

Plugins Designed to be Run by Other Plugins

The RunPluginEntry Edition

Bob Zawalich October 14, 2022

TLDR Summary

If you run a plugin that has a **New Macro/Plugin** button, pressing it will generate a “command line” that will allow you to later run that plugin with the current dialog setting but without showing the dialog.

The command line needs to be turned into a macro or plugin using the plugin **Execute Commands**. See [an example here](#).

Overview

Sibelius plugins have traditionally been designed to be run by humans. You or I can choose options from plugin dialogs and make other decisions.

In some cases, such as batch plugins, one would prefer not to have to check and OK a dialog for every file processed. Some plugins were given a “**Do not show dialog**” checkbox that would show the dialog the first time the plugin was run in a Sibelius session, and then you could turn the dialog off. Every time the plugin was run again in the same session it would run without bringing up the dialog, using the settings you had chosen when it was first run.

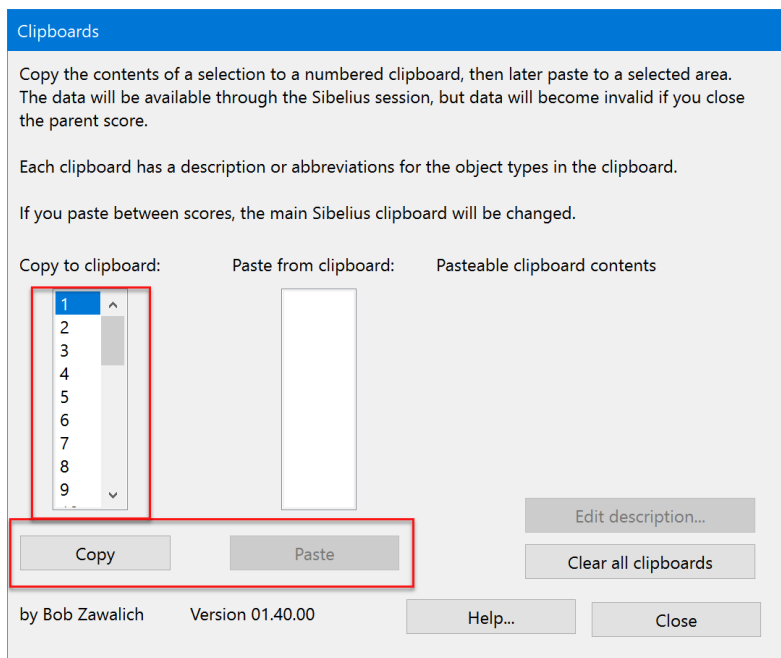
When it became possible for plugins to run Sibelius commands (in Sibelius Ultimate 2021.9), the plugin **Execute Commands** made it possible to set up a macro that would run a number of commands, including plugins, in sequence. In this context it is desirable to be able to run a plugin with a specific set of plugin settings every time, without ever needing any user interaction.

Imagine a case where you create a macro that runs plugins and then you give it to someone else to use. If they had to enter the correct settings in the dialogs of each plugin in the macro, it would be both error prone and difficult for you to explain.

I tried various way to make this work better. One way to do this was to set up a full-featured plugin with a dialog, which I referred to as a **Parent** plugin, and then write a number of small **Child** plugins that do nothing except call the **Parent** with a specific set of dialog settings.

An example of this is the **Clipboards** plugin, whose dialog is shown below.

To run it using the dialog you need to choose one of 30 clipboards and say whether you want to copy or paste. It became clear that one would be likely to want to set up a number of clipboards and then copy to them and use that data to paste into scores. I wrote a number of Child plugins which could each run one combination of clipboard number and either Copy or Paste, bypassing the dialog. I provided Copy and Paste children for the first 5 clipboards, which were installed along with the parent plugin.



