

# Using Plugins with a “New Plugin” Button to Generate a Custom Macro or Plugin

*Bob Zawalich October 23, 2022 Updated March 2, 2024*

## Contents

Using Plugins with a “New Plugin” Button to Generate a Custom Macro or Plugin.....	1
Contents .....	1
Introduction .....	1
Example: <b>Add Instrument Change With Names</b> .....	2
Required Tools .....	2
The <b>Add Instrument Change With Names</b> dialog.....	2
Specifying the Instrument Change.....	3
Appending the command line to the Command List in Execute Commands.....	6
What’s with the long crazy names? .....	7
How a command line is constructed .....	7
Components of a File or Menu name .....	8
What if I hate the long names? .....	9
How to change a File or Menu Name .....	9
How to find and run your custom plugin.....	10

## Introduction

The document goes into detail about using the **New 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 *March 2, 2024* the published plugins that support this feature are:

- AddInstrumentChangeWithNames.plg
- AddTextToBlankPage.plg
- ChangeLivePlayback.plg
- ConvertLegacyChordSymbols.plg
- FilterNotesByBeat.plg
- FilterNotesByDuration.plg
- FilterNotesByPosition.plg
- FilterOther.plg
- FilterWithDeselect.plg
- FlipSelectedNotes.plg
- MinimumPluginParentRPE\_Enabled.plg
- MovePitchesToTransposedMidLine.plg
- MulticopyDynamic.plg
- ReplaceNoteheadStyle.plg
- StoreDataForWildcard.plg

## Example: Add Instrument Change With Names

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 **Instrument Names** and **Staff Names** along with the **Instrument Change**.

This document will explain how to use the **New 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

To enable the New Plugin... button in any of these dialogs you will need to install the current versions of these plugins. The minimum version number for these is shown. If using **File>Plugins>Install Plug-in**, try re-installing these. If you have the most recent version published, the installer will tell you so; if not it will install the most recent version.

- **ExecuteCommands.plg** (category **Developers' Tools**, version 01.81.44)
- **cmdutils.plg** (category **Developers' Tools**, version 01.49.60)
- **NewPluginLib.plg** (category **Developers' Tools**, version 01.20.50)

To follow along with this example you will also need to install the most recent version of

- **AddInstrument ChangeWithNames.plg** (category **Engravers' Tools**, version 01.09.50)

All these plugins, and any of the other “New Plugin” plugins can be installed using **File>Plug-ins>Install Plug-ins**.

## The Add Instrument Change With Names dialog

Add Instrument Change With Names - Version 01.09.40 - by Bob Zawulich

Add an instrument change at the location of the first selected object, optionally changing the Full and Short Instrument and Staff Names used. The list of instruments is in alphabetical order, and there is a search box. Press Tab to move from the list box to the Search box. The list box selection will not change until you press the search button and find a match.

'Use Previous Staff Names' recovers the Staff Names in effect before this Instrument Change would be added. Use this if you want previous Staff Names to continue.

picco Search ☒ Match only at start of name

Orchestral Bells  
Organ (Great)  
Organ (Swell)  
Organ [manuals]  
Oud [notation]  
Oud [tab]  
Pad 1 (New Age)  
Pad 2 (Warm)  
Pad 3 (Polysynth)  
Pad 4 (Choir)  
Pad 5 (Bowed)  
Pad 6 (Metallic)  
Pad 7 (Halo)  
Pad 8 (Sweep)  
Pandeiro [2 lines]  
Panpipes  
Patsch  
Ped. [Organ pedals]  
Pedal Steel Guitar [notation]  
Pedal Steel Guitar [tab]  
Percussion [1 line]  
Percussion [2 lines]  
Percussion [3 lines]  
Percussion [4 lines]  
Percussion [5 lines]  
Percussive Organ  
Piano  
Piccolo  
Piccolo Saxophone in Bb [Soprillo]

Full Instrument Name  
Piccolo  
Short Instrument Name  
Picc.  
Full Staff Name  
1\n2  
Short Staff Name  
1\n2  
☒ Use Previous Staff Names  
☒ Add Clef (if necessary)  
☒ Show instrument name following change  
☒ Announce at last note of previous instrument  
To  
  
☐ Do not show dialog (this Sibelius session)  
New Plugin...  
Cancel OK

You can choose an instrument and various properties and then insert an Instrument Change.

