Tutorial: Execute Commands and cmdutils

Communicating between Cmdutil Commands

Bob Zawalich June 22, 2021

Sibelius Commands just run. They take no input and they return no values. you need to be sure to set the score up so the commands will have what they need, and you have to assume that they worked.

If you need more communication than that, you can write a ManuScript plugin that calls commands, and you can sometimes indirectly find out what happened, or when feasible, just write a conventional ManuScript plugin.

Cmdutils Commands can take fixed strings of text as parameters. They return useful values when called by plugins, but in macros they do not return any information to the next command that runs.

Here are a few ways that **Cmdutils Commands** can provide information to succeeding commands.

AddSelect_ commands

AddSelect_ commands create an object, and once created, they change the selection so that only the new object is selected. Since commands apply to the current selection, you will know that either the new object will be the only thing selected, or the selection will be cleared on failure.

If there is a selection, then only the new object will be selected, so you can manipulate it if you have appropriate commands, such as the **SetTextFormat** commands for a text object.

You can save the current selection before running an **AddSelect_** command, and restore it afterward, so you could manipulate the new objects, and then put the selection back, like this:

```
SaveSelection_cu()
AddSelect_Text_Technique_cu(I am a bold one!)
SetTextFormat_Bold_cu()
RestoreSelection_cu()
```

You really cannot know for sure that the **AddSelect_** command was successful, but you could test the selection with one of the **Command Exit** commands.

Here are the current AddSelect commands

AddSelect _Line_cu(styleTextOrId)

AddSelect _Line_8va_cu()
AddSelect _Line_Box_cu()
AddSelect_Line_Bracket_Vertical_Left_cu()
AddSelect_Line_Bracket_Vertical_Right_cu()
AddSelect_Line_Ending_First_cu()
AddSelect_Line_Ending_Second_cu()
AddSelect_Line_Hairpin_Crescendo_cu()
AddSelect_Line_Hairpin_Diminuendo_cu()
AddSelect_Line_Plain_cu()
AddSelect_Line_Slur_cu()
AddSelect_Line_Trill_cu()
AddSelect_Line_Vertical_cu()

AddSelect_StaffSymbol_cu (nameOrIndexSymbol)
AddSelect_SystemSymbol_cu (nameOrIndexSymbol)

AddSelect_Text_cu(strText)
- **Adds text using the text style that was set by running the TextStyleDefaultForCommands _cu(styleTextOrId) command, or adds Technique text if the text style is not set.**
- **It is best to always run TextStyleDefaultForCommands _cu immediately before running this command.**

AddSelect_Text_Dynamics_cu (strText)
AddSelect_Text_Expression_cu (strText)
AddSelect_Text_Technique_cu (strText)

Command Exit commands

These routines will check for an empty or non-passage selection, or a passage selection that does not include specific staves or bars, or whether a plugin is installed. If found, they will give a warning and either Exit, or ask if you want to continue, possibly after selecting the entire score. This these commands communicate data about the current state of the selection, and perform some action if the selection is not what you want.

In the example above, we know that if an **AddSelect_** command fails, there will be no selection. We can follow

AddSelect_Text_Technique_cu(I am a bold one!)

with

ExitIfSelection_Empty_cu(The text could not be added, and the selection is empty. The plugin or macro will now exit.)

So we could have:

SaveSelection_cu()
AddSelect_Text_Technique_cu(I am a bold one!)
ExitIfSelection_Empty_cu(The text could not be added, and the selection is empty. The plugin or macro will now exit.)
SetTextFormat_Bold_cu()
RestoreSelection_cu()

You could also leave out **ExitIfSelection_Empty_cu**, and the plugin would continue as if you had never run the **AddSelect** command, since the original selection would be restored. What you do depends on your tolerance for not knowing why the text was not added.

Here are the current **Command Exit** commands

ContinueIfSelection_Empty_cu(Nothing is selected. Choose Yes to continue, No to stop the plugin.)