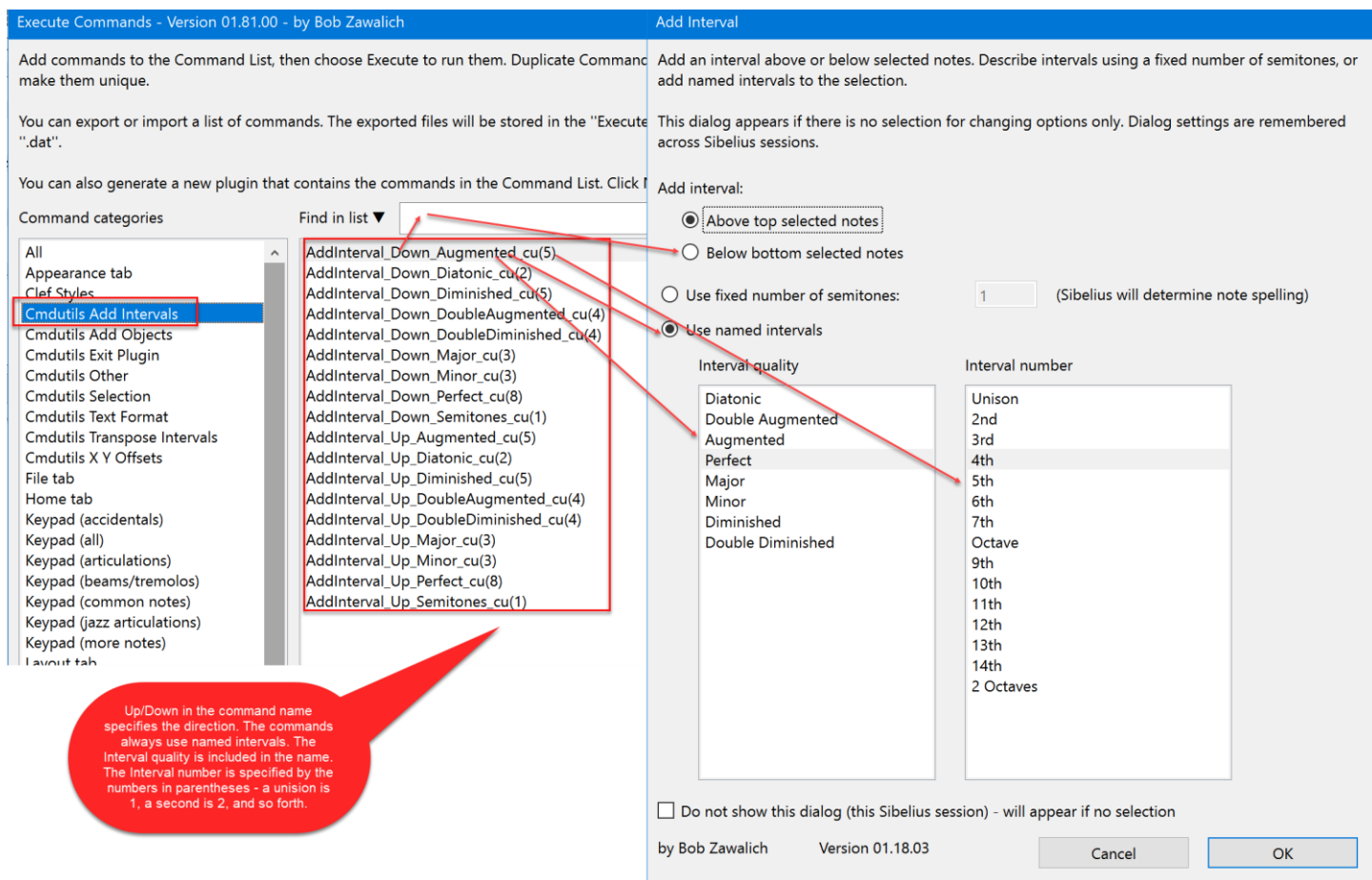
Clip Copy 1
Clip Copy 2
Clip Copy 3
Clip Copy 4
Clip Copy 5
Clip Paste 1
Clip Paste 2
Clip Paste 3
Clip Paste 4
Clip Paste 5
Clipboards

This works ok, but you can only use the provided children for a small set of clipboards. I gave instructions for making clones of the children and changing the clipboard number. It was not really satisfactory, but it would not be practical to ship 60 child plugins to cover all the combinations. And many plugins have many more combinations of dialog options than 60.

Later I created a large library plugin, **cmdutils**, containing extended commands, which could be called from **Execute Commands** or other plugins. Most of the code for the commands was completely contained within **cmdutils**, but a few times I found existing plugins that could do what I wanted, and so I adapted those plugins so that **cmdutils** could call them without a dialog.