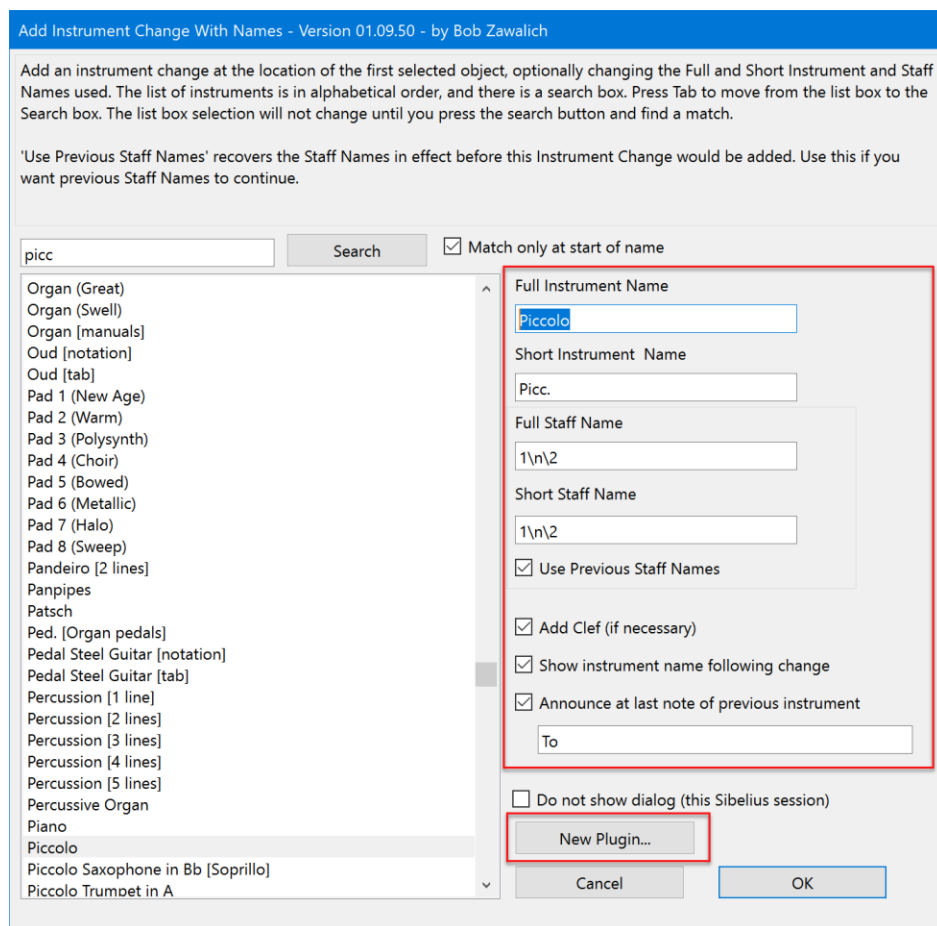
## Specifying the Instrument Change

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\n2”. 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 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. **New Plugin...** will only be enabled if the plugin **NewPluginLib.plg** has been properly installed.

Press **New Plugin...** and you will see some text being written to the plugin trace window (which you can ignore), and this dialog box:

New Plugin / Append to Command List

You can generate a custom plugin that runs the command line that has just been traced.

You can also add the command line to the Command List used by the plugin Execute Commands. The added command line will be at the bottom of the Command List the next time Execute Commands is run.

