows:

64 to 127	**Legato Mode**	
1 to 63	**Solo Mode**	
0	**SIPS OFF**	**Note: This mode also disables the SVS**

Understanding the Crossfade Contouring Controls

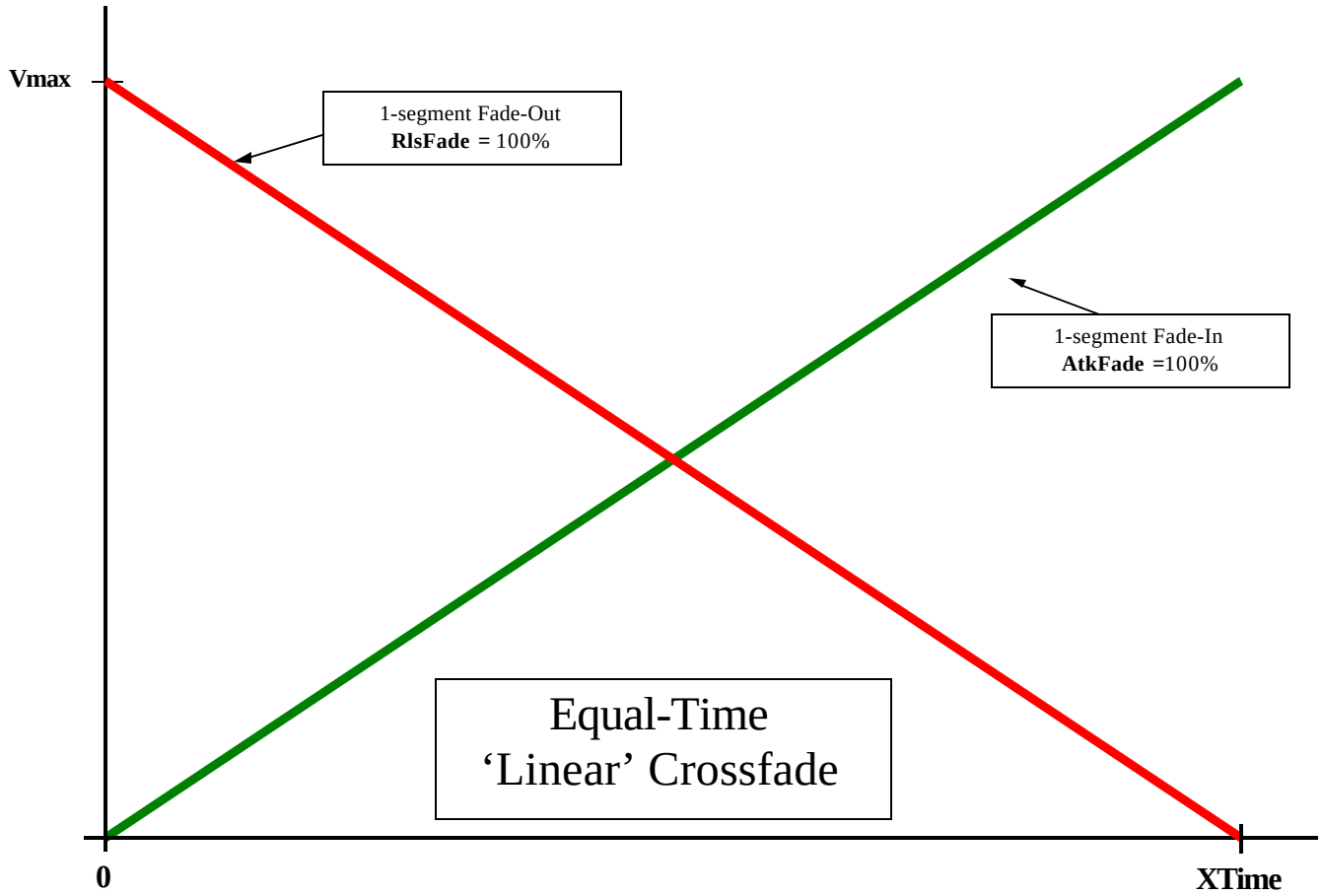
This section of the User's Guide will focus on the Crossfade Function and its parameters. The Bend Function and its parameters will be discussed in the next section. The purpose of Crossfading is to gradually get rid of the old note while simultaneously welcoming the new note. When this script was written, the change_vol() function was too noisey to be used for crossfading so the **SLS** was designed to use the fade functions with a novel, 2-segment shaping. Since this resulted in being so musically satisfying, there is little incentive to rewrite this code just for the sake of using the now-improved change_vol() function.

The simplest form of a crossfade is depicted in **Figure-1**. The old note is faded out over the period **XTime** and the new note is faded in over the same period of time. Now, before proceding, it should be pointed out that the precise volume versus time relationship produced by the KSP fade functions has not been disclosed. By ear they sound fairly smooth and thus probably follow some log/linear relationship. But for convenience of discussion and graphical depiction, we'll refer to the KSP fade in/out curves as **linear** and thus show them graphically as straight lines.

In **Figure-1**, the solid **Red Line** depicts the old note fading out from its full level, **Vmax**, to silence. The solid **Green Line** depicts the new note fading in from silence to **Vmax**. This type of linear crossfade, when done with the KSP fade functions, provides a fairly decent effect and has been used as the basis of several simple legato scripts. Naturally this 'bare-bones' form of crossfade works better with some kinds of instruments than it does with others.

The next logical improvement in getting to a more convincing legato transistion is to remove the attack transient of the new note. With the KSP this can be done by starting the new note's playback farther into the sample. The **SLS** provides two automatic and two manual modes for determining the amount of sample-start offset. The **Auto Modes** produce an offset equal to the time since the start of the prior note or the time since the start of the current phrase. In the **Manual Modes**, the offset is determined by the setting of the **Offset Knob** alone or together with a randomly chosen add-on. These four modes are selected with the drop-down list button beneath the **Offset Knob**. Note that the sample-start offset feature **only works when K2 is in Sampler Mode and not in DFD mode**. So if the instrument you are using requires sample offset to sound right, you will need to operate that instrument's groups in **Sampler mode**. Conversely, if you must operate the instrument in DFD mode, the crossfade alone will have to try to cover up the attack transient (since the sample-start offset settings will have no effect).

For most instruments it may be difficult to get a smooth legato transition without removing the attack portion of the 'inside' samples. So, even if you must eventually operate an instrument in K2's DFD mode, you may want to consider the following alternative. While you are designing your preset, operate the instrument in K2's Sampler mode and use the **SLS** Offset Knob (Manual Mode setting) to try to find a fixed sample start offset that produces a nice smooth legato sound. You can then note the time setting you are using for the Offset knob and write it down somewhere. Then, go to work editing your instrument as follows. Create a duplicate set of instrument samples and edit these samples in your favorite sample editor by removing the sample starts for the time interval you just wrote down. Put the edited samples in a new group(s) and arrange it so you can select either the normal group(s) or the edited group(s) when you play (for example by using a keyswitch). Then, you should be able to run K2 in the DFD mode (where it will now ignore the Offset Knob setting). However, if you now manually switch to the edited group(s) while you play 'inside notes', the fixed sample start offset (that you had when running in Sampler mode) will be provided by the edited samples instead. Since these samples have a fixed amount of time removed from their beginning, the effect ought to sound about the same now in DFD mode as it did in Sampler mode.



XTime is total crossfade time and **Vmax** is normal sample volume

Red - Prior Note fading out Green - New Note fading in

Figure 1

Different instruments require different **XTime** settings. For example, something on the order of 60ms works out well for a Clarinet but strings usually require a much longer crossfade time. For example, a Cello will typically need **XTime** settings from 300 to 600ms. For short **XTime** settings, the simple linear crossfade of **Figure-1** can be made to work quite well. However, as **XTime** gets longer, several undesireable things begin to happen. In order to avoid a 'chorusing-like' effect, as well as a general mudiness in the transistion sound, it is necessary to shorten the fade-out time relative to the fade-in time. So if the total fade-in time is always considered to be **XTime**, then the fade-out time must often be reduced to some fraction of **XTime**.

One way to do this is to provide an adjustment that sets the ratio of fade-out time to **XTime**. With such a knob set to say 25%, the crossfade profile would be as shown in **Figure-2**. The problem with this scheme is that a serious dip in volume occurs before the fade-in gets up high enough. But, apart from this volume dip, the legato transistion itself sounds much better than with the equal-time crossfade of **Figure-1** (at least for longer **XTime** values).

Another way to shorten the fade-out time is depicted in **Figure-3**. Here the old note fades out along the same path as it would in **Figure-1** until 25% of **XTime** elapses. Then, the note is terminated with the note_off() function. This causes K2 to act as though the key was released which in turn advances the sample playback to the 'release' phase of its envelope. Thus, this scheme effectively has a 2-segment fade-out curve which greatly shores up the volume dip with little or no unfavorable effects. For high **XTime** settings, this scheme provides a better overall quality legato transistion, yet greatly reduces the volume dip problem.

