

The cmdutils library in Execute Commands

Bob Zawalich 27 April, 2021, Updated 23 March 2024

Contents

Overview.....	2
Using cmdutils commands in Execute Commands	3
Cmdutils Commands.....	4
Command Parameters.....	4
GetUserInput_cu and <i>parameter variables</i>	6
“Full” methods in cmdutils.plg (not available in Execute Commands).....	8
Classes of Cmdutils Commands	10
Add Object and “Add and Select” Commands.....	10
Identifying objects to be added	10
Add Object Commands.....	12
“Add and Select” Commands.....	13
Add Intervals Commands	14
Transpose Intervals Commands.....	14
Command Selection commands	16
Command Exit Commands	18
Text formatting commands	19
View Tab Commands.....	20
Horizontal (X) and Vertical (Y) Offset commands.....	21
User Input Commands	21
Mute/Unmute Commands	22
“Other” Commands	22
Alphabetical full list of commands called from Execute Commands (as of March 23, 2024).....	24
Commands in category: <i>Cmdutils Add Objects</i>	25
Commands in category: <i>Cmdutils Add Intervals</i>	26
Commands in category: <i>Cmdutils Selection</i>	26
Commands in category: <i>Cmdutils Exit Plugin</i>	27
Commands in category: <i>Cmdutils Text Format</i>	28
Commands in category: <i>Cmdutils Transpose Intervals</i>	28
Commands in category: <i>Cmdutils View</i>	28
Commands in category: <i>Cmdutils X Y Offsets</i>	28
Commands in category: <i>Cmdutils Other</i>	29
Making plugins that use a single call to a cmdutils command	30
Debugging aids.....	30
Sharing Command Macro and Command Plugins Files.....	30
<i>Tip: use Command Macro format (.dat files) for your “source code”</i>	31
A sample plugin that uses cmdutils calls: Add Line And Dynamic Text	32

Overview

cmdutils.plg is a library of ManuScript commands, written by Bob Zawalich, intended to be called by other plugins, particularly plugins that make use of the Command/**Sibelius.Execute()** facilities made available in Sibelius Ultimate 2021.2. You install it with **Install Plug-ins**, like any other plugin.

You can call the routines in **cmdutils.plg** in any plugin in Sibelius6 or later. To use it with **Execute Commands** you need Sibelius Ultimate 2021.2 or later.

The purpose of the library is to increase what is possible to do with sequences of commands.

It provides some features that are not available as Sibelius Commands, in part because most of the new features take parameters and have return values, which Commands do not use.

The return values will only be accessible in plugins that use the commands, not in Command macros, which are sequences of commands with no intervening ManuScript code.

Most of these new commands can be accessed directly from the plugin **Execute Commands**, using the “**Cmdutils**” categories. These will only appear in the list of categories if **cmdutils.plg** is installed.

You can trace and then copy and print the list of the commands for each category using the **Trace** button below the middle list box.

For an overview on using the **Execute Commands** plugin, see the Scoring Notes post <https://www.scoringnotes.com/tutorials/using-commands-in-plugin-ins-in-sibelius-ultimate/>

The screenshot shows the 'Execute Commands' plugin window, version 01.26.00 by Bob Zawalich. The window has a blue header and a white body. On the left is a 'Command categories' list with tabs like 'All', 'Appearance tab', 'File tab', etc. The 'Cmdutils' category is expanded, showing a list of commands with a '.cu' suffix. A red callout points to this list, stating: 'Cmdutils commands all have a .cu suffix. They may have a text parameter that can be edited with Edit Command...'. Another red callout points to the 'Cmdutils' category in the list, stating: 'These are available only if cmdutils.plg is installed'. The main area of the window contains a 'Find' box, a 'Find' button, and a list of commands. A red callout points to a comment line in the list: 'Cpmmnt line added by Add Comment'. Another red callout points to a command in the list: 'Edit the placeholder text of commands with parameters using Edit Command...'. At the bottom of the window are buttons for 'Trace', 'Execute Current Command', 'Export List', 'Import List', 'Add comment', 'Edit command', 'New Plugin', and 'Help'. A red callout points to the 'Trace' button, stating: 'Write a list of the Cmdutils commands in each category to the plugin Trace window. Copy to a document for reference.' Another red callout points to the 'Execute Current Command' button, stating: 'Run only the command selected in the middle list'.

Using cmdutils commands in Execute Commands

Many of the **_cu** commands take no parameters and can be used like any other command. Choose a **Cmdutils** category and add the command you want to call, such as

```
ExtendSelection_Left_cu()  
MuteSelectedNotes_cu()  
SetTextFormat_Bold_cu().
```

to the Command List.

Some **_cu** commands, such as the **Add** and **AddSelect** commands, need to be told what to add, and so you must add some text between the parentheses (). The commands on the list that need such text will be given *placeholder parameters* that you will probably need to replace to accomplish what you want to do. For example, these commands have text that you might need to change after adding them to the Command List.

```
ExitIfSelection_NotPassage_cu(A passage selection is required. This plugin will now exit.)  
AddSelect_Text_Technique_cu(legato)  
Add_Line_cu(line.staff.hairpin.crescendo)
```

You can do simple editing of the placeholder parameters using the **Edit Command...** button.

In most cases, the placeholder text will produce a reasonable value, even if it is not what you want.

RunPluginHideDialog_cu(Valid Plugin Menu Name), however, might cause problems if it used a valid plugin name, because if you ran a plugin without showing its dialog, and the plugin changed your score, you would likely be unhappy. Thus, if you do not change the placeholder, you will be told that the command is invalid and you will have to change it.

You cannot access the return values of the commands in **Execute Commands**, or use commands that need objects instead of strings for parameters. To access those facilities, you will need to create a plugin and edit these commands in the plugin.

You can generate a **New Plugin** from a sequence of commands in **Execute Commands**, or otherwise create a plugin in the traditional manner, and then add calls to these commands from within the plugin code. If you then edit that plugin in **File>Plug-ins>Edit Plug-ins** or a text editor you will find the commands that had been in the Command List are in the command **Process Score**.

The **Execute Command** command form does not use quotes or semicolons, but when code is written to a **New Plugin** it will be made into syntactically correct **ManuScript** code.

You can also call other routines in **cmdutils** in a plugin with ManuScript code. They will not have a **_cu** suffix in their names, since that designation is reserved for commands that can be used in **Execute Commands**. The **_Full** variants of the cmdutils commands are not available in **Execute Commands**, but can be called directly in a plugin.

Here is an example of a **_cu** command with edited text in the parameter:

```
Add_Text_Technique_cu(Adding this text at start of the selection)
```

and the **ManuScript** code used in a plugin to call this **cmdutils** command:

```
txtNew = cmdutils.Add_Text_Technique_cu("Adding this text at start of the selection");
```

This will add **Technique** text to the start of the selection, and return the created object.

You should ensure the **cmdutils.plg** is installed on the machine of anyone using such a plugin, and typically **ExecuteCommands.plg** also needs to be installed. Any plugins generated by **New Plugin** in **Execute Commands** will add code to check for these plugins if any of the commands require them.

Cmdutils Commands

The **_cu** commands in **cmdutils** process the first selected object in the current selection of the currently active score, mimicking what Sibelius does when adding object to a score.

As in Sibelius itself, **many of these routines treat staves hidden by Focus In Staves or Hide Empty Staves as if they are not hidden.** In particular the **ExtendSelection Up/Down** and **Contract Selection Up/Down** routines will just include or exclude the next staff they find, irrespective of the hidden state. Be careful when copying and pasting in selections that include hidden staves.

If the current selection is not a passage selection, the location of the object will be that of the first object found in the selection. *If the selection is not a passage, the order of these is determined by the order in which the objects were selected, which may not be in chronological score order.*

If you want to make sure the new objects will be added to the position of the chronologically first or last selected object, you can first call **SelectPassageForCommands_cu**, which will turn the multiple selection into a passage, or use one of the **Select_First/Last_Object** commands, which will sort the selected objects and select only the first or last of the originally selected objects. Now you can run an **Add Object** command knowing where it will actually go.

The commands listed below have either a **_cu** suffix or a **_Full** suffix. The **_cu** commands call the **_Full** commands, with a hard-coded set of parameters, so the **_cu** routines are less fussy to deal with. If you need the full range of parameters, you can call the **_Full** routines directly from a Manuscript plugin. **_Full** routines are displayed in *Italic* in this document.

Command Parameters

One thing that distinguishes **cmdutils** Commands from Sibelius Commands is that **cmdutils** Commands can accept parameters.

Most **cmdutils** commands have no parameters; you simply add the command to the Command List and run it. The names of such commands will end with “_cu()”, with nothing between the parentheses.

Here are some commands that have no parameters:

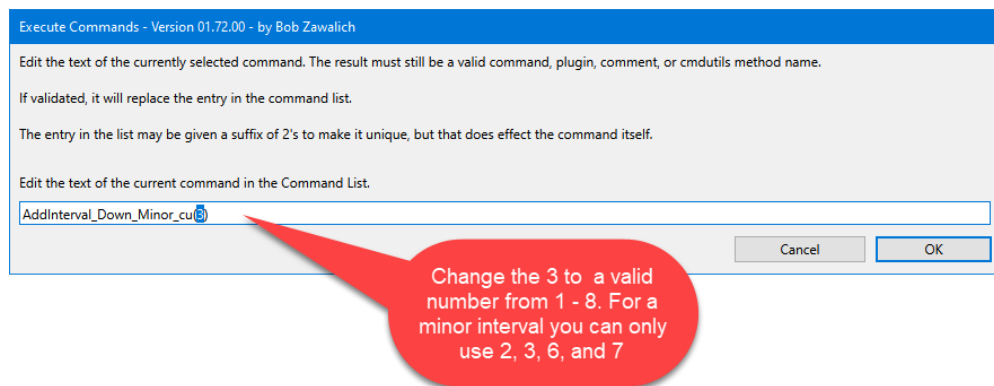
```
ExtendSelection_Left_cu()  
GoToFirstBar_cu()  
SetTextFormat_Bold_cu()  
MuteSelectedNotes_cu()
```

Other commands will have a single text parameter between the parentheses, which allows the commands to work with a number of inputs. While the command **Add_Line_Plain_cu()** has no parameter, since the line style is implied by the command name, **Add_Line_cu()** needs a style name or id, such as **Add_Line_cu(Glissando (wavy))** or **Add_Line_cu(line.staff.arrow.black.right)**. This allows **Add_Line_cu** to create a large number of line styles, at the cost of needing to very precisely specify the style – misspelled style names will cause the command to fail.

