# Tutorial: Execute Commands

## Writing Language Independent Command Macros and Command Plugins

*Bob Zawalich June 4, 2021*

## Language issues and Command Ids

If you are running commands from the Ribbon, or add several commands to the Command List in Execute Commands and then choose Execute, you can use the command names as they are shown, in the current language, such as English or German.

If you export and import macros, or create new plugins, then the command names will only work if you are running on a machine in the same language. This is probably how it will be most of the time.

However, if you have a set of useful macros files, or plugins that used Commands, and want to share them with someone running Sibelius in another language, the command names will not be recognized. For example, "Move Down Chromatically" will not be found on a German machine.

To deal with this problem, each Sibelius Command Name was given a language -independent Command Id. If your plugin uses Command Ids consistently, it will run in any language.

In a plugin you can call **Cmd(<command name>)** to convert the Sibelius command to a command id, and use **Sibelius.Execute(<cmdId>)** to execute the command in a plugin.

This means that you can run **Sibelius.Execute(Cmd<command name>))**to execute a command, but note that **<command name>** will be in the local language, so plugins that use such commands will be limited to running in a single language.

**Plugin commands and commands from the cmdutils plugin are not recognized by Sibelius** itself, and the **Sibelius.Execute()** instruction cannot be used to execute these commands.

**As a workaround, I have invented language-independent ids for these commands, and mechanisms for running such commands.**

- The plugin command name is the plugin Menu Name with a suffix of (<plugin number string>), such as **"Add Pickup Bar(Plug-in 198)"**. Menu Names are translated into the local language for shipping plugins but downloadable plugins almost never have the menu names translated, and plugin file names are never translated.
- For plugins, only shipping plugins will fail to run in a different language if you use the command name instead of an id.

- **For plugins, the id is the plugin file name with no path**. It will always include the extension ".plg". For example, the plugin "Add Pickup Bar" has the id "AddPickupBar.plg".
    - To execute a call to a plugin in a ManuScript plugin, use the routine ExecuteCommands.RunPlugin("<plugin command name or id>").
    - Example: **ExecuteCommands.RunPlugin("AddPickupBar.plg"); // Add Pickup Bar (Plug-in 198)**
    - **ExecuteCommands.plg** must be installed for this to work.

- Cmdutils routine names are never translated, so the id is the same as the command name.
    - To execute a call to a cmdutils routine in a ManuScript plugin, use the routine cmdutils.<command name or id>(["<parameter>"]);
    - Example: **cmdutils.Add_Line_cu("line.staff.arrow.black.right");**
    - **cmdutils.plg** must be installed for this to work.

## Finding an Id for a command name

- When writing language-independent ManuScript code you need to know the Id that corresponds to a specific local command name.

- When **Execute Commands** exports a Command Macro (**Export List**) or generates a plugin (**New Plugin**), it converts Command Names to Ids. If I am writing a plugin that uses commands I will often bring up **Execute Commands**, add the command to the Command List, and trace it so I can copy the correctly formatted lines of code with the translation to an id done for me automatically.

- The only situation where full translation is not done is when the **parameter** of a **cmdutils** command is itself a local name.
    - **cmdutils.Add_Line_cu("line.staff.arrow.black.right");** is language independent, because it uses a Line Style Id.
    - **cmdutils.Add_Line_cu("Arrow");** is language dependent, because is uses a translatable Style Name As Text.

- These commands can use Command names or ids as parameters (for Symbols or Noteheads, the id is an index).

    - Add_Line_cu(line.staff.arrow.black.right)
    - Add_StaffSymbol_cu(Choral divide arrow)
    - Add_SystemSymbol_cu(Coda)

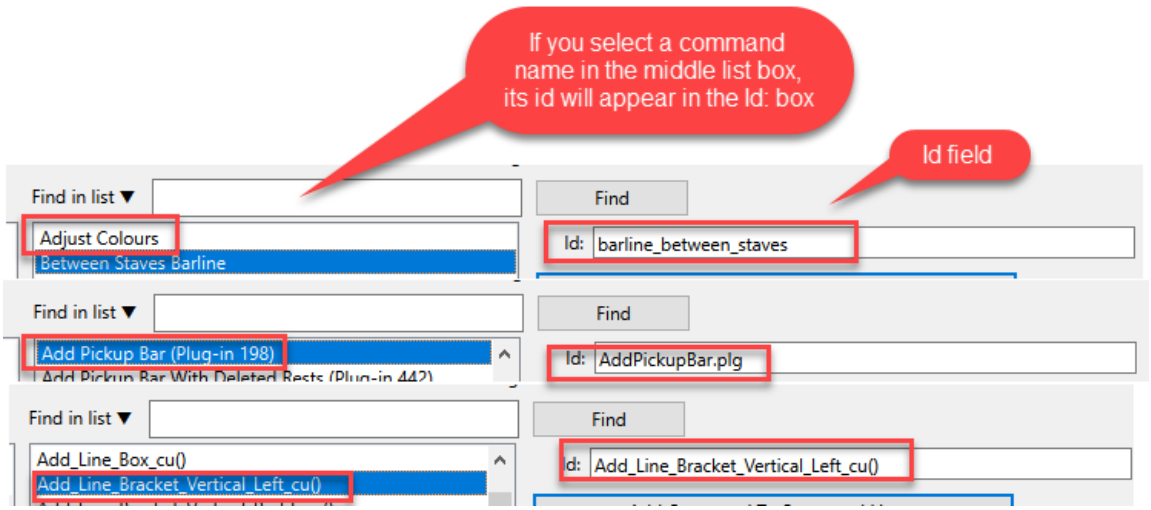
    - AddSelect_Line_cu(line.staff.arrow.black.right)
    - AddSelect_StaffSymbol_cu(Choral divide arrow)
    - AddSelect_SystemSymbol_cu(Coda)