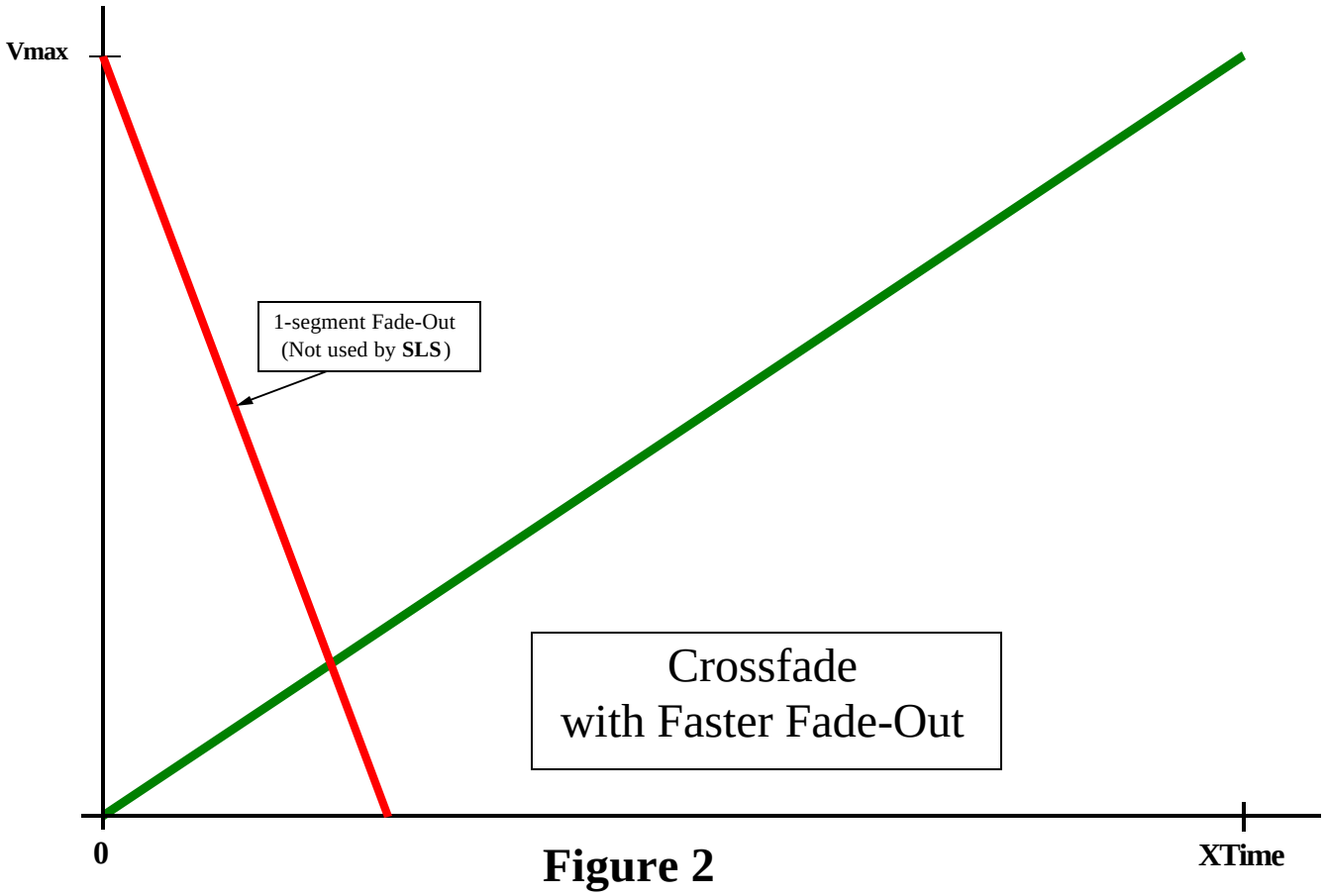


Figure 2

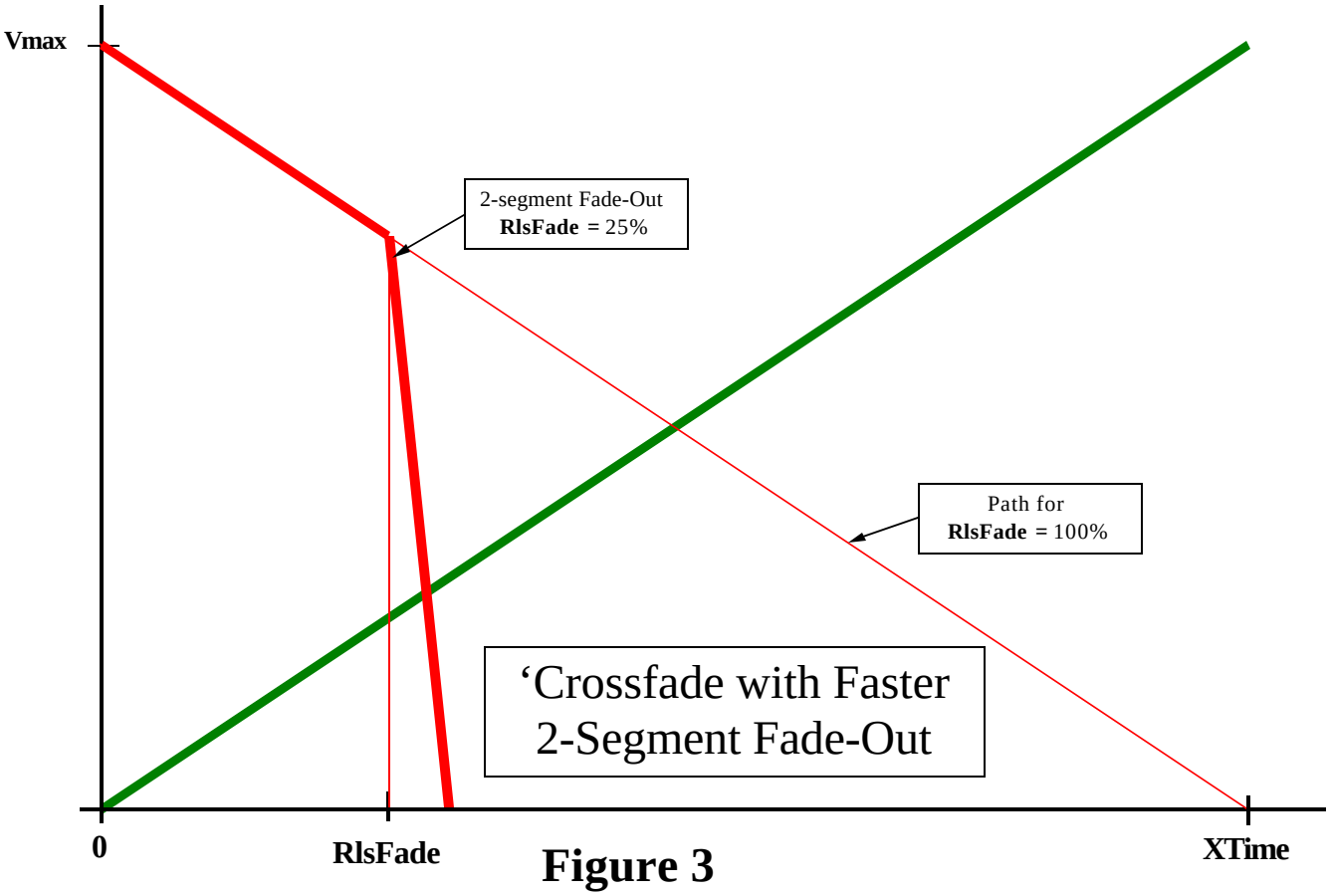


Figure 3

The reason this scheme is so beneficial is that the volume of the old note stays up where it is needed the most and yet the note is reduced to silence in a period not much longer than that depicted in **Figure-2**. The 2-segment fade-out shown in **Figure-3** is the technique used in the **SLS**. In addition, the **SLS** provides two different mechanisms for specifying what percentage of **XTime** is used for the first segment (the pre-release fade-out time). The selection is made with the release mode drop-down menu just under the **RlsFade** Knob. When the **Knob Setting** mode is selected, the **RlsFade** knob sets the fraction of **XTime** given to the **pre-release** fade-out time. For example, if **XTime** is set for 500ms and **RlsFade** is set to 25%, then the pre-release fade-out interval will last for 125ms before the prior note moves on to the envelope's release phase.

When **Key-Lift** is selected as the release mode, the **RlsFade** knob is ignored and instead, the **pre-release** fade-out ends when the prior note's key is released.. The **Key-Lift** time is governed by your playing style and how much you typically overlap the notes when playing legato. So, if you hold the prior note longer (ie make the overlap longer), the **pre-release** fade will be longer and if you make the overlap shorter, the **pre-release** fade will be shorter. Thus, you can vary the **pre-release** in 'real time' as you play (by varying the length of 'overlap' that you use). Depending on your keyboard skills and/or your playing style, you might find this mode to be fairly comfortable to use and it may give you some additional measure of control over the sound.

Now, while the fade-out depicted in **Figure 3** has two segments, unlike the first segment, the second segment is not directly under control of the script. The time of the second segment is governed by the release phase of the sample itself which in turn may be shortened by the release phase of its envelope. However, in spite of the fact that the script cannot directly set this release time period, the scheme of **Figure 3** for controlling the fade-out still works out to be quite musical in most situations. One reason this is so is because instruments that require a shorter total fade-out time than the fade-in time (ie **XTime**) are those instruments that have a slower attack time and a correspondingly slower release time. For such instruments, the natural release time is usually such that you can find a **pre-release** time (**RlsFade*XTime**) that dovetails nicely with it.

So, for faster attack instruments (which usually use rather short **XTime** settings), a **RlsFade** setting near 100% (as depicted in **Figure 1**) usually works out quite well. And, for many of the slower attack instruments (requiring larger **XTime** settings), all that is needed is to reduce the **RlsFade** setting as depicted in **Figure 3**. However, for some instruments that need higher **XTime** settings, the scheme depicted in **Figure-3** has a few shortcomings. During the legato transition, there may be a *small* (but undesireable) volume dip that occurs. Moreover, when **XTime** and **RlsFade** are set for the smoothest and best sounding legato effect, sometimes the response at high **XTime** values becomes kind of sluggish for playing faster passages. To overcome these problems, we need to get the new note up faster and yet not materially reduce the total fade-in time (to preserve the nice smooth legato sound). To accomplish this, the **SLS** provides a 2-segment fade-in that is faster at first and slower later. This is sort of the inverse of the fade-out curve (together providing something akin to an equal-power crossfade). Both the 2-segment fade-out (red) and the 2-segment fade-in (green) curves are depicted in **Figure-4**.

Two adjustment knobs are provided for setting the fade-in contour. The **AtkFade** knob sets the percentage of **XTime** over which the first segment of the fade-in curve rises from silence to where the 2nd segment starts. The **NodeVol** knob sets the percentage of **Vmax** that the first segment of the curve rises to before it changes its slope. Thus segment-2 of the curve rises from **NodeVol*Vmax** to **Vmax** over the remainder of **XTime**. With these two knobs a variety of different contours can be set, and with the proper settings, you will usually be able to overcome the volume dip and also make the script more responsive to faster passages (even at long **XTime** settings). Note that if you set **AtkFade to 100%**, the value of the **NodeVol setting is ignored** and a single-segment fade-in curve such as that depicted in **Figure-3** is obtained.