If you generate a plugin you will be given a chance to review and edit the new plugin location and name. You can just press OK/Enter in that dialog to accept the default name and location.

If you generate a plugin you will need to close and restart Sibelius before you can use it.

Current command line:

```
RunPluginEntry_cu(AddInstrumentChangeWithNames.plg, API_ProcessSelection, AddClef, yes, FullIName, Piccolo, FullSName, 1\n2, InstName, Piccolo, ShortIName, Picc., ShortSName, 1\n2, ShowText, yes, ShowWarning, yes, WarningLabel, To)
```

☒ Generate a new plugin from the command line

☐ Append the command line to the Command List in Execute Commands

Cancel OK

So what does this all mean?

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.

Right now 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 now 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.

New Plugin - Generate and Install New Plugin - Version 01.81.44

Generate a new plugin that will run the commands specified in the Command List.

Enter the plugin file name (no spaces), the menu name (normally the file name with spaces between words), and the plugin category(subfolder) where the new plugin will be installed. File and menu names produced by New Plugin must have, or will be given, the suffix "\_CP" - see Help in the main dialog.

The generated plugin uses Sibelius.Execute commands that can either use a language-independent command id, (the default) or a language-dependent, but possibly more readable, local command name.

You will need to close and restart Sibelius before you can edit or run the new plugin.

Name (without spaces):  (LCP) Menu name from file name

Menu name:  (LCP) File name from menu name

Names in category:

Plugin category (subfolder) name:

- Capo\_Notation
- Chord Symbols
- Clipboards
- Close\_Open
- Color
- Composing Tools**
- Delete
- Developers'\_Tools
- Drafts\_neutered\Tests and Drafts-plx neutered
- Engravers\_Tools
- Enumerators
- Execute\_Commands
- Execute\_Commands\Publish Macros
- Export
- Filter and Find

Format of Sibelius.Execute statements

☒ <command ids>

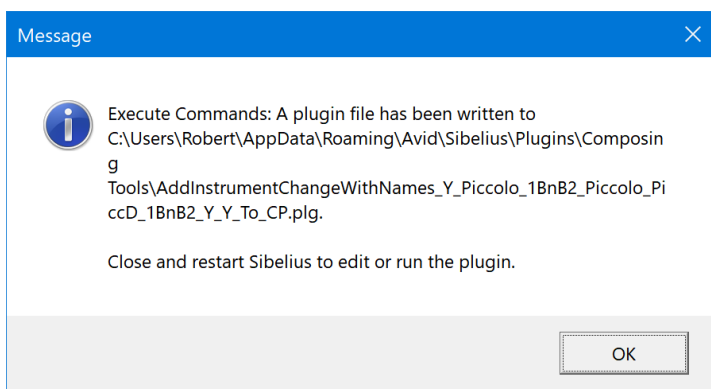
☐ Cmd(<local command name>)

☐ Include command comments in code

☐ Trace generated plugin instructions

Cancel OK

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 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:



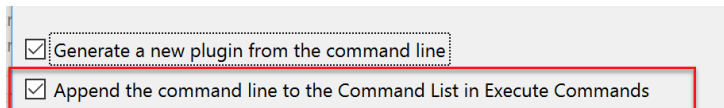
**Execute Commands** will also write the text in this message box to the trace window so you can find it after the message box is closed.

The message tells you the name of the custom plugin (**AddInstrumentChangeWithNames\_Y\_Piccolo\_1BnB2\_Piccolo\_PiccD\_1BnB2\_Y\_Y\_To\_CP.plg**), and where it is installed (my plugin subfolder **Composing Tools**). This is the same plugin folder where **Add Instrument Change With Names**, the **parent** of this new plugin, is installed.

If you go back to 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.

## Appending 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 Plugin...** button, **Append the command line to the Command List in Execute Commands**.

