The “If Condition” Commands in cmdutils and Execute Commands

Bob Zawalich June 15, 2024 updated September 12, 2024

Executive Summary

cmdutils.plg contains several new routines that use a “**condition**” to determine what to do. The conditions available are defined in the new plugin **Evaluate Plugin Condition**, which must be installed for these commands to work.

These are the cmdutils routines that support conditions:

ExitIfConditionFalse_cu(condition,message)
ExitIfConditionFalse_YesNo_cu(condition,message)

ExitIfConditionTrue_cu(condition,message)
ExitIfConditionTrue_YesNo_cu(condition,message)

RunCommand1IfConditionFalseElseCommand2_cu(notes_selected,MessageBox_cu(Command1 was run),MessageBox_cu(Command2 was run),trace_no)

RunCommand1IfConditionTrueElseCommand2_cu(notes_selected,MessageBox_cu(Command1 was run),MessageBox_cu(Command2 was run),trace_no)

RunCommandAndExitIfConditionFalse_cu(notes_selected,MessageBox_cu(Command was run - will exit),trace_no)
RunCommandAndExitIfConditionTrue_cu(notes_selected,MessageBox_cu(Command was run - will exit),trace_no)

RunCommandIfConditionFalse_cu(notes_selected,MessageBox_cu(Command was run),trace_no)
RunCommandIfConditionTrue_cu(notes_selected,MessageBox_cu(Command was run),trace_no)

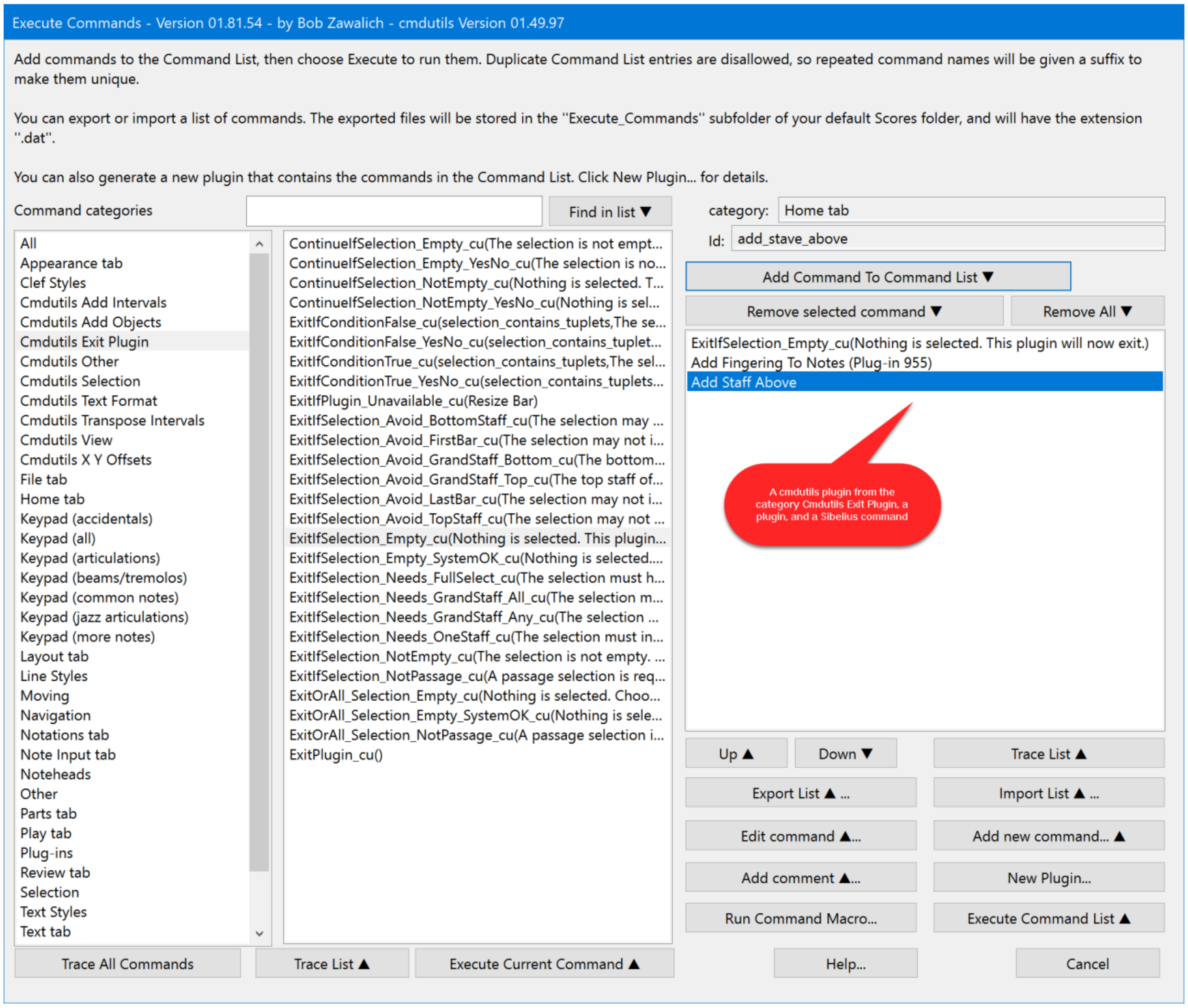
To use these commands, you will need the latest versions of these plugins:

- **Execute Commands**
- **cmdutils**
- **Evaluate Plugin Condition**

Install them (once they are available) using **File>Plug-ins>Install Plug-ins**. You will need to close and restart Sibelius before you use these new plugins, even though you usually do not need to do that when using the installer.

Overview of “If” commands

The plugin **Execute Commands** can run a series of “commands”, which can include Sibelius commands, plugins, and cmdutils commands, as shown below.



Execute Commands - Version 01.81.54 - by Bob Zawalich - cmdutils Version 01.49.97

Add commands to the Command List, then choose Execute to run them. Duplicate Command List entries are disallowed, so repeated command names will be given a suffix to make them unique.

You can export or import a list of commands. The exported files will be stored in the "Execute_Commands" subfolder of your default Scores folder, and will have the extension ".dat".

You can also generate a new plugin that contains the commands in the Command List. Click New Plugin... for details.

Command categories

All
Appearance tab
Clef Styles
Cmdutils Add Intervals
Cmdutils Add Objects
Cmdutils Exit Plugin
Cmdutils Other
Cmdutils Selection
Cmdutils Text Format
Cmdutils Transpose Intervals
Cmdutils View
Cmdutils X Y Offsets
File tab
Home tab
Keypad (accidentals)
Keypad (all)
Keypad (articulations)
Keypad (beams/tremolos)
Keypad (common notes)
Keypad (jazz articulations)
Keypad (more notes)
Layout tab
Line Styles
Moving
Navigation
Notations tab
Note Input tab
Noteheads
Other
Parts tab
Play tab
Plug-ins
Review tab
Selection
Text Styles
Text tab

