Using New macro/plugin to Generate a Custom Macro or Plugin

Bob Zawalich October 23, 2022

Introduction and preparation

The document goes into detail about using the **New macro/plugin** button in certain plugins to generate a custom macro or plugin that will be the equivalent of running a plugin without seeing its dialog, but with specific dialog options selected. It will use the plugin **Add Instrument Change With Names** as an example of this type of plugin. As of this writing, the plugins that support this feature are:

- Add Text to Blank Page version version 01.14.00
- Add Instrument Change With Names version 01.09.01
- Convert Legacy Chord Symbols version 01.44.00
- Filter Notes By Beat version 01.13.00
- Filter Notes By Duration version 01.16.00
- Filter Notes By Position version 01.13.00
- Filter Other version 03.25.00
- Filter With Deselect version 01.68.00
- Flip Selected Notes version 01.12.00
- Move Pitches to Transposed Mid Line version 02.01.00

The installable plugin **Add Instrument Change With Names**, which requires Sibelius Ultimate 2022.9 or later, lets you create an **Instrument Change** while specifying non-default Full and Short I**nstrument Names** and **Staff Names** along with the **Instrument Change**.

This document will explain how to use the **New macro/plugin** button to generate a plugin can be run to create the same **Instrument Change** with all its properties, without needing to see and fill the plugin dialog each time.

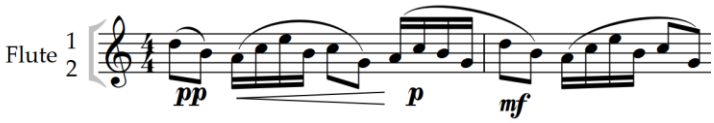
Required Tools

You will need to install the most recent versions of these plugins:

- **Add Instrument Change With Names**
- **Execute Commands**
- **cmdutils**
- **Command Search Shortcut**

Specifying the Instrument Change.

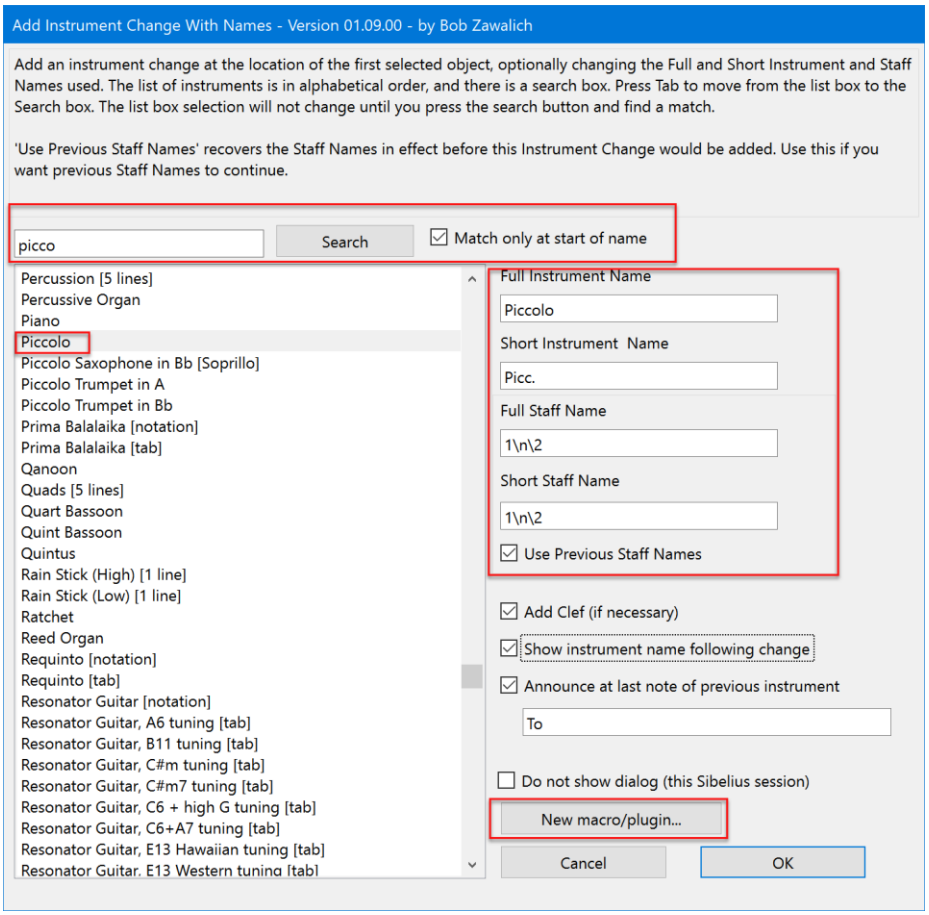
Let's say we have a **Flute** staff that uses Staff Names 1/2,



and we want to create a **Piccolo** Instrument Change with the same Staff Names.