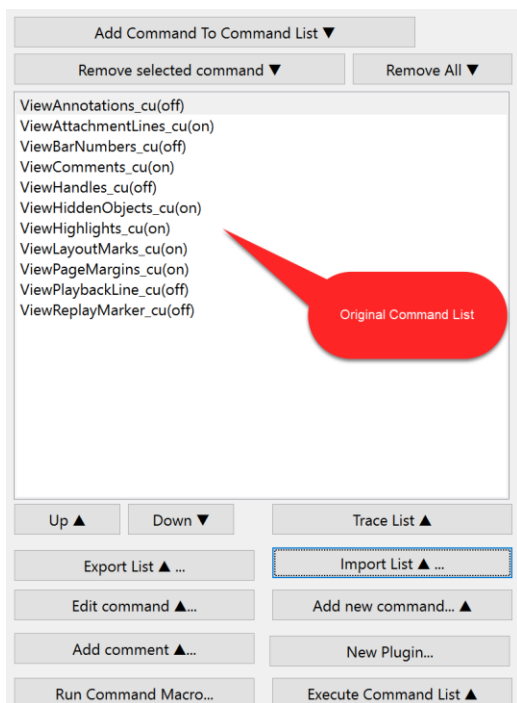


If you choose this option, the **RunPluginEntry\_cu** command line will be added to the end of the Command List for the **Execute Commands** plugin and will be present the next time **Execute Commands** is run in this Sibelius session. (The Command List is cleared at the start of the next Sibelius session,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

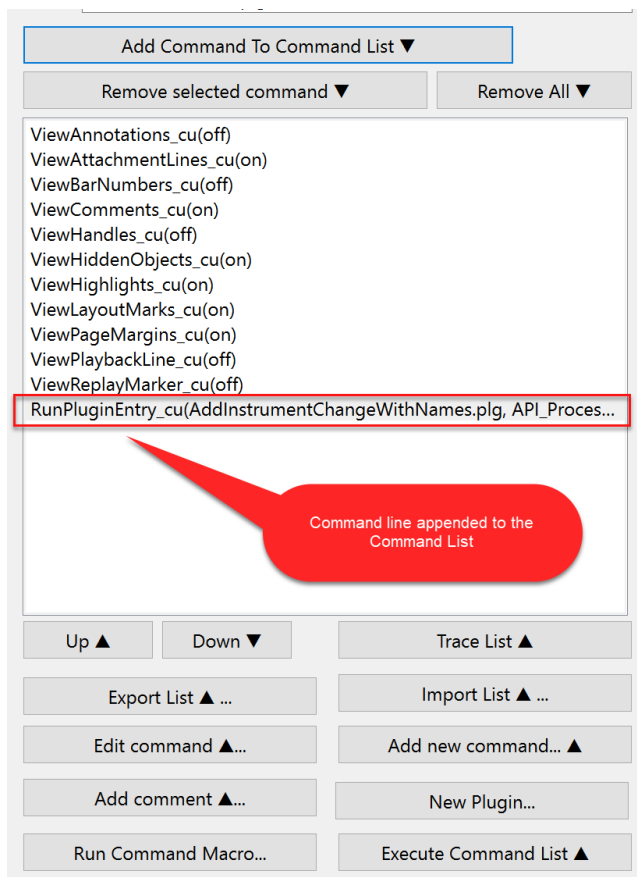
This can be useful if you want to combine the command line with other commands to make a larger macro or plugin.

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 in 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 something useful about the plugin being named.

Bear in mind that these plugins will not bring up a dialog when you run them, so if you want to create a plugin and use it in the future, **it is useful to choose a name that can give you some idea of what it will do.** Otherwise, you may have to just start over, or use the Plugin Editor to look inside the plugin to try to figure out what it does.

The fields in the **RunPluginEntry\_cu** command line and in the suggested **Plugin file name** and **Menu name** allow one to reconstruct the settings that were used when the command line or names were created.