ContinueIfSelection_Empty_cu(The selection is not empt...
ContinueIfSelection_Empty_YesNo_cu(The selection is no...
ContinueIfSelection_NotEmpty_cu(Nothing is selected. T...
ContinueIfSelection_NotEmpty_YesNo_cu(Nothing is sel...
ExitIfConditionFalse_cu(selection_contains_tuplets,The se...
ExitIfConditionFalse_YesNo_cu(selection_contains_tuplet...
ExitIfConditionTrue_cu(selection_contains_tuplets,The sel...
ExitIfConditionTrue_YesNo_cu(selection_contains_tuplets...
ExitIfPlugin_Unavailable_cu(Resize Bar)
ExitIfSelection_Avoid_BottomStaff_cu(The selection may ...
ExitIfSelection_Avoid_FirstBar_cu(The selection may not i...
ExitIfSelection_Avoid_GrandStaff_Bottom_cu(The bottom...
ExitIfSelection_Avoid_GrandStaff_Top_cu(The top staff of...
ExitIfSelection_Avoid_LastBar_cu(The selection may not i...
ExitIfSelection_Avoid_TopStaff_cu(The selection may not ...
ExitIfSelection_Empty_cu(Nothing is selected. This plugin...
ExitIfSelection_Empty_SystemOK_cu(Nothing is selected....
ExitIfSelection_Needs_FullSelect_cu(The selection must h...
ExitIfSelection_Needs_GrandStaff_All_cu(The selection m...
ExitIfSelection_Needs_GrandStaff_Any_cu(The selection ...
ExitIfSelection_Needs_OneStaff_cu(The selection must in...
ExitIfSelection_NotEmpty_cu(The selection is not empty. ...
ExitIfSelection_NotPassage_cu(A passage selection is req...
ExitOrAll_Selection_Empty_cu(Nothing is selected. Choo...
ExitOrAll_Selection_Empty_SystemOK_cu(Nothing is sele...
ExitOrAll_Selection_NotPassage_cu(A passage selection i...
ExitPlugin_cu()

category: Home tab
Id: add_stave_above

ExitIfSelection_Empty_cu(Nothing is selected. This plugin will now exit.)
Add Fingering To Notes (Plug-in 955)
Add Staff Above

Execute Commands can run commands in a sequence; there is no if-then-else mechanism to let you do different things depending on some condition or property of a selected object.

To provide a bit of control in addition to running commands in sequence, I had created a group of **ExitIf** commands, which cause the sequence of commands to stop if some condition, such as the selection being empty, is satisfied. Here is the annotated original set of **ExitIf** commands:

The Original Exit Plugin 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, or some other criteria. 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.

ContinueIfSelection_Empty_cu(strMsgExit)
ContinueIfSelection_Empty_YesNo_cu(strMsgYesNoContinue)
ContinueIfSelection_NotEmpty_cu(strMsgExit)
ContinueIfSelection_NotEmpty_YesNo_cu(strMsgYesNoContinue)

ContinueIfSelection_Full (score, selection, strMsgYesNoContinue, fIncludeSystemStaff, fNoteRestRequired, fIfEmpty, fYesNo)

- 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_Avoid_Full(score, selection, strMessageIn, arrOptions)

- This is called by the **Avoid** commands and can be called directly by ManuScript plugins.

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.

TestSelection_Needs_Full(score, selection, strMessageIn, fRequireOneStaff, fRequireFullSelect, valRequireGrandStaff)
- This is called by the **Needs** commands and can be called directly by ManuScript plugins.

Description of the ExitIf Commands

The simplest of these is

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

which causes a plugin to immediately stop. This can be useful when you are debugging a sequence of commands (which I will hereafter call a **macro**). You can drag the command into the sequence, and when you run the macro it will stop and the **ExitPlugin_cu** command, and you can look at the score and see if everything looks the way you think it should. If not, you need to figure out why not.

All the other **ExitIf** commands will exit or continue if the current selection meets a specific condition.

ExitIfSelection_Empty_cu (strMessageIfEmpty) is the most likely one of these to be used. Many commands and plugins require a selection, and things can go awry if nothing is selected. A user can edit the message that appears when the plugin decides to exit. Here is the command with its default message that appears when you select the command in Execute Commands:

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

You can change the message by selecting the command in the **Command List** and pressing **Edit Command**. The message should at least explain that the plugin is exiting. This sort of thing is critical if you plan to share your macros or plugins with other users. If they are only for your own use, you can decide for yourself if you need the warning.

The other **ExitIf** messages are essentially the same thing, just checking on different conditions. There are some, like **ExitOrAll_Selection_Empty_cu(strMessageIfEmpty)**, that will put up a message box to ask you if you want to exit if there is no selection, or if you want to select the entire score and then continue.

There is also a small set of routines that will **continue**, rather than **exit**, if the selection is either empty or not empty. These also have a **YesNo** form. Instead of just continuing or exiting, a **YesNo** message box comes up, and the user can decide whether to exit or continue.

ContinueIfSelection_Empty_cu(strMsgExit)
ContinueIfSelection_Empty_YesNo_cu(strMsgYesNoContinue)
ContinueIfSelection_NotEmpty_cu(strMsgExit)
ContinueIfSelection_NotEmpty_YesNo_cu(strMsgYesNoContinue)

The “If Condition” commands

These new commands,

ExitIfConditionFalse_cu(tuplets_selected,The selection does not contain tuplets. This plugin will now exit.)
ExitIfConditionFalse_YesNo_cu(tuplets_selected,The selection does not contain tuplets. This plugin will now exit.)

ExitIfConditionTrue_cu(tuplets_selected,The selection contains tuplets. This plugin will now exit.)
ExitIfConditionTrue_YesNo _cu(tuplets_selected,The selection contains tuplets. This plugin will now exit.)

RunCommand1IfConditionFalseElseCommand2_cu(notes_selected,MessageBox_cu(Command1 was run),MessageBox_cu(Command2 was run),trace_no)

RunCommand1IfConditionTrueElseCommand2_cu(notes_selected,MessageBox_cu(Command1 was run),MessageBox_cu(Command2 was run),trace_no)

RunCommandAndExitIfConditionFalse_cu(notes_selected,MessageBox_cu(Command was run - will exit),trace_no)
RunCommandAndExitIfConditionTrue_cu(notes_selected,MessageBox_cu(Command was run - will exit),trace_no)

RunCommandIfConditionFalse_cu(notes_selected,MessageBox_cu(Command was run),trace_no)
RunCommandIfConditionTrue_cu(notes_selected,MessageBox_cu(Command was run),trace_no)

are different. These commands allow you to specify a **condition name**, chosen from **a limited set of conditions**, described below, and decide to exit or continue based on whether the condition evaluates to True or False, or to run additional commands depending on how the condition evaluates.

In one case you can choose 2 commands: one that will be run when the evaluation succeeds, and another that runs when it fails.

In this example,

ExitIfConditionFalse_cu(**tuplets_selected**,The selection does not contain tuplets. This plugin will now exit.)

The condition is **tuplets_selected**, and **The selection does not contain tuplets. This plugin will now exit.**, separated from the condition by a comma, is a warning message.

If the selection does contain tuplets, the condition will evaluate as True and the plugin will continue. If there are no tuplets, the condition evaluates as False, warning messages will appear, and the plugin will exit.

Why add conditions? A bit of history

I was considering adding a version of the **ExitIf** commands that could run a command and then exit if a "condition" were satisfied. This would be something of the form eventually used by

RunCommandAndExitIfConditionTrue_cu()

However, at the time I did not have separate conditions, and to make this work I would have needed to make *and document* modified copies of around 20 **ExitIf** commands, having a **RunCommandAndExitIf**... equivalent for each of them. I was also considering some variants like **RunCommandIf....** and **RunCommand1If...ElseCommand2...,** each of which would require 20 commands.

This was not practical, so I decided I could make the condition a separate parameter rather than having it be part of the command name. So instead of having, for example, **RunCommandAndExitIfSelectionEmpty_cu(),** I could have **RunCommandAndExitIfConditionTrue_cu(selection_is_empty...),** which takes a condition name as a parameter instead of having the condition be part of the name. The new command could replace all 20 of the commands I would need, as long as I could figure out a way to define "conditions".