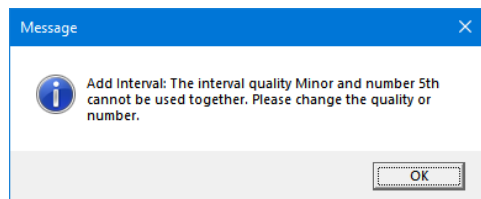
When a command needs a parameter, it will always appear in the command list with a “placeholder” parameter.

In some cases, that placeholder can be the one you need, as in **AddInterval_Down_Minor_cu(3)**, where the “3” is a placeholder for the interval you want, if you do not want to add a minor 3rd, you can edit the command to change the parameter. For the **AddInterval** commands, you can change the placeholder to be a whole number from 1 (unison) to 8 (octave).

Changing the placeholder can be awkward, since **Execute Commands** is not set up to edit in place. The supported editing mechanism is to select the command in the Command List and press **Edit Command**. You will see a dialog like this:



Edit the **3** to be the interval you want and press OK. For some commands, like **AddInterval**, there will be restrictions of the values of parameters, which will only be enforced when the command is run. If, for example, you asked for an interval of a minor 5th, you would see this message.



If I am writing a macro that has a lot of parameters I will need to change, I enter the commands into the Command List, use **Export List** to create a text (.dat) file, then open that file in a text editor, make the edits all at once, then save the file in the text editor, and load it back into **Execute Commands** using **Import List**.

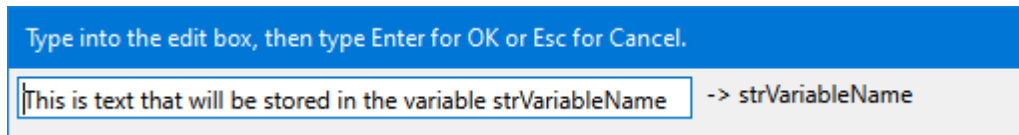
Note that parameters can only be text characters or numbers. They cannot include single or double quote characters (due to internal Manuscript limitations), so sometimes you need to be clever to work around the text (use **do not** instead of **don't**, for example, and rewrite to avoid possessives.) The parameter string in the command **RunPluginEntry_cu** cannot include commas, except those that separate sub-parameters. With great power comes great responsibility!

GetUserInput_cu and *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:



Type into the edit box, then type Enter for OK or Esc for Cancel.

This is text that will be stored in the variable strVariableName -> strVariableName

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, **strVariableName** will resolve to the text *strVariableName*. So don't do that!

You can also change the heading used in the **GetInput** dialog by calling **SetUserInput_Heading_cu()**. The default heading, if this is not called, is “Type into the edit box, then type Enter for OK or Esc for Cancel.” Once you call **SetUserInput_Heading_cu**, the heading will be used by **GetUserInput_cu** for the remainder of the Sibelius session, or until **SetUserInput_Heading_cu** is called again.

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.

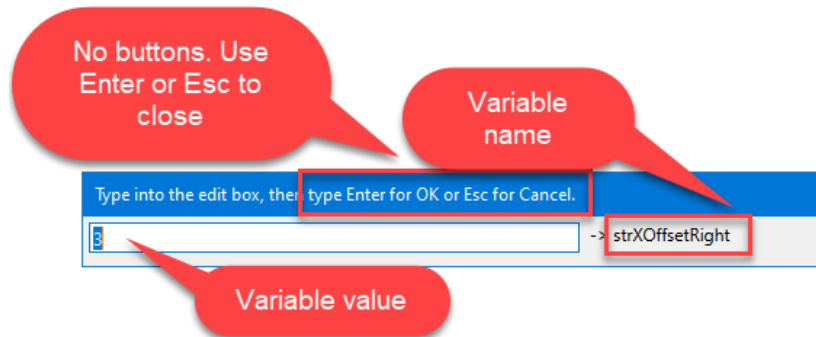
This mechanism gives you the ability to create a macro or plugin that will accept small amounts of text input without needing to call a plugin with a big dialog.

An Example of a parameter variable

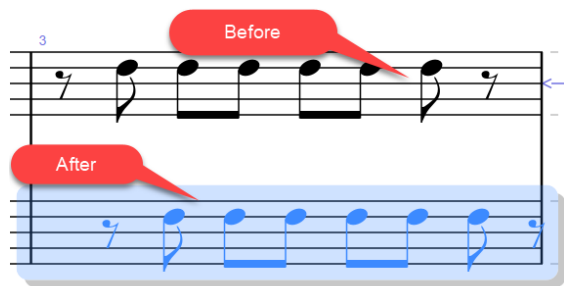
You could write something like this, to get an offset value to use in the command **SetXOffsets_Right_Relative_cu**:

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

You would see 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 offset (the message box could be optional if this was for your own use only). If you wanted the same offset all the time for some cases, you could have separate macros/plugins for that.

You could write a separate macro for Left offsets, and for Y offsets as well. Entering a color name for **ApplyNamedColor_cu** would be another good use for this.

“Full” methods in cmdutils.plg (not available in Execute Commands)

These are called by the **_cu** routines and can be called directly by Manuscript plugins, but may not be called in Execute Commands or as part of a macro.

Many, but not all these methods are described in the previous **cmdutils commands** section.

In this document, these routines are displayed in *Italic*.

Something like **Add_Text_Full (score, selection, strText, styleTextOrId, fSelectNewObject)** can be used within a Manuscript plugin to create more types of text. You could write a plugin that has different styles and puts up a user input edit box, and then use the resulting plugin in **Execute Commands**.

AddInterval_Full (score, iQuality, degreeOrSemitones, fUpward, strQuality)

Add_Line_Full (score, selection, styleTextOrId, fSelectNewObject)

Add_Symbol_Full (score, selection, nameOrIndexSymbol, fUseSystemStaff, fSelectNewObject)

Add_Text_Full (score, selection, strText, styleTextOrId, fSelectNewObject)

ApplyNamedColor_Full (score, selection, fApplyColor, fColorNameFromList, strNamedColor, fAlphaChannel, strAlpha, fTraceResults)

ApplyNoteheadStyle_Full (score, selection, strIdNote)

BracketText_Full (score, selection, strOpenBracket, strCloseBracket)

ContinueIfSelection_Empty_Full (score, selection, strMsgYesNoContinue, fIncludeSystemStaff, fNoteRestRequired)

ContractSelection_Full (score, selection, fLeft)

ContractSelection_UpDown_Full (score, selection, valDirection, valHidden)

DataForWildcard_Full (score, selection, fSetValue, strParameters, strWildcardName, strText)

DeleteSelection_Full (score, selection, fRestorePassage)

ExitIfPlugin_Unavailable_Full (score, strPluginMenuName, strMessagePluginUnavailable)

ExitIfSelection_Avoid_Full (score, selection, strMessageIn, arrOptions)

ExitIfSelection_Empty_Full (score, selection, fIncludeSystemStaff, fNoteRestRequired, strMessageIfEmpty)

ExitIfSelection_NotPassage_Full (score, selection, strMessageIfNonPassage)

ExitOrAll_Selection_Empty_Full (score, selection, fIncludeSystemStaff, fNoteRestRequired, strMessageIfEmpty)

ExitOrAll_Selection_NotPassage_Full (score, selection, strMessageIfNonPassage, fIncludeSystemStaff)

ExportPDF_Full (score, selection, iPart)

ExtendSelection_Full (score, selection, fLeft)

ExtendSelection_FullBar_Full (score, selection, fFullLeft, fFullRight)

ExtendSelection_ObjTo_Full (score, selection, fNeedsNotes, strOption)

ExtendSelection_Pages_Full (score, selection)

ExtendSelection_UpDown_Full (score, selection, valDirection, valHidden)

Filter_Duration_Full (score, selection, strCompare, strDuration, strType)

GetFirstOrLastSelectedObjectEachStaffScoreOrder_Full (score, selection, voice, valLimitObjects, fFirst, fVisibleStaffOnly)

GetFirstOrLastSelectedObjectScoreOrder_Full (score, selection, voice, valLimitObjects, fFirst)

GetFirstSelectedNoteRest_Full (score, selection, fNeedsNotes)

GetUserInput_Full (score, selection, strHeading, strVariableName)

HideAllInvisibles_Full (score, selection, valAction, fTrace)

GoToBar_Full (score, selection, barnumInternalFirstIn, valSelectionType)

GoToPage_Full (score, selection, pagenumExternalNew, valSelectionType)

IsEmptySelection_Full (score, selection, fIncludeSystemStaff, fNoteRestRequired)

MagneticLayout_Full (score, selection, valueML)

MakePassageSelection_Full (score, selection)
MakeSystemPassageSelection_Full (score, selection)

Panorama_Full(score, selection, fPanorama)

ReduceSelectionToObject_Full (score, selection, fSelectFirstObject, fSelectPassage)

RunPluginEntry_Full(score, selection, strParameters)
RunPluginHideDialog_Full (score, strPluginMenuNameCmdOrId, strMessagePluginUnavailable)

SelectFirstOrLastSelectedObjectEachStaffScoreOrder_Full (score, selection, voice, valLimitObjects, fFirst, fVisibleStaffOnly)
SelectFirstOrLast_ObjectScoreOrder_Full (score, selection, voice, valLimitObjects, fFirst)

SelectOneObject_Full (score, selection, obj, fSelectPassage)

Select_All_Passage_Full (score, selection, fSystemPassage)
Select_Bars_Full (score, selection, barnumInternalFirstIn, fSelectFirstBarFullyOnly, fForceSystemSelect)

SetPlayOnPass_Full(score, selection, fNoteRestOnly, arrPassOnOff))

SetScoreRedraw_Full (score, fRedraw)

SetTextCase_Full (score, selection, iCase)
SetTextFormat_Full (score, selection, fResetToDefault, fBold, fItalic, fUnderlined, strSubSuperScript)

SetXOffsets_Full (score, selection, xOffsetInSpaces, iActionForLines, fRight, fRelative)
SetYOffsets_Full (score, selection, yOffsetInSpaces, iActionForLines, fUp, fRelative)

SplitBarRestsAtBeat_Full (score, selection, strBeatToSplit, voice, fSplitHiddenBars)
SplitBarRestsForPassage_Full (score, selection, voice, fSplitHiddenBars)

TestSelection_Avoid_Full (score, selection, strMessageIn, arrOptions)
TestSelection_Needs_Full (score, selection, strMessageIn, fRequireOneStaff, fRequireFullSelect, valRequireGrandStaff)

TracePlugins_HideableDialog_Full (fFillGlobalList)
TraceSelection_Full (score, selection)
Trace_LineStyleIdFromName_Full (score, selection, strName, fWarn)
Trace_NoteStyleIndexFromName_Full (score, selection, strName, fWarn)
Trace_Object_Type_Name_StyleOrIndex_Full (score, selection)
Trace_SymbolIndexFromName_Full (score, selection, strName, fWarn)
Trace_TextStyleIdFromName_Full (score, selection, strName, fWarn)

TransposeInterval_Full(score, selection, strQuality, strNumber, fKeepDoubleAccidentals, fTransposeKeys, fDown, fReportError)

ViewInvisibles_Full(name, param)

Classes of Cmdutils Commands

There are several classes of Cmdutils Commands.