## How a command line is constructed

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.

Finally, all such names end with **\_CP** (for **Command Plugin**). This identifies a generated plugin and helps prevent choosing a name that will cause a plugin to call itself in an endless loop in an endless loop in an endless loop....

Both the plugin file name and the Menu name start with the parent plugin name (**AddInstrumentChangeWithNames**), but not the entry name, and are followed by a representation of the **values**, but not the **names**, for each dialog control.

In the names below, in the command line, 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. Periods and backslashes and most other special characters have been stripped or replaced 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** (file) name will be

**AddInstrumentChangeWithNames\_Y\_Piccolo\_1BnB2\_Piccolo\_PiccD\_1 BnB2\_Y\_Y\_To\_CP.plg**

The **Menu** name will be

**AddInstrumentChangeWithNames Y Piccolo 1BnB2 Piccolo PiccD 1 BnB2 Y Y To CP**

The plugin with the **New Plugin...** button **will automatically generate the RunPluginEntry\_cu command line** for you. All you need to do is choose your dialog options and press the button.

### Components of a File or Menu name

You don't need to know the following to use these command lines, 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 recognize when the time comes to find and run the generated plugin.

The plugin name and menu name are

**AddInstrumentChangeWithNames\_Y\_Piccolo\_1BnB2\_Piccolo\_PiccD\_1 BnB2\_Y\_Y\_To\_CP.plg**

and

**AddInstrumentChangeWithNames Y Piccolo 1BnB2 Piccolo PiccD 1 BnB2 Y Y To CP**

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



little space. Some additional replacement of characters are made so that the file name can be used by a plugin to call this plugins.

In the file name, the original characters **1\n\2** and converted to **1BnB2**; the backslashes were replaced by **B**. **Picc**. Was replaced by **PiccD**. You may find many other examples of characters that were replaced for practical reasons that leave the string unintelligible.

## What if I hate the long names?

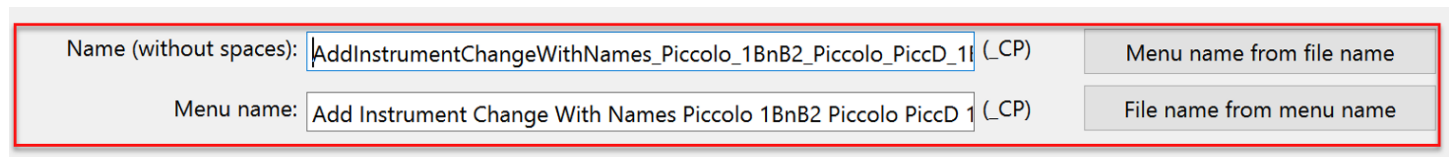
Feel free to choose a totally different naming scheme, or location.

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.

These plugins will not bring up a dialog when you run them, so if you want to create a plugin and use it in the future, **it is useful to choose a name that can give you some idea of what it will do**. Otherwise, you may have to just start over, or use the Plugin Editor to look inside the plugin to try to figure out what it does.

## How to change a File or Menu Name

You can change either of the names in the **New Plugin - Generate and Install New Plugin** dialog:



Put the cursor into one of the edit boxes and type **ctrl/cmd+a** to select the entire name.

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. The file and menu names should be related, so you can change only one of the names, and use the appropriate “**name from**” button to generate a reasonable choice for the other name.

These plugins will not bring up a dialog, so if you want to create a plugin and use it in the future, **it is useful to choose a name that can give you some idea of what it will do**. Otherwise, you may have to just start over, or use the Plugin Editor to look inside the plugin to try to figure out what it does.

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, and which must also be changed.