For example, I wanted to make a set of commands to add a larger set of intervals than Sibelius provides, and I realized I had all the code I needed in the existing plugin **Add Interval**. I restructured that plugin so it could either be run by a human with its dialog, or it could be called by other plugins who passed in the same kind of data that was provided by the dialog.

I ended up giving **cmdutils** a set of **AddInterval** commands, each of which could duplicate any combination of dialog settings in **Add Intervals**. Most of the options were specified by the command names, but the commands also took a **parameter**, the Interval number, so each name could have 15 variations.

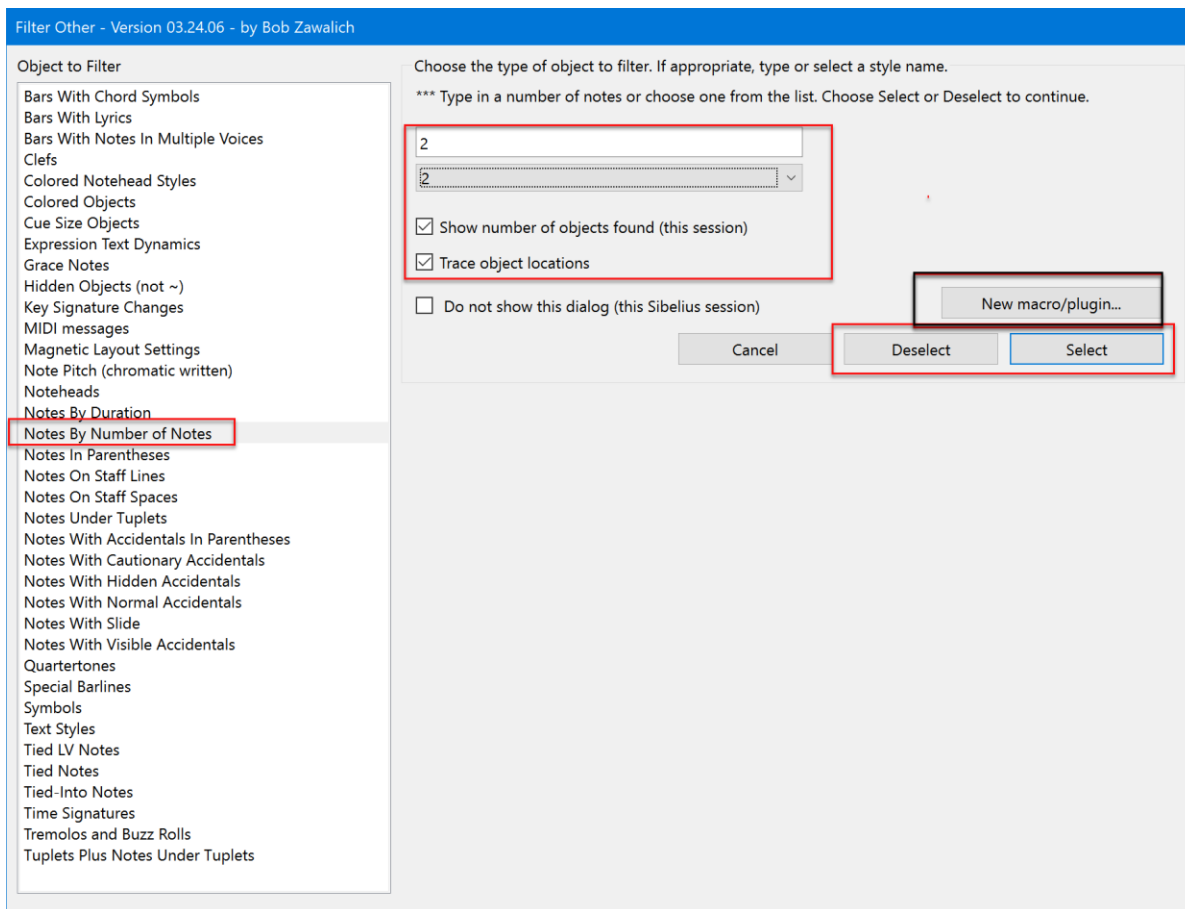


Since the **AddInterval** commands were all contained within **cmdutils**, I did not need to provide 18 separate child plugins, but I did need to add all those commands to **cmdutils**, which was still a fair bit of work.

