Execute Commands Plugin: Command List vs. Command Macro vs. Command Plugin

Bob Zawalich May 2, 2021

In the **Execute Commands** plugin you can create a sequence of commands in the **Command List**, which is the rightmost of 3 list boxes in the dialog. You create the sequence by selecting a command from the 2nd listbox and using the **Add Command to Command List** button to add the command to the Command List. The dialog is designed so that this button is always the Default Button, so you can always just type Enter to add a command.

Command Classes

In the current version of **Execute Commands** (1.28.00 or later), there are 3 "classes" of commands available for adding. These are:

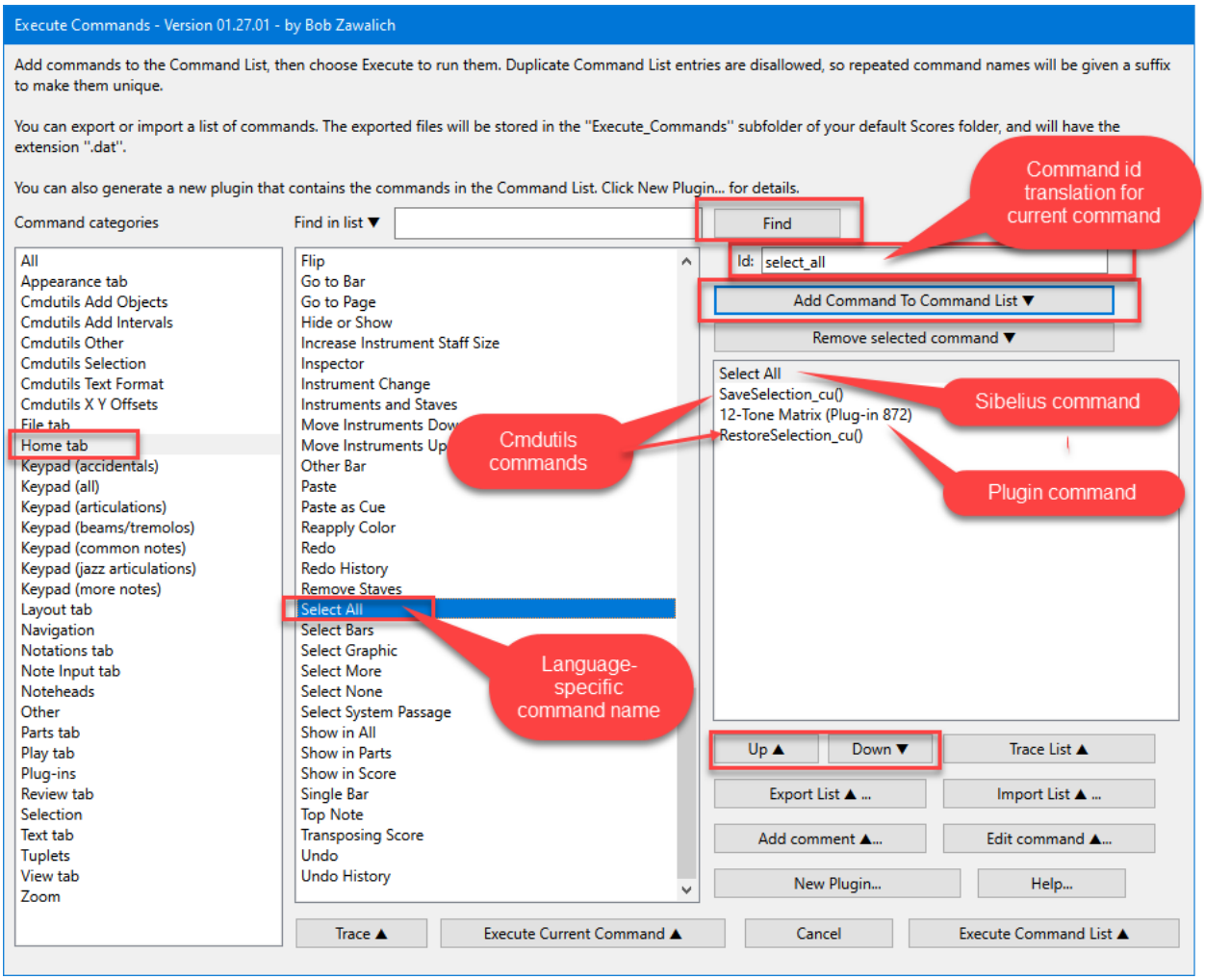
- **Sibelius commands**: These appear in Home >Commands>Commands, and are mostly the same as the commands in File>Preferences>Keyboard Shortcuts.
- **Plug-in commands**: These also appear in Home >Commands>Commands. These are the "menu name" of the plugin followed by the plugin number in parentheses, such as *12-Tone Matrix (Plug-in 872)*. The number itself is not really useful. It will be different for the same plugin name on different machines, but its presence identifies this command as a plug-in.
- **Cmdutils commands**: These are new as of **Execute Commands** version 1.28. They are a set of special routines designed to increase the abilities of Command Macros and Command Plugins. All cmdutils commands have a **_cu** suffix, followed by parentheses. Some of these commands have a text parameter between the parentheses, such as *AddSelect_Text_Dynamics_cu(mf),* which lets you specify the specific dynamics text you want. By default it adds ***mf***, but you can change that with the **Edit Command** button.