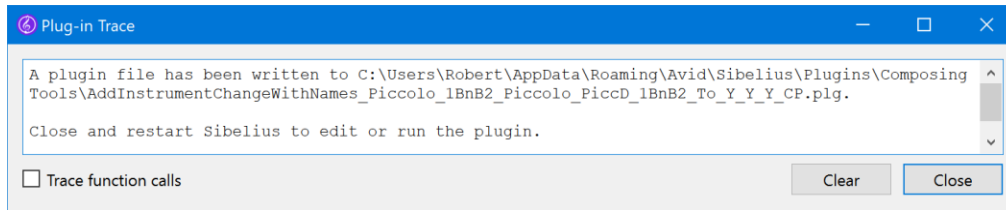
If you really want to change the names, I recommend running the **parent plugin** again with the same parameters and change the names when the plugin is being generated. If you really want to change the menu name of a **generated** plugin, use **File>Plug-ins>Edit Plugins** while running Sibelius. Edit the plugin you want to change and 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. You might see:



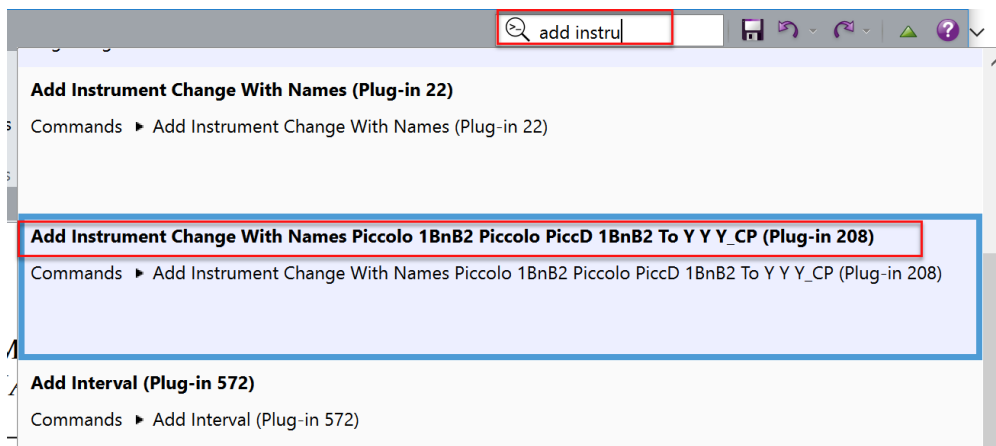
From this we can see that the file:

**AddInstrumentChangeWithNames\_Piccolo\_1BnB2\_Piccolo\_PiccD\_1BnB2\_To\_Y\_Y\_Y\_CP.plg**

will be in the plugin subfolder/category **Composing Tools**.

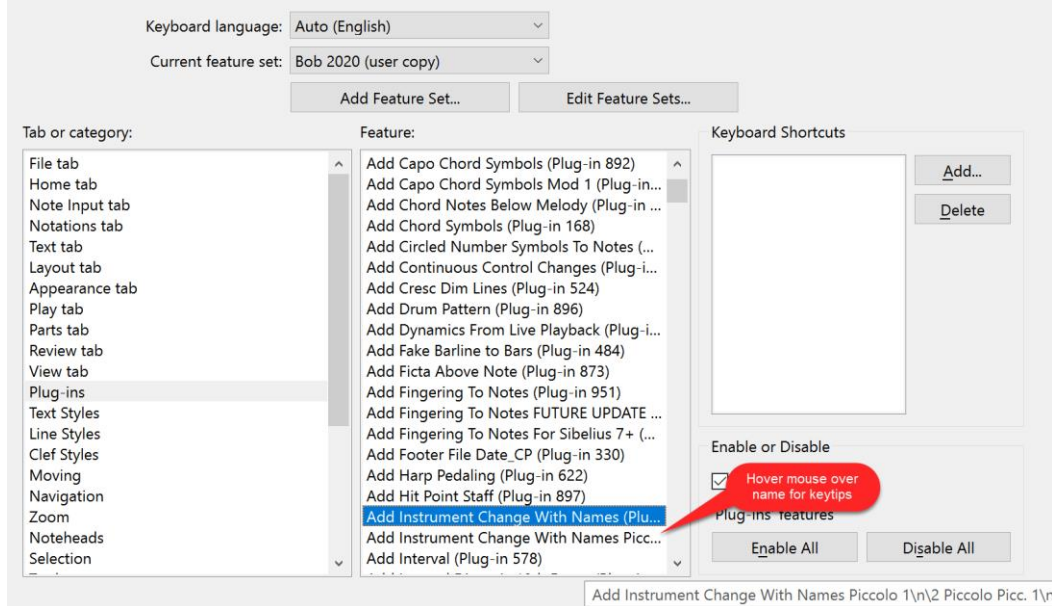
Admittedly these long names (both the plugin file name and its menu name) seem to be awkward to deal with, but in fact, most of the ways you can run a plugin will handle the long names pretty well. Here are some examples.