Eventually, I realized that if I expanded the parameter list of a command to include all the settings it would be possible to create a single command in **cmdutils** that could call **any** specially-designed plugin. The command could pass in settings that **matched every control in the dialog** of the called Parent plugin. I wrote the universal child plugin called **RunPluginEntry_cu**.

This command has a list of parameters that specify which Parent plugin is to be called, which routine within the plugin is to be called, and which combination of dialog settings should be used.

Let's look at an example. The most recent version of the plugin **Filter Other** was adapted to support **RunPluginEntry_cu**. It has a dialog like this:



In a macro, we want to be able to run a plugin with the same dialog settings every time, without ever seeing the dialog. For **Filter Other**, we would want to capture the settings of the controls marked in red.

Plugins that support **RunPluginEntry_cu** are required to generate appropriate parameters for the command, and all the one I have written so far have a **New Macro/Plugin** button to generate and trace a command line that enables **RunPluginEntry_cu** to run a parent plugin like **Filter Other** with the specified dialog options.

If you press the **New macro/plugin** button, the plugin will write a line of text to the Plugin Trace Window. With the options chosen above for **Filter Other**, the command line looks like this.

```
RunPluginEntry_cu(FilterOther.plg, API_ProcessObjects, str_Action, select, str_FilterType, Notes By Number of Notes, str_StyleName, 2, str_fShowResults, yes, str_fTraceLocation, yes)
```

This looks pretty intimidating, but you really don't need to do anything with it except to copy the generated text and get in into the **Execute Commands** plugin.

Without getting into too much detail, the command line tells **RunPluginEntry_cu** the **file name of the plugin** to run (FilterOther.plg, the **routine, or entry point, within the plugin to run** (API_ProcessObjects), and the **values of each of the controls** we care about. The command line has 2 entries for each control in the dialog, representing the name of the control, and the value of the control.

In this example, the **list box** control is called **str_FilterType** and its current value is **Notes By Number of Notes**, so the command line has 2 entries for that control. **Filter Other**, the "parent plugin", determines which controls to put in the command line, and what their names should be, and this is true for all such Parent plugins, so you do not have to figure that out.

The details of the command line don't matter to you. What you need to do is to set up the options that you want in the dialog, and then press the **New macro/plugin**.

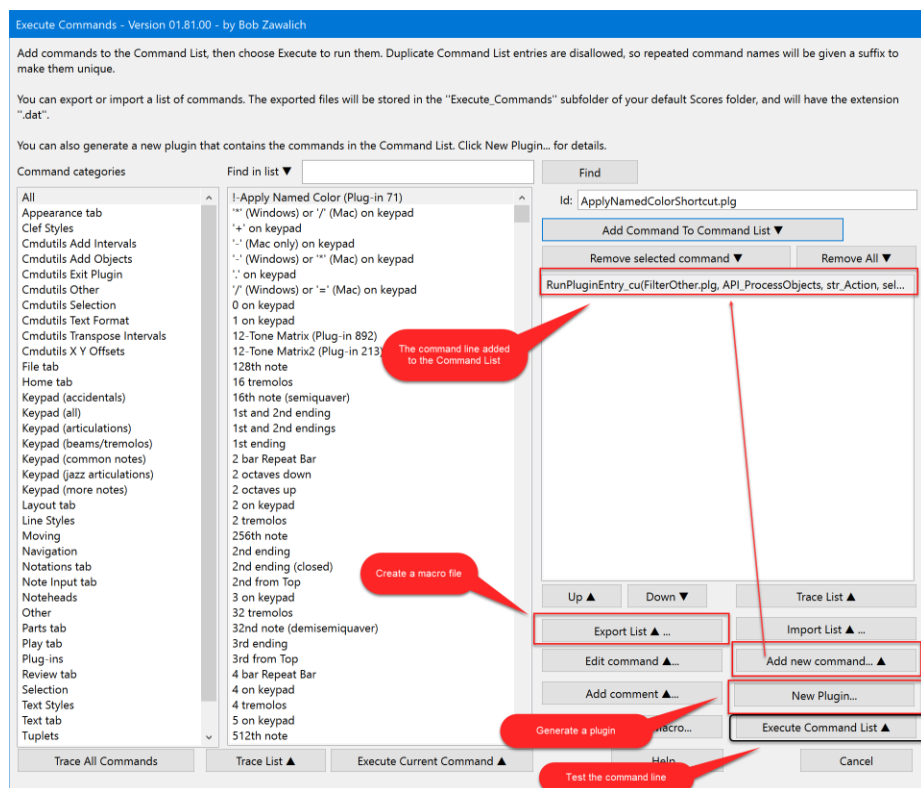
What do you do with the command line?