ExitIfPlugin_Unavailable_cu(Resize Bar)

ExitIfSelection_Avoid_BottomStaff_cu(The selection may not include the bottom staff in the score. This plugin will now exit.)
ExitIfSelection_Avoid_FirstBar_cu(The selection may not include the first bar in the score. This plugin will now exit.)
ExitIfSelection_Avoid_GrandStaff_Bottom_cu(The bottom staff of the selection may not be the bottom staff of a multi-staff instrument. This plugin will now exit.)
ExitIfSelection_Avoid_GrandStaff_Top_cu(The top staff of the selection may not be the top staff of a multi-staff instrument. This plugin will now exit.)
ExitIfSelection_Avoid_LastBar_cu(The selection may not include the last bar in the score. This plugin will now exit.)
ExitIfSelection_Avoid_TopStaff_cu(The selection may not include the top staff in the score. This plugin will now exit.)

ExitIfSelection_Empty_cu(Nothing is selected. This plugin will now exit.)

ExitIfSelection_Needs_FullSelect_cu(The selection must have all bars fully selected. This plugin will now exit.)
ExitIfSelection_Needs_GrandStaff_All_cu(The selection must include all the staves of a multi-staff instrument, including ossias. This plugin will now exit.)
ExitIfSelection_Needs_GrandStaff_Any_cu(The selection must include only staves of a single multi-staff instrument, including ossias. This plugin will now exit.)
ExitIfSelection_Needs_OneStaff_cu(The selection must include only a single staff. This plugin will now exit.)

ExitIfSelection_NotPassage_cu(A passage selection is required. This plugin will now exit.)

ExitOrAll_Selection_Empty_cu(Nothing is selected. Reply Yes to select all and continue, or No to exit.)
ExitOrAll_Selection_NotPassage_cu(A passage selection is required. Reply Yes to select all and continue, or No to exit.)

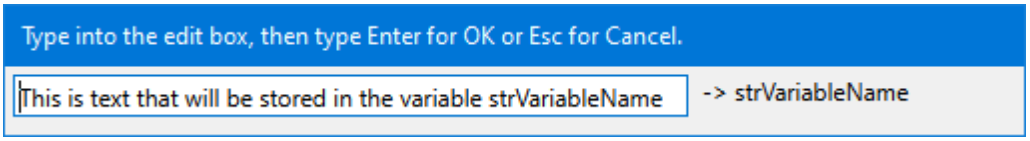
ExitPlugin_cu()

Parameter Variables

Parameters can only be changed if you edit the command itself. What if you want to be able to have the parameter change while a macro is running? **Parameter variables** provide a way to pass user input to a Cmdutils Command.

GetUserInput_cu(variableName), will put up a dialog you can type into, and will store the result in a ***parameter variable*** whose name is the parameter of **GetUserInput_cu**. A **parameter variable**, for our purposes, is a *name* that can hold a *value*.

For **GetUserInput_cu,** the *name* of the variable is its parameter and the *value* is the text that you type into the edit box . If I ran **GetUserInput_cu(strVariableName),** and typed in this text:



there would now be a variable with the name **strVariableName** that holds the value “This is text that will be saved in strVariableName”.

The parameter variable **strVariableName** will exist with its current value until the end of the Sibelius session in which it is created. If you try to use it in the next session without calling **GetUserInput_cu** first, the variable **strVariableName** will resolve to the text *strVariableName*. So don’t do that!

You can have any number of parameter variables, all with different names. If you use **GetUserInput_cu** with the same variable name, it will overwrite the previous setting.

The names of variables you create with GetUserInput_cu can be used as the parameter in any _cu command that takes a parameter.

After running **GetUserInput_cu(strVariableName)**, you could use any of these commands with **strVariableName** as a parameter:

Add_Text_Technique_cu(strVariableName)
Trace_cu(strVariableName)
MessageBoxYesNo_cu(strVariableName)

// These would need properly formatted text in strVariableName
ApplyNamedColor_cu(strVariableName)
AddInterval_Down_Diatonic_cu(strVariableName)
SetXOffsets_Left_Absolute_cu(strVariableName)

The text that is saved needs to be appropriate to the command using it. **Add_Text_Technique_cu, Trace_cu,** and **MessageBox_cu** will work with any text, but **ApplyNamedColor_cu** needs a specific valid color name, and the others need numbers, so be aware of what text the variable holds.