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

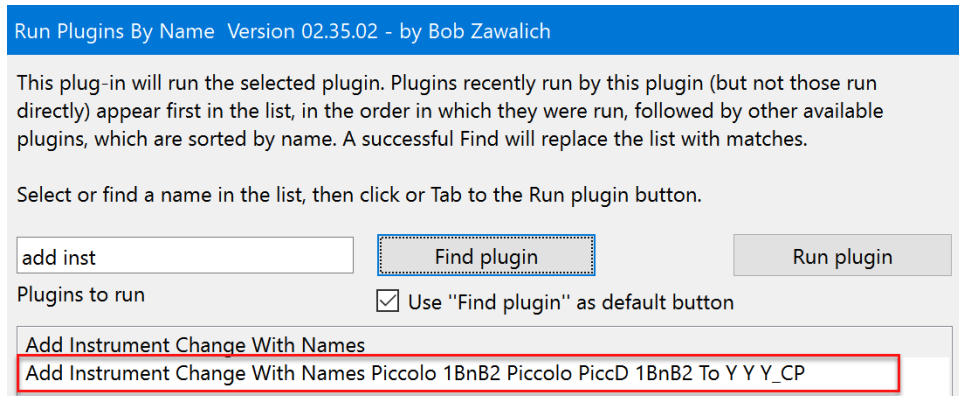


You could assign a shortcut to run your plugin in **File>Preferences>Keyboard Shortcuts**. The name will be cut off in the listbox, but if you hover the mouse over an entry, the full name will appear in a key tip:

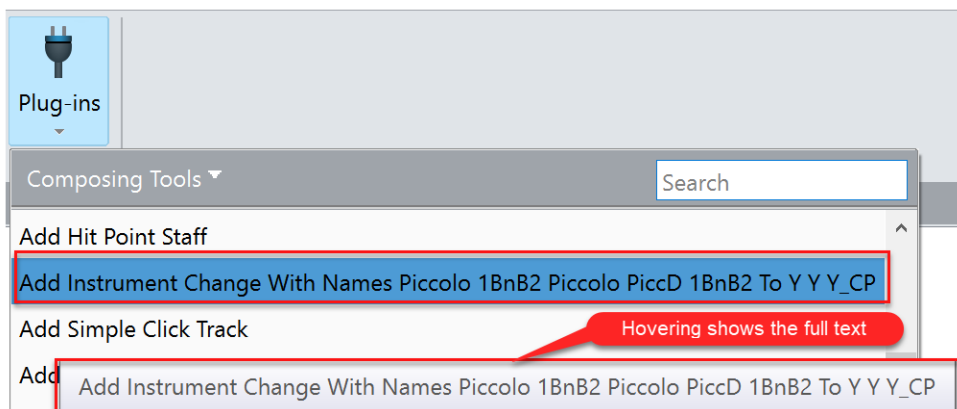
## Keyboard Shortcuts



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.



You may never need all this information, but if something goes wrong, you may find it to be helpful. May nothing ever go wrong!