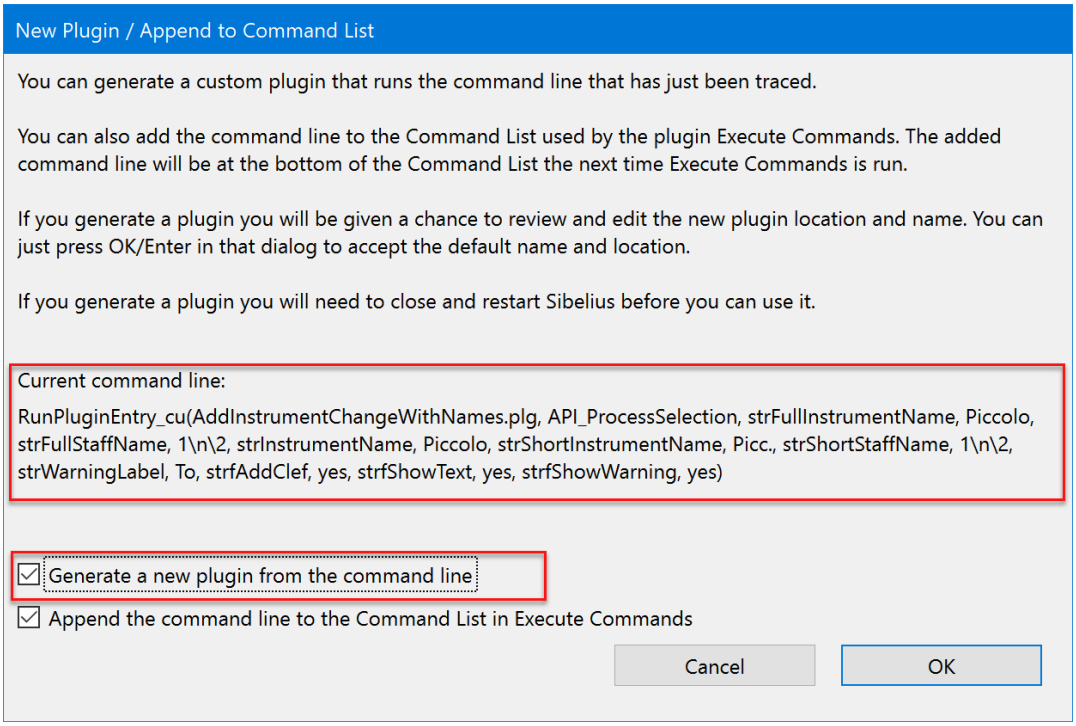
Run **Add Instrument Change With Names**, and search for **Piccolo** in its list. Once found, check **Use previous staff names.** This will fill the Staff Name fields with the Staff Names, if any, currently in use, and in this example, we get "1\n\2". Check any of the other check boxes as desired. If you were to click **OK**, the **Instrument Change** would be added at the start of the current selection.

But let's not do that this time.



Instead, we will use the **New macro/plugin** button in the lower right to create a custom plugin that will recreate this **Instrument Change**, with all its current dialog settings, whenever the custom plugin is run.

Press **New macro/plugin,** and if the plugin **Custom Search Shortcut** had been properly installed, you will see some text being written to the plugin trace window (which you can ignore), and this dialog box:

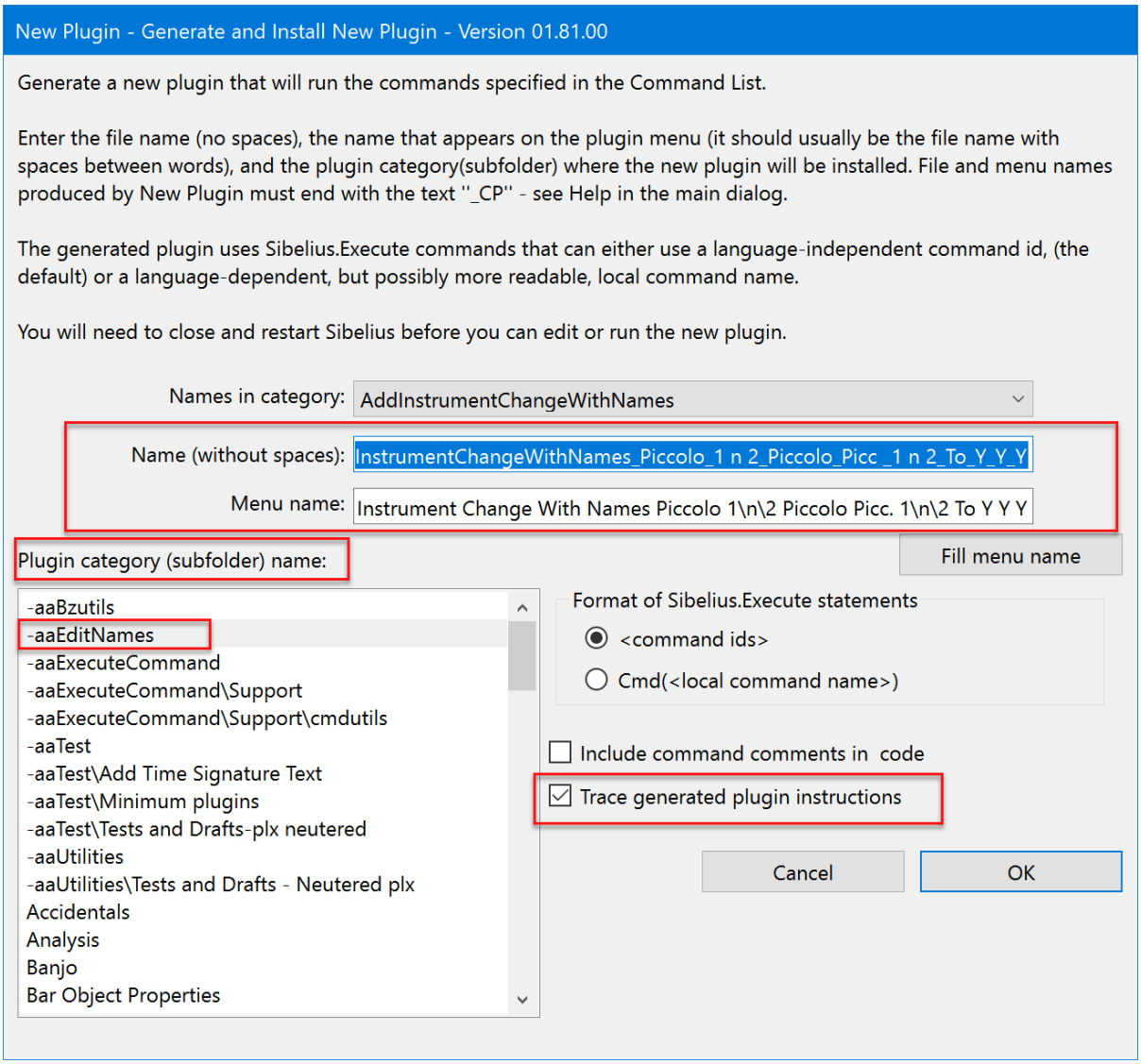


Take a deep breath and check it out…

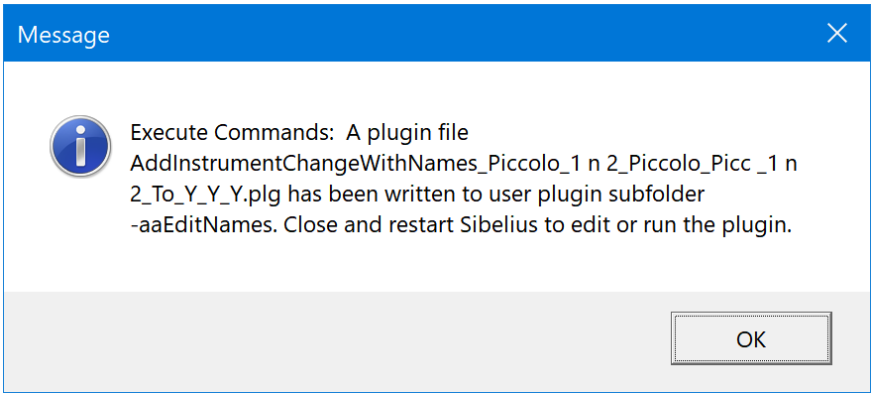
The **Current command line,** which I also call the **RunPluginEntry_cu command line,** is the text that was written to the trace window. It is a set of instructions that will reproduce the choices you made in the **Add Instrument Change With Names** dialog. It is somewhat "human readable", and one can match up the text to the options in the dialog. There is no real need to do that unless something goes wrong, which we hope will not happen.