Add Object and “Add and Select” Commands

These add objects to the current selection, mimicking the way Commands work. These routines use the current selection of **Sibelius.ActiveScore**, but they all call lower-level routines where the score, selection, and certain more advanced features may be explicitly specified. I have tried to make the top level routines use as few parameters as possible.

Add Object commands typically return the object created.

If the **AddSelect** form is used, the created object will be selected (non-passage selection), but if the add fails the selection will be cleared. You can call **ContinueIfSelection_Empty_cu(strMsgYesNoContinue)**, or **ExitIfSelection_Empty_cu(strMessageIfEmpty)** to exit the plugin if the selection had been cleared, indicating that the add had been unsuccessful.

Using the **AddSelect** forms in combination with **SaveSelection_cu** and **RestoreSelection_cu** can be an effective way to add and manipulate an object, and then continue. Here is an example:

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

Notice that the parameter in **AddSelect_Text_Technique_cu** has been edited from its original placeholder value.

You need to be careful about adding objects into a selection while you are walking through the selection in a plugin’s “for each” loop, since adding or deleting objects will change the selection and disturb the loop.

There are many predefined **Line** and **Text** style commands available in **Execute Commands**.

You can also call **Add_Line_cu(styleTextOrId)**, which adds a line of the requested line style. **styleTextOrId** must be a StyleAsText or StyleId string which is valid in the score, spelled EXACTLY as Manuscript expects it to be. It is very easy to get the **styleTextOrId** wrong, so only use this if you really need the flexibility.

If you need more Text styles, call **Add_Text_cu([styleId],strText)** to add the text.

Originally, there was no way to specify the style id in this command, but you can now specify a Text styleId at the start of the parameter string, followed by a comma. If you do not explicitly set the text style, **Add_Text_cu** will use **Technique** text style or the text style set by **TextStyleDefaultForCommands_cu**. The text style set by **TextStyleDefaultForCommands_cu** will remain active for the remainder of the Sibelius session unless it is changed by another call to **TextStyleDefaultForCommands_cu**.

Text formatting wildcards such as \B\ and \I\ can be included in **strText**.

Identifying objects to be added

You can apply **Notehead Styles** or add symbols in **Execute Commands**, but you need to know the **Style** or other identifier for the object you are creating, and it must be spelled EXACTLY as Manuscript expects it to be.

In general, objects have a language-specific name (often called **StyleAsText**) and a language-independent identifier, often called a **StyleId**. To make macros or plugins as portable as possible, you should use the

StyleId rather than the **StyleAsText**, but either will work in the same language in which the macro or plugin was written. User-defined styles also will not work if that style is not defined in the current score.

If you know the **StyleAsText** name for Lines, Text, Symbols, or Noteheads, you can find the **StyleId** or index for such objects by running one of these Trace commands, editing the command to contain the name, and looking in the trace window.

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.

Now you can edit the Add command to use the language-independent id or index.

Alternatively, you can select some objects that you want the identifier for and call

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.

Add Object Commands

Some of these commands have placeholder parameters, and any parameter can be edited with **Edit Command**. Some can work with the placeholder, but most of the commands in this category will always need to be changed. For example, **Add_Line_cu(styleTextOrId)** must be given a valid line style name or id to replace **styleTextOrId**.

Add_Bars_At_End_cu(1)

Add_Line_cu(styleTextOrId)

- Adds a line of the requested style. styleTexOrId must be a StyleAsText or StyleId valid in the score, spelled EXACTLY as Manuscript expects it to be.

Add_Line_8va_cu()

Add_Line_Box_cu()

Add_Line_Bracket_Vertical_Left_cu()

Add_Line_Bracket_Vertical_Right_cu()

Add_Line_Ending_First_cu()

Add_Line_Ending_Second_cu()

Add_Line_Hairpin_Crescendo_cu()

Add_Line_Hairpin_Diminuendo_cu()

Add_Line_Plain_cu()

Add_Line_Slur_cu()

Add_Line_Trill_cu()

Add_Line_Vertical_cu()

- Adds a line of the specified style to the selection
- Some of the lines, especial vertical lines like brackets and box lines are given slightly different (and better, in my opinion) positions that Sibelius uses in the Lines menu.

Add_Line_Full (score, selection, styleTextOrId, fSelectNewObject)

- Adds a line of the requested style. styleTexOrId must be a StyleAsText or StyleId valid in the score, spelled EXACTLY as Manuscript expects it to be. Can only be called in Manuscript plugins.

Add_StaffSymbol_cu (nameOrIndexSymbol)

Add_SystemSymbol_cu (nameOrIndexSymbol)

Add_Symbol_Full (score, selection, nameOrIndexSymbol, fUseSystemStaff, fSelectNewObject)

- Adds a SymbolItem or SystemSymbolItem to the selection. nameOrIndexSymbol is either a symbol name or an index into the symbol table, spelled EXACTLY as Manuscript expects it to be. Can only be called in Manuscript plugins.

Add_Text_cu([styleId],strText)

- Add a Text object to the selection.
- This is an unusual command that can specify both a style id and the text to be displayed. If the optional Text styleId is present, it must be followed by a comma but no spaces. Any spaces following the comma will be considered to be part of the text to be displayed.
- If the optional styleId is not present, this command will use the text style that was set by running the **TextStyleDefaultForCommands _cu(styleTextOrId)** command, or will add **Technique** text if the text style is not set.

Add_Text_Dynamics_cu (strText)

- Add a dynamics text object with Expression text style and **MusicText** character style, so the text will be correctly formatted.

Add_Text_Expression_cu (strText)

- Add an Expression text object to the selection

Add_Text_Technique_cu (strText)

- Add a Technique text object to the selection

Add_Text_Full (score, selection, strText, styleTextOrId, fSelectNewObject)

- Add a text or system text object (determined by the style id). styleTexOrId must be a StyleAsText or StyleId defined in the score. Can only be called in Manuscript plugins.

ApplyNoteheadStyle_cu (strIdNote)

ApplyNoteheadStyle_Full (score, selection, strIdNote)

- Changes the notehead style of all selected notes. strIdNote is either a numeric index or a valid Note Style Name, spelled EXACTLY as Manuscript expects it to be. Can only be called in Manuscript plugins.

TextStyleDefaultForCommands _cu(styleTextOrId)

- The commands AddSelect_Text_cu and AddSelect_Text_cu can now specify a style id in their parameter list, so this command should no longer be needed.

- Stores the value of `strTextOrId` for the remainder of the Sibelius session so it can be used by **Add_Text_cu** and **AddSelect_Text_cu**
- It is best to always run this immediately before running **Add_Text_cu** or **AddSelect_Text_cu** or any other command that may use this value.

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.

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.

“Add and Select” Commands

These work the same way as the Add commands except that at the end the added object will be the only thing selected. It can be useful to call **SaveSelection_cu** before making such a call, and then after having manipulated the new object you can restore the previous selection using **RestoreSelection_cu**.

For example, you can create a Text object with **Add_Text_Technique_cu (strText)**, but you may want to change some properties that were not set when you created the object. If you wanted the text to be bold, you could write something like this:

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

which adds and selects a **Technique** Text object, makes it bold, and then restores the original selection.

Some of these commands have placeholder parameters, and any parameter can be edited with **Edit Command**. The placeholder parameters for the commands in this category will always need to be changed. For example, **AddSelect_StaffSymbol_cu (nameOrIndexSymbol)** must be given a valid symbol name or number to replace **nameOrIndexSymbol**.

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 ([styleId],strText)

- Add a Text object to the selection.
- This is an unusual command that can specify both a style id and the text to be displayed. If the optional Text styleId is present, it must be followed by a comma but no spaces. Any spaces following the comma will be considered to be part of the text to be displayed.
- If the optional styleId is not present, this command will use the text style that was set by running the **TextStyleDefaultForCommands_cu(styleTextOrId)** command or will add Technique text if the text style is not set.

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

Add Intervals Commands

These commands have placeholder parameters, and any parameter can be edited with **Edit Command**. The placeholder parameters for the commands in this category will often need to be changed. Numbers from 1 (unison) to 8 (octave) are allowed, though each interval type has some restrictions. You cannot add a Perfect 2nd or a Minor 5th, for example.

AddInterval_Down_Augmented_cu(5)
AddInterval_Down_Diatonic_cu(2)
AddInterval_Down_Diminished_cu(5)
AddInterval_Down_DoubleAugmented_cu(2)
AddInterval_Down_DoubleDiminished_cu(2)
AddInterval_Down_Major_cu(3)
AddInterval_Down_Minor_cu(3)
AddInterval_Down_Perfect_cu(8)
AddInterval_Down_Semitones_cu(1)

AddInterval_Up_Augmented_cu(5)
AddInterval_Up_Diatonic_cu(2)
AddInterval_Up_Diminished_cu(5)
AddInterval_Up_DoubleAugmented_cu(4)
AddInterval_Up_DoubleDiminished_cu(4)
AddInterval_Up_Major_cu(3)
AddInterval_Up_Minor_cu(3)
AddInterval_Up_Perfect_cu(8)
AddInterval_Up_Semitones_cu(1)

- These add a specified interval above the top note or below the bottom note of a chord, just as Sibelius does when you type numbers on the numeric keypad. They call into the downloadable plugin **Add Interval**, which must be installed for these to work. It provides more combinations of intervals than Sibelius itself does.
- The command names tell you the type of interval to add, and the parameters are numbers, where 1 is a unison, 2 a second, 3 a third, and so forth, including octaves for number greater than 7.