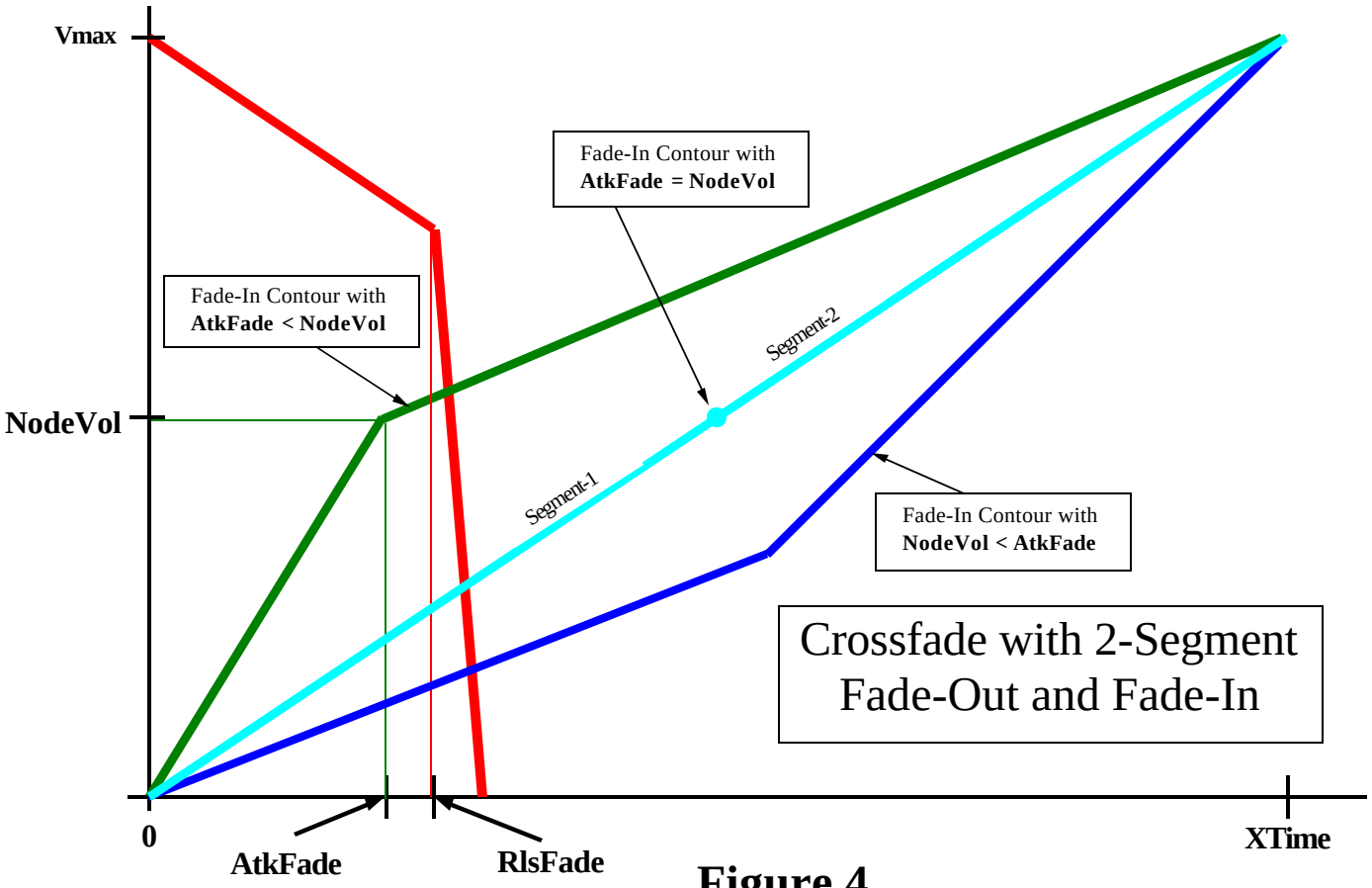


Figure 4

It should be mentioned that while these two controls can be set many useful ways, they can also be set in many **nonsensical** ways that should be avoided. To clarify this a little, consider the following. When **AtkFade** is set to the same value as **NodeVol**, the 2-segment fade-in contour coalesces effectively into a one-segment fade-in because segment 1 and segment 2 both have the same slope. For example, if **AtkFade** and **NodeVol** are both set to 50%, the (light blue) fade-in curve of **Figure 4** will be the result even though there are two segments. The first segment will traverse the first 50% of the straight-line and the second segment will traverse the last 50% of the **same straight line**. Now if **NodeVol** is set to a value lower than **AtkFade,** the contour will dip at the node which is just the opposite of what you will likely be trying to do with these controls. This sort of situation is depicted by the dark blue contour shown in **Figure 4**.

Thus when attempting to correct for side effects of the simple, 1-segment linear fade-in contour, you might begin by setting both **AtkFade** and **NodeVol** to 50%. This will start you out with the same sound as you had when **AtkFade** was still at 100% (ie a single-segment linear fade-in). Then, as you begin to tweak these controls, you will want to move them in such a way that keeps **AtkFade** less than **NodeVol** so that you don't get an inverse contour like that of the dark-blue curve in **Figure 4**. Remember that what you are trying to do is to make the segment-1 slope steeper and the segment-2 slope flatter as it approaches **Vmax**.

MIDI Control of the Crossfade Function

The **SLS** also provides for MIDI control of **XTime**. The *assignment-button* beneath the **XTime** knob allows you to assign a MIDI CC to provide 'add-on' control of **XTime**. The range of the add-on effect for **XTime** can be set with the companion edit box named **CC Range**. For complete details about MIDI add-on control, see the section titled **SIPS MIDI Control** in the introductory discussion of **SIPS** (see page 6).

Understanding the Bend Contouring Controls

When musicians play legato passages on real instruments, there is usually a certain amount of pitch bend that takes place in moving from the prior note to the new note. Just as the prior note smoothly crossfades into the new note, the prior note's pitch also starts to bend toward the new note and the new note bends toward its target value (from the side of the prior note). Thus, if an up interval is played, the old note starts to sharpen as it fades out while the new note fades-in somewhat flat as it bends toward the desired center value. Conversely, if a down interval is played, the old note starts to flatten as it fades out while the new note fades in somewhat sharp as it bends toward the desired center value.

The amount of bend depends on the instrument class but, for a given instrument, the amount of bend usually increases as the played interval widens. The **SIPS Legato Script** allows you to set the amount of bend for a played interval of a minor 2nd (ie one semitone) and also lets you specify the linear rate at which the bend will increase as the played interval increases. **Figure 5** illustrates the relationship between bend amount and the played interval.

Figure 5

Pitch bend as a function of the Interval
between a pair of notes played Legato

In **Figure 5**, the left side annotation identifies the amount of bend for a played interval of one semitone as **Bend1** and the amount of bend for a played interval of one octave as **Bend12**. The right-hand side annotation shows these same two bend levels related to the **SLS** parameters of **Bend** and **Slope**. The **Bend** edit box value is precisely the same as the **Bend1** value (ie the amount of bend for a played interval of a minor 2nd). So, you should set the **Bend** edit box for the number of cents of bend you desire for a minor 2nd. Then you can set **Slope** as follows. Determine how much bend you want for a played interval of one octave (Bend12). **Slope** should then be set to **Bend12 / Bend.** For example, if you want a bend of 20 cents for a minor 2nd, and a bend of 45 cents for an octave, set **Bend** = 20 and **Slope** to 45 / 20 = 2.3. Note that if you want the same amount of bend for any played interval you can accomplish that by simply setting **Slope** = 1.0.

Now in addition to setting **Bend** and **Slope**, you also need to specify the time interval over which the bend takes place. The bend always starts when the crossfade starts (ie when **XTime** begins) and continues for the time interval set by the **BTime** knob. Thus the rate of pitch bend is governed by the amount of bend and the time over which that bend takes place. Usually you will want a **BTime** setting close to that of **XTime** but you can set it shorter or longer as the occasion warrants. Since **BTime** and **XTime** can both be assigned to a CC, if you always want **BTime** to track with **XTime**, you could simply assign the same CC to control both.