I defined the conditions in a separate plugin (**Evaluate Plugin Condition**), which contained a list of available condition names. The cmdutils **IfCondition** commands can call that plugin to evaluate the condition. New conditions could be added to **Evaluate Plugin Condition** (by me) if more conditions were needed in the future, without needing to change **cmdutils**.

I ended up with 10 cmdutils **IfCondition** commands, which is equivalent to 200 commands if I had not separated out the conditions.

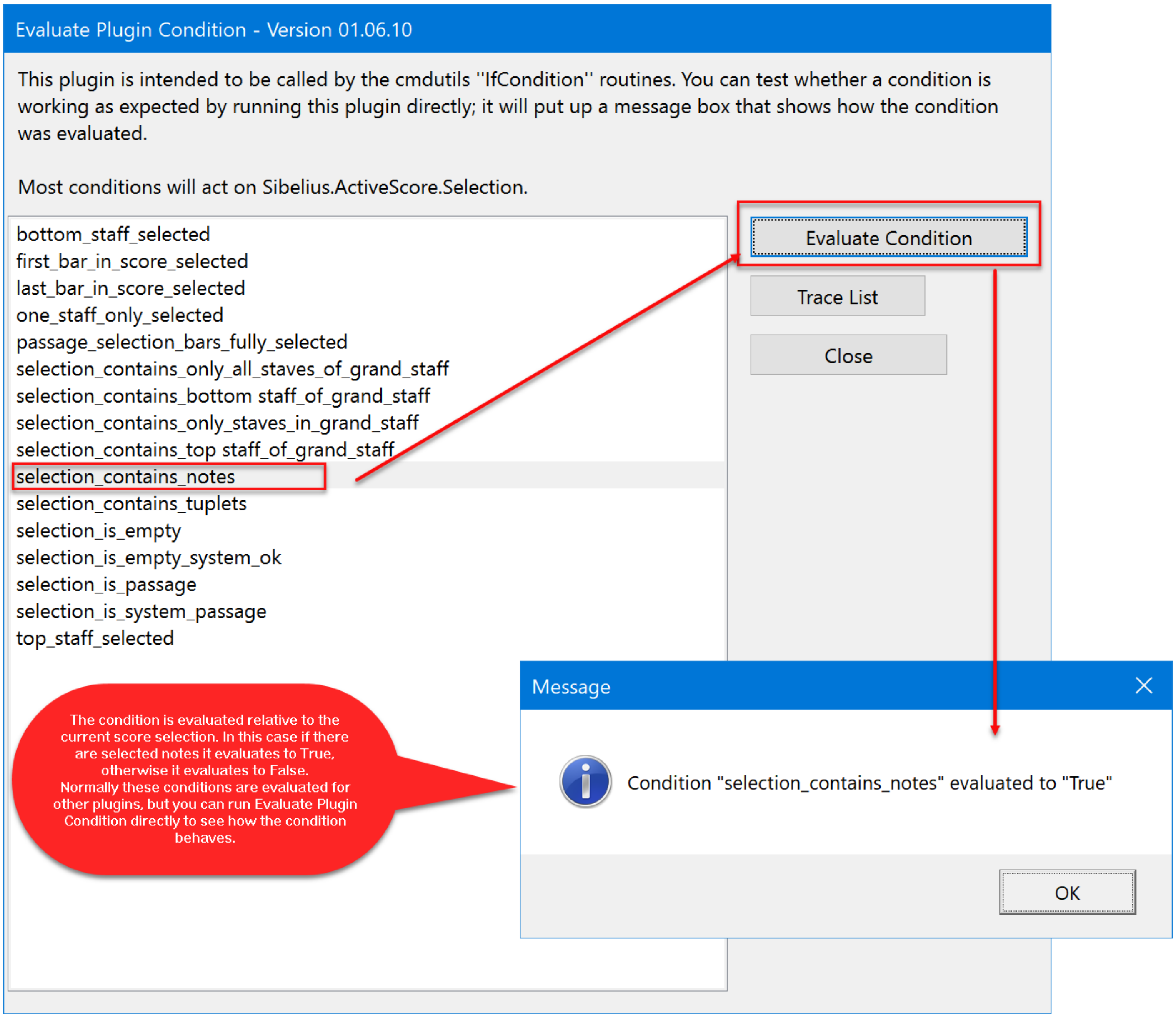
I considered replacing the existing **ExitIf** commands with condition-type commands, but decided not to break existing macros, and that the commands with conditions in the command names were actually easier to work with than the new ones, since you did not need to find the correct condition name.

I created a set of conditions that would be equivalent to nearly all of the existing **ExitIf** commands so the **IfCondition** commands could be used where the **ExitIf** commands had been used. I show a mapping of the **ExitIf** names to conditions in **Appendix 2: Mapping ExitIf commands to use conditions.**

Conditions

The conditions available are defined and displayed in the plugin **Evaluate Plugin Condition**, which must be installed for these commands to work.

Evaluate Plugin Condition may be run directly so you can see which conditions are available or to test out new conditions, but you would usually only run it indirectly from one of the **IfCondition** commands. When running **Evaluate Plugin Condition** directly, pressing **Evaluate condition** will evaluate the condition selected in the list box and tell you if it currently returns True or False, based most often on what is selected in the score.


Evaluate Plugin Condition - Version 01.06.10

This plugin is intended to be called by the cmdutils "IfCondition" routines. You can test whether a condition is working as expected by running this plugin directly; it will put up a message box that shows how the condition was evaluated.

Most conditions will act on Sibelius.ActiveScore.Selection.

bottom_staff_selected
first_bar_in_score_selected
last_bar_in_score_selected
one_staff_only_selected
passage_selection_bars_fully_selected
selection_contains_only_all_staves_of_grand_staff
selection_contains_bottom staff_of_grand_staff
selection_contains_only_staves_in_grand_staff
selection_contains_top staff_of_grand_staff
selection_contains_notes
selection_contains_tuplets
selection_is_empty
selection_is_empty_system_ok
selection_is_passage
selection_is_system_passage
top_staff_selected

The condition is evaluated relative to the current score selection. In this case if there are selected notes it evaluates to True, otherwise it evaluates to False. Normally these conditions are evaluated for other plugins, but you can run Evaluate Plugin Condition directly to see how the condition behaves.

Message

Condition "selection_contains_notes" evaluated to "True"

There will only be one form of a condition in the list. There may be **selection_is_empty**, but not **selection_is_not_empty.** There are True and False versions of all the **IfCondition** commands, so you could use **ExitIfConditionTrue_cu** to exit if **selection_is_empty** is True or **ExitIfConditionFalse_cu** to exit if **selection_is_empty** is False.