Transpose Intervals Commands

These commands have placeholder parameters, and any parameter can be edited with **Edit Command**. The placeholder parameters for the commands in this category will often need to be changed. Numbers from 1 (unison) to 8 (octave) are allowed, though each interval type has some restrictions. You cannot add a Perfect 2nd or a Minor 5th, for example.

TransposeInterval_Down_Augmented_cu(5)
TransposeInterval_Down_Diatonic_cu(2)
TransposeInterval_Down_Diminished_cu(5)
TransposeInterval_Down_DoubleAugmented_cu(4)
TransposeInterval_Down_DoubleDiminished_cu(4)
TransposeInterval_Down_Major_cu(3)
TransposeInterval_Down_Minor_cu(3)
TransposeInterval_Down_Perfect_cu(8)

TransposeInterval_Up_Augmented_cu(5)
TransposeInterval_Up_Diatonic_cu(2)
TransposeInterval_Up_Diminished_cu(5)
TransposeInterval_Up_DoubleAugmented_cu(4)
TransposeInterval_Up_DoubleDiminished_cu(4)
TransposeInterval_Up_Major_cu(3)
TransposeInterval_Up_Minor_cu(3)
TransposeInterval_Up_Perfect_cu(8)

- These transpose the notes in the selection. They call into the downloadable plugin **Transpose By Interval**, which must be installed for these to work. It provides more combinations of intervals than Sibelius itself does.
- The command names tell you the type of interval to add, and the parameters are numbers, where 1 is a unison, 2 a second, 3 a third, and so forth, up to 14, which is the limit for **Transpose By Interval**.

Command Selection commands

This is a large group, which could be subdivided into a few subgroups which could be called Select All, Contract Selection, Expand Selection, Go To, Select First Or Last, and Other (including SaveSelection and RestoreSelection). A number of the **Command Exit** commands are also related to selections.

Unlike **ExtendSelection**, there are no **Visible_XHidden routines** for **ContractSelection**. Contracting to the next visible staff will always exclude any hidden staves that were skipped over.

These command have no parameters, except for the “_Full” versions, which are only accessible in Manuscript plugins.

ContractSelection_Left_cu ()

- Move the right end of the selection one note/rest to the left

ContractSelection_Right_cu ()

- Move the left end of the selection one note/rest to the right

ContractSelection_Down_cu()

- Move the top staff in the selection down 1 staff. There is no special handling for hidden staves. Converts non-passage selection to a passage before contracting.

ContractSelection_Up_cu()

- Move the bottom staff in the selection up 1 staff. There is no special handling for hidden staves. Converts non-passage selection to a passage before contracting.

ContractSelection_Visible_Down_cu()

- Move the top staff in the selection down to the next visible staff. Converts non-passage selection to a passage before contracting. Any hidden staves skipped are excluded from the selection, but this will not produce a discontinuous passage selection.

ContractSelection_Visible_Up_cu()

- Move the bottom staff in the selection up to the next visible staff. Converts non-passage selection to a passage before contracting. Any hidden staves skipped are excluded from the selection, but this will not produce a discontinuous passage selection.

-

ContractSelection_Full (score, selection, fLeft)

- Reduce the size of the selection by moving the end left or right. It will convert a non-passage selection to a passage selection. Can only be called in Manuscript plugins.

ContractSelection_UpDown_Full (score, selection, valDirection, valHidden)

- Reduce the size of the selection by moving the top or bottom staff up or down 1 staff. Converts non-passage selection to a passage before contracting. Can only be called in Manuscript plugins.

Copy_cu()

- Duplicates the Sibelius Copy command

Cut_cu()

- Duplicates the Sibelius Cut command

DeleteSelection_cu()

- Deletes all the selected objects, but does not delete the staff even if the entire staff is selected.

ExtendSelection_Left_cu ()

- Move the left end of the selection one note/rest to the left

ExtendSelection_Right_cu ()

- Move the right end of the selection one note/rest to the right

ExtendSelection_Down_cu()

- Move the bottom staff in the selection down 1 staff. Converts non-passage selection to a passage before extending.

ExtendSelection_Up_cu()

- Move the top staff in the selection up 1 staff. Converts non-passage selection to a passage before extending.

ExtendSelection_Visible_Down_cu

- Move the bottom staff in the selection down to the next visible staff. Converts non-passage selection to a passage before extending.

ExtendSelection_Visible_Up_cu

- Move the top staff in the selection up to the next visible staff. Converts non-passage selection to a passage before extending.

ExtendSelection_Visible_XHidden_Down_cu

- Include the next visible staff down in the selection but exclude hidden staves (discontinuous selection). Converts non-passage selection to a passage before extending.

ExtendSelection_Visible_XHidden_Up_cu

- Include the next visible staff up in the selection but exclude hidden staves (discontiguous selection). Converts non-passages selection to a passage before extending.

ExtendSelection_Full (score, selection, fLeft)

- Extend the size of the selection by moving the end left or right. It will convert a non-passages selection to a passage selection. These mimic Selection Commands that are not currently accessible to plugins, Can only be called in Manuscript plugins.

ExtendSelection_UpDown_Full (score, selection, valDirection, valHidden)

- Extend the size of the selection by moving the top or bottom staff up or down. Converts non-passages selection to a passage before extending. Can only be called in Manuscript plugins.

ExtendSelection_ObjTo_Bar_cu()

ExtendSelection_ObjTo_Page_cu()

ExtendSelection_ObjTo_Staff_cu()

ExtendSelection_ObjTo_System_cu()

ExtendSelection_ObjTo_Full(score, selection, fNeedsNotes, strOption) (Can only be called in Manuscript plugins.)

- These make a passage selection starting from the first Note, Rest or BarRest in the current selection. The selection will include the bar, staff, page, or system containing the first selected object.
- These are analogous to clicking in white space of a bar (*_Bar*), double clicking in a bar (*_System*), triple clicking in a staff (*_Staff*).
- ***ExtendSelection_ObjTo_Page_cu*** has no Sibelius analog; it selects the entire page that contains the first selected object. Unlike ***ExtendSelection_Pages_cu*** it only extends from the first selected object, and not from both the first and last object, so it will always select a single page.

ExtendSelection_Pages_cu ()

ExtendSelection_Pages_Full (score, selection)

- Get the first bar number on the page containing the first selected object, and the last bar number on the page containing the last selected object, and make a passage selection between those bars. Can only be called in Manuscript plugins.

ExtendSelection_FullBar_Left_cu()

- Make the first bar in the selection be fully selected

ExtendSelection_FullBar_LeftRight_cu()

- Make the first and last bar in the selection be fully selected

ExtendSelection_FullBar_Right_cu()

- Make the last bar in the selection be fully selected

ExtendSelection_FullBar_Full(score, selection, fFullLeft, fFullRight)

- Make the first and/or last bar in the selection be fully selected

FilterAllSelected_cu ()

- Turns a passage selection into a group of separately selected objects, as a filter would do.

GoToFirstScoreObject_cu()

- Selects the first staff object in the score (not a passage selection), and brings the selection into view.

GoToFirstScoreObject_SystemOK_cu()

- Selects the first staff or system object in the score (not a passage selection), and brings the selection into view.

GoToLastScoreObject_cu()

- Selects the last staff object in the score (not a passage selection), and brings the selection into view.

GoToLastScoreObject_SystemOK_cu()

- Selects the last staff or system object in the score (not a passage selection), and brings the selection into view.

GoToFirstBar_cu()

GoToLastBar_cu()

GoToNextBar_cu()

GoToPreviousBar_cu()

- Makes a system passage selection in the desired bar, and brings the selection into view.

GoToNextPage_cu()

GoToPreviousPage_cu()

- Makes a system passage selection in first bar of the desired page, and brings the selection into view.

MakePassageSelection_cu ()

- Turns a non-passages selection into a passage selection that includes all the previously selected objects, or converts a system selection to a passage selection.

MakeSystemPassageSelection_cu()

- Turns a non-passages selection into a system selection that includes all the previously selected objects, or converts a passage selection to a system selection.

Paste_cu

- Duplicates the Sibelius Paste command

SelectAll_Passage_cu()

- Makes a non-system passage selection of the entire score

SelectAll_Passage_System_cu()

- Makes a system passage selection of the entire score

SelectAll_NonPassage_cu()

- Makes a non-system passage selection of the entire score excluding objects in the system staff

SelectAll_NonPassage_System_cu()

- Makes a non-passage selection of the entire score

SelectAll_Full(score, selection, fSystemPassage)

- Makes a passage selection of the entire score. Can only be called in Manuscript plugins.

SaveSelection_cu ()

- Saves the current selection so it can be restored later. This is most useful for a passage selection.

RestoreSelection_cu ()

- Restores the most recent selection saved by SaveSelection_cu.

Select_First_NoteChordNoRest_cu ()

Select_First_NoteChordNoRest_EachStaff_cu ()

Select_First_NoteChordRest_cu ()

Select_First_NoteChordRest_EachStaff_cu ()

Select_First_Object_cu ()

Select_First_Object_EachStaff_cu ()

- These put the selected objects into chronological score order if needed, then selects the first object in the selection or in each staff in the selection.

Select_Last_NoteChordNoRest_cu ()

Select_Last_NoteChordNoRest_EachStaff_cu ()

Select_Last_NoteChordRest_cu ()

Select_Last_NoteChordRest_EachStaff_cu ()

Select_Last_Object_cu ()

Select_Last_Object_EachStaff_cu ()

- These put the selected objects into chronological score order if needed, then selects the last object in the selection or in each staff in the selection.

SelectFirstOrLast_ObjectOneStaffScoreOrder_Full (score, selection, staffnum, voice, valLimitObjects, fFirst, fVisibleStaffOnly)

SelectFirstOrLast_ObjectScoreOrder_Full (score, selection, voice, valLimitObjects, fFirst)

SelectFirstOrLastSelectedObjectEachStaffScoreOrder_Full(score, selection, voice, valLimitObjects, fFirst, fVisibleStaffOnly)

- These put the selected objects into chronological score order if needed, then select the first or last object in the selection or in each staff in the selection. Can only be called in Manuscript plugins.