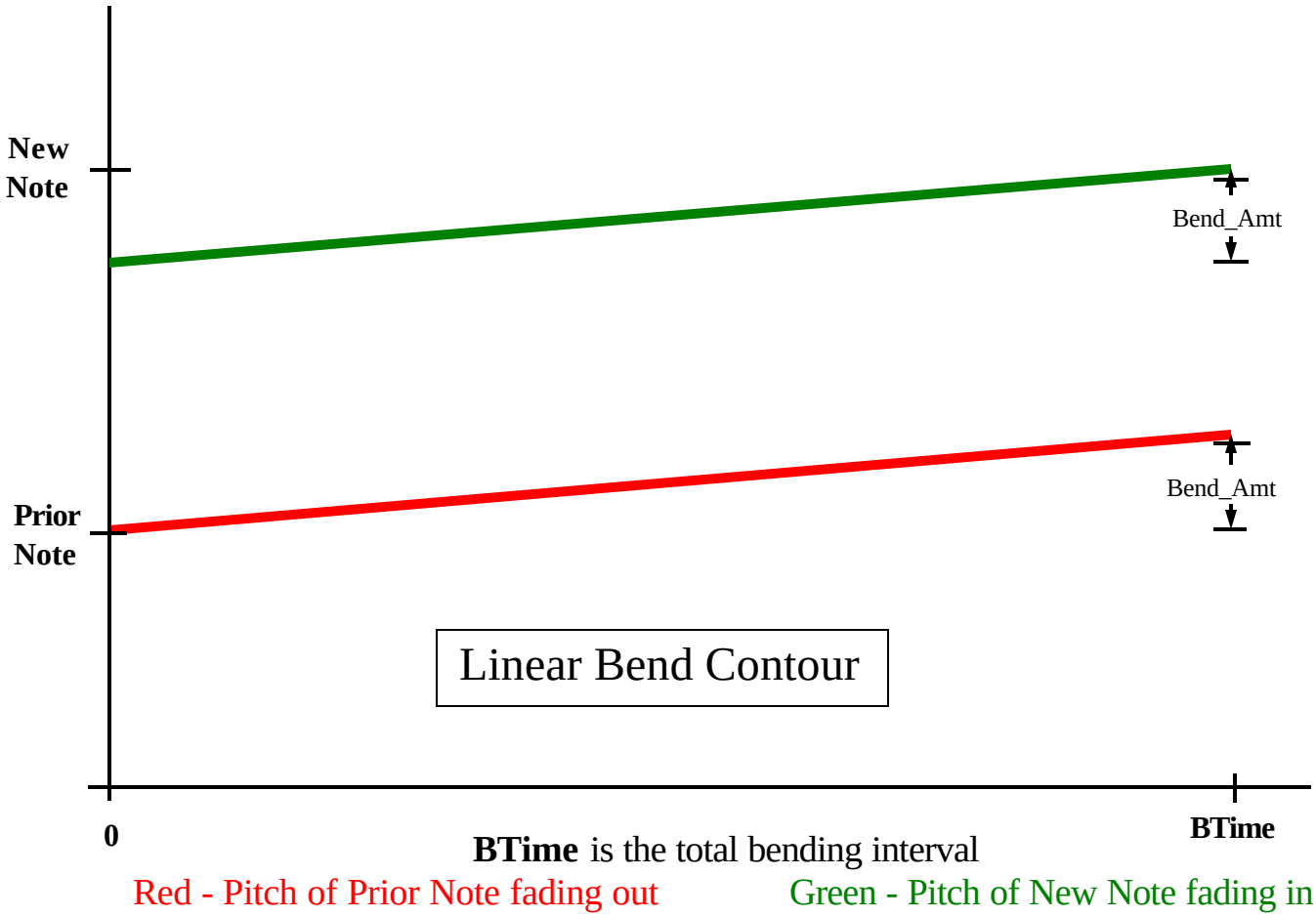


Figure 6

The remaining issue that needs to be discussed is how the bend is shaped over the **BTime** interval. Currently, the **SLS** uses a simple linear bend contour as depicted for an up-interval in **Figure 6**. This is by no means the only way to apply the bend over time but, it does provide a fairly 'musical' sound for most situations. One could, for example, have both the Prior and New notes track the same pitch (from the prior to the new) and follow something like an S-shaped contour over the **BTime** interval. Any number of such schemes could be devised, but it seems there is a psychoacoutical phenomenom at work here that leads to a discrepancy between what **should** sound good and what actually **does**. Some early experiments with various bend shapes (that were theoretically promising) produced disappointing results from a musical point of view. On the other hand, the simple scheme that was adopted seems to be quite musical. However, if someone experimenting with this idea should come up with another musical-sounding curve that is superior to the linear contour used, there is room for one more drop-down box where the **For Minor 2nd** >> label is now. This dropdown could be used to select from several bend contours.

MIDI Control of the Bend Function

The **SLS** provides for MIDI control of both the bend time and bend amount. The *assignment-button* below the **BTime** knob allows you to assign a MIDI CC to provide 'add-on' control of **BTime**. Similarly, the *assignment-button* below the **Slope** and **Bend** controls allows you to assign a MIDI CC to provide 'add-on' control of **Bend**. The range of the add-on effect for either of these parameters is set with the corresponding edit boxes named **CC Range**. For more details about MIDI add-on control, see the section titled **SIPS MIDI Control** in the introductory discussion of **SIPS** (page 6).

Guidelines For Making Legato Settings

Coming up with the right settings for a great-sounding legato preset is not unlike programming a synthesizer. By studying what the various synthesizer controls do and how they interact you can then experiment with settings and learn by listening to the results. Synthesizers usually come with a number of factory presets and it's often easier to pull up a preset for an instrument or sound that's similar to the one you are trying to develop and then try to intelligently tweak it.

Similarly, when setting up the **SLS** for use with a new instrument patch, the easiest way to begin is to pull up a preset for an instrument that has similar characteristics and then tweak the settings for the best sound. However, in order to do this, you must have a good grasp of what each control does and how they interact. So obviously you should first read the various technical discussions presented in this **User's Guide** to gain a good understanding of the various parameters from a theoretical point of view.

While tweaking an existing preset may seem like it will provide a fairly quick way to bring up a new instrument, you'll probably find this to be the case only after you have acquired some 'hard won' experience with the process. In order to do this, it is suggested that you try to bring up a few instruments from scratch using a procedure similar to the following.

NOTE: be sure to set the instrument range before you begin. Throughout the legato preset design, disable all other member scripts such as the **SVS** by clicking the KSP's Bypass button for each script. This is to make sure that you aren't being confused by some other effect being added by another script. Next, set up the **SLS** parameters as follows. Turn off the bend function by setting **Bend** = 0. Then set the release mode to **Knob Setting** and set **RlsFade** = 100%. Next set the offset mode to **Manual Mode** and set the **Offset** knob to 0 ms. Set **AtkFade** = 100% (remember **NodeVol** is igmored when **AtkFade** = 100%). Finally, set **XTime** to about 100 ms as a starting point.

Now, before changing any settings, disable the legato effect entirely (use the **SLS Mode Menu** in the lower right-hand corner of the control panel) and select **Solo Mode**. Next play a few legato phrases (ie play with overlapping notes) to get familiar with how the patch sounds when playing with just the **Solo Mode** active. When you play a new note in **Solo Mode**, any prior note still held receives a note-off command causing it to proceed to its envelope's release phase. Thus the notes are packed tightly together to the extent that the prior note's release occurs during the attack of the new note. In particular listen to how sharp an attack the patch was sampled with.

Now, enable the legato function (using the **SLS Mode Menu**) and play a legato phrase. Listen to how well or how poorly the crossfade softens the note transitions compared to the **Solo Mode**. If the instrument in **Solo Mode** has a fairly sharp attack, you can usually set **XTime** rather low (50 to 150ms or so). On the other hand, if the attack is rather slow in **Solo Mode**, **XTime** may have to be set somewhat higher. However, resist the urge to fix everything by raising **XTime** excessively. High values of **XTime** bring with it all sorts of bad side effects that have to be dealt with. One of the first things to try before raising **XTime**, is to raise the sample offset to see if getting rid of the attack portion of the 'inside' notes helps. But, don't raise **Offset** any higher than needed either. Rather use just enough of both **XTime** and **Offset** but don't overdo it. Also, remember, that **Offset** only works if the instrument groups in K2 are operating in **Sampler** mode, **offset is always ignored and treated as zero when K2 is in DFD mode**.

It can't be emphasized enough that setting **XTime** too high and then trying to compensate with the other parameters is a prescription for a 'less than stellar legato preset. You can always raise **XTime** a little as needed during your final rounds of tweaking. Now, once you have established a minimal starting point for **XTime** and **Offset** that seem to provide fairly smooth legato transistions, it's probably a good time to introduce the bending effect. For a starting point, set **BTime** to the same value as **XTime** and then set **Slope** = 1.0 and **Bend** = 20 cents.

Now play some minor 2nds (legato) and listen to the bend effect. Raise or lower the **Bend** value to provide the right amount of bend (for when you play an interval of a minor 2nd). Now play some octaves (legato of course) and slowly raise the value of **Slope** until the amount of bend sounds good for when you play an octave interval. Then, start listening to how wide legato intervals sound (5ths to Octaves) and listen for sounds of harmony or chorusing during the crossfade interval. If you have set **XTime** fairly low, it's unlikely you will hear anything like this but it's not uncommon for higher **XTime** settings. If you hear some undesireable effect along these lines, try lowering the value of **RlsFade**. You may need to set the value as low as 10% (but don't set it any lower than necessary) to subdue any chorusing or harmony artifacts, especially when you play wider intervals.

Now as you play lots of legato phrases both slow and fast listen for any problems. For example if everything sounds fine for slow passages but fast passages sound a little sluggish (usually when you have **XTime** fairly high), you may have to introduce a 2-segment fade-in contour. Another reason you may have to do this is if you seem to get a little volume dip between notes (again this usually occurs only for high **XTime** values). If you feel you need to introduce a 2-segment fade-in contour, begin with about 50% for both **AtkFade** and **NodeVol**. Then slowly lower **AtkFade** relative to **NodeVol** (ie lower **AtkFade** or raise **NodeVol**). **Generally you will want to avoid settings where NodeVol is set lower than AtkFade.** You may also find that once you introduce a 2-segment fade-in contour, you will be able to raise **XTime** a little without as much ill-effect as there was with a 1-segment fade-in contour. Since all 3 of these controls interact quite a bit, try changing all of them iteratively in small amounts rather than changing one a lot. In the end your ears will have to be your guide but you should also have a good feel for what to expect when you change something. As already discussed in the section titled **'Understanding the Crossfade Contouring Controls'** there are a lot of **nonsensical** settings for these controls which should be avoided.

On the bend function side of things listen to not only the bend amount but the bend rate. If you think a faster rate would sound better, first try lowering **BTime** a little rather than raising **Bend**. On the other hand if you think a slower rate would sound better try the opposite. Of course you may now want to tweak the **Bend** and/or **Slope** setting. Also, somewhere along the line you might want to assign the Pitch Wheel (or some other MIDI CC) for add-on bend control. Then as you play legato phrases you can increase the bend effect as you play when it seems musically appropriate. Initially try this with the companion add-on range, ie **CC Range** (for the **Bend CC**) set to 100%. Try to set **CC Range** such that you are using most of the range of the CC but never running out of headroom. If you are running out of CC headroom, increase the **CC Range** and if you are using too small a fraction of the CC's range, decrease the setting of **CC Range**.

When you think you have the settings pretty close to optimum, you might want to listen to the effect of using something other than the fixed, manual offset mode. The **Manual +Rand** mode can sometimes help reduce the legato-form of the machine-gun problem. Depending on your playing style, you might also want to try the **Key-Lift** release mode. Of course you can also assign MIDI CCs for **XTime** and **BTime** so that more things can be varied in real time. Obviously you may be limited as to how many CCs you can manipulate at one time but, if you are recording to a sequencer, you can always make multiple passes if necessary. **The more real time controls you can use artfully, the more realistic and convincing your legato passages will be compared with a 'set it and forget it' approach.** After you get the hang of it, you'll no doubt settle down to your own style of making legato settings but the plan just outlined will serve as a good introduction to the basics.

SIPS Vibrato Script

Introduction

The Vibrato effect modulates both the pitch and volume of a held note. The **SVS** allows you to set the amount of pitch and volume deviation independently and in addition, the modulation rate and its basic waveshape can be controlled. The basic modulation is trapezoidal but the 'flat-top' duty cycle, **Pk Dwell**, can be altered to provide waveshapes ranging from triangular to nearly square as depicted in **Figure 7**.