This command line will be inserted into the custom plugin we are now generating, so when you run that plugin, this command will be executed.

At this point we want to leave the **Generate a new plugin…** checkbox checked, and uncheck the **Append the command line…** checkbox, then press OK. We will see another dialog. If you are familiar with the **Execute Commands** plugin, you may recognize this as the **New Plugin** dialog from that plugin. We are actually running **Execute Commands**, and if this is the first time it has run in this Sibelius session, it will take a few seconds to build its command lists, so please be patient.



This dialog is only presented so you can see **what the plugin will be named** (Name (without spaces) and Menu name), **and where it will be installed** (Plugin category)**.** You can change all of these things, and I will discuss naming later, but for now let's just accept everything on the dialog and press **OK**. You will see this final dialog:



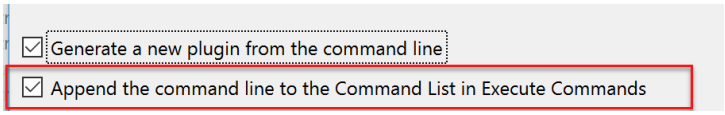
Execute Commands will write the text in this message box to the trace window so you can find it after the message box is closed.

The message tells you the name of the custom plugin (**AddInstrumentChangeWithNames_Piccolo_1 n 2_Piccolo_Picc _1 n 2_To_Y_Y_Y.plg**), and where is it installed (my plugin subfolder **-aaEditNames**). This is the same plugin folder where **Add Instrument Change With Names**, the parent of this new plugin, is installed.

Now, back in the plugin **Add Instrument Change With Names,** you can either cancel the dialog, or OK it if you want to create the **Instrument Change** right now. As the message box said, you will need to close and restart Sibelius before you will be able to access the generated plugin.

Append the command line to the Command List in Execute Commands

There is another check box in the dialog that was brought up by the **New macro/plugin** button, **Append the command line to the Command List in Execute Commands**.

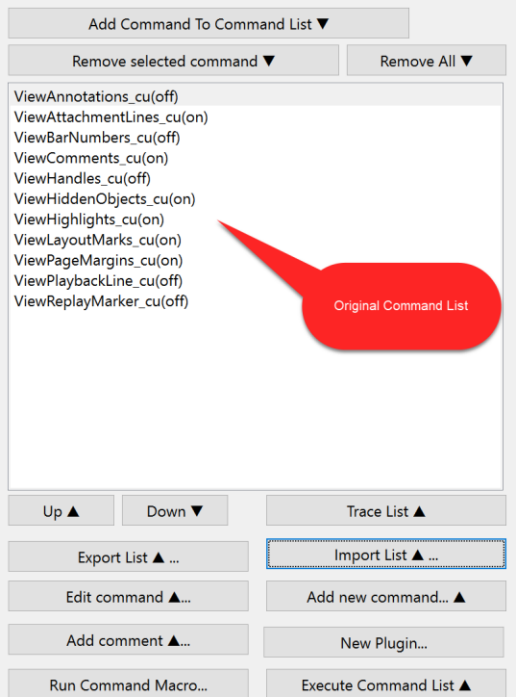


If you choose this option, the **RunPluginEntry_cu** command line will be added to the end of the Command List for the **Execute Commands** plugin and will be present the next time **Execute Commands** is run in this Sibelius session. (The Command List is cleared at the start of a Sibelius session, so you need to act fairly quickly).

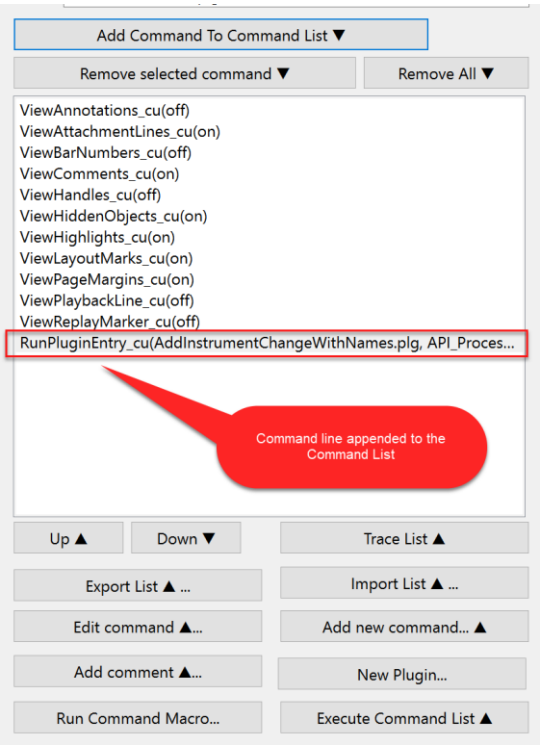
This can be useful if you want to combine the command line with other commands to make a larger macro or plugin.