TraceSelection_cu ()

TraceSelection_Full (score, selection)

- Write a description of the current selection to the Plugin Trace Window. Can be useful for debugging. Can only be called in Manuscript plugins.

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.

Most of these commands have placeholder parameters, and any parameter can be edited with **Edit Command**. The placeholder parameters for the commands in this category will always need to be changed. The “_Full” versions of these commands are only used in Manuscript plugins, not in Command Macros or Command Plugins. They will appear in *Italic* font style in this document.

ContinuelfSelection_Empty_cu (strMsgYesNoContinue)

ContinueIfSelection_Empty_Full (score, selection, strMsgYesNoContinue, fIncludeSystemStaff, fNoteRestRequired)

- Exits the plugin if there is no selection and the user responds No in the message box. strMsgYesNoContinue is the message the user will see. Can only be called in Manuscript plugins.

ExitIfSelection_Empty_cu (strMessageIfEmpty)**ExitIfSelection_Empty_Full (score, selection, fIncludeSystemStaff, fNoteRestRequired, strMessageIfEmpty)**

- Exits the plugin if there is no selection. Can only be called in Manuscript plugins.

ExitIfSelection_NotPassage_cu (strMessageIfNotPassage)**ExitIfSelection_NotPassage_Full (score, selection, strMessageIfNonPassage)**

- Exits the plugin if there is no passage selection. Can only be called in Manuscript plugins.

ExitOrAll_Selection_Empty_cu(strMessageIfEmpty)**ExitOrAll_Selection_Empty_Full(score, selection, fIncludeSystemStaff, fNoteRestRequired, strMessageIfEmpty)**

- Exits the plugin if there is no selection, or selects the entire score (non-system selection) and continues. Can only be called in Manuscript plugins.

ExitOrAll_Selection_NotPassage_cu(strMessageIfNotPassage)**ExitOrAll_Selection_NotPassage_Full(score, selection, strMessageIfNonPassage, fIncludeSystemStaff)**

- Exits the plugin if there is no passage selection, or selects the entire score (non-system selection) and continues. Can only be called in Manuscript plugins.

ExitPlugin_cu

- Exits the plugin immediately. Can be useful when debugging as a way to run a part of a macro and then stop.

ExitIfPlugin_Unavailable_cu (strPluginMenuName)**ExitIfPlugin_Unavailable_Full (score, strPluginMenuName, strMessagePluginUnavailable)**

- Exits the plugin if a required plugin is not installed. Can only be called in Manuscript plugins.

ExitIfSelection_Avoid_BottomStaff_cu(strMessage)**ExitIfSelection_Avoid_FirstBar_cu(strMessage)****ExitIfSelection_Avoid_LastBar_cu(strMessage)****ExitIfSelection_Avoid_TopStaff_cu(strMessage)****ExitIfSelection_Avoid_GrandStaff_Bottom_cu(strMessage)****ExitIfSelection_Avoid_GrandStaff_Top_cu(strMessage)**

- These will exit the plugin or macro if there is a selection that includes a “forbidden” staff or bar. If the selection is not a passage selection, it will be temporarily converted into a passage selection that includes all the selected objects, and then restored after the tests are complete.

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.)**

- This will exit the plugin or macro if there is a selection that does not contain specific staves in a multi-staff instrument, such as a grand staff. If the selection is not a passage selection, it will be temporarily converted into a passage selection that includes all the selected objects, and then restored after the tests are complete.

ExitIfSelection_Needs_OneStaff_cu(strMessage)

- This will exit the plugin or macro if there is a selection that contains anything other than a single staff. If the selection is not a passage selection, it will be temporarily converted into a passage selection that includes all the selected objects, and then restored after the tests are complete.

Text formatting commands

Except for the “_Full” commands, these commands do not have parameters.

BracketText_cu()

- Puts parentheses () around all selected text objects.
- Requires the **Bracket Text** plugin (version 01.94.00 or later).

SetTextCase_Lower_cu()

SetTextCase_Upper_cu()

SetTextCase_ToggleCase_cu()

SetTextCase_WordInitialUpper_cu()

- Changes case of selected text objects, including text with formatting
- FOR UNACCENTED (ASCII) Upper and Lower case only
 - UNLESS the plugin MakeChangecase is installed, in which case international characters can be translated. MakeChangecase is installed automatically if you install the Change Case plugin.
 - Code was taken from the ChangeCase plugin.
- Text using fonts that do not support upper or lower case may have problems
- Dynamics text is not changed
- Wildcards are not changed

SetTextFormat_BoldItalic_cu()

SetTextFormat_Bold_cu()

SetTextFormat_Italic_cu()

SetTextFormat_Normal_cu()

SetTextFormat_Underlined_cu()

SetTextFormat_Position_Subscript_cu()

SetTextFormat_Position_Superscript_cu()

SetTextFormat_Full(score, selection, fResetToDefault, fBold, fItalic, fUnderlined, strSubSuperScript)

- Sets Text formatting properties on any selected Text, SystemTextItem, or LyricItem objects.
- No way to remove individual properties, but can reset formatting to normal and then add back.
- Can only be called in Manuscript plugins.

View Tab Commands

HideAllInvisibles_Toggle_cu()

- Toggles the state of the Hide All Invisibles button by running the command View ► Invisibles ► Hide All.

HideAllInvisibles_Hide_cu()

- Sets the state of the Hide All Invisibles button to “hide”

HideAllInvisibles_Show_cu()

- Sets the state of the Hide All Invisibles button to “show”

HideAllInvisibles_Store_Current_Setting_cu()

HideAllInvisibles_Restore_Setting_cu()

- These store the current setting of Hide All Invisibles (for the current Sibelius session only), or restore any store setting. These are comparable to Store Current Selection and Restore Selection, and allow you to change the setting as needed and then restore the original setting.

HideAllInvisibles_Current_cu()

- Returns the current state of the Invisibles Hide All Invisibles setting (“show” or “hide”) after the operation is done. This value is only accessible when invoked in a Manuscript plugin, so it is not included in the commands available to Execute Commands.

HideAllInvisibles_Full(score, selection, valAction, fTrace)

- Called by the HideAllInvisibles... methods above. The return value is “show” or “hide”, reflecting the state of the Invisibles Hide All Invisibles setting after the operation is done. Can only be called in Manuscript plugins.

PanoramaOff_cu()

- Sets View Panorama off. Provides non-toggling setting.

PanoramaOn_cu()

- Sets View Panorama on. Provides non-toggling setting.

Panorama_Full(score, selection, fPanorama)

- Called by the Panorama _cu routines. Returns fPanorama. Can only be called in Manuscript plugins.

ViewAnnotations_cu(toggle/on/off)

ViewAttachmentLines_cu(toggle/on/off)

ViewBarNumbers_cu(toggle/on/off)

ViewComments_cu(toggle/on/off)

ViewHandles_cu(toggle/on/off)

ViewHiddenObjects_cu(toggle/on/off)

ViewLayoutMarks_cu(toggle/on/off)

ViewPageMargins_cu(toggle/on/off)

ViewPlaybackLine_cu(toggle/on/off)

ViewReplayMarker_cu(toggle/on/off)

ViewHighlights_cu(toggle/on/off)

- Allows a plugin to set the appropriate **View Invisible** setting to toggle, on, or off. Edit the parameter to “on” or “off”. Leaving it as it is will toggle the setting. The return value for these routines is True or False, reflecting the state of the **Invisibles** setting after the toggle/on/off operation is done.
- If Hide All Invisibles is active, these commands will have no effects and will always return False. Run **HideAllInvisibles_Show_cu()** f

ViewInvisibles_Full(name, param)

- Called by the View... methods above. The return value is True or False, reflecting the state of the Invisibles setting after the toggle/on/off operation is done. Can only be called in Manuscript plugins.

Horizontal (X) and Vertical (Y) Offset commands

These commands have placeholder parameters, and any parameter can be edited with **Edit Command**. The placeholder parameters for the commands in this category will need to be changed to a valid number of spaces, which need not be a whole number. The “_Full” versions of these commands are only used in Manuscript plugins, not in Command Macros or Command Plugins.

SetXOffsets_Left_Absolute_cu (xOffsetInSpaces)

SetXOffsets_Left_Relative_cu (xOffsetInSpaces)

SetXOffsets_Right_Absolute_cu (xOffsetInSpaces)

SetXOffsets_Right_Relative_cu (xOffsetInSpaces)

SetXOffsets_Full (score, selection, xOffsetInSpaces, iActionForLines, fRight, fRelative) (Can only be called in Manuscript plugins.)

- These are similar to using the X offsets in the Inspector on a selection of objects. iActionForLines: 0 = move both ends, 1 = move left end only, 2 = move right end only. The non-full routines will move both ends of a line.
- Relative commands add the offset to the current offset of the selected objects. Absolute commands set the object offsets to that value.
- These expect a positive number. You can put in a negative number and it will make the offset reverse direction
 - Left will go right, right will go left

SetYOffsets_Down_Absolute_cu (yOffsetInSpaces)

SetYOffsets_Down_Relative_cu (yOffsetInSpaces)

SetYOffsets_Up_Absolute_cu (yOffsetInSpaces)

SetYOffsets_Up_Relative_cu (yOffsetInSpaces)

SetYOffsets_Full (score, selection, yOffsetInSpaces, iActionForLines, fRight, fRelative) (Can only be called in Manuscript plugins.)

- These are similar to using the Y offsets in the Inspector on a selection of objects. iActionForLines: 0 = move both ends, 1 = move left end only, 2 = move right end only. The non-full routines will move both ends of a line.
- Relative commands add the offset to the current offset of the selected objects. Absolute commands set the object offsets to that value.
- These expect a positive number. You can put in a negative number and it will make the offset reverse direction
 - Up will go down, down will go up

User Input Commands

You will need to use these in a plugin where you can examine and manipulate the data they return. Other than **MessageBox_cu** they will not be useful in Command macros, since there is no way to access the return values. See the **Exit Plugins** commands for examples of commands that ask for input and respond based on at least Yes/No responses. Here is what the user dialog (as generated by **SetUserInput_Heading_cu** and **GetUserInput_cu**) looks like. The OK/ Cancel buttons are offscreen. The user types, then types Enter to OK or Esc to exit. Pretty compact.



These commands have placeholder parameters, and any parameter can be edited with **Edit Command**. The placeholder parameters for the commands in this category will need to be changed to appropriate text strings.