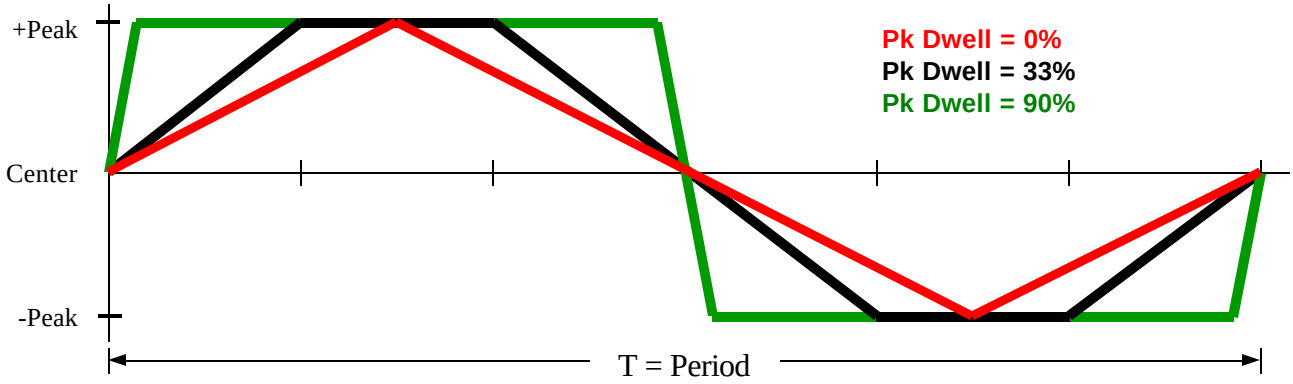


Figure 7

Modulation Waveforms Available with the **SVS**

The maximum amount of pitch and volume deviation is set by the **Width** and **Depth** panel knobs respectively. The frequency of these modulations and the waveform shape is set by the **Speed** and **Pk-Dwell** knobs respectively. To these four parameters, various kinds of random variations can be applied over time using the edit boxes named **Width-Var, Depth-Var, Speed-Drift/Speed-Per,** and **Wave-Drift/Wave-Per**. The combination of all these modulations and variations produce the 'Vibrato Effect' and the overall intensity of this effect can then be controlled over time in several different ways.

With the **Vibrato Control Menu** set to **Vib-Amt Only**, the overall intensity of the 'Vibrato Effect' (from 0 to 100%) is determined by the (MIDI controllable) knob named **Vib-Amt**. In this mode you can use the assigned MIDI CC to vary the amount of the 'Vibrato Effect' over time in any way you desire. With the **Vibrato Control Menu** set to **Envelope Only**, the amount of the 'Vibrato Effect' is controlled over time by a programmable envelope. Finally, with the **Vibrato Control Menu** set to **Vib-Amt + Env**, the overall amount of the 'Vibrato Effect' is controlled over time by the combined effects of the **Envelope** and the **Vib-Amt** knob (or its assigned CC).

The **Envelope** has four parameters which can be set to control the vibrato intensity over time. The **Onset** edit box can be used to set an onset delay from when the note starts until the effect is brought in. During the **Onset** time (in vibrato cycles), the intensity of the vibrato will be zero. For example, if vibrato **Speed** is set to 4.0 Hz, setting **Onset** = 8 will produce an **Onset** delay of 2 seconds. The **Rise** time edit box can be used to set the time it takes (after the onset delay) for the vibrato to ramp up to max. The units for **Rise** time, like **Onset** delay are vibrato cycles. Once the vibrato is fully established (ie the **Onset** delay has expired and the **Rise** time has passed), the vibrato intensity can be reduced slowly over time to simulate the 'relaxation' of vibrato often used by performers when very long notes are held. The **Decay** and **Sustain** parameters are used to set this 'relaxation' effect. The **Sustain** edit box sets the 'final level' (as a percentage of max) that the vibrato intensity will decay toward, and, the **Decay** edit box sets the **time** it takes for the decay to occur (in vibrato cycles).

Combining Vib-Amt and Envelope Control

The MIDI controllable **Vib-Amt** knob and the **Envelope** generator can be thought of as a pair of cascaded attenuation controls for the intensity of the vibrato effect. Thus, if **VI** is the maximum intensity of the effect, then the intensity heard is given by:

(1) Intensity Heard = VI * (Vib-Amt / 100) * (Env / EnvMax)

Note that if **Vib-Amt** is minimum, ie its value is **0%**, then no vibrato effect will be heard regardless of what value the envelope generator assumes. Similarly, if the envelope generator value is zero, then no effect will be heard regardless of the setting of **Vib-Amt**. Thus if you are using both the **Envelope** and **Vib-Amt,** and you set an **Onset Delay** for the **Envelope**, you will **not** be able to raise the vibrato intensity with the **Vib-Amt** knob (or its assigned CC) during the onset delay since the **Envelope** will keep the output level at zero. The effect of the **Vib-Amt** can be eliminated by setting it to100% or by selecting **Envelope Only** with the **Vibrato Control Menu**. The effect of the **Envelope** can be eliminated by selecting **Vib-Amt** only (with the **Vibrato Control Menu**).

Humanizing the Vibrato Effect

To add more realism to the vibrato effect, several humanizing functions provide pseudo-random changes to the parameters. You can add a small amount of random deviation to either the **Width** or **Depth** (or both independently) using the **Width-Var** and **Depth-Var** edit boxes to specify a percentage limit for the variations. Also, the vibrato **Speed** can be varied with a small, random 'drift' component to simulate the human fatigue and imperfection factor (using the **Speed-Drift/Speed-Per** edit boxes). For even more realism (for non-zero **Pk-Dwell** settings) you can apply a random drift to the vibrato 'waveshape' using the **Wave-Drift/Wave-Per** edit boxes.

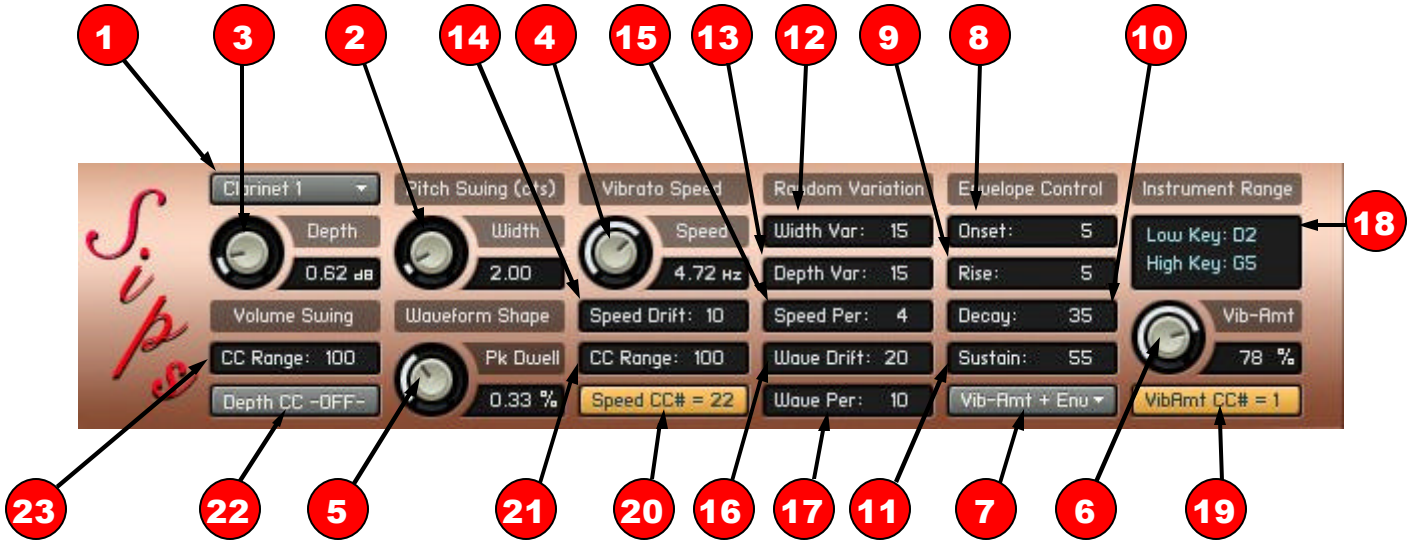
Setting Random Drift

Two parameters, **Speed** and **Pk-Dwell,** can be varied with a special kind of random drift component. For example, **Speed** drift is set with the two edit boxes labeled **Speed-Drift** and **Speed-Per. Speed-Per** specifies a maximum percentage (+/- of the **Speed** knob setting) that **Speed** is allowed to change during one vibrato cycle. However, unlike the random changes made to **Width** or **Depth** (which are always made from the knob value), **Speed-Per** changes are made from the last value (ie the knob value plus the accumulated random changes). Since there will be on average an equal number of ups as there will be downs, you might think that over the long haul, the 'average' value of **Speed** would remain near its knob setting. However, it is the nature of random numbers that periodic 'streaks' of more ups than downs (and vice versa) will occur over time. When this happens, there will be an accumulated 'drift' from the center value. The **Speed-Drift** edit box sets an upper limit on this accumulated drift that will be allowed (so that **Speed** doesn't 'drift' too far from nominal). So, generally, you will want to set **Speed-Per** to something less than **Speed-Drift**. This type of random variation in vibrato speed is much closer to what happens in the real world and therefore lends more realism than a straight randomization 'from the center'.

Similarly, you can apply the same kind of 'random drift' to the waveform shape using the two edit boxes named **Wave-Drift** and **Wave-Per**. The **Wave-Per** setting sets the maximum percentage (+/- of the **Pk-Dwell** parameter) that can occur in a single vibrato cycle. Remember that **Pk-Dwell** determines the percentage of the vibrato cycle during which the trapezoidal waveform is flat-topped. The **Wave-Drift** setting determines the overall maximum accumulated drift allowed as a percentage of the **Pk-Dwell** value.

This kind of drift away from the center value is known mathematically as a 'random walk' and often appears in math textbooks as a humorous puzzle known as the **Drunkard's Walk**. It has to do with an intoxicated man standing under a lamppost and taking steps in purely random directions (both away from and toward the lampost). One question often asked is: 'after some time, how far will he be from the lamppost'? Since his moves are purely random, at first you might think that he would pretty much stay put. But, the surprise answer is that his average distance from the lamppost will always continue to increase over time!

The Vibrato Script Control Panel



(1) **Preset Selector** drop-down menu. Selecting an Instrument or Instrument Class from this menu will set certain key parameters to a good starting point for you to further customize by ear. For a list of key parameters included in the **SVS** built-in presets, see page 13. For complete information on **User Presets** and other features of the **SIPS Preset System**, see page 8.

(2) Vibrato **Width** knob. Sets the maximum amount of pitch modulation (+/- in cents).
NOTE: pitch modulation is always synchronous with any volume modulation set with **(3)**.

(3) Volume (tremolo) **Depth** knob. Sets the peak amount of volume modulation (+/- in db).
NOTE: volume modulation will be synchronous with any pitch modulation set with **(2)**.