RunPluginEntry_cu command lines were designed to be used in the plugin **Execute Commands**. The command line could be added into the **Execute Commands command list** by copying the text in the trace window, running **Execute Commands**, pressing the button **Add New Command**, pasting the command line into its dialog, and then pressing OK. At this point the command line can be used like any other command. It can be saved as a macro or plugin or incorporated into a set of commands to make up a larger macro or plugin.

This seems like a lot of work, but if you are building a macro that you will use over and over, it will take you less than a minute to create the command line and incorporate it into **Execute Commands**, and then it will be something you can use forever.

Here is what **Execute Commands** would look like after adding this command line. You can quickly test what it does by pressing the **Execute Command List** button. If it does what you want, use **Export List** to save the command as a macro, or **New Plugin** to generate a plugin that will run this command.

The command line is typically too long to be fully shown in the listbox. Hover your mouse near the line and you should see the full line, or you can press **Edit Command** to see the line in a longer edit box. But in general, you really don't care what it says.



How is this useful?

If you are not writing macros or plugins you will not care about this. You probably stopped reading this a long time ago, but if not, feel free to stop now.

If you are writing macros or plugins and you want to run another plugin without seeing the dialog, you can use this mechanism to specify any combination of dialog controls you want. **Filter Other** has about 40 entries in the list box, and many of those entries have a list of values in the smaller **Style List**.

RunPluginEntry_cu can run **any** plugin that is structured according to its specifications, so new Parent plugins can be written at any time, and you can use **RunPluginEntry_cu** command lines to specify exactly which plugin to run, and which dialog settings you want from any supported plugin.

Technical specifications for parent plugins

ManuScript developers can find [documentation on how to structure a Parent plugin here](#). There are also a number of [documents on the Execute Command plugin family here](#). The document [The cmdutils library in Execute Commands](#) should be especially useful.

The plugin **Minimum Plugin Parent RPE_Enabled** is a freely modifiable template plugin you can install and copy for a template.

Simplifying the use of a command line

While it was definitely possible to [paste a command line into Execute Commands and from there to use that command to create a macro or plugin](#), I wanted to find a way to make it simpler to create a plugin that would run just the generated command, and an easier way to get the command line into the command list in **Execute Commands**.

I added code to the plugin **Custom Search Shortcut** that other plugins could use. If **Custom Search Shortcut** is installed, any plugin that has a **New Macro/Plugin** button will be able to generate a plugin that runs this command line directly, or to insert the command line directly into **Execute Commands**, so that it will be in the command list the next time **Execute Commands** is run.