If the Command List has commands in it that you don't want to include with your command line, you can run Execute Commands before you append a new command. Save any commands you want to keep using Export List, then either clear the entire list using the **Remove all** button or select specific commands and clear them with **Remove selected command.**

Let's say that the **Execute Commands** Command List looked like this the last time **Execute Commands** was run:



If you run **Execute Commands** after **Append the command line to the Command List in Execute Commands** was used, you would see this.



You can hover your mouse over the command line to see more of it, or press the **Edit Command...** button to see it in a larger edit box, or use **Export List...** to save all the commands as a macro (.dat) file. The .dat file is really a text file, so you can open it an any convenient text editor and examine the contents of the command line.

Export List... will turn the Command List into a macro you can run with the **Run Command Macro** plugin. You can also press **New Plugin...** and create a plugin that will run all the commands in the Command List.

What's with the long crazy names?

Names are important, and my goal is usually to make names that are unique and that also contain enough information to tell me something useful about the object being named. The fields in the **RunPluginEntry_cu** command line and in the Plugin file name and Menu name allow one to reconstruct the settings that were used when the command line or names were created.

The **RunPluginEntry_cu command line** starts with the name of the plugin to be called (**AddInstrumentChangeWithNames**) followed by the name of a routine, or entry point, in the plugin that is being called (**API_ProcessSelection**, in this case). These are followed by a set of comma-separated fields. There are 2 fields per setting, the first being a **name** for the control in the dialog, and the other expresses the **value** of that control.

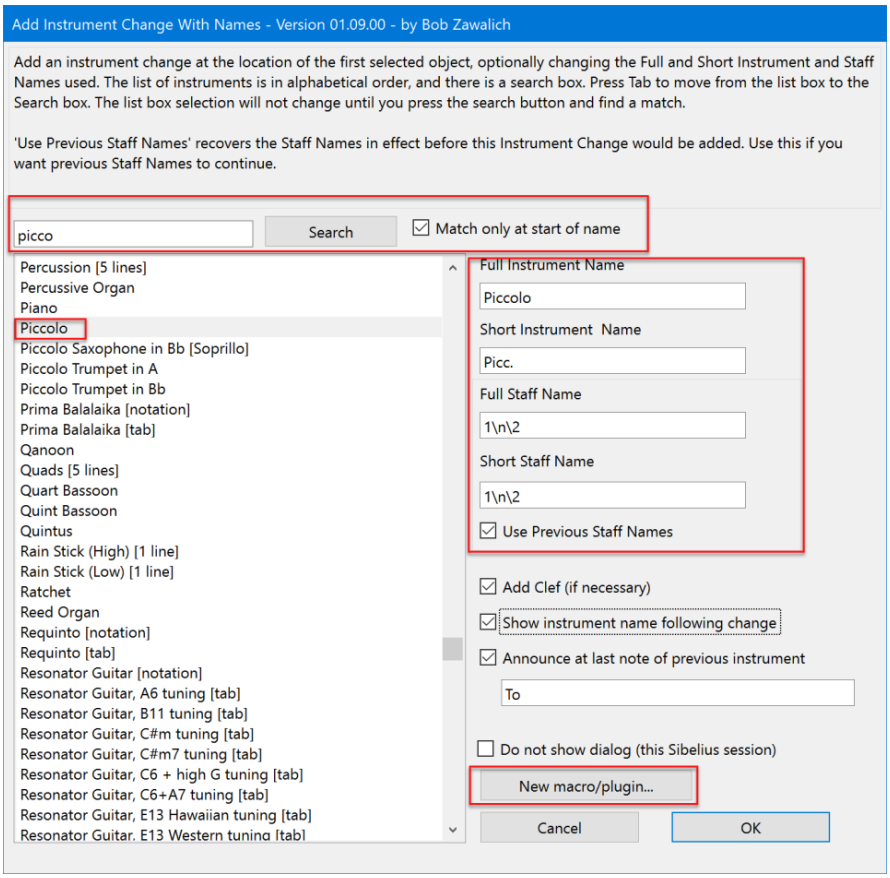
Both the plugin file name and the Menu name start with the parent plugin name (**AddInstrumentChangeWithNames)** and are followed by the **values** for each dialog control.