    - ApplyNoteheadStyle_cu(2)

- The lower-level routines for adding objects are not exposed to **Execute Commands**, and if called directly they will need to have parameters set to ids for the code to be language independent.

**Execute Commands provides several tools to translate a Command Name to a Command Id.**

- There are several places where the id will be calculated and written to a location where you can copy it and paste in into a command.
- There is an **"Id: "** field above the Command List that shows the Id for the currently selected Command Name.




- The **Trace List** button below the Command List will trace data for the Command Names in the Command List, and will show these results:
  - The local Command name, as shown in the Command List
  - The ids for these commands
  - Copyable ManuScript statements for executing the commands, using the local command names
  - Copyable ManuScript statements for executing the commands, using the ids.
- Here is an example showing the Command List trace when the list contains one Sibelius Command, one Plugin, and one cmdutils command:

  - Move Down a Staff
  - Add or Replace System Text (Plug-in 739)
  - GoToNextPage_cu()

  - cross_stave_move_down
  - AddReplaceSystemText.plg
  - GoToNextPage_cu()

  - Sibelius.Execute(Cmd("Move Down a Staff")); // cross_stave_move_down
  - ExecuteCommands.RunPlugin('AddReplaceSystemText.plg'); // Add or Replace System Text (Plug-in 739)
  - cmdutils.GoToNextPage_cu();

  - Sibelius.Execute("cross_stave_move_down"); // Move Down a Staff
  - ExecuteCommands.RunPlugin('AddReplaceSystemText.plg'); // Add or Replace System Text (Plug-in 739)
  - cmdutils.GoToNextPage_cu();

- The **Trace** button in the middle listbox will trace all the commands in that list, which are the commands in a single category. Here are the commands for the **"Other"** category, with the command names listed first, followed by the Command Ids. The ManuScript commands are not traced here.

  - Accessibility Settings
  - Activate Command Search
  - Edit Text
  - Fret number 0 on tab
  - Minimize/Expand Ribbon
  - Next tie style

- Toggle L.V. tie
- accessibility_preferences
- command_search
- edit_text
- tab_fret_number0
- minimize_expand_ribbon
- change_tie_style
- toggle_lv_tie.

- The downloadable plugin **Execute Command Ids To Names** will trace all the available Sibelius Command Ids and show the corresponding Command Name and Category, and can be imported into a spreadsheet and sorted in different ways.
  - Only Sibelius Commands are listed here, not plugins or cmdutils routines.

- For the cmdutils commands such as **Add_Line_cu** that need ids or names for objects, there are these commands in the **CmdUtils Add Objects** category in **Execute Commands**:
  - **Trace_LineStyleIdFromName_cu(Glissando (wavy))**
  - **Trace_NoteStyleIndexFromName_cu(Diamond)**
  - **Trace_SymbolIndexFromName_cu(Mordent)**
  - **Trace_TextStyleIdFromName_cu(Technique)**
  - These can be used with a properly spelled name to get the language-independent StyleId of Index for the name.

  - **Trace_Object_Type_Name_StyleOrIndex_cu()**
  - This can be used by selecting appropriate objects in your score to get the language-independent StyleId of Index for the objects. This is probably the easiest way to find such ids.

  - For the first set of these you call the routine with a local name and get an appropriate id (or an empty string if the name is not valid). To use **Trace_Object_Type_Name_StyleOrIndex_cu(),** you need to select objects in the score that have the properties you want (add them temporarily if need be), and run the command on the selection and find the data in the plugin trace window.

You really only need to be concerned about the language issues if you give a plugin or macro to a user in another language. If you use Execute Commands to generate your Command Macros and Command Plugins, it will do the translation for you in almost all cases, and such macros and commands should run in any language.