The “_Full” versions of these commands are only used in Manuscript plugins, not in Command Macros or Command Plugins.

GetUserInput_cu (strVariableName)

- This puts up a small dialog box with an edit box that the user can type into. The text that will appear in the edit box when the dialog is displayed will be what was used the last time, or empty.

GetUserInput_Full (score, selection, strHeading, strVariableName)

- **GetUserInput_Full** lets you specify some text to appear at the top of the dialog. There is not much room for the text, so check before publishing anything to be sure the text will fit.
- **GetUserInput_Full** creates a parameter variable whose name is what is passed in for strVariableName, and whose value is the text entered into the edit box. See the section on parameter variables for details.
- Can only be called in Manuscript plugins.

MessageBox_cu(strMsg);

ok = MessageBoxYesNo_cu(strMsgYesNo); // ok will be True for Yes, False for No

- Puts up a message box with the string displayed.
- Yes/No value can only be seen in plugin code
- See also **ContinuelfSelection_Empty_cu (strMsgYesNoContinue)**

MessageBoxYesNo_Exit_No_cu(strMsgYesNo)

MessageBoxYesNo_Exit_Yes_cu(strMsgYesNo)

- Puts up a message box with the string displayed.
- The No version will exit the plugin if the user replies No.
- The Yes version will exit the plugin if the user replies Yes.

SetUserInput_Heading_cu(strHeading)

- Sets the heading text for the dialog brought up by **GetUserInput_cu**

Mute/Unmute Commands

MuteSelectedNotes_cu ()

UnmuteSelectedNotes_cu ()

MuteAnySelectedObjects_cu()

UnmuteAnySelectedObjects_cu()

MuteOrUnmuteSelectedNotes_cu(1,1,1,1,1,1,1,1)

MuteOrUnmuteAnySelectedObjects_cu(1,1,1,1,1,1,1,1)

SetPlayOnPass_Full(score, selection, fNoteRestOnly, arrPassOnOff))

- Mutes or unmutes selected notes or objects by setting all Play On Pass passes on or off.
- The MuteOrUnmute routines take a parameter string of 8 fields of 0 or 1 separated by commas.
- Can only be called in Manuscript plugins.

“Other” Commands

Most of these commands have placeholder parameters, and any parameter can be edited with **Edit Command**. The placeholder parameters for the commands in this category will often need to be changed. The “_Full” versions of these commands are only used in Manuscript plugins, not in Command Macros or Command Plugins.

ApplyNamedColor_cu(colorName)

- Applies color to selected objects. **colorName** can only be one of these color names (without quotes): "Black", "Blue", "Brown", "Dark blue", "Dark cyan", "Dark green", "Dark magenta", "Dark salmon", "Gray", "Green", "Indigo", "Light blue", "Light green", "Light slate gray", "Medium blue", "Medium green", "Olive", "Orange", "Orange red", "Pink", "Purple", "Red", "Tan", "Violet", "Yellow", or "White".
- The spelling of the colors names must be exact matches, though case differences are ignored.
- To remove coloring, set the color to Black.
- The plugin **Apply Named Colors** (version 01.22.00 or later) must be installed to use this command.

ApplyNamedColor_List_cu()

- Applies color to selected objects, but it will put up a small dialog with a list of color names.
- Choose a color name from the list, the press Enter to accept the color, or Esc to cancel.

ExportPDF_cu()

- Exports the current score or part as PDF

ExportPDF_DateTime_cu()

- Exports the current score or part as PDF and appends the current date and time to the file name.

ExportPDF_FullScore_cu()

- Exports the full score of the current score as PDF

ExportPDF_FullScore_DateTime_cu()

- Exports the full score of the current score as PDF and appends the current date and time to the file name.
-

ExportPDF_Full(score, selection, iPart)

- Saves in the same folder as the score. Can only be called in Manuscript plugins.

Ese_cu

- A synonym for **Sibelius.Execute("cancel_stop_selectnone")**; It cancels a selection, among other things.
- This is no longer available as of May 2022. Call the Sibelius command **"Cancel/Stop/Select None / "cancel_stop_selectnone"** directly.

Filter_Duration_Notes_Equal_cu(e)

Filter_Duration_Notes_Greater_cu(q)

Filter_Duration_Notes_GreaterEqual_cu(x)

Filter_Duration_Notes_Less_cu(h.)

Filter_Duration_Notes_LessEqual_cu(w..)

- These filter notes or chords (not rests) that are of the desired duration. The duration must be specified as
- w = whole, h = half, q = quarter, e = 8th, x = 16th, y = 32nd, optionally followed by 1 to 3 periods (rhythm dots).
- This calls the plugin **Filter Notes By Duration**, which must be installed for these commands to work.

MagneticLayout_Default_cu()

MagneticLayout_Off_cu()

MagneticLayout_On_cu()

MagneticLayout_Full(score, selection, valueML) (Can only be called in Manuscript plugins.)

- Sets Magnetic Layout to On, Off, or Default for all selected objects.
- These are mostly duplicates of the **Sibelius Magnetic Layout** commands: *Turn off Magnetic Layout for item*, *Turn on Magnetic Layout for item*, and *Use default Magnetic Layout settings*..
- You can use **MagneticLayout_Full**, using the special value of -1 for valueML to find the current Magnetic Layout state of the first selected object.
- There is no sure way to **toggle** the settings, because if an object is set to Default, it can be On or Off, depending on the object.
- This will be applied to Notes as well as other objects (as Sibelius does in **Layout>Magnetic Layout**) . The setting has no practical effect on notes, but be aware that it happens. You may want to filter to deselect notes.
- A user could write a plugin to toggle, based on their own needs, for example, setting Off to On, and On or Default to Off, but that is not reliable enough in general for me to provide that function.

RunPluginEntry_cu(strParameters)

RunPluginEntry_Full(score, selection, strParameters) (Can only be called in Manuscript plugins.)

- Runs an entry point within an installed plugin, passing parameters through a dictionary object.
- This is a universal **Child** plugin for any plugin method that is set up to receive text parameters through a Dictionary of named fields.
- strParameters is a special string with comma separated fields. The format is:
 - <plugin command id>,<plugin method name>,strName1, strVal1, strName2, strVal2, ... strNamen, strValn
 - DANGER!! strName and strVal entry may not contain commas. Use Notes and Chords and Rests, not Notes, Chords, and Rests, as I once did. Messes up the parser. There can be no single or double quotes either.
- Here is an example of a call into the API_ProcessSelection method in Filter Notes By Duration:
 - **RunPluginEntry_cu(FilterNotesByDuration.plg, API_ProcessSelection, str_Duration, q, str_Operator, =, str_Type, Notes and chords only)**
- See the document **Parent/Child plugins** for more details.

RunPluginHideDialog_cu (strPluginMenuName)

RunPluginShowDialog_cu (strPluginMenuName)

RunPluginHideDialog_Full (score, strPluginMenuNameCmdOrId, fHideDialog, strMessagePluginUnavailable)

- Runs the named plugin, and requests it to show or not show its dialog
- Only plugins that have been updated to support this feature will hide the dialog
- Unsupported plugins will be run as usual.
- Can only be called in Manuscript plugins.

GetDataForWildcard_cu(strWildcardName)

- Gets the current value stored in the specified Score Info field

SetDataForWildcard_cu(strWildcardName, strText)

- Sets the current value of the specified Score Info field to the text that follows the comma in the parameter list

DataForWildcard_Full (score, selection, fSetValue, strParameters, strWildcardName, strText)

- Called by **GetDataForWildcard** and **GetDataForWildcard**
- **<strText>** is any text that follows the comma. Avoid most punctuation, and never use single quotes, double quotes, or parentheses
- **< strWildcardName>** must be one of these names (the letters can be upper case or lower case). To get the list of acceptable names in real time, run the command with a bogus wildcard name, such as ?, and the error message will show the available names.

"Arranger"
"Artist"
"Composer"
"ComposerDates"
"Copyist"
"Copyright"
"Dedication"
"Lyricist"
"MoreInfo"
"OpusNumber"
"PartName"
"Publisher"
"Subtitle"
"Title"
"YearOfComposition"

TracePlugins_HideableDialog_cu

- Traces the names of any plugins you have installed that support **RunPluginHideDialog_cu**.

Trace_cu(Write to trace window)

- Writes the parameter string to the Plugin Trace Window

SetScoreRedraw_False_cu ()

SetScoreRedraw_True_cu ()

SetScoreRedraw_Full (score, fRedraw).

- If score.Redraw is not set True, any actions taken by a plugin or command that change the score will cause the score to be redrawn in real time, which greatly slows the process. Normally this is handled in plugins .
- Can only be called in Manuscript plugins.

ShowCmdutilsVersion_cu()

- Traces the current version of cmdutils.plg that is being called.

SplitBarRestsAtBeat_cu(2.0)

- Splits selected bar rests at the specific beat position (beat size depends on the time signature). Split at beat 1 to convert a bar rest into one or more normal rests. This will process hidden staves.

SplitBarRestsForPassage_cu()

- Splits bar rests enclosed by a partial-bar passage selection so that any portions of the bar rests in the first or last selected bar that are out of the selection will be split off as “normal” rests. This will process hidden staves.

Alphabetical full list of commands called from Execute Commands (as of March 23, 2024)

This list was created by choosing the **Cmdutils categories** in the plugin **Execute Commands**, and clicking on **Trace**. I then copied the contents of the **Trace** window to a file. These are the commands that can be executed in **Execute Commands**.

The text strings used as arguments in some commands are placeholders, which will often need to be changed to be an appropriate value using Edit Command or a text editor. When these commands are called directly in a **Manuscript** plugin, the text in between () will need to be enclosed in double quotes, and the last closing parenthesis would be followed by a semicolon.

As an example, the command shown as **AddSelect_Line_cu(line.staff.arrow.black.right)** would have to be changed in Manuscript plugin code to be **cmdutils.AddSelect_Line_cu(“line.staff.arrow.black.right”);**

This is done automatically by the **New Plugin** mechanism in **Execute Commands**.

The command definitions in the file **cmdutils.plg** are the final authority on what the parameters should be. This list was correct at the time of writing (March 23, 2024).