(4) Vibrato **Speed** knob. Sets the nominal frequency of the vibrato (pitch and volume modulation) in Hz.

(5) **Pk Dwell** waveform shape knob. Sets the duty cycle of the flat top and bottom of the trapezoidal waveform used for Vibrato modulation. With **Pk Dwell** set to 0.33, the waveshape is that of an equilateral trapezoid (see the Black curve in **Figure-7**, on Page 27).

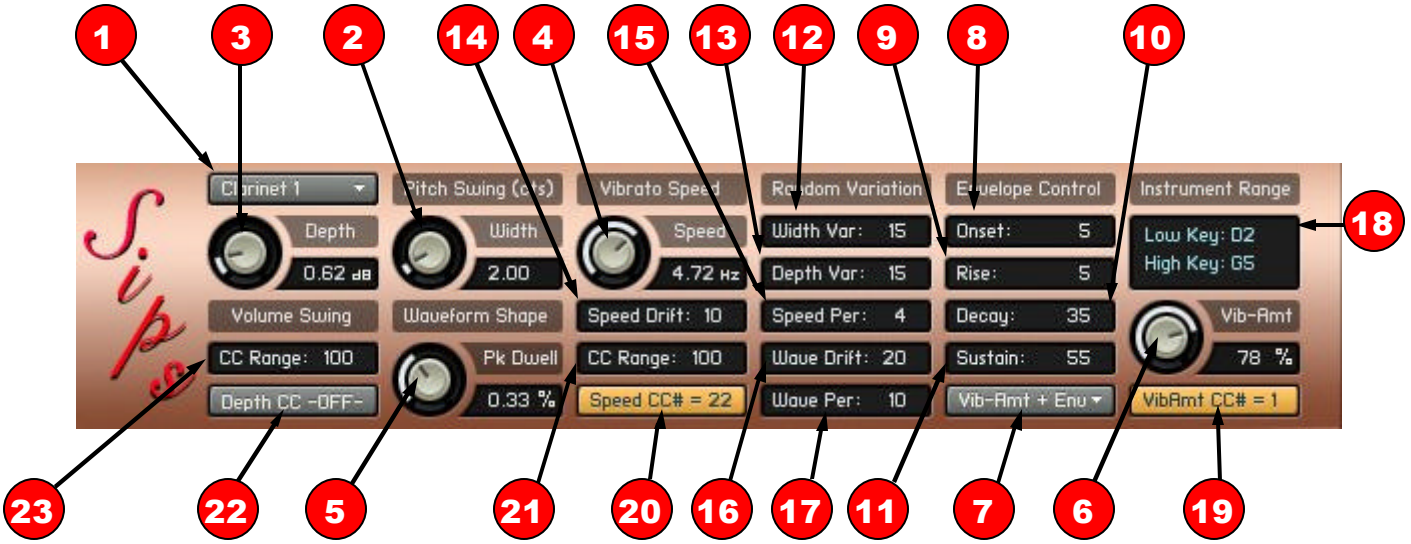
(6) **Vibrato Amount** knob. When selected by **(7)**, controls the total overall amount (0 to 100%) of the 'Vibrato Effect' that will be heard. This knob can be assigned to a MIDI CC using **(19)** to manually control the overall vibrato intensity over time.

(7) **Vibrato Control Menu** selections include: **Vib-Amt Only**, **Envelope Only,** and **Vib-Amt + Env**. When set to **Envelope Only** or **Vib-Amt + Env**, the control envelope is programmed with the settings of edit boxes **(8), (9), (10)**, and **(11)**

(8) Envelope **Onset** edit box. When a new note is played, vibrato isn't added until after the onset delay period set with this edit box. This delay time is in vibrato cycles. For example if **(4)** is set for 5.00 Hz, a delay setting of 8 will delay the onset attack of vibrato for 1.6 seconds.

(9) Envelope **Rise** edit box. After the onset delay expires, the 'Vibrato Effect 'will ramp up to maximum over the time period set by this edit box (again in vibrato cycles). In other words this parameter governs how slowly or abruptly the vibrato effect is brought in after the **Onset** delay.

(10) Envelope **Decay** edit box. After the 'Vibrato Effect' is established (ie after the onset delay and ramp up), the envelope can be made to decay slowly over time. This edit box allows you to set the number of vibrato cycles over which the envelope decays to the level specified by **(11)**.



(11) Envelope **Sustain** level edit box. This box sets the percentage of the full 'Vibrato Amount' that the intensity envelope will decay to over the number of cycles specified in **(10)**.

(12) **Width Var**. sets the limit for random variations (as a percentage +/-) of vibrato **Width.**

(13) **Depth Var**. sets the limit for random variations (as a percentage +/-) of vibrato **Depth.**

(14) **Speed Drift** sets an upper limit (as a percentage +/- of speed) that can occur from **accumulated** random variations in **Speed**.

(15) **Speed Per** sets the amount of random change in **Speed** (as a percentage +/-) that can occur in **one vibrato cycle**.

(16) **Wave Drift** sets an upper limit (as a percentage +/-) that can occur from **accumulated** random variations in **Pk Dwell**.

(17) **Wave Per** sets the amount of random change in **Pk Dwell** (as a percentage +/-) that can occur in **one vibrato cycle**.

(18) **Instrument Range Display Box**
Displays the Low Key and High Key of the range set (by the **SLS**) for the Instrument.

(19) **Vib-Amt CC** *assignment-button.* Allows you to assign a MIDI CC to control the overall 'Vibrato Effect' as controlled by the **Vib-Amt** knob **(6)**

(20) **Speed CC** *assignment-button*. Allows you to assign a MIDI CC for add-on control of **Speed**. The range of control provided by the assigned CC is set by **(21)**. **NOTE:** If the **Pitch Wheel** is assigned, it will be used in the **uni-polar** mode (see page 6).

(21) **CC Range** (for **Speed**) edit box. Sets the maximum percentage of **Speed** that can be added by an assigned MIDI CC **(20)**.

(22) **Depth CC** *assignment-button*. Allows you to assign a MIDI CC for add-on control of **Depth**. The range of control provided by the assigned CC is set by **(23)**. **NOTE:** If the **Pitch Wheel** is assigned, it will be used in the **uni-polar** mode (see page 6).

(23) **CC Range** (for **Depth**) edit box. Sets the maximum percentage of **Depth** that can be added by an assigned MIDI CC **(22)**.

Guidelines For Making Vibrato Settings

Generally, the easiest way to setup the **SVS** is to recall a preset for a related instrument family and then tweak it for the desired instrument. However, to familiarize yourself with what each parameter does, and for those situations where you may just want to start building a preset from scratch, this section presents a set of simple guidelines for making **SVS** settings.

A good way to start building a preset is to select the Preset named '*** Basic Setup ***'**.** This calls up a basic vibrato configuration by turning off the **Envelope** and all random variations. It also turns off volume modulation (**Depth**) and sets the waveform to a equilateral trapezoid (33% dwell). It leaves just a simple 4 Hz pitch modulation of about +/- 3 cents. After recalling the **Basic Setup**, assign a convenient CC (such as the **Mod-Wheel**) to control the **Vib-Amt** knob and turn off MIDI control of **Speed** and **Depth** so these parameters won't change accidentally. For the rest of this discussion, we'll assume you assigned the **Mod-Wheel** to **Vib-Amt**. Now, push the **Mod-Wheel** to max (so that **Vib-Amt** will be 100%) and then you can begin the prelimnary setup.

While playing and holding a note, adjust the **Width** knob for just a little more intensity of the effect than you would ever want for a maximum. You may also want to adjust the **Speed** knob (if 4.0 Hz is not a suitable vibrato rate) for the instrument you are setting up. You may then want to experiment with different waveshapes for the vibrato by adjusting the **Pk-Dwell** knob.

Now reduce the **Mod-Wheel** a bit until you have a 'pleasant' amount of vibrato going and then raise the **Depth** control to add some synchronous volume modulation to the pitch modulation. **Don't overdo this**. Usually only a small amount of volume modulation is needed for a realistic effect. Now, start playing some phrases and use the **Mod-Wheel** in the conventional way to control the intensity of the effect and see if you have things approximately right.

Next, try adding small random variations to **Width** and **Depth** using the **Width-Var** and **Depth-Var** edit boxes. Also, set a pair of values for vibrato speed 'drift' (ie **Speed-Drift** and **Speed-Per**) and, if **Pk-Dwell** isn't set to zero,. you can try adding some 'drift' to the waveshape (by adjusting **Wave-Drift** and **Wave-Per**). This can provide a particularly realistic-sounding variation to the vibrato (if you don't overdo it). The drift edit boxes are set as percentages of the parameters they modify so **if Pk-Dwell is zero, there can be no variation produced by the settings of Wave-Per and Wave-Drift.**

If you want to avoid multiple sequencer recording passes, and you think you might have your hands full 'riding the CCs' for the **SLS**, you might be able to automate most of the **Mod-Wheel** movements you would use by employing the **SVS Envelope** feature. To setup an appropriate envelope, first use the **Vibrato Control Menu** to select the **Envelope Only** mode. Then, dial in some appropriate settings for **Onset** delay and **Rise** time and play and hold a long note while you experiment with the **Decay** and **Sustain** settings. You might just be able to come up with suitable settings that will handle most situations without the need for riding the **Mod Wheel**.

Now when it comes to actually using your setup with a specific instrument, you may also want to assign additional MIDI CCs to allow you to vary vibrato **Speed** and possibly **Depth** in real time as you play (or under control of your MIDI sequencer) to lend additional variation related to musical context. Finally, you may want to do some iterative tweaking of the parameters and when you get it sounding the way you want, save it as a **User Preset** and **Rename** it accordingly. Then, so you won't lose your new preset, you can either re-save the **.nkp** for the **SVS** or, you can re-save the instrument **.nki** (along with the script).

Besides the **Vib-Amt Only** and **Envelope Only** modes, there might be situations where some combination of the two may be useful. If so, you can use the **Vibrato Control Menu** to select the **Vib-Amt + Env** mode and experiment with both control mechanisms active. Review **Combining Vib-Amt and Envelope Control** on page 28 so you will know what to expect.

SIPS Tips, Techniques, and Musings
Theo Krueger

SIPS provides such an amazing series of tools that you'll quickly notice it can elevate your sample libraries to new levels of realism and natural phrasing. To help you get the most out of it, here are some guidelines that I have found helpful.