Additional conditions could be added to **Evaluate Plugin Condition** by editing the ManuScript code of the plugin. A new condition can be added by

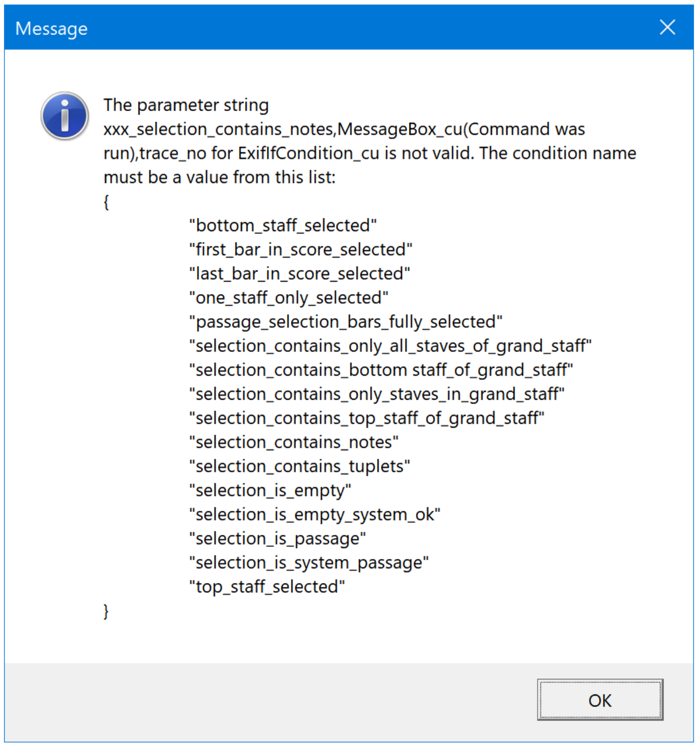
1. Add a new condition name to the array **dlg_lstConditionNames**, preferably in alphabetical order.
2. Add a new case statement to code in **API_TestCondition** to evaluate the new condition. Return 1 if the condition is satisfied, 0 if not.

A few more details and examples can be found in the method **How_To_Add_New_Conditions** in **Evaluate Plugin Condition** , which can be seen by editing **Evaluate Plugin Condition.** Writing a correct condition can be tricky, but that is the only real work you would need to do. The **ManuScript Language Reference**, available at **File>Plug-ins> ManuScript Language Reference**, will be your friend, and you can also peruse the code in other plugins for examples (plugin .plg files are plain text files).

As of August 26, 2024, the available conditions are (“-----” are visual separators in the list)

```
dlg_lstConditionNames
{
"bottom_staff_selected"
"top_staff_selected"
"first_bar_in_score_selected"
"last_bar_in_score_selected"
"one_staff_only_selected"
"-------"
"passage_selection_bars_fully_selected"
"-------"
"selection_contains_only_all_staves_of_grand_staff"
"selection_contains_bottom staff_of_grand_staff"
"selection_contains_only_staves_in_grand_staff"
"selection_contains_top_staff_of_grand_staff"
"-------"
"selection_is_empty"
"selection_is_empty_system_ok"
"selection_is_passage"
"selection_is_system_passage"
"-------"
"notes_selected"
"notes_or_rests_selected"
"bar_rests_selected"
"rests_selected"
"rests_or_bar_rests_selected"
"tuplets_selected"
"tuplets_or_child_notes_selected"
"tuplets_or_child_notes_or_rests_selected"
"tuplets_or_child_rests_selected"
"-------"
"voice_1_objects_selected"
"voice_2_objects_selected"
"voice_3_objects_selected"
"voice_4_objects_selected"
}
```

If a macro or plugin runs an **IfCondition** command with an invalid condition name, the error message will display all the valid condition names.



```
The parameter string
xxx_selection_contains_notes,MessageBox_cu(Command was
run),trace_no for ExifIfCondition_cu is not valid. The condition name
must be a value from this list:
{
        "bottom_staff_selected"
        "first_bar_in_score_selected"
        "last_bar_in_score_selected"
        "one_staff_only_selected"
        "passage_selection_bars_fully_selected"
        "selection_contains_only_all_staves_of_grand_staff"
        "selection_contains_bottom staff_of_grand_staff"
        "selection_contains_only_staves_in_grand_staff"
        "selection_contains_top_staff_of_grand_staff"
        "selection_contains_notes"
        "selection_contains_tuplets"
        "selection_is_empty"
        "selection_is_empty_system_ok"
        "selection_is_passage"
        "selection_is_system_passage"
        "top_staff_selected"
}
```

The “ExitIfCondition” Commands

ExitIfConditionFalse_cu(condition,message)
ExitIfConditionFalse_YesNo_cu(condition,message)

ExitIfConditionTrue_cu(condition,message)
ExitIfConditionTrue_YesNo_cu(condition,message)

These commands evaluate a condition, and based on the results, will make the plugin exit or continue. On exit these will put up a message box.

The “RunCommandIfCondition” Commands

RunCommand1IfConditionFalseElseCommand2_cu(condition,command1,command2,traceNoYes)

RunCommand1IfConditionTrueElseCommand2_cu(condition,command1,command2,traceNoYes)

RunCommandAndExitIfConditionFalse_cu(condition,command,traceNoYes)
RunCommandAndExitIfConditionTrue_cu(condition,command,traceNoYes)

RunCommandIfConditionFalse_cu(condition,command,traceNoYes)
RunCommandIfConditionTrue_cu(condition,command,traceNoYes)

These commands evaluate a condition, and based on the results, will run a command before returning. These commands can be Sibelius commands, cmdutils commands, or plugins. Plugins could be very useful as commands to be run. You can also use the cmdutils routine **RunMacro_cu** to run a macro .dat file.

The condition name must be spelled exactly the same as a condition in the plugin **Evaluate Plugin Condition.** The command name must be a valid command name. I suggest using Execute Commands to enter the command you want to run into the Command List following the RunCommand… command, then cut the command name, use **Edit Command** to edit the RunCommand… command, and replace the placeholder command name with the one you had cut, which will now be in the clipboard.

In the “placeholder parameters” for these commands, the condition is **“notes_selected”** and the command to be run is **“MessageBox_cu(Command was run)”**. These placeholders will do no harm if a user forgets to update the parameters.

The parameter **traceNoYes** can only be one of the 2 strings (not in quotes): **trace_no** (the default) or **trace_yes**. Use **trace_yes** if you are debugging the command and want to see some traced debug data. In a finalized command, this parameter should always be **trace_no**.

When you set up one of these commands you need to spell the condition and any command names exactly, or the command will return an error message and cause the running plugin or macro to exit.

The command that is run can be one that will cause the plugin to exit. In the simplest case, you can run the command **ExitPlugin_cu()**. In a more complex case, you can create a plugin that includes either

```
ExitPlugin();
```

or

```
cmdutils.ExitPlugin_cu();
```

either of these should cause the entire plugin to exit. You can instead use one of the commands

RunCommandAndExitIfConditionFalse_cu(notes_selected,MessageBox_cu(Command was run - will exit),trace_no)
RunCommandAndExitIfConditionTrue_cu(notes_selected,MessageBox_cu(Command was run - will exit),trace_no)

which will run a command and then exit. That way you don’t need to write a new plugin to force the plugin to exit but can run an ordinary command and let the **RunCommandAndExit** command handle the exit. An example of this could be if you wanted to clear the selection before exiting if a condition fails.

RunCommandIfConditionFalse_cu(condition,command,traceNoYes)
RunCommandIfConditionTrue_cu(condition,command,traceNoYes)

- runs the specified command if the condition evaluates to False or True, then allows the plugin or macro to continue.

RunCommandAndExitIfConditionFalse_cu(condition,command,traceNoYes)
RunCommandAndExitIfConditionTrue_cu(condition,command,traceNoYes)

- runs the specified command if the condition evaluates to False or True, then exits the plugin or macro.

RunCommand1IfConditionFalseElseCommand2_cu(condition,command1,command2,traceNoYes)
RunCommand1IfConditionTrueElseCommand2_cu(condition,command1,command2,traceNoYes)

- These supply 2 commands. Command1 will run if the condition evaluates as desired (True for the True form, False for the False form). Command 2 will run otherwise. This is a crude form of an if-then-else mechanism, which could provide much more control over a plugin sequence than was previously possible.