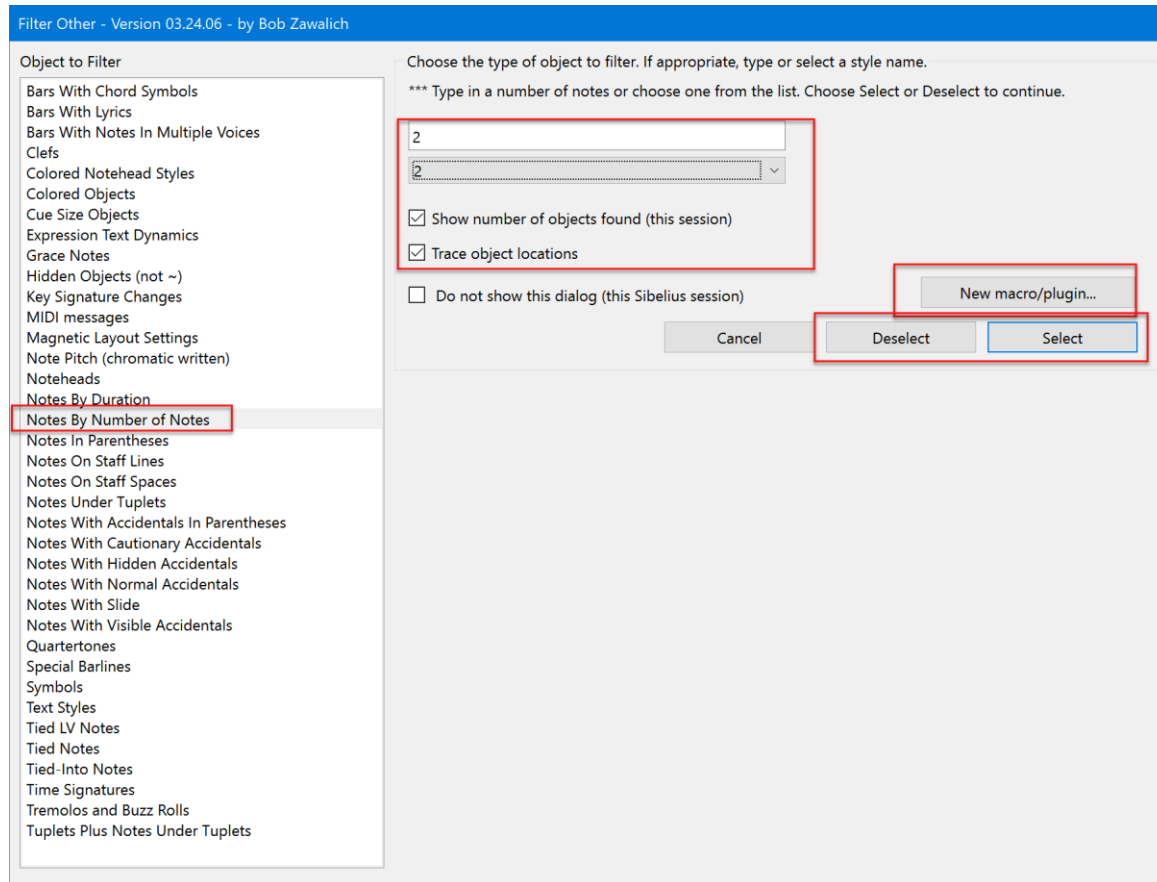
Currently the plugins **Filter Other**, **Move Transposed Notes To Mid Line**, **Flip Selected Notes**, and **Add Instrument Change With Names** support this facility, and several plugins that have a **Trace Macro** button will be updated in the near future.

An Example

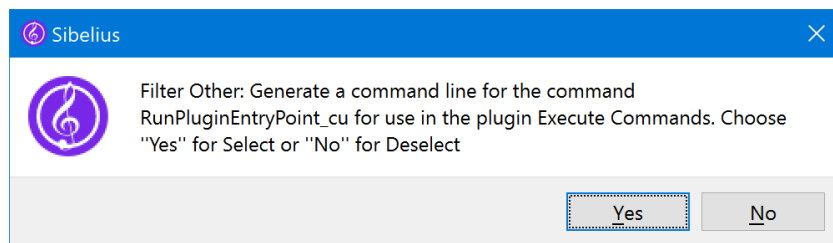
Here is the **Filter Other** plugin that traces the command line when the plugin **Custom Search Shortcut** is installed:

```
RunPluginEntry_cu(FilterOther.plg, API_ProcessObjects, str_Action, select, str_FilterType, Notes By Number of Notes, str_StyleName, 2, str_fShowResults, yes, str_fTraceLocation, yes)
```

to the Plugin Trace Window:



When you have chosen the dialog options you want and press the **New macro/plugin** button in **Filter Other**, you will see a dialog like this, which needs to determine if you wanted to **Select** or **Deselect** the filtered material, since you would not have pressed one of those buttons yet.



Once you choose, the command line is immediately traced to the Plugin Trace Window, and it can be manually pasted into **Execute Commands** as described earlier.

If the plugin **Custom Search Shortcut** is installed, some more things will happen.

You will see a dialog appear that will show you the current command line, and let you choose to generate a plugin that will run this plugin, and/or to make the command line appear in the command list of **Execute Commands** the next time that plugin is run.