Shoot for Equal-Energy Transitions

Shoot for Equal-Energy Transitions. The first and most important thing to keep in mind when experimenting is, "Equal Energy". We want every transition we make from note to note to be as seamless as possible and with equal volume to get the best legato effect. Try loading a Flute patch and keep on playing the same two notes. The first thing you will realize is that there is a small fade-in or a small volume 'jump' whenever a new note comes in. Depending on the XTime you have set, you will need to set the AtkFade and NodeVol accordingly in order to make the transitions seamless and with 'equal energy'. The best way to hear this is to listen at the transients of a sound — the violin bow or the flute breath noise, etc. Once you get those noises to be consistent you are on the right path.

Generally, for a small XTime (of 0.150 and below) you will need a small AtkFade and NodeVol to get the best results, but, if you need a bigger XTime you will realize that these three values are somewhat proportional. So, for an XTime of say 0.500, you will need both a larger AtkFade and NodeVol to keep the transitions seamless. It is important that your samples are 'compatible' with this equal volume thinking. In modern libaries there are a lot of expressive samples with rises and drops in volume. It will be much harder to get good results with those because moving from sample to sample will have volume inconsistencies. Generally, in testing, we found that simple "Sustain" or "sustain vibrato" samples worked the best since they are the most uniformly recorded.

Bending and intervals:

Depending on how lyrical and lively you want your bends to be, you can set the bending value and the time in which it occurs to more or less. Although setting the bend to cover the full interval at first seems like the most reasonable way to think -- since two notes should be connected like that in reality--, for most situations less bend produces better results. When the instrument is in a mix, the bending happens 50% with the script and the other 50% of it is psychoacoustical. So even though a transition with the script might not have as much bend as in reality, your brain will recreate the rest and you will actually hear it. Also, sections require much less bending (around 5-15) while solo instruments can be more expressive with up to 50-70.

The Xtime knob:

Setting Xtime depends mostly on how fast the instrument you choose can play or will play in your song. If it's a piccolo, you will have XTime very low so it can be used even in really fast runs and trills. For a french horn and Cello you can increase the XTime more. Don't forget that you can always change these parameters in realtime by using midi CC's from within the sequencer. No one setting can be perfect for all occasions.

Setting the optimal sample position:

Generally, in order to get better legato phrasing, the sample position (sample-start offset) should be set after the attack of the instrument. You will soon find that this defeats the "overlapping notes" philosophy you might've been following untill now... **what a relief!!** Depending on what result you are after, some instruments will need to have the start position right on the attack and others way after it, so if you want the trombone blowing sound to be heard in each transition, setting the position right at the start will give you that effect. If you are programming instruments with embedded vibrato (violin, oboe, etc.), the best sample position is right after the attack, before the vibrato kicks in. When going from vibrato to vibrato there can be some ugly sounding effects while from straight tone to straight tone it is less likely.

SIPS Tips, Techniques, and Musings
Andrew Keresztes

All of my preset demos were created using East West Quantum Leap Symphonic Orchestra XP Pro to illustrate the SIPS Legato tool. These are just examples of what can be done with the SIPS in real time without tons of editing to get a legato type sound. These demos are not benchmarks… like any instrument, with time and practice using SIPS, **I'm sure much better results can be achieved**. Most of the patches I used had modulation cross-fade features that would allow the instrument to be more expressive. For example, if the modulation controller was at 127, the violin would have a sharp attack and if the mod wheel was at 0 then it would have a slow attack.

Prepare Your Library Instrument

So, first things first. In order to incorporate the SIPS script with EWQLSO I had to eliminate any release groups in the patch. Otherwise, there would be a legato pitch-bend **and** a reverb decay **without a pitch-bend** playing back at the same time. The release groups can be identified by the suffix 'rel' in the group names when you click on the Group Editor button inside the patch editor. Select all the groups that have a "rel" appended to them and under the Group edit drop-down menu choose "Delete Selected Group(s)". Since so many libraries now include release samples, I thought this might be useful information.

Use MIDI CCs

Now, from the Script Editor, load in the SIPS Legato tool. In all my mp3 examples, I used continuous controllers (CC) to vary the way SIPS controlled the legato passages. In my case, I assigned CC 110 to both XTime CC **and** BTime CC (both controlled by the same slider). I then assigned CC 111 to Bend CC. This way I was able to ride faders as I was playing parts and vary the amount of the cross-fade and Bend Time between the parent and child note (the 1rst and 2nd note) with just one slider. To make the legatos into more of a glissando, I would then ride the CC 111 fader (assigned to Bend CC) as I was playing. It can be a little tricky, but it becomes more intuitive as you do it. Again, this was just my approach to it…. I'm sure there are better ways that we will hopefully all share with each other.

So, if you were (for example) playing my solo violin SIPS Legato preset, you would be able to play faster passages with the sliders (in this case CC110 and CC111) all the way down. Then to get a slower glissando effect, you would ride both sliders up 2/3s or all the way up. This is all based upon the initial settings of the SIPS Legato Patch. You can customize it anyway you'd like.

Use an Appropriate Patch

I want to stress how important it is to select the right sound patch to be used in conjunction with SIPS. If your patch has a really slow attack, you won't be able to play faster legato passages…. So… always bear in mind what kind of sounds you want associated with SIPS.

Have fun.

Thonex

Installing A Third-Party Script
(Windows)

NKP Files

If you have a script that's supplied in the **.nkp** file format, you can make it available to K2 as follows. Launch Windows Explorer and navigate to the main K2 program directory. For a typical default installation of K2 the path would be: "C:\Program Files\Native Instruments\Kontakt 2" but you may have put it elsewhere. Once you have located the Kontakt 2 directory, navigate further to:

C:\Program Files\Native Instruments\Kontakt 2\presets\scripts.

You can now put a copy of the **.nkp** file anywhere in the scripts folder (or any of its sub-folders). Or, you can create a new sub-folder of your own and put the **.nkp** file in it (just so the file is accessible **via** the 'scripts' folder). **NOTE: If you do this while K2 is running you may have to close and re-launch K2 before you will see your new script in K2's list.**

Running A Script

Once a 3rd-party script is installed as a **.nkp** file in K2's 'scripts' folder, you can run it just like any other script. For example, to add the script to one of your instruments, do the following. Launch K2 and then load the desired instrument. Open the instrument editor (by clicking on the Wrench Icon) and then Open the Script Editor. Up to 5 scripts can be installed for any given instrument so choose one of the 5 tabs to select in which of the 5 available slots you wish to install the new script (usually this will just be the first slot). Now, click the **Script** button on the left side of the KSP Editor panel and navigate through the folders of the drop-down menu to find the desired script and click on it. The script's control panel (if any) will appear and the script should be ready to use with the current instrument. Depending on various details of how the script is implemented, you may need to adjust some knobs and make other settings for the best performance with the loaded instrument.

If you wish to retain your custom settings for the script (for the next time you use it) there are two ways to do this but both of them involve re-saving the script. You can either resave the **.nkp** file itself or you can resave the instrument as a **.nki** file. If you do either of these, you may want to resave with an altered name so that you will still have the original version of the script and/or instrument intact. To resave just the customized script as a new **.nkp** file, click on the **Script** button and then select **Save Preset**. This will open a standard Explorer dialog where you can now save the script under any name that you wish. To save the customized script with the instrument, simply use K2's **Load/Save** button as you would to save any instrument. If you save the script with the instrument, the next time you load that instrument the script will be loaded with it and, the customized panel settings should also be restored. If you have resaved the script as a new **.nkp** file, you can load it just as you did initially. That is, first load an instrument that you want to use the script with, and then load the script. The only difference between this and when you first loaded the supplied **.nkp** file, is that the resaved **.nkp** file will load with your customized settings (that you had when you saved the **.nkp**) still intact.

NOTE: In order for your customized panel settings to be saved and recalled as described above, the script must be written properly using 'persistent' variables in all the appropriate places. Most scripts will be written this way but, if you have one that isn't, your customizations may not be properly saved and recalled. Also, regarding using more than one script with an instrument, be advised that in general, you may not be able to use several scripts together unless the scripts are specifically designed to be used together. See page 5 of this User's Guide for more information on chaining or cascading scripts.

Installing A Third-Party Script
(Windows)

Source Files

If you have a script that is supplied as a **.txt** file, the procedure to install it is different from that of installing a **.nkp** script file. Source code files can usually be viewed in any plain text editor such as 'Note Pad' and what you will see is the series of instructions that the script's author used when writing the script. If the source code was written using all the syntax rules of the KSP scripting language, it is known as a **K2-Ready** or simply a **K2-Source** file. To install and use such a source file with an instrument, launch K2 and then load the instrument you want to use the script with. Open the instrument editor (by clicking on the Wrench Icon) and then Open the Script Editor. Up to 5 scripts can be installed for any given instrument so choose one of the 5 tabs to select in which of the 5 available slots you wish to install the new script (usually this will just be the first slot). Assuming that the slot you choose is empty (ie it doesn't already contain a script), click on the Edit button to open the KSP Editor's empty text-entry window. If the slot you choose already contains a previously loaded script, the editor's window will have text in it. If this is the case, fiirst click on the **Script** button and choose the **Empty** preset. This will clear-out the old script so that the KSP Editor's text entry window will now be empty.

With the **.txt** file loaded into 'Note Pad', hit **ctl-A, ctl-C** to select all the text and put it in the Windows clipboard. If you are using a text editor other than 'Note Pad', use whatever procedure is appropriate to select all the text and place it into the Windows clipboard. Now click on the empty KSP Editor text window (to give it the focus) and then hit **ctl-V** to paste the clipboard into the KSP text window. Then, click the **'Apply'** button and the orange square to its left should go out -- indicating a successful compilation of the **K2** source code (the script's control panel, if any, should now appear). Next, name the script by double-clicking the title box and typing the desired name followed by the enter key. Finally, you can close the text window and save the script as a **.nkp** file To do this, click the **Script** button and select **'Save Preset**' at the bottom of the drop-down list. An Explorer-type window should open with K2's 'script' folder at the top and several sub-folders under it. You can now select a sub-folder (or create a new one) and then give the preset a suitable name and save it. In order for K2 to find it later, you must save it in the **'scripts'** folder or any subfolder under it. You may now proceed as described in **'Running A Script'**.