RunCommandIf Examples

RunCommandIfConditionTrue_cu(selection_is_empty,MessageBox_cu(Nothing is selected),trace_no)

If the selection is empty, the command **MessageBox_cu** will be run, which will put up a warning message. The plugin will then continue, and not find anything to process.

RunCommandAndExitIfConditionTrue_cu(notes_selected, select_none,trace_no)

If the selection contains notes, the command **select_none** will be run, which will clear the selection. The plugin will then exit. If no notes had been selected, the plugin continues.

RunCommand1IfConditionFalseElseCommand2_cu(tuplets_selected,PluginNoTuplets.plg,PluginTuplets.plg,trace_no)

Here the commands called are 2 plugins, and this command is acting as a switcher. If the selection does **not** contain tuplets (the **False** command form is being used), the command **PluginNoTuplets.plg** will be run. If tuplets **had** been selected, the command **PluginTuplets.plg** will be run. In either case, the original plugin will continue, unless one of the called plugins calls **ExitPlugin()**. I would expect that if you were using this command as a switcher then it would be the last command in the calling plugin or macro, so it would terminate immediately when control returned to it.

In some cases, you will need to write a custom plugin to serve as the command that is to be called, because the existing commands may not be able to do enough. But I think there can be interesting possibilities here.

To use these commands, you will need the latest versions of these plugins:

- **Execute Commands**
- **cmdutils**
- **Evaluate Plugin Condition**

Install them (once available) using **File>Plug-ins>Install Plug-ins**. You will need to close and restart Sibelius before you use these new plugins, even though you usually do not need to do that when using the installer.

Appendix 1: examples of the ManuScript code used to evaluate conditions (TECHNICAL)

In case you are interested in adding additional conditions to the plugin (for your own use)

Each condition in **Evaluate Plugin Conditions** is evaluated by adding a **case** statement to the **switch** statement in the method **API_TestCondition** in the plugin **Evaluate Plugin Condition.**

The name used in the case() statememt must match exactly an entry in the array **dlg_lstConditionNames.** The names are English-only.

The case statement should return 1 if it evaluates to True, and 0 (zero) if evaluating to False.

Here is a simple example that **checks whether the selection is a passage selection**

```
    case ("selection_is_passage")
    {
        if (selection.IsPassage)
        {
            return 1;  // condition met
        }

        return 0;
    }
```

Most of the other conditions in **API_TestCondition** reproduce the conditions implicitly used in the cmdutils **ExitIf** commands, and they call code in cmdutils to do the work. Here is code for the condition "one_staff_only_selected".

```
    case ("one_staff_only_selected")
    {
        fRequireOneStaff = True;
        fRequireFullSelect = False;
        valRequireGrandStaff = 0; // not used
        strMessageIn = "not_used";

        //TestSelection... will convert to a passage selection temporarily if we start with a non-passage selection

        strMessageOut = cmdutils.TestSelection_Needs_Full(score, selection, strMessageIn, fRequireOneStaff, fRequireFullSelect,
valRequireGrandStaff);
        if (strMessageOut = "")
        {
            return 1;
        }
        return 0;
    }
```

There will certainly be simpler ways to implement these conditions, but in this case I wanted these conditions to match the results of the comparable cmdutils **ExitIf** routine - in this example, **ExitIfSelection_Needs_OneStaff_cu**, whose code is pretty much exactly the same as this **case** statement. If you are writing your own conditions, you will not be restrained in the same way, and you can write any code you want, as long as it returns 1 for True, and zero for False.

You can also call into the cmdutils routines if you like, but it can be a bit challenging.

Another source for condition-like code in Filtering plugins, especially something like **Filter With Deselect**, which has lots of filters. In **Filter With Deselect** , **API_ProcessObjects** is the main object processing loop, and for each selected object it calls **IsDesiredObject**, where the analogue of the condition is defined. You can find a lot of good models for new conditions in that code.

Appendix 2: Exactly what the conditions test

Here is the actual code in the plugin **Evaluate Plugin Conditions** that evaluates conditions as of this writing (September 12, 2024)

This is described somewhat in the previous appendix, but here is the main Switch statement in API_TestCondition.

A number of conditions, such as "bottom_staff_selected", mimic what non-condition cmdutils commands do, and these end up calling code in the cmdutils plugin. Some, like “tuplets_selected”, arer straightforward ManuScript code, as:

```
    case ("tuplets_selected")
    {
        for each Tuplet tup in selection
        {
            return 1;  // condition met
        }

        return 0;
    }
```

If you really need to find exactly what the condition looks at edit the plugin **Evaluate Plugin Conditions,** look in the routine **API_TestCondition,** and if the code in the **switch** statement calls other routines, find those routines and analyze them until you get to the lowest level. All the plugins that are called can be edited using **File>Plug-ins>Edit Plug-ins**. Just be careful not to change anything unless you are confident that you know what you are changing.