You add any of these commands to the Command List the same way, and in practice they all behave the same way. Cmdutils commands are the only ones that have useful parameters. Plug-in commands do have the plug-in name in the parentheses, but it is not used for anything except for identifying itself as a plugin.

Here is an annotated **Execute Command** command sequence, where there are 4 commands in the Command List:

```
Select All
SaveSelection_cu()
12-Tone Matrix (Plug-in 872)
RestoreSelection_cu()
```

Select All is a Sibelius command. *SaveSelection_cu()* and *RestoreSelection_cu()* are cmdutils commands, and *12-Tone Matrix (Plug-in 872)*is a Plug-in command.



Command Names and Command Ids

Translation issues

If you are running command with the Command Search bar, it is fine that the commands are in the native language, but with Execute Commands you can save a command sequence as a Command Macro or Command plugin, which you can then give to another Sibelius Ultimate user, who may be using a different language.

If you are only using Command Macros and Command Plugins in your own language, you don't really need to be concerned about this, but understanding it will make it easier for you to grasp what happens when command are exported or made into plugins.

French Sibelius, for example, will not recognize Sibelius commands written in English.

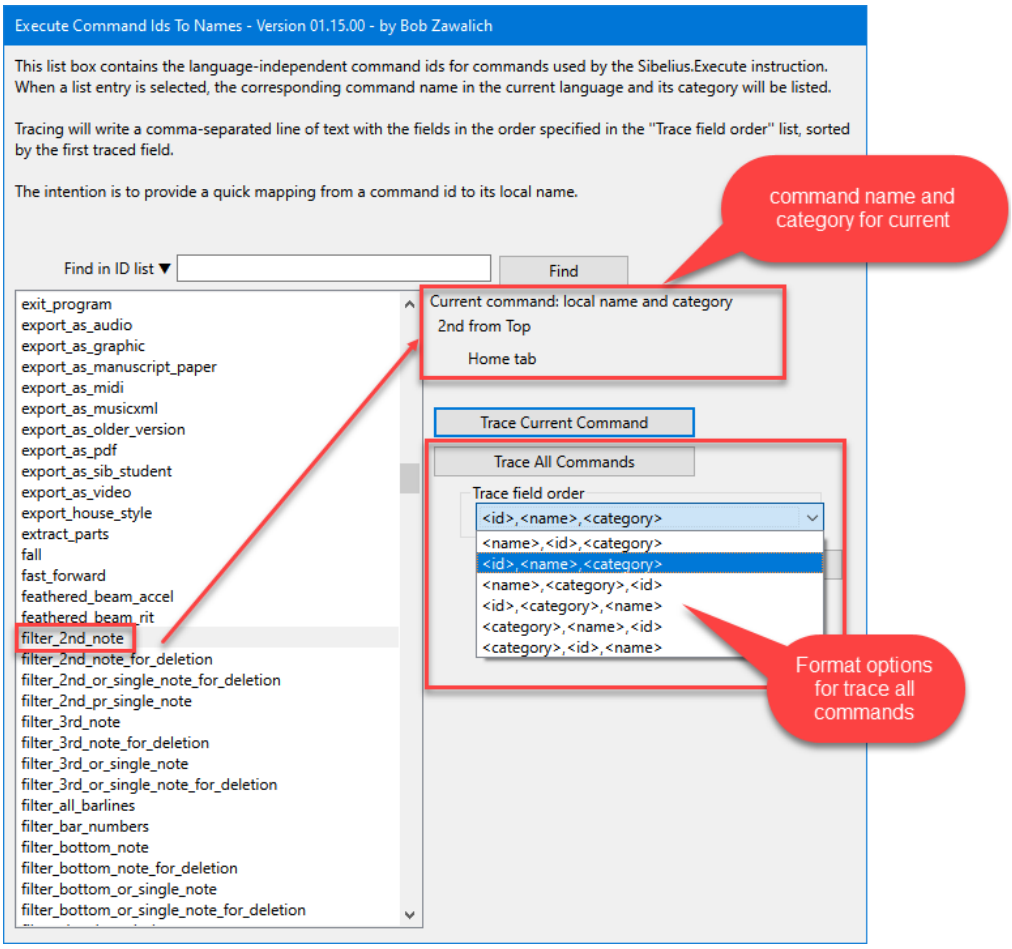
Feel free to skip to the next major section if this is not useful.

- Sibelius commands, and the command categories in the first list box are translated into the language that Sibelius is running under, and so will be different in English or German or Spanish.
- Plug-in commands will be translated for shipping plugins, but plugins that you have downloaded and installed will appear in the language in which the plugin was written. All my downloadable plugins will have English names, and so will their Plug-in commands.