Commands in category: *Cmdutils Add Objects*

Add_Bars_At_End_cu(1)

Add_Line_8va_cu()

Add_Line_Box_cu()

Add_Line_cu(line.staff.arrow.black.right)

Add_Line_Bracket_Vertical_Left_cu()

Add_Line_Bracket_Vertical_Right_cu()

Add_Line_Ending_First_cu()

Add_Line_Ending_Second_cu()

Add_Line_Hairpin_Crescendo_cu()

Add_Line_Hairpin_Diminuendo_cu()

Add_Line_Plain_cu()

Add_Line_Slur_cu()

Add_Line_Trill_cu()

Add_Line_Vertical_cu()

Add_StaffSymbol_cu(Choral divide arrow)

Add_SystemSymbol_cu(Coda)

Add_Text_cu(text.staff.expression,legato)

Add_Text_Dynamics_cu(mf)

Add_Text_Expression_cu(pizz)

Add_Text_Technique_cu(legato)

AddSelect_Line_8va_cu()

AddSelect_Line_Box_cu()

AddSelect_Line_cu(line.staff.arrow.black.right)

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(Choral divide arrow)

AddSelect_SystemSymbol_cu(Coda)

AddSelect_Text_cu(text.staff.expression,legato)

AddSelect_Text_Dynamics_cu(mf)

AddSelect_Text_Expression_cu(pizz)

AddSelect_Text_Technique_cu(legato)

ApplyNoteheadStyle_cu(2)

TextStyleDefaultForCommands_cu(text.staff.technique)

Trace_LineStyleIdFromName_cu(Glissando (wavy))

Trace_NoteStyleIndexFromName_cu(Diamond)

Trace_Object_Type_Name_StyleOrIndex_cu()

Trace_SymbolIndexFromName_cu(Mordent)

Trace_TextStyleIdFromName_cu(Technique)

Commands in category: *Cmdutils Add Intervals*

AddInterval_Down_Augmented_cu(5)
AddInterval_Down_Diatonic_cu(2)
AddInterval_Down_Diminished_cu(5)
AddInterval_Down_DoubleAugmented_cu(2)
AddInterval_Down_DoubleDiminished_cu(2)
AddInterval_Down_Major_cu(3)
AddInterval_Down_Minor_cu(3)
AddInterval_Down_Perfect_cu(8)
AddInterval_Down_Semitones_cu(1)

AddInterval_Up_Augmented_cu(5)
AddInterval_Up_Diatonic_cu(2)
AddInterval_Up_Diminished_cu(5)
AddInterval_Up_DoubleAugmented_cu(4)
AddInterval_Up_DoubleDiminished_cu(4)
AddInterval_Up_Major_cu(3)
AddInterval_Up_Minor_cu(3)
AddInterval_Up_Perfect_cu(8)
AddInterval_Up_Semitones_cu(1)

Commands in category: *Cmdutils Selection*

ContractSelection_Down_cu()
ContractSelection_Left_cu()
ContractSelection_Right_cu()
ContractSelection_Up_cu()

ContractSelection_Visible_XHidden_Down_cu()
ContractSelection_Visible_XHidden_Up_cu()

Copy_cu()
Cut_cu()

DeleteSelection_cu()

ExtendSelection_Down_cu()

ExtendSelection_FullBar_Left_cu()
ExtendSelection_FullBar_LeftRight_cu()
ExtendSelection_FullBar_Right_cu()
ExtendSelection_Left_cu()

ExtendSelection_ObjTo_Bar_cu()
ExtendSelection_ObjTo_Page_cu()
ExtendSelection_ObjTo_Staff_cu()
ExtendSelection_ObjTo_System_cu()

ExtendSelection_Pages_cu()
ExtendSelection_Right_cu()
ExtendSelection_Up_cu()
ExtendSelection_Visible_Down_cu()
ExtendSelection_Visible_Up_cu()
ExtendSelection_Visible_XHidden_Down_cu()
ExtendSelection_Visible_XHidden_Up_cu()

FilterAllSelected_cu()

GoToFirstBar_cu()
GoToFirstScoreObject_cu()
GoToFirstScoreObject_SystemOK_cu()
GoToLastBar_cu()
GoToLastScoreObject_cu()
GoToLastScoreObject_SystemOK_cu()

GoToNextBar_cu()
 GoToNextPage_cu()
 GoToPreviousBar_cu()
 GoToPreviousPage_cu()

 MakePassageSelection_cu()
 MakeSystemPassageSelection_cu()

 Paste_cu()

 RestoreSelection_cu()
 SaveSelection_cu()

 Select_All_NonPassage_cu()
 Select_All_NonPassage_System_cu()
 Select_All_Passage_cu()
 Select_All_Passage_System_cu()

 Select_First_NoteChordNoRest_cu()
 Select_First_NoteChordNoRest_EachStaff_cu()
 Select_First_NoteChordRest_cu()
 Select_First_NoteChordRest_EachStaff_cu()
 Select_First_Object_cu()
 Select_First_Object_EachStaff_cu()
 Select_First_Object_SystemOK_cu()

 Select_Last_NoteChordNoRest_cu()
 Select_Last_NoteChordNoRest_EachStaff_cu()
 Select_Last_NoteChordRest_cu()
 Select_Last_NoteChordRest_EachStaff_cu()
 Select_Last_Object_cu()
 Select_Last_Object_EachStaff_cu()
 Select_Last_Object_SystemOK_cu()

 ShiftSelectionNextBar_cu()
 ShiftSelectionPreviousBar_cu()

TraceSelection_cu()

Commands in category: *Cmdutils Exit Plugin*

ContinuelfSelection_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()

Commands in category: *Cmdutils Text Format*

BracketText_cu()

SetTextCase_Lower_cu()

SetTextCase_ToggleCase_cu()

SetTextCase_Upper_cu()

SetTextCase_WordInitialUpper_cu()

SetTextFormat_Bold_cu()

SetTextFormat_BoldItalic_cu()

SetTextFormat_Italic_cu()

SetTextFormat_Normal_cu()

SetTextFormat_Position_Subscript_cu()

SetTextFormat_Position_Superscript_cu()

SetTextFormat_Underlined_cu()

Commands in category: *Cmdutils Transpose Intervals*

TransposeInterval_Down_Augmented_cu(5)

TransposeInterval_Down_Diatonic_cu(2)

TransposeInterval_Down_Diminished_cu(5)

TransposeInterval_Down_DoubleAugmented_cu(4)

TransposeInterval_Down_DoubleDiminished_cu(4)

TransposeInterval_Down_Major_cu(3)

TransposeInterval_Down_Minor_cu(3)

TransposeInterval_Down_Perfect_cu(8)

TransposeInterval_Up_Augmented_cu(5)

TransposeInterval_Up_Diatonic_cu(2)

TransposeInterval_Up_Diminished_cu(5)

TransposeInterval_Up_DoubleAugmented_cu(4)

TransposeInterval_Up_DoubleDiminished_cu(4)

TransposeInterval_Up_Major_cu(3)

TransposeInterval_Up_Minor_cu(3)

TransposeInterval_Up_Perfect_cu(8)

Commands in category: *Cmdutils View*

HideAllInvisibles_Hide_cu()

HideAllInvisibles_Restore_Setting_cu()

HideAllInvisibles_Show_cu()

HideAllInvisibles_Store_Current_Setting_cu()

HideAllInvisibles_Toggle_cu()

PanoramaOff_cu()

PanoramaOn_cu()

ViewAnnotations_cu(toggle/on/off)

ViewAttachmentLines_cu(toggle/on/off)

ViewBarNumbers_cu(toggle/on/off)

ViewComments_cu(toggle/on/off)

ViewHandles_cu(toggle/on/off)

ViewHiddenObjects_cu(toggle/on/off)

ViewLayoutMarks_cu(toggle/on/off)

ViewPageMargins_cu(toggle/on/off)

ViewPlaybackLine_cu(toggle/on/off)

ViewReplayMarker_cu(toggle/on/off)

ViewHighlights_cu(toggle/on/off)

Commands in category: *Cmdutils X Y Offsets*

SetXOffsets_Left_Absolute_cu(1)
SetXOffsets_Left_Relative_cu(1)
SetXOffsets_Right_Absolute_cu(1)
SetXOffsets_Right_Relative_cu(1)

SetYOffsets_Down_Absolute_cu(1)
SetYOffsets_Down_Relative_cu(1)
SetYOffsets_Up_Absolute_cu(1)
SetYOffsets_Up_Relative_cu(1)

Commands in category: *Cmdutils Other*

ApplyNamedColor_cu(Red)
ApplyNamedColor_List_cu()

ExportPDF_cu()
ExportPDF_DateTime_cu()
ExportPDF_FullScore_cu()
ExportPDF_FullScore_DateTime_cu()

Filter_Duration_Notes_Equal_cu(e)
Filter_Duration_Notes_Greater_cu(q)
Filter_Duration_Notes_GreaterEqual_cu(x)
Filter_Duration_Notes_Less_cu(h.)
Filter_Duration_Notes_LessEqual_cu(w..)

GetUserInput_cu(strVariableName)

GetDataForWildcard_cu(strWildcardName)

MagneticLayout_Default_cu()
MagneticLayout_Off_cu()
MagneticLayout_On_cu()

MessageBox_cu(You need to know this)
MessageBoxYesNo_cu(Choose Yes or No)
MessageBoxYesNo_Exit_No_cu(Choose No to exit this plugin or macro)
MessageBoxYesNo_Exit_Yes_cu(Choose Yes to exit this plugin or macro)

MuteAnySelectedObjects_cu()
MuteOrUnmuteAnySelectedObjects_cu(1,1,1,1,1,1,1,1)
MuteOrUnmuteSelectedNotes_cu(1,1,1,1,1,1,1,1)
MuteSelectedNotes_cu()

RunPluginEntry_cu(strParameters)
RunPluginHideDialog_cu(Valid Plugin Name Or Id)
RunPluginShowDialog_cu (Valid Plugin Name Or Id)

SetDataForWildcard_cu(strWildcardName, strText)

SetScoreRedraw_False_cu()
SetScoreRedraw_True_cu()

SetUserInput_Heading_cu(Type into the edit box, then type Enter for OK or Esc for Cancel.)

ShowCmdutilsVersion_cu()

SplitBarRestsAtBeat_cu(2.0)
SplitBarRestsForPassage_cu()

Trace_cu(Write to trace window)
TracePlugins_HideableDial

UnmuteAnySelectedObjects_cu()
UnmuteSelectedNotes_cu()

Making plugins that use a single call to a cmdutils command

You can include calls to **cmdutils** commands anywhere in your plugin or macro code. You can write a plugin that includes only one or more **cmdutils** commands calls, and then use that plugin in your own **Command Plugins** or **Macros** that you create using **Execute Commands**.