NOTE: When you load a new K2 source text file as described above, you will get a **'fresh'** install whereby the new script will be compiled with all the panel settings and parameters initialized to their **default values** (as originally established by the script's author). When you are installing a new source script into a slot that has had a previous script installed in it and you want a **'fresh'** install, **it is important that you first load the Empty preset** before pasting the new script into the text window. If instead of clearing out the old script, you simply use **ctl-A, ctl-V** to paste the new text **'over'** the old text, when you hit **'Apply'**, unexpected results may occur. Since K2.1, NI has added double-buffering of persistent variables. So, if the previously loaded script uses persistent variables with the same name as used in the new script, the previous values assigned to these variables will be retained when the **'Apply'** button is hit. This may or may not be what was intended. For a **'fresh'** install where you want the new script to be set to its defaults, you won't want previous values to be retained. And, since it's not always easy to tell if the old script has persistent variables with the same name, if you want a **'fresh'** install, **you should always be sure to load the Empty script first**. On the other hand, when you are updating to a new version of the same script (and the script has been written properly), this retention of persistent variables can be profitably exploited. See page 9 for a discussion of how **SIPS** uses this feature to perform an **Auto-Import Update**.

Installing A Third-Party Script
(Windows)

Special Source Files

While source code files can be created in the KSP text editor, script authors will seldom do that for anything but a very small script. The KSP editor is not too friendly and writing a non-trivial script in such an environment would be impractical. Therefore, source text files are usually created using an 'external' editor.

If the script is developed in a standard external editor such as 'Note Pad', the source code will be written in **K2 format** and can be pasted into K2 as previously described. However, **Nils Liberg** has written a very powerful editor, specially geared toward writing K2 scripts. His **KScript Editor** provides a whole plethora of very powerful syntax extensions and features that make writing and organizing scripts so much easier than without it. The result is that almost every serious script writer is now using the **NL Editor** to write their source code. However, **NL** source code cannot be directly pasted into the KSP Editor and run. **NL** source code must first be processed and converted to **K2** source code. If you have the **NL Editor** this is very easy to do by simply hitting the **F5** key on your keyboard. This causes the **NL Editor** to compile the source code into **K2-Ready** code and place it in the windows Clipboard.

So, even if you don't write scripts, you should get a copy of the **NL Editor** in case you need to install a source file written with **NL** extensions. Moreover, even for scripts written without extensions, you may still want to use the **NL Editor** because you will be able to view and/or print the code with syntax highlighting and enjoy the many other useful features as well. And, since **Nils** has graciously made his editor available as a free download, every K2 user that intends to use or write scripts should have a copy of this editor. http://nilsliberg.se/ksp/ Once you've had a chance to use Nils' KScript Editor (not to mention the many other very useful scripts available on his site), maybe you'll want to click his PayPal button and make a generous donation to show your appreciation for all the wonderful contributions he has made and continues to make for the benefit of the K2 community.

Placing the K2-Ready Source in the Clipboard

For purposes of installing and/or updating a script, it is often necessary to put a copy of the script's source code (in **K2 format**) into the clipboard. From the clipboard the source code can then be entered into to the KSP editor's text window where it can be compiled and run. 3rd-party scripts may be supplied as source code (a **.txt** file) or in NI's preset format (a **.nkp** file) or both. When supplied as source text, it may be either in **K2** or **NL** format as already discussed. The following will describe some of the ways you can place the K2 source into the clipboard.

If you were supplied with a **K2-Source-code** file for the script, simply load it into any general editor such as **Note Pad.** Then, whenever you need to place a copy into the clipboard, simply use **ctl-A, ctl-C** (while in **Note Pad**). If you were supplied with an **NL Source-code** file, simply load it into the **NL Editor**. Then, whenever you need to place a copy into the clipboard, simply hit **F5** (while in the **NL Editor**).

If the script is supplied only as a **.nkp** file, you may or may not be able to place the **K2 Source** into the clipboard. If the script is **not** 'Password Locked', you can use the following procedure to obtain the **K2 Source**. Launch K2 and load some instrument. Open the instrument editor (by clicking on the Wrench Icon) and then Open the Script Editor. Select the first of the 5 script tabs and then click the **Script** button on the left side of the KSP Editor panel. Navigate through the folders of the drop-down menu to find the desired script and click on it. Now open the KSP editor's text window where you should see the **K2 Source** code. If you now use **ctl-A, ctl-C**, the text will be selected and placed in the clipboard. You can now open Note Pad and paste the text into it with a **ctl-V**. After this, whenever you need to place a copy into the clipboard, simply use **ctl-A, ctl-C** (while in **Note Pad**). **NOTE: If the script is Password Locked, you cannot use this procedure to obtain the source code** (unless of course you know or can determine the Password)**.**

More than you ever wanted to know

About Big Bob

I'm including this page, about my background and interests, because my motivation for using K2 and writing scripts may be quite different from most of you. As a result, what may work very well for me, may not be suitable for what you are trying to do. However, to the extent that one of my scripts may be useful to others, I offer them freely to the K2 community. I also try to provide a complete documentation package so that anyone skilled in the art of programming can easily adapt and/or improve upon any of my scripts for similar or other purposes without the need for a lot of 'reverse engineering' effort. So, if you find one or more of my scripts not quite 'hitting the mark' for your situation and you want to tweak it, knowing what my objectives were when I wrote it might be helpful.

I'm a retired Engineer (BSEE) and hobby musician. I retired in 1990 after about 30 years that was almost equally divided between Analog and Digital Circuit Design and then (in the latter half of my career), Microprocessor and Software Engineering. On the musical side of things, I've been making (or attempting to make) 'one-man-band' recordings since about 1951 via various forms of 'multitracking'. I've always used real acoustic instruments but, since I'm getting up in years, my 'chops' are beginning to wane. So, slowly over the last 10 years or so, I've become very interested in sampling technology as a way to allow me to continue recording past my prime. However, I won't use synthesis unless I can make it sound like the 'real thing' and, until recently, extremely realistic synthesis has been very hard to do, especially for wind instruments. So my main focus has been in that area. Since I'm an old geezer, I like doing older musical styles. For example, I do a lot of Big-Band Swing and Dixieland stuff. While I can still play all the needed instruments, one of the consequences of 'old age' setting in is that I'm quickly starting to lose manual dexterity (arthritus in the joints and all that sort of stuff). So, I know it's only a matter of time before the quality will begin to suffer noticeably.

I'm telling you all this so you will know where I'm coming from. To do convincing Dixieland, for example, the trombone has to use the slide a lot and to syththesize a realistic trombone glissando, I needed to do formant-corrected bending. That was my original motive for designing the PCE. Similarly, I'm hoping that my new **SIPS** scripts will eventually provide convincing legato and vibrato emulation; simple LFO vibrato just doesn't sound realistic enough. And, good sampled vibrato is too hard to come by (and too inflexible). So I've been recording my own wind-instrument playing and analyzing the vibrato and legato sounds and trying to get a handle on what I need to do to simulate it. However, the thing I want to emphasize here is that I may be trying to make a sampled clarinet sound good and you may want to apply these scripts to a string patch. While legato and vibrato techniques vary considerably between instrument families, there is also much common ground. So I'm hoping that the **SIPS** Scripts will eventually have enough flexibility to be used on any instrument. But, keep in mind that it was initially crafted for wind instruments, especially Trombone and Clarinet (my trumpet chops are still fairly good so I'll tackle that last). Another thing I should emphasize is that I intend to use these scripts with Sequencer-controlled Playback, not Live Playing.

All that being said, I'm hopeful that by making my scripts available to the K2 community and encouraging an open discussion about ways of improving them, we may all benefit in the end. I know that many of you are much more accomplished musicians than I am and your input will be most valuable. As a community we are blessed with a number of members who are also excellent programmers and many of these have shown a willingness to share their work with us also. So, I'd like to encourage as many of you as can, to participate in future script development and testing. The Good Lord willing, we may reap a rich harvest of very useful tools and, if nothing else, we may get to know each other a little better, and, you can never have too many friends!

Bob

The Best Things In Life Are Free

SIPS is distributed free of charge, all you have to do is download it and print out the **.pdf** manual and you'll be all set to go. **SIPS** is also 'open source', which means you can alter it or add to it as you wish. To facilitate that, **SIPS** comes with a complete documentation package including design flow charts and heavily-commented source code.

So, you can either just use **SIPS** as it is or, if you are skilled in the art of programming, you can easily modify **SIPS** without having to do a lot of 'reverse engineering'. However, if you decide to modify or build on **SIPS**, I would like to ask you a favor. Please personalize what you do to distinguish it from the original. By that I mean at least change the Title Block of the source code and use your name as the author. It would also be nice if you would give the new script a different name, and version number series, such as **'Son of SIPS'**. The reason I'm asking you to do this is that I intend to continue the **SIPS** series (at least for a while) and I want everyone to be able to distinguish the original from yours so that the waters don't get too muddy.

My intention from the beginning was to give **SIPS** free to anyone who might benefit from it. However, knowing the effort that goes into something like this, many well-meaning friends have suggested that I at least ought to allow grateful users of **SIPS** to make some kind of gift (if they were so inclined). To that I have always responded that I really didn't want to receive any monetary compensation, knowing that others were benefiting from my effort is reward enough. After all, it's more blessed to give than receive you know.

But, persistent bunch that they are, the next suggestion was that I should at least encourage **SIPS** users who felt the desire (and had the means), to make a donation to one of my favorite charities. Well, after prayerful consideration of this idea, I see no harm in it. It's no secret that I'm an evangelical Christian and I'm always interested in raising money for the Lord's work. So, here's the deal. If you are using **SIPS** and you find it useful, and, you feel an overwhelming urge to want to do something nice for me in return, please click on one of the links below and make a nice contribution to one of these fine organizations. Whether or not you do this will just be between you and the Lord. This is strictly voluntary and I don't want anyone to feel under any obligation whatsoever to do this, but God Bless you if you do.

Have a fantastic day,

Bob

1. Make a donation to the Billy Graham Evangelistic Association.

https://www.billygraham.org/donate.asp

2. Make a donation to The Moody Bible Institute of Chicago.

https://safeweb.moody.edu/support/index.php?afterset=1

3. Make a donation to Bible League Organization.

http://www.bibleleague.org/donate/index.php

4. Make a donation to Union Rescue Mission, Los Angeles.

https://giving.silaspartners.com/donatenow/unionrescuemission/

5. Make a donation to Salvation Army.

https://secure2.salvationarmy.org/donations.nsf/donate?openform&t=US_USC*USE*USS*USW&redirect=1

6. Make a donation to Grace Brethren International Missions.

https://donatelinq.net/donate/gbim-donate.asp?mid=gbimorg