In the names below, the value fields from the dialog have been marked in red. Compare the red values fields in the command line with the corresponding fields in the file name. Note that the periods and backslashes have been stripped in the file name.

RunPluginEntry_cu(AddInstrumentChangeWithNames.plg, API_ProcessSelection, strFullInstrumentName, Piccolo, strFullStaffName, 1\n\2, strInstrumentName, Piccolo, strShortInstrumentName, Picc., strShortStaffName, 1\n\2, strWarningLabel, To, strfAddClef, yes, strfShowText, yes, strfShowWarning, yes)

The plugin name will be **AddInstrumentChangeWithNames_Piccolo_1 n 2_Piccolo_Picc _1 n 2_To_Y_Y_Y.plg**

The Menu name **will be Add Instrument Change With Names Piccolo 1\n\2 Piccolo Picc. 1\n\2 To Y Y Y**



Please note that the plugin with the **New macro/plugin** button **will automatically generate the command line** for you. All you need to do is choose your dialog options and press the button.

You don't need to know the following to use these names, but I will explain them anyway.

In the dialog, not all controls are captured in the command line. The **Search** box is an example of this. Of the controls that are captured, the list box of **Instrument Names** is named **strInstrumentName**, and its value is **Piccolo**, and both these fields are in the command line. Similarly, check boxes like **Add Clef (if necessary)** have a name and a value. For the command line, a checked box will have the value **yes**, and an unchecked box will display **no**. The **Add Clef** checkbox has the name **strAddClef** and its value is **yes**.

The names are chosen for my convenience as a programmer rather than yours, but each control will have a unique name that should be fairly easy to map to.

The plugin name and menu name are

AddInstrumentChangeWithNames_Piccolo_1 n 2_Piccolo_Picc _1 n 2_To_Y_Y_Y.plg

and

Add Instrument Change With Names Piccolo 1\n\2 Piccolo Picc. 1\n\2 To Y Y Y

The **menu name** is derived from the **file name** with spaces added for readability. The file name has been stripped of any special characters, like periods and backslashes, that might confuse the file system, and spaces are replaced with underscores. The values **yes** and **no** are replaced by **Y** and **N** to save a little space.

What if I hate the long names?

Using this naming scheme I can generate a unique file and menu name for each combination of values in a dialog, so when it comes time to generate the plugin you do not need to come up with a memorable and unique name. Similarly, having the plugins installed in the same folder as the parent plugin is easy for me to program, and to me seems to be a sensible location.

If you don't like either the name or the location, you can change them in the **New Plugin** dialog box that comes up from **Execute Commands.** Be aware that the file and menu names need to be unique. It is easy to create duplicate plugins, and these will not work the way you expect them to.

You can change the file name of an existing plugin in the file system, but that will not change the menu name, which appears in most of the places that run plugins.

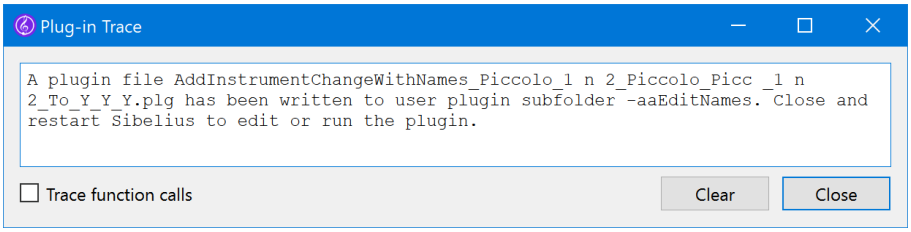
If you really want to change the names, I recommend running the **parent plugin** again with the same parameters and change the names when the plugin is being generated. If you really want to change the menu name of a generated plugin, use **File>Plug-ins>Edit Plugins** while running Sibelius. Look for the **Data** block and find **_PluginMenuName**.

Double click on it and you can edit the text. The changeable text will be in **double** quotes. Change just the text between the double quotes, being careful not to use any single or double quotes in your text, and be sure you retain both double quotes, or the plugin is likely to not work, and probably will not be loaded by Sibelius the next time it starts up.