- cmdutils commands will always be in English.
- Sibelius command names that have command ids can be converted to command ids in a plugin by using the **Cmd()** instruction.

- Plugin commands are not given command ids by Sibelius, but I have made up a language independent id for plugins, taking their file name, with no path and no extension, and appending ".plg" to it. **Sibelius.Execute()** does not work with Plugin commands; instead, **Execute Commands** and generated plugins make a call to the **Execute Commands** method **Run Plugin,** such as *ExecuteCommands.RunPlugin("HarpGlissPlayback.plg");*
- cmdutils commands are also not run by **Sibelius.Execute()**. Instead the commands are executed by cmdutils.plg itself.
    - In a plugin, the command *SaveSelection_cu()* would appear as *cmdutils.SaveSelection_cu();*

To work around these language issues, Sibelius invented language independent Command Ids for each Sibelius command. These are listed in the **Manuscript Language Reference**, and you can also see them by installing and running the plugin **Execute Command Ids To Names**.

As shown in the screenshot, it provides a list of all command ids. If you select one in the list, its local name and category will be displayed.



If you click on **Trace Current Command**, you will see something like this:

```
filter_2nd_note,2nd from Top,Home tab
Sibelius.Execute("filter_2nd_note"); // 2nd from Top
Sibelius.Execute(Cmd("2nd from Top")); // filter_2nd_note
```

The first line is what you see in the dialog, with the id,local name,and category, separated by commas.

The next 2 lines are translations into commands that can be copied into a Sibelius plugin. *Sibelius.Execute* runs a Sibelius command in a plugin, and takes a Command Id as a parameter, so these calls will always be language independent.