The easiest way to do this is to add a **_cu** command to the **Command List** in **Execute Commands** and use **New Plugin...** to generate and install a new plugin file. You will need to close and restart Sibelius before you can use the plugin.

Now you can test the plugin. Make a selection in the score, and use **Command Search** on the Ribbon to find the name of your plugin and run it, or bring up **Execute Commands** and run the plugin from there (use **Execute Current Command**).

Once it works as expected (which often takes several tries), you can use it like any other plugin as a command in **Execute Commands**.

If you edit a generated plugin you will find that all the commands you wrote are in the method **ProcessScore(score)**.

Debugging aids

There are not many, but adding comments to a macro sequence can help you remember why you did things.

You can also use **Edit Command...** to edit a command temporarily and add “//” to the start of the command, which will turn it into a comment. That way you can see what happens when that specific instruction is removed.

I often add an **ExitPlugin_cu** command to my Command List when debugging. Start it at the bottom of the list where it will really have no effect, then slide it up to a place where you want the macro to stop, so you can examine the score at that point. Change its vertical position as you like, and leave it at the bottom, delete it, or comment it out when you are finished debugging.

Trace_cu will write text to the plugin trace window, and you can write information at various points in the sequence to show how far you are getting. **ExitPlugin_cu** can be very useful: Add an **ExitPlugin_cu** command to a sequence, and slide it up or down so that only parts of the plugin run, and you can effectively halt partway through.

You can use **MessageBox_cu** or **MessageBoxYesNo_cu** or other **User Input** commands to send or receive information to or from the user of the macro. You can only access the Yes/No return value in **ManuScript** code.

MessageBoxYesNo_Exit_No_cu and **MessageBoxYesNo_Exit_Yes_cu** can be added for testing. It will allow you to stop the plugin if the user replies either Yes or No.

Sharing Command Macro and Command Plugins Files

Command macros are stored in the **Execute_Commands** subfolder of your default **Scores** folder (as set up in **File>Preferences>Saving and Exporting**, or in one of that folder’s subfolders. Find them there (they are Text files with .dat extensions), and send copies away. The recipient will need to have create an **Execute_Commands** subfolder in their default **Scores** folder, and copy the .dat files to that subfolder.

Each subfolder in **Execute_Commands** is treated as an independent group in **Run Command Macros**, so you can have any number of working sets of macros, each with its own set of shortcuts.

To share a Command Plugin, you will need to find the .plg file in the plugin folder it was saved in, and send it. In my experience if you are emailing such files it is useful to zip the file first, or the email system may reject it. The recipient will need to copy it into an appropriate plugin subfolder, or use the downloadable plugin [Install New Plugin](#) to move it to an appropriate location. They will need to close and restart Sibelius before that plugin can be used, since Sibelius loads all plugins at startup.

Tip: use Command Macro format (.dat files) for your “source code”

I have found that it is often best to export a sequence of commands to a **Command Macro** (*dat*) file even if you are planning to generate a plugin from them. If you want a plugin, you can import the *dat* file and use that to produce a **New Plugin**. You can easily trade *dat* files, and those can be read in without needing to restart. The receiving user can then generate their own plugin if they so desire.

Keeping the data as a **Command Macro** means it is easy to modify it within **Execute Commands**, whereas once you create a plugin, there is no easy way to get that data back into **Execute Commands**. I think of the Command Macro File as the main source file, and the plugin is derived from that.

Once you need to change the plugin code in Manuscript, though, you are out of the realm of **Execute Commands**, and you will need to edit the code in a text editor or the Sibelius plugin editor (I recommend using the Sibelius plugin editor). I am still trying to get useful code that can be made entirely as sequences of commands, and in that case being able to go easily in and out of **Execute Commands** is a good thing.

A sample plugin that uses cmdutils calls: **Add Line And Dynamic Text**

This plugin adds a “p” in a dynamics style , then a hairpin that extends for the duration of the selection, and finally an “f” dynamic text. The biggest problem was trying to figure out a good way to separate the 3 objects so they were close enough and not too far apart. This version was the best compromise I came up with. It uses several of the **cmdutils** routines as well as regular commands and plugins.

This command could be called by another plugin. If this routine were generated in **Execute Commands** by **New Plugin**, this would be the code in the **ProcessScore()** command.

This example was created in **Execute Commands**, and all the lines of code were produced by adding commands to the **Command List**. The command **Add_Text_Dynamics_cu(f)** was edited with Edit Command to change the default text from *mf* to *p* or *f*.

I present it first as just a sequence of commands, as produced in **Execute Commands**, and stored in a **Command Macro** file using **Export List**, then as a plugin with no comments, and then I provide an annotated version of the plugin that explains what is going on at each step.

Command sequence of Add Line And Dynamic Text

Any commands with (Plug-in nnn) or (xxx) or .plg suffixes are plugins. The commands with **_cu** suffixes call commands in **cmdutils.plg**

```
// Add Line And Dynamic Text
// by Bob Zawalich and Ilkay Bora Oder
//
// Add to the selection dynamics text p, then
// a crescendo hairpin, then a dynamics f
//
ExitIfSelection_NotPassage_cu(A passage selection is required. This plugin will now exit.)
SaveSelection_cu()
Add_Line_Hairpin_Crescendo_cu()
// Add first text at line start position and select it
AddSelect_Text_Dynamics_cu(p)
ExitIfSelection_Empty_cu(The Add failed, and the selection was cleared. The plugin will now exit.)
// shift text 1 space to the left
SetXOffsets_Left_Relative_cu(1)
RestoreSelection_cu()
Select_Last_Object_cu()
select_next_object
Add_Text_Dynamics_cu(f)
RestoreSelection_cu()
ExtendSelection_Right_cu()
```

Uncommented ProcessScore() method of Add Line And Dynamic Text plugin

These lines are essentially the previous commands with added syntax so that Manuscript will run the commands. Parameters will be in quotes (single quotes in the file, but double quotes if you look in the Sibelius Plugin editor), and the commands themselves will be arguments to a **cmdutils** or **ExecuteCommands** routine, or for **Sibelius.Execute**.

```
// Add Line And Dynamic Text
// by Bob Zawalich and Ilkay Bora Oder
//
// Add to the selection dynamics text p, then
// a crescendo hairpin, then a dynamics f
//
cmdutils.ExitIfSelection_NotPassage_cu('A passage selection is required. This plugin will now exit.');
```

```
cmdutils.AddSelect_Text_Dynamics_cu('p');
cmdutils.ExitIfSelection_Empty_cu("The Add failed, and the selection was cleared. The plugin will now exit.');"
// shift text 1 space to the left
cmdutils.SetXOffsets_Left_Relative_cu('1');
cmdutils.RestoreSelection_cu();
cmdutils.Select_Last_Object_cu();
Sibelius.Execute("select_next_object"); // Select next object
cmdutils.Add_Text_Dynamics_cu('f');
cmdutils.RestoreSelection_cu();
cmdutils.ExtendSelection_Right_cu();
```

Cmdutils commands that take parameters, such as **ExitIfSelection_NotPassage_cu** and **ExitIfSelection_Empty_cu**, are given default parameter values which will serve as a reasonable placeholder so that the command can be run immediately, even if it adds the wrong text.

My intention is that if you generate a plugin from these commands you will be able to edit the text to your liking in the editor you use for plugins. You can also select the command in the **Command List** in **Execute Commands** and edit it with the **Edit Command** button, which will allow some simple text editing. In this example, I changed the default “*mf*” to “*p*” in **Add_Select_Text_Dynamics**, and changed the warning string for **ExitIfSelection_Empty_cu** using **Edit Command**, so all this code was written in **Execute Commands**.

Several other command lines might have needed editing, but the default parameters were what I wanted already.

Annotated version of Add Line And Dynamic Text's ProcessScore() method

```
// Add to the selection a dynamics text p, then
// a crescendo hairpin, then a dynamics f
```

These comments give an overview of the macro. Comments need to be on separate lines, and start with `//`. Select a command in the Command List, and use Add Comment to insert a comment into the list immediately above the selected command.

The **ExitIfSelection_NotPassage_cu** and **ExitIfSelection_Empty_cu** calls are not strictly required. If you are running this only on your machine and you know what to expect, you can leave them out. But if you are going to give macros or plugins to someone else, some error checking saves your friend inconvenience and frustration, and lessens the chance that you will have to answer questions and debug code that is not working as expected.

```
// We check for a passage selection and exit if there is none. We could probably get away with only checking for any selection, but I
prefer to limit it further.
```

```
cmdutils.ExitIfSelection_NotPassage_cu('A passage selection is required. This plugin will now exit.');
```

```
// We want to be able to reuse the original selection, so save it away here
```

```
cmdutils.SaveSelection_cu();
```

```
// We add the hairpin line to the full selection. The creation might fail and if this were to be a published plugin
// I would probably error check this as well
```

```
cmdutils.Add_Line_Hairpin_Crescendo_cu();
```

```
// The 'p' text goes at the start of where the line went. We both add the text and select it
// so we can manipulate it. If that Add/Select failed, the selection will be cleared, so we can
// use ExitIfSelection_Empty_cu to warn and exit the plugin
```

```
cmdutils.AddSelect_Text_Dynamics_cu('p');
cmdutils.ExitIfSelection_Empty_cu("The Add failed, and the selection was cleared. The plugin will now exit.');
```

// Adjust the new Text object, which is now selected, one space to the left to give the line a bit of room

cmdutils.SetXOffsets_Left_Relative__cu('1');

// Now restore the original (hopefully passage) selection. To be able to select the following

// object, we first call Select_Last_Object_cu() to select the last of the selected

// objects, then we can use Select next Object to get to the Note, Rest ,or Bar Rest that follows the selection.

cmdutils.RestoreSelection_cu();

cmdutils. Select_Last_Object_cu();

Sibelius.Execute('select_next_object');

// add the final text ""f at the location of the following object

cmdutils.Add_Text_Dynamics_cu('f');

// restore the original selection, which is good to do if this will be run in a

// sequence of commands. Finally, extend the selection to the right

// so the *f* is contained in the selection

cmdutils.RestoreSelection_cu();

cmdutils.ExtendSelection_Right_cu();