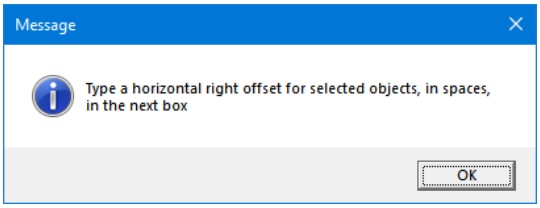
This gives you the ability to create a macro or plugin that will accept small amounts of text input without needing to call a plugin with a big dialog.

An Example of using a parameter variable

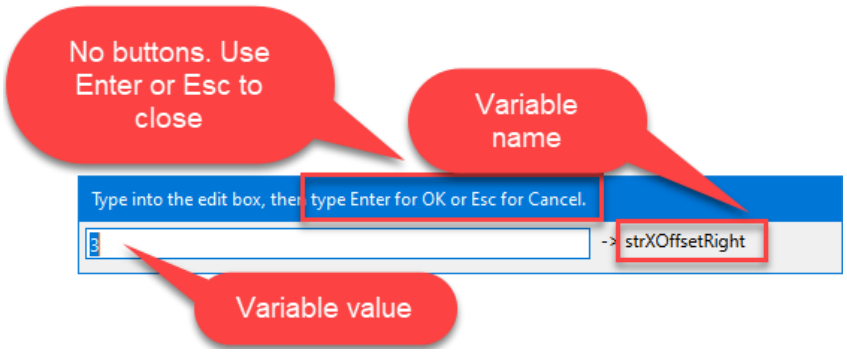
You could write something like this, to get an offset value to use in the command **SetXOffsets_Right_Relative_cu**:

```
MessageBox_cu(Type a horizontal right offset for selected objects, in spaces, in the next box)
GetUserInput_cu(strXOffsetRight)
SetXOffsets_Right_Relative_cu(strXOffsetRight)
```

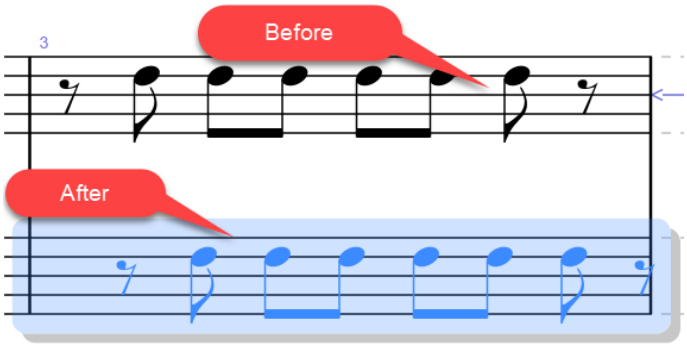
And you would see the message box,



Then the dialog that accepts the text,



And now the variable **strXOffsetRight** can be used as a command parameter for the rest of the Sibelius session. In this case it will be used to shift selected objects right, and you might see the selected notes shifted 3 spaces to the right in this example:



You would thus not need to write separate macros for each offset value you want to use, at the cost of having to enter a value for the offsets. If you wanted the same offset all the time for some situation, you could have separate macros/plugins for that situation.

You could write a separate macro for Left offsets, and for Y offsets as well.

The Offset commands expect a positive number. You can, though, put in a negative number, and it will make the offset reverse direction. Left will go right, right will go left. Up/down also respond to a minus sign. You thus need fewer commands if you choose to use negative numbers.

Entering a color name for **ApplyNamedColor_cu** would be another good use for this.

A variation on Parameter Variables - TextStyleDefaultForCommands _cu(styleTextOrId)

The command **Add_Text_cu(strText)** had a problem. It needed to specify both its text style name and the text itself, but cmdutils limits the parameters to a single string of text. I created the command

TextStyleDefaultForCommands_cu(styleTextOrId)

to work around that. **TextStyleDefaultForCommands_cu** takes its parameter, which is a text style name or id, and saves it into a global variable that can be accessed by **Add_Text_cu** and **AddSelect_Test_cu**. Like the parameter variables, the global variable will hold its value until Sibelius is closed, but I recommend calling **TextStyleDefaultForCommands_cu** immediately before running any command that needs that variable.

Parameter variables are a generalization of the way **TextStyleDefaultForCommands _cu** works.