That said, here is the essential conditions code:

```
switch (strConditionName)
{
    case ("bottom_staff_selected")
    {
        arrOptions[0] = 1; // fTopStaffSelected
        arrOptions[1] = 0; // fBottomStaffSelected
        arrOptions[2] = 0; // fFirstBarSelected
        arrOptions[3] = 0; // fLastBarSelected
        arrOptions[4] = 0; // fGrandStaffTopSelected;
        arrOptions[5] = 0; // fGrandStaffBottomSelected

        valRet = TestObjectsSelected(score, selection, arrOptions);
        return valRet;
    }
    case ("first_bar_in_score_selected")
    {
        arrOptions[0] = 0; // fTopStaffSelected
        arrOptions[1] = 0; // fBottomStaffSelected
        arrOptions[2] = 1; // fFirstBarSelected
        arrOptions[3] = 0; // fLastBarSelected
        arrOptions[4] = 0; // fGrandStaffTopSelected;
        arrOptions[5] = 0; // fGrandStaffBottomSelected

        valRet = TestObjectsSelected(score, selection, arrOptions);
        return valRet;
    }
    case ("last_bar_in_score_selected")
    {
        arrOptions[0] = 0; // fTopStaffSelected
        arrOptions[1] = 0; // fBottomStaffSelected
        arrOptions[2] = 0; // fFirstBarSelected
        arrOptions[3] = 1; // fLastBarSelected
        arrOptions[4] = 0; // fGrandStaffTopSelected;
        arrOptions[5] = 0; // fGrandStaffBottomSelected

        valRet = TestObjectsSelected(score, selection, arrOptions);
        return valRet;
    }
    case ("one_staff_only_selected")
    {
        fRequireOneStaff = True;
        fRequireFullSelect = False;
        valRequireGrandStaff = 0; // not used
        strMessageIn = "not_used";

        //TestSelection... will convert to a passage selection temporarily if we start with a non-passage selection

        strMessageOut = cmdutils.TestSelection_Needs_Full(score, selection, strMessageIn, fRequireOneStaff, fRequireFullSelect,
valRequireGrandStaff);
        if (strMessageOut = "")
        {
            return 1;
        }
        return 0;
    }
    case ("passage_selection_bars_fully_selected")
    {
        fRequireOneStaff = False;
        fRequireFullSelect = True;
        valRequireGrandStaff = 0;   // not used
        strMessageIn = "not_used";

        strMessageOut = cmdutils.TestSelection_Needs_Full(score, selection, strMessageIn, fRequireOneStaff, fRequireFullSelect,
valRequireGrandStaff);
        if (strMessageOut = "")
        {
            return 1;
        }
        return 0;
    }
    case ("selection_contains_only_staves_in_grand_staff")
    {
        fRequireOneStaff = False;
        fRequireFullSelect = False;
        valRequireGrandStaff = 3;  //only staves from a single Grand Staff
        strMessageIn = "not_used";

        strMessageOut = cmdutils.TestSelection_Needs_Full(score, selection, strMessageIn, fRequireOneStaff, fRequireFullSelect,
valRequireGrandStaff);
        if (strMessageOut = "")
        {
            return 1;
        }
        return 0;
    }
    case ("selection_contains_only_all_staves_of_grand_staff")
    {
        fRequireOneStaff = False;
        fRequireFullSelect = False;
        valRequireGrandStaff = 4;  // all staves required
        strMessageIn = "not_used";

        strMessageOut = cmdutils.TestSelection_Needs_Full(score, selection, strMessageIn, fRequireOneStaff, fRequireFullSelect,
valRequireGrandStaff);
        if (strMessageOut = "")
        {
            return 1;
        }
        return 0;
    }
     case("selection_contains_bottom staff_of_grand_staff")
     {
        arrOptions[0] = 0; // fTopStaffSelected
        arrOptions[1] = 0; // fBottomStaffSelected
        arrOptions[2] = 0; // fFirstBarSelected
        arrOptions[3] = 0; // fLastBarSelected
        arrOptions[4] = 0; // fGrandStaffTopSelected;
        arrOptions[5] = 1; // fGrandStaffBottomSelected

        valRet = TestObjectsSelected(score, selection, arrOptions);
        return valRet;
    }
     case("selection_contains_top_staff_of_grand_staff")
     {
        arrOptions[0] = 0; // fTopStaffSelected
        arrOptions[1] = 0; // fBottomStaffSelected
        arrOptions[2] = 0; // fFirstBarSelected
        arrOptions[3] = 0; // fLastBarSelected
        arrOptions[4] = 1; // fGrandStaffTopSelected;
        arrOptions[5] = 0; // fGrandStaffBottomSelected

        valRet = TestObjectsSelected(score, selection, arrOptions);
        return valRet;
    }
    case ("notes_selected")
    {
        for each Note n in selection
        {
            return 1;  // condition met
        }
        return 0;
    }
    case ("notes_or_rests_selected")
    {
        for each NoteRest nr in selection
        {
            return 1;  // condition met
        }
        return 0;
    }
     case ("bar_rests_selected")
    {
        for each BarRest br in selection
        {
            return 1;  // condition met
        }
        return 0;
    }
    case ("rests_selected")
    {
        for each NoteRest nr in selection
        {
            if (nr.NoteCount = 0)
            {
                return 1;  // condition met
            }
        }
        return 0;
    }
    case ("rests_or_bar_rests_selected")
    {
        for each NoteRest nr in selection
        {
            if (nr.NoteCount = 0)
            {
                return 1;  // condition met
            }
        }
        for each BarRest br in selection
        {
            return 1;  // condition met
        }
        return 0;
    }
    case ("tuplets_selected")
    {
        for each Tuplet tup in selection
        {
            return 1;  // condition met
        }

        return 0;
    }
    case ("tuplets_or_child_notes_or_rests_selected")
    {
        for each Tuplet tup in selection
        {
            return 1;  // condition met
        }
        for each NoteRest nr in selection
        {
            if (nr.ParentTupletIfAny != null)  // note/chord or rest ok
            {
                return 1;  // condition met
            }
        }
        return 0;
    }
    case ("tuplets_or_child_notes_selected")
    {
        for each Tuplet tup in selection
        {
            return 1;  // condition met
        }
        for each NoteRest nr in selection
        {
            if (nr.ParentTupletIfAny != null)
            {
                if (nr.NoteCount != 0) // note or chord but not rest
                {
                    return 1;  // condition met
                }
            }
        }
        return 0;
    }
    case ("tuplets_or_child_rests_selected")
    {
        for each Tuplet tup in selection
        {
            return 1;  // condition met
        }
        for each NoteRest nr in selection
        {
            if (nr.ParentTupletIfAny != null)
            {
                if (nr.NoteCount = 0) // rest not notes or chords
                {
                    return 1;  // condition met
                }
            }
        }
        return 0;
    }
    case ("selection_is_empty")
    {
        fIncludeSystemStaff = False;
        fNoteRestRequired = False;
        val = TrueFalseAsNumber(cmdutils.IsEmptySelection_Full(score, selection, fIncludeSystemStaff, fNoteRestRequired));
        return (val);
    }
    case ("selection_is_empty_system_ok")
    {
        fIncludeSystemStaff = True;
        fNoteRestRequired = False;
        val = TrueFalseAsNumber(cmdutils.IsEmptySelection_Full(score, selection, fIncludeSystemStaff, fNoteRestRequired));
        return (val);
    }
    case ("selection_is_passage")
    {
        if (selection.IsPassage)
        {
            return 1;  // condition met
        }

        return 0;
    }
    case ("selection_is_system_passage")
    {
        if (selection.IsSystemPassage)
        {
            return 1;  // condition met
        }

        return 0;
    }
    case ("top_staff_selected")
    {
        arrOptions = CreateArray();
        arrOptions[0] = 1; // fTopStaffSelected
        arrOptions[1] = 0; // fBottomStaffSelected
        arrOptions[2] = 0; // fFirstBarSelected
        arrOptions[3] = 0; // fLastBarSelected
        arrOptions[4] = 0; // fGrandStaffTopSelected;
        arrOptions[5] = 0; // fGrandStaffBottomSelected

        valRet = TestObjectsSelected(score, selection, arrOptions);
        return valRet;
    }
    case ("voice_1_objects_selected")
    {
        valRet = (VoiceSelected(selection, 1));
        //trace("API_TestCondition Voice1Selected valRet: " & valRet);
        return valRet;
    }
    case ("voice_2_objects_selected")
    {
        return(VoiceSelected(selection, 2));
    }
    case ("voice_3_objects_selected")
    {
        return(VoiceSelected(selection, 3));
    }
    case ("voice_4_objects_selected")
    {
        return(VoiceSelected(selection, 4));
    }
    case ("voice_1_selected")  // the next 4 statements are here only to support macros made before the commands were renamed
    {
        valRet = (VoiceSelected(selection, 1));
        //trace("API_TestCondition Voice1Selected valRet: " & valRet);
        return valRet;
    }
    case ("voice_2_selected")
    {
        return(VoiceSelected(selection, 2));
    }
    case ("voice_3_selected")
    {
        return(VoiceSelected(selection, 3));
    }
    case ("voice_4_selected")
    {
        return(VoiceSelected(selection, 4));
    }
}
```

Appendix 3: Mapping ExitIf commands to use conditions

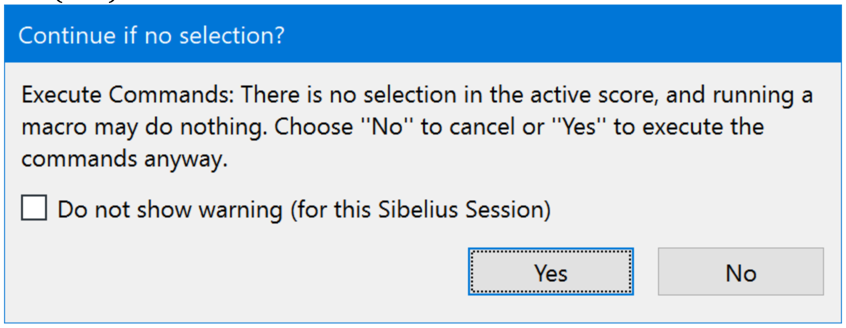
I mentioned above that I set up conditions so they could do the equivalent of the existing ExitIf commands. It is a bit tricky to map the ExitIf name to the appropriate condition and ExitIfConditionTrue/False command. You need to choose both the condition and the appropriate True/False form of the command.