In this case it uses *"filter_2nd_note"*.

The 3rd line

```
Sibelius.Execute(Cmd("2nd from Top")); // filter_2nd_note
```

shows the use of the *Cmd()* function, which, in ManuScript, translates a local name to a command id. This can be easier to remember than a command id, but be aware that if you use this form of the **Sibelius.Execute()** instruction in a plugin, it will only work in the original language.

Trace All Commands traces the command id, command name, and category for all the Sibelius commands, written as comma separated fields, which could be loaded into a spreadsheet, where you can sort the fields in different ways.

You can trace in different field orders. The default <id>,<name>,<category> starts out like this:

```
128th_note,128th note,Keypad (more notes)
16_tremolos,16 tremolos,Keypad (beams/tremolos)
16th_note,16th note (semiquaver),Keypad (common notes)
256th_note,256th note,Keypad (more notes)
2_tremolos,2 tremolos,Keypad (beams/tremolos)
32_tremolos,32 tremolos,Keypad (beams/tremolos)
32nd_note,32nd note (demisemiquaver),Keypad (common notes)
4_tremolos,4 tremolos,Keypad (beams/tremolos)
512th_note,512th note,Keypad (more notes)
64th_note,64th note (hemidemisemiquaver),Keypad (more notes)
8_tremolos,8 tremolos,Keypad (beams/tremolos)
8th_note,Eighth note (quaver),Keypad (common notes)
accent,Accent,Keypad (articulations)
accessibility_preferences,Accessibility Settings,Other
acciacatura,Acciaccatura,Keypad (more notes)
add_2nd_above,Add interval 2nd above,Note Input tab
```

You can instead use <name>,<id>,<category> , which starts like:

```
'+' on keypad,keypad_+,Keypad (all)
'.' on keypad,keypad_.,Keypad (all)
0 on keypad,keypad0,Keypad (all)
1 on keypad,keypad1,Keypad (all)
128th note,128th_note,Keypad (more notes)
16 tremolos,16_tremolos,Keypad (beams/tremolos)
16th note (semiquaver),16th_note,Keypad (common notes)
2 bar Repeat Bar,repeat_2_bars,Keypad (jazz articulations)
2 on keypad,keypad2,Keypad (all)
2 tremolos,2_tremolos,Keypad (beams/tremolos)
256th note,256th_note,Keypad (more notes)
2nd from Top,filter_2nd_note,Home tab
3 on keypad,keypad3,Keypad (all)
32 tremolos,32_tremolos,Keypad (beams/tremolos)
32nd note (demisemiquaver),32nd_note,Keypad (common notes)
3rd from Top,filter_3rd_note,Home tab
4 bar Repeat Bar,repeat_4_bars,Keypad (jazz articulations)
4 on keypad,keypad4,Keypad (all)
4 tremolos,4_tremolos,Keypad (beams/tremolos)
5 on keypad,keypad5,Keypad (all)
512th note,512th_note,Keypad (more notes)
6 on keypad,keypad6,Keypad (all)
64th note (hemidemisemiquaver),64th_note,Keypad (more notes)
7 on keypad,keypad7,Keypad (all)
8 on keypad,keypad8,Keypad (all)
8 tremolos,8_tremolos,Keypad (beams/tremolos)
9 on keypad,keypad9,Keypad (all)
Accent,accent,Keypad (articulations)
Accessibility Settings,accessibility_preferences,Other
```