Press **OK** to commit the edit of this variable, then then **OK** to commit to changing the plugin.

How to find and run your custom plugin

When you successfully create a plugin, you will see a message box, and the message text will also be written to the plugin trace window. So you might see:



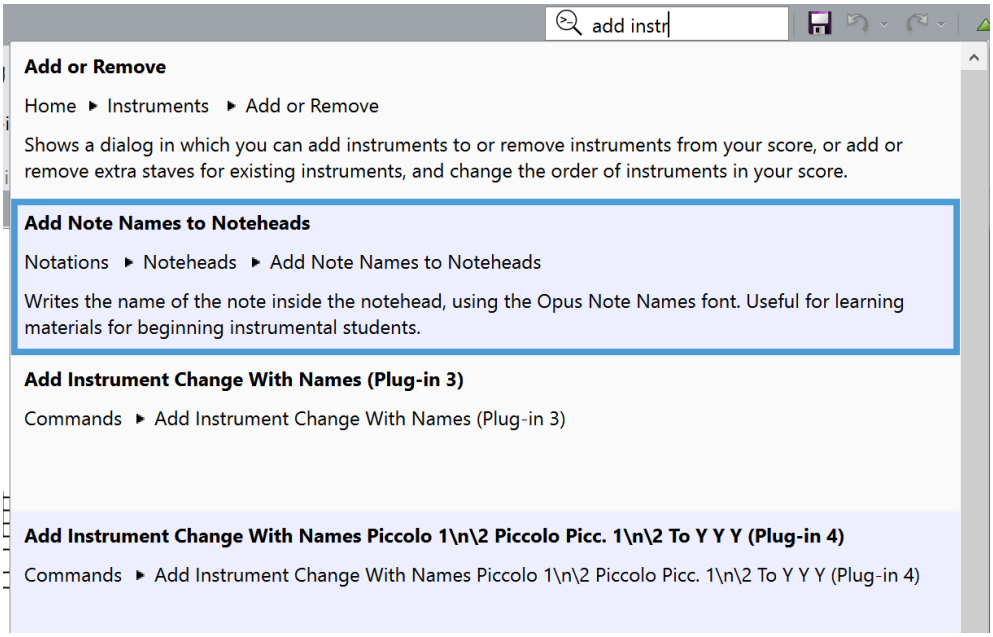
From this we can see that the file:

AddInstrumentChangeWithNames_Piccolo_1 n 2_Piccolo_Picc _1 n 2_To_Y_Y_Y.plg

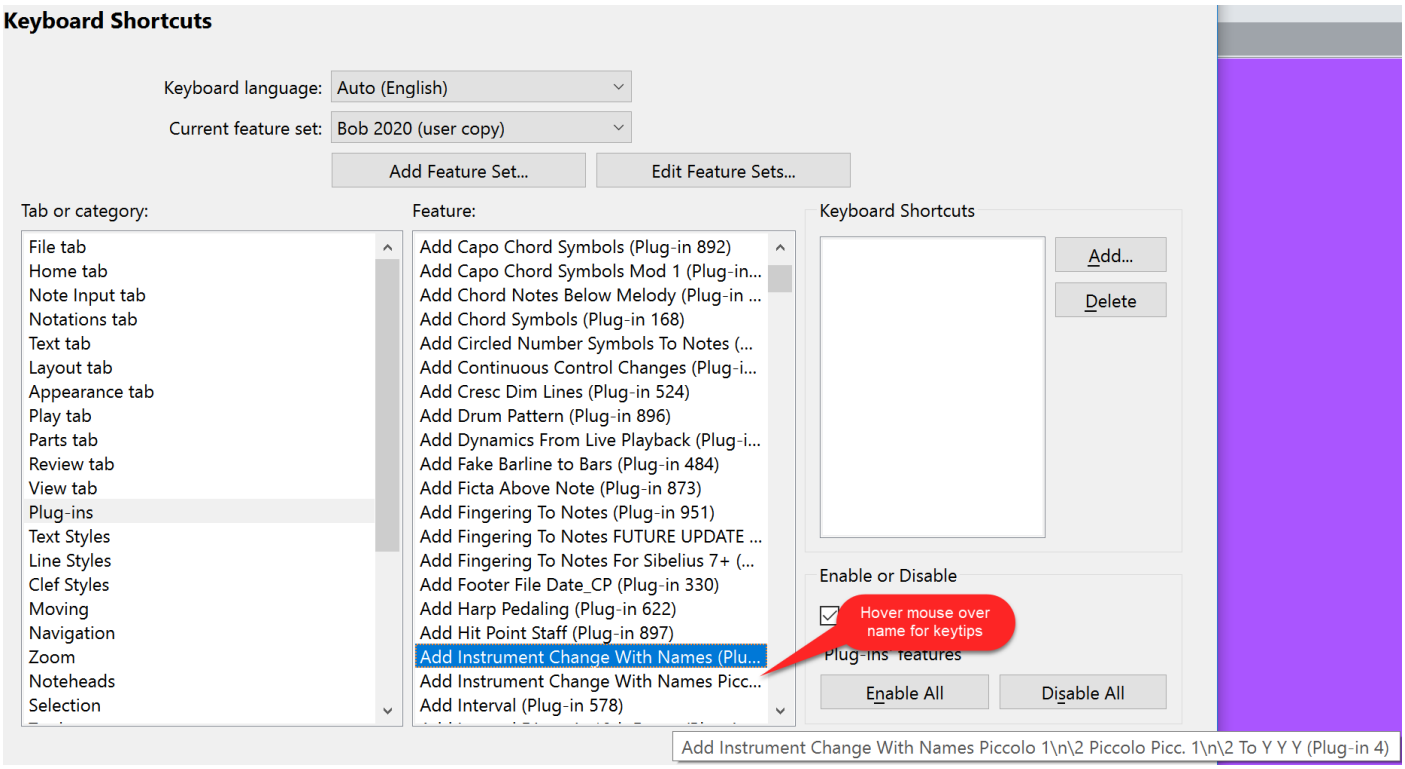
will be in the plugin subfolder/category **-aaEditNames.**