Here is how they are related:

Exit or Continue If Command	**Equivalent using <condition>**
ContinueIfSelection_Empty_cu()	ExitIfConditionFalse_cu(selection_is_empty,<msg>)
ContinueIfSelection_Empty_YesNo_cu()	ExitIfConditionFalse_YesNo_cu(selection_is_empty,<msg>)
ContinueIfSelection_NotEmpty_cu()	ExitIfConditionTrue_cu(selection_is_empty,<msg>)
ContinueIfSelection_NotEmpty_YesNo_cu()	ExitIfConditionTrue_YesNo_cu(selection_is_empty,<msg>)
//ExitIfPlugin_Unavailable_cu()	*No equivalent*
ExitIfSelection_Avoid_BottomStaff_cu\()	ExitIfConditionTrue_cu(bottom_staff_selected,<msg>)
ExitIfSelection_Avoid_FirstBar_cu()	ExitIfConditionTrue_cu(first_bar_in_score_selected,<msg>)
ExitIfSelection_Avoid_GrandStaff_Top_cu()	ExitIfConditionTrue_cu(selection_contains_top staff_of_grand_staff,<msg>)
ExitIfSelection_Avoid_GrandStaff_Bottom_cu()	ExitIfConditionTrue_cu(selection_contains_bottom staff_of_grand_staff,<msg>)
ExitIfSelection_Avoid_LastBar_cu()	ExitIfConditionTrue_cu(last_bar_in_score_selected,<msg>)
ExitIfSelection_Avoid_TopStaff_cu()	ExitIfConditionTrue_cu(top_staff_selected,<msg>)
ExitIfSelection_Empty_cu()	ExitIfConditionTrue_cu(selection_is_empty,<msg>)
ExitIfSelection_Empty_SystemOK_cu()	ExitIfConditionTrue_cu(selection_is_empty_system_ok,<msg>)
ExitIfSelection_Needs_FullSelect_cu()	ExitIfConditionFalse_cu(passage_selection_bars_fully_selected,<msg>)
ExitIfSelection_Needs_GrandStaff_All_cu()	ExitIfConditionFalse_cu(selection_contains_only_all_staves_of_grand_staff,<msg>)
ExitIfSelection_Needs_GrandStaff_Any_cu()	ExitIfConditionFalse_cu(selection_contains_only_staves_in_grand_staff,<msg>)
ExitIfSelection_Needs_OneStaff_cu()	ExitIfConditionFalse_cu(one_staff_only_selected,<msg>)
ExitIfSelection_NotPassage_cu()	ExitIfConditionFalse_cu(selection_is_passage,<msg>)
ExitIfSelection_NotEmpty_cu()	ExitIfConditionFalse_cu(selection_is_empty,<msg>)
ExitOrAll_Selection_Empty_cu()	*No equivalent**
ExitOrAll_Selection_Empty_SystemOK_cu()	*No equivalent**
ExitOrAll_Selection_NotPassage_cu()	*No equivalent**
//ExitPlugin_cu()	*No equivalent*

* These commands can be simulated with a fair bit of work.
ExitOrAll_Selection_Empty_cu(), for example, does this:

- Checks if selection is empty.
    - If not, it returns without exiting
    - If empty
        - Put up a YesNo message box asking the user to exit (No) or to select all and continue (Yes)



Continue if no selection?

Execute Commands: There is no selection in the active score, and running a macro may do nothing. Choose "No" to cancel or "Yes" to execute the commands anyway.

☐ Do not show warning (for this Sibelius Session)

If No, the plugin exits
If Yes, the plugin passage selects (not a system passage selection, just the blue box) the entire score and then continues/does not exit.

One could approach this by using

RunCommandIfConditionTrue_cu(selection_is_empty,<command>,trace_no)

If the selection is empty, it will run the command, if not, it will just continue. What would be needed would be a command that would put up the message box, exit on No, and if Yes, passage select all and continue.

There is no existing command that would do this, so you would likely need to write a small ManuScript plugin to do this. You could use the code in **cmdutils.ExitOrAll_Selection_Empty_cu()** as a model, but it is probably more trouble than it's worth. Just use **ExitOrAll_Selection_Empty_cu()**.

What works well in this situation is to have 2 commands. One will test the condition and put up a message box to provide a warning but will not run a functional command or exit. The next will check the same condition and if True will run a real command and then exit.

This effectively gives the IfCondition commands a mechanism to warn before exiting.

In the following example, if notes are selected, the first command will just put up a message box and stop. The second command will test the same condition and will clear the selection and exit.

If the condition had evaluated to False, neither command would have performed any action.

RunCommandIfConditionTrue_cu(notes_selected,MessageBox_cu(Exiting because selection contains notes),trace_no)

RunCommand*AndExit*IfConditionTrue_cu(notes_selected,Select All,trace_no)

Appendix 4: Equivalent non-condition commands

If there is no available condition for what you want, you can sometimes use **Filters** to accomplish what you need.

This is pretty intense, so feel free to skip it.

You may have been using something like this:

RunCommandAnd ExitIfConditionTrue_cu(tuplets_selected,<command>,trace_no)

Which would run <command> and then **exit if the selection contained tuplet objects.** Say that you wanted to test for tuplet objects or notes/chords that were children of tuplets, and there was no condition that could test this, but there was a filter that would filter for tuplet objects or notes/chords that were children of tuplets.

You could save the selection, then run a filter, and test for whether anything had been selected.

If nothing were selected, i.e., the selection was empty, no such tuplets or notes would be selected, so we would want to restore the original selection and continue. If the selection were not empty, we would want to restore the original selection, run the specified command, and exit.

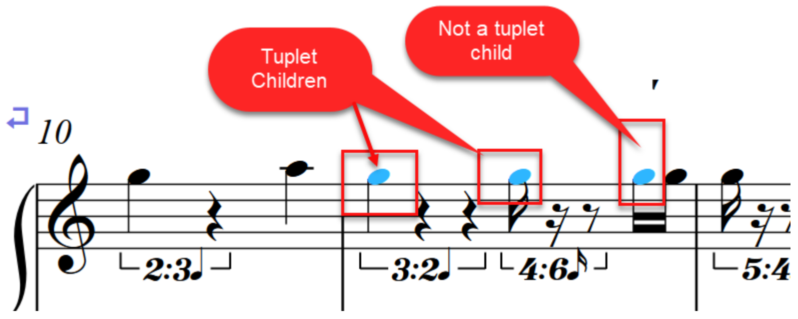
The details of this are tricky, so pause a moment to be sure it makes sense.