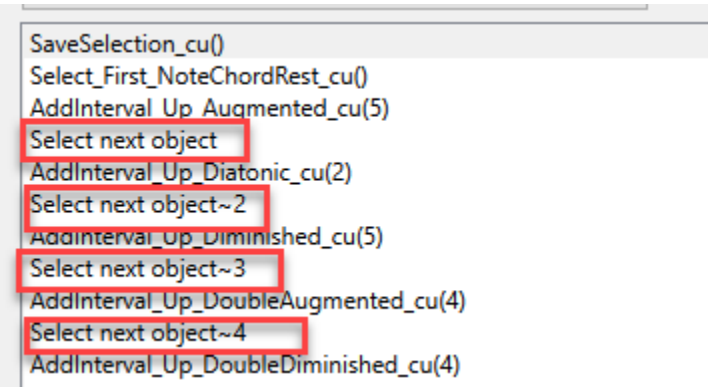
Duplicate commands in the Command List

In the previous example, all the commands in the command list were different. If we wanted to run the same command more than once, there would be a major problem because of the way Sibelius gives plugins access to list box entries.

Plugins get only the text of the selected list box entry. If more than one list box entry has the same text, then the plugin is always given the first matching entry.

In **Execute Commands**, the plugin can delete the selected entry or move it up or down, and if there are duplicates it would often try to process the wrong entry.

To work around this, Execute Commands adds a suffix consisting of a tilde (~) and a number whenever a duplicate command is added to the list, or if a Command Macro is imported.
You may see something like this, where there are multiple copies of *Select next object*:



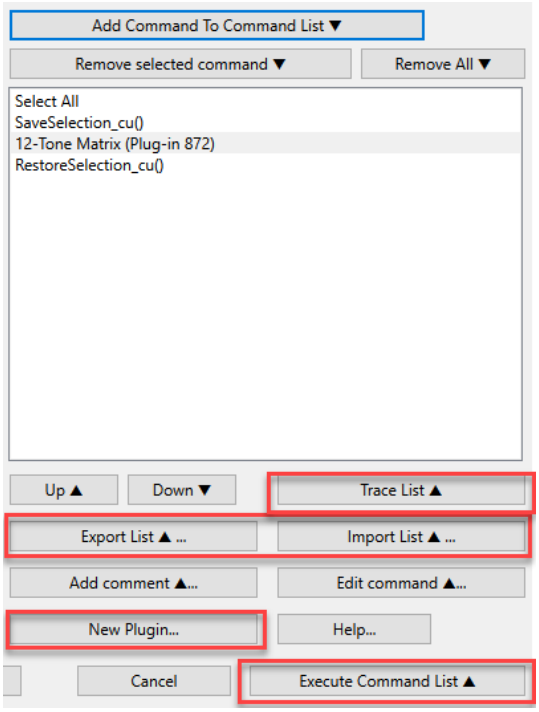
These suffixes only appear in the Command List, and are only there so it can manipulate the list box entries.

One peculiarity: if you **remove** an entry with a suffix, such as *Select next object~2*, the numbers of the other suffixes will not change. I tried doing that and it was disturbing to see all the entries change, so now I just leave a gap. If you add another *Select next object* command, it will not fill the gap, but will start at the next number, in this case *5*.

Another peculiarity: the suffixes are not included when a Command List is **Exported** or used in **New Plugin**. If you **Import** a Command Macro, it will regenerate the suffixes, and there will be no gap.

Suffixes do not matter. *Plug-in numbers do not matter*. They are only present in the Command List so the contents of the list can be manipulated. I apologize for any confusion they may cause.

Executing the Command List



You can run, or execute, the commands in the Command list. They will be executed in order, and will process the current selection in the active score, just as commands run from Command Search do.

Execute Command List causes **Execute Commands** to close its dialog, run the commands in the Command List, and exit.

You can also execute a single command, without needing to bring it into the Command List, by choosing a command in the middle list box and using **Execute Current Command**. This will also cause **Execute Commands** to exit. (A plugin cannot execute commands while a dialog is up).

Once you add commands to the Command List, they will be there for the current Sibelius session only. You can close **Execute Command**s, and then run it again, and they will still be there, but if you close Sibelius, restart it, and run **Execute Commands**, the Command List will be empty.