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(FilterOther.plg, API_ProcessObjects, str_Action, select, str_FilterType, Notes By Number of Notes, str_StyleName, 1, str_fShowResults, yes, str_fTraceLocation, yes)

☒ Generate a new plugin from the command line

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

Cancel OK

If you choose to generate a plugin, a new plugin name and menu name are generated from the Parent plugin name (**Filter Other** in this case) plus some text that represents the values passed to the command line. The plugin will be installed into the same folder into which the parent plugin is installed. You will then see the **New Plugin** dialog from **Execute Commands**, containing the names and subfolder/category.

New Plugin - Generate and Install New Plugin - Version 01.81.00

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

Enter the file name (no spaces), the name that appears on the plugin menu (it should usually be 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 end with the text "_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.

Names in category: CustomFilter

Name (without spaces): FilterOther_select_Notes_By_Number_of_Notes_2_Y_Y

Menu name: Filter Other select Notes_By_Number_of_Notes 2 Y Y

Plugin category (subfolder) name:

- Color
- Composing Tools
- Delete
- Developers' Tools
- Downloads New
- Engravers' Tools
- Enumerators
- Execute_Commands
- Execute_Commands\Export
- Execute_Commands\Publish Macros
- Filter and Find**
- Fingering for Instruments
- Focus On Staves
- Focus On Staves\FocusSet
- Graphical MIDI Tools

Format of Sibelius.Execute statements

☒ <command ids>

☐ Cmd(<local command name>)

☐ Include command comments in code

☒ Trace generated plugin instructions

Cancel OK

The names are unwieldy but descriptive and should be unique. If you will only use this plugin in other macros or plugins you can leave the unwieldy names, otherwise you can change them to something more wieldy.

Hint: fill in **Name (without spaces)**, and then press **Fill menu name**, which will copy the file name to the menu name edit box, separating words with spaces. You can also change the category of the generated plugin.

After you create the plugin, you will need to close and restart Sibelius before you can access it.

If you chose **Append the command line to the Command List in Execute Commands**, then the next time you run **Execute Commands** in the current Sibelius session, you will find the command line at the end of the command list.

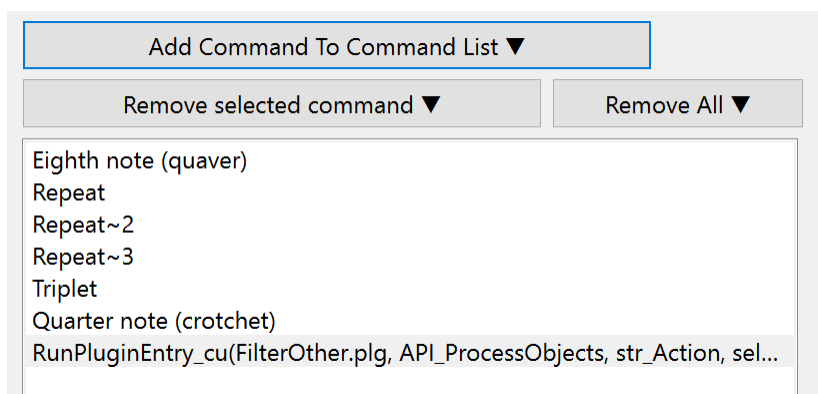
If the command list in Execute Commands had looked like this:

Add Command To Command List ▼

Remove selected command ▼ Remove All ▼

- Eighth note (quaver)
- Repeat
- Repeat~2
- Repeat~3
- Triplet
- Quarter note (crotchet)

and then you use **Append the command line to the Command List in Execute Commands**, then run **Execute Commands** again, you will see:



with the command line at the bottom of the command list, ready to be part of a macro or plugin.

Plugins to Install

In addition to any Parent plugin you want to run (as of this writing, **Filter Other**, **Move Transposed Notes To Mid Line**, **Flip Selected Notes**, and **Add Instrument Change With Names**), you will also need:

- Execute Commands
- cmdutils
- Custom Search Shortcut
- Run Command Macro

These can all be installed using **File>Plug-ins>Install Plug-ins**. Use the **Search** box to find them in the list of **all plugins**, or go to the category **Developers' Tools**.