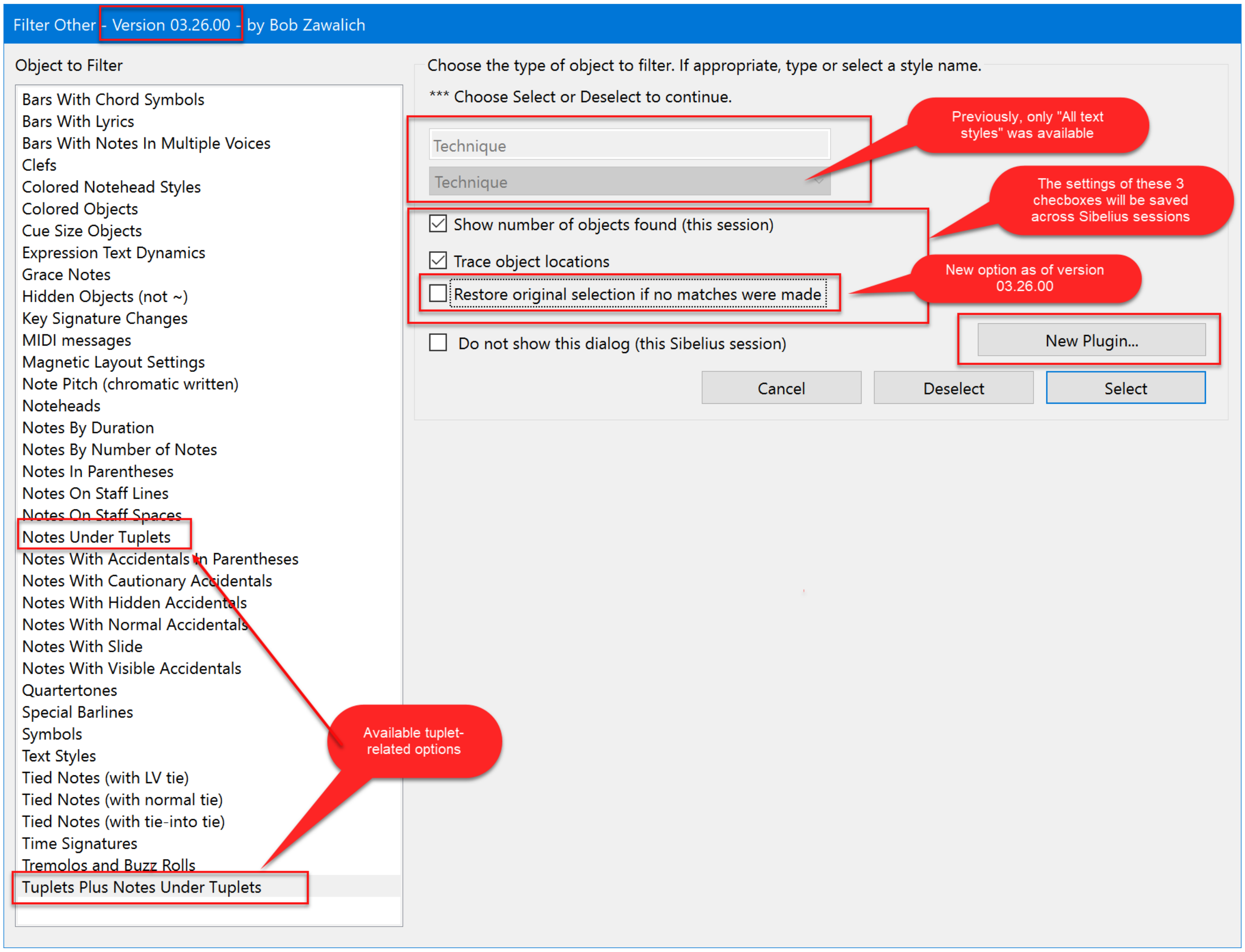
Note that **Saving** and **Restoring** a selection works well for a passage selection but will not restore the selection in a non-passage selection if any of the selected objects were changed or added or deleted. You really need to understand what is happening to the selection to make this work.

In this example, we will not change any selected objects, so it will work even with a non-passage selection.

Let us deal with the worst case and start with a non passage selection. For a filter, we could run any Sibelius filtering command, such as **filter_nonspecial_barlines.** This gives you access to all the filters on **Home>Filters**. You could also run any filtering plugin, ideally one that does not bring up a dialog. For some more complex filters, you could write your own filtering plugin using one of the **Custom Filter** plugins as a template. You may find useful options in the **Filter Other** or **Filter With Deselect** plugins, and for these you could use the **New Plugin…** button to generate a command line for a single filter with no dialog.

In this example we will use the plugin **Filter Other** and have it generate a command line for **Tuplets and Notes Under Tuplets** with the **New Plugin…** button.




Filter Other - Version 03.26.00 - by Bob Zawalich

Object to Filter

- Bars With Chord Symbols
- Bars With Lyrics
- Bars With Notes In Multiple Voices
- Clefs
- Colored Notehead Styles
- Colored Objects
- Cue Size Objects
- Expression Text Dynamics
- Grace Notes
- Hidden Objects (not ~)
- Key Signature Changes
- MIDI messages
- Magnetic Layout Settings
- Note Pitch (chromatic written)
- Noteheads
- Notes By Duration
- Notes By Number of Notes
- Notes In Parentheses
- Notes On Staff Lines
- Notes On Staff Spaces
- Notes Under Tuplets
- Notes With Accidentals in Parentheses
- Notes With Cautionary Accidentals
- Notes With Hidden Accidentals
- Notes With Normal Accidentals
- Notes With Slide
- Notes With Visible Accidentals
- Quartertones
- Special Barlines
- Symbols
- Text Styles
- Tied Notes (with LV tie)
- Tied Notes (with normal tie)
- Tied Notes (with tie-into tie)
- Time Signatures
- Tremolos and Buzz Rolls
- Tuplets Plus Notes Under Tuplets

Choose the type of object to filter. If appropriate, type or select a style name.

*** Choose Select or Deselect to continue.

Technique

Technique

☑ Show number of objects found (this session)

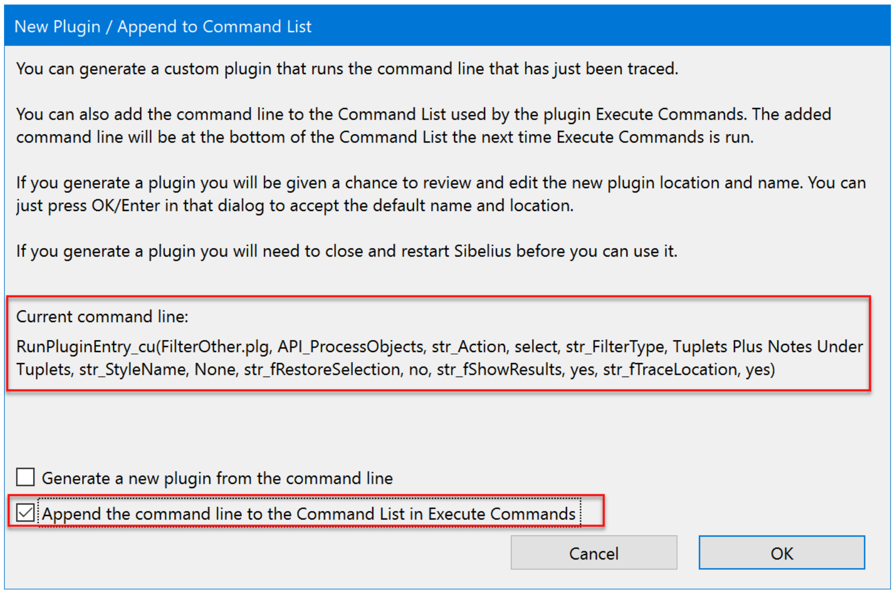
☑ Trace object locations

☐ Restore original selection if no matches were made

☐ Do not show this dialog (this Sibelius session)

New Plugin...
Cancel
Deselect
Select
Previously, only "All text styles" was available
The settings of these 3 checboxes will be saved across Sibelius sessions
New option as of version 03.26.00
Available tuplet-related options

Here is the dialog that appears when you press **New plugin…** and tell it to append the command line to the **Execute Command** Command List:


New Plugin / Append to Command List

You can generate a custom plugin that runs the command line