If you want to clear the list and start over, either click on **Remove All**, or Import a previously exported **Command Macro**.

Exporting a Command List as A Command Macro or Command Plugin

If you want to be able to reuse the command sequence, the best thing to do is to use **Export List**, which will save the commands in a text file with a *".dat"* extension, in the **Execute_Commands** folder of your default **Scores** folder (as declared in **File>Preferences>Saving and Exporting>Saving Scores**). It will create that folder if it does not exist. (The default **Scores** folder was chosen to stores these because it is easy to find, and should make sharing *dat* files easier).

You can recover the command sequence using **Import List**.

Execute Commands refers to sequences exported as *dat* files as **Command Macros**.

You can edit these with a text editor if you like, or bring them back into **Execute Commands** to add, remove, or reorder the commands.

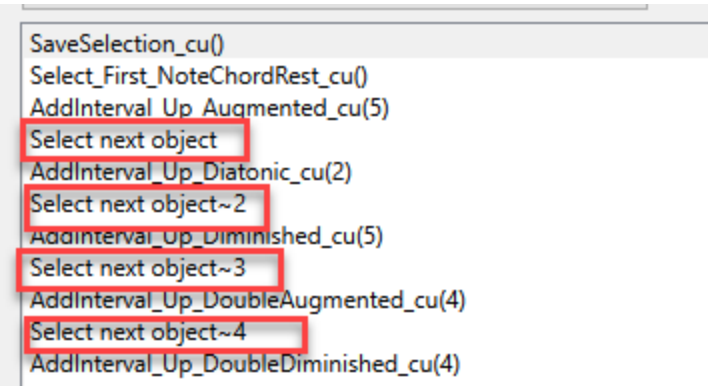
You can execute an imported Command Macro immediately, but for general use of Command Macros, the plugin **Run Command Macro** is probably more convenient. It is discussed in the **Scoring Notes** blog post Using commands in plug-ins in Sibelius Ultimate.

You can also use the **New Plugin** button to convert the command sequence into an executable plugin. While Command Macros need a plugin (**Execute Commands** or **Run Command Macro**) to run them, the plugin created by **New Plugin** is self-sufficient. Once it is correctly installed and Sibelius has been closed and restarted, it behaves like any other plugin.

What do Command Macro and Command Plugin files actually look like?

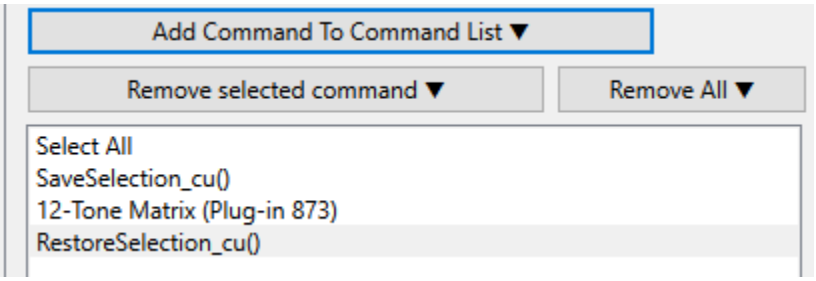
Command sequences are transformed by **Export List** and **New Plugin**.

The first thing that happens is that any ~ suffixes are stripped off, so if we started with, for example,



All the *Select next object* lines would lose the ~ suffixes.

Let us looks at this (useless) example macro that contains one Sibelius command, 2 cmdutils commands and one plugin.



When this sequence is exported to a Macro Command data file or a plugin is made from the commands, they are all translated into language -independent forms if possible, so that they can be shared across different native languages.

- **Sibelius Command Names** are translated to their Command Id
- **Cmdutils Command Names** are unchanged, since they are always in English
- **Plugin Commands** are translated to a form of their file name, which is never translated by Sibelius.

These commands:

```
Select All
SaveSelection_cu()
12-Tone Matrix (Plug-in 873)
RestoreSelection_cu()
```

are Exported to a data file as:

```
select_all
SaveSelection_cu()
12ToneMatrix.plg
RestoreSelection_cu()
```

Command Macro Files

- *Exporting Command Macros*
  - As explained above, the command names are translated into command ids or other language-independent representations of the command names, but they will produce the same effect as if you had executed the language-dependent Command Names.
- *Importing Command Macros*
  - When a *dat* file is imported, a reverse translation takes place, and if there are duplicate commands, the "~" suffixes will be rebuilt. If there had been a gap in the numbering before exporting, the duplicates will now be renumbered with no gap.
  - With respect to translation:
    - Command Ids are replaced with Command Names
    - cmdutils commands are unchanged
    - Plugin ids are replaced with a Command Name consisting of the plugin's Menu Name followed by its current plugin number, which may be different than the number it had when the file was exported, but that is not a problem.

Command Plugin Files

When a plugin is created by New Plugin, a fair bit of "boilerplate" code is written, independent of the Commands, which will also be translated to their language independent forms. The Initialize() method will always be the same, and Run() will be the same, except that there may be checks to see if Execute Command and/or cmdutils are installed, in case calls will be made to them.

The commands themselves, in a modified form, will be found in the method ProcessScore. Here is what a generated plugin with the 4 command we used as an example looks like when viewed in a text editor:

```
{
Initialize "(){
AddToPluginsMenu(_PluginMenuName, 'Run');
if (Sibelius.ProgramVersion > 20200600) {
   SetInterpreterOption(TreatSingleCharacterAsString);
   SetInterpreterOption(SupportHalfSemitonePitchValues); }
}"
Run "(){
if (Sibelius.Plugins.Contains('ExecuteCommands') = False){
   Sibelius.MessageBox(utils.Format(_msgNoPlugin, 'ExecuteCommands'));
   return False;}
if (Sibelius.Plugins.Contains('cmdutils') = False){
   Sibelius.MessageBox(utils.Format(_msgNoPlugin, 'cmdutils'));
   return False;}

score = Sibelius.ActiveScore;
score.Redraw = False;
valRet = ProcessScore(score); // user code will be found here
score.Redraw = True;
return(valRet);
}"

ProcessScore "(score){
Sibelius.Execute('select_all');
cmdutils.SaveSelection_cu();
ExecuteCommands.RunPlugin('12ToneMatrix.plg');
cmdutils.RestoreSelection_cu();
}"
```

```
zg_VersionNumber "010000"
_PluginMenuName "Command Type Sample"
_msgNoPlugin "Please install the plugin %s and run this plugin again."
}
```

If you need to edit this code, you will likely only need to change what is in the **ProcessScore()** block, which is where the specific commands for this plugin were added. Briefly, though:

The **Initialize** code is called when the plugin is loaded at startup. It registers the plugin menu name, in this case Command Type Sample, and its Run() method, and it adds some features, including handling of quartertones, that are now pretty much standard.

Run() is normally the main routine, but for generated plugins only the code that checks whether the plugins Execute Commands and/or cmdutils have been installed is different between plugins.Run() calls ProcessScore() which is where the action takes place.

You will recall that we started with these commands;

Select All
SaveSelection_cu()
12-Tone Matrix (Plug-in 873)
RestoreSelection_cu()

And in ProcessScore we get:

Sibelius.Execute('select_all');

- Select All is a Sibelius Command, we use Sibelius.Execute() and the Command Id form 'select_all'

cmdutils.SaveSelection_cu();

- This and the final command are cmdutils commands. The generated code starts with "cmdutils." concatenated to the unchanged Command Name.

ExecuteCommands.RunPlugin('12ToneMatrix.plg');

- This is a plugin. We call ExecuteCommands.RunPlugin and the language-independent command form '12ToneMatrix.plg'.

cmdutils.RestoreSelection_cu();