Admittedly these long names (both the plugin file name and its menu name) seem to be awkward to deal with, but in fact, most of the ways you can run a plugin handle the long names pretty well. Here are some examples.

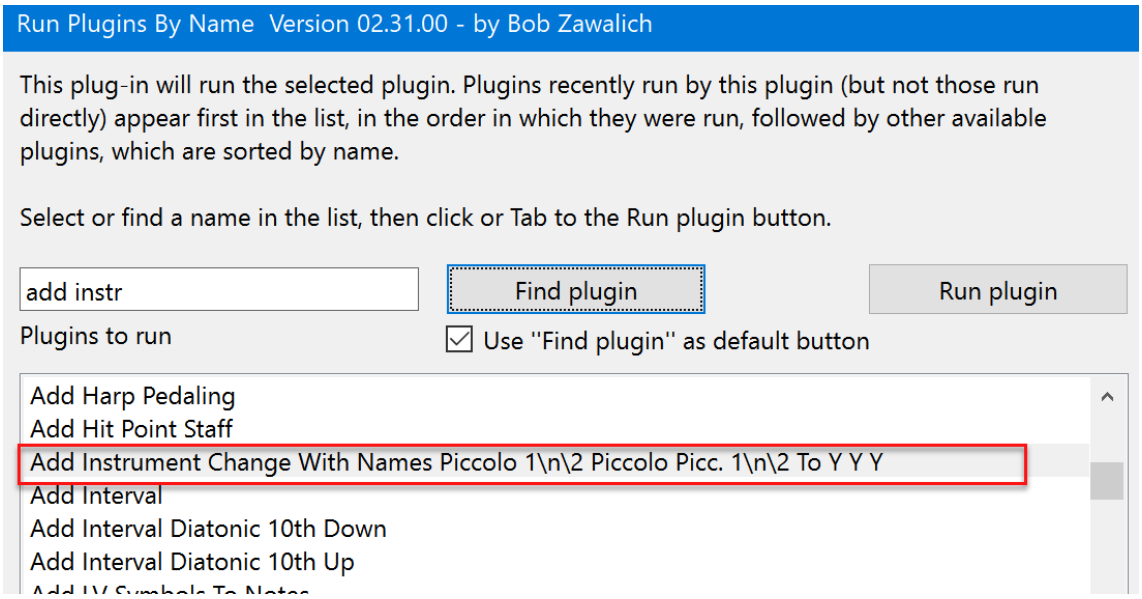
Command Search, which uses the menu name for its key, works well with these names, as shown below.



You could assign a shortcut to run your plugin in **File>Preferences>Keyboard Shortcuts**. The name will be cut off in the listbox, but if you hover the mouse over an entry, the full name will appear in a key tip:



You can run plugins with another plugin like **Run Plugins By Name**, which has room for long names.



Even plugin menus seem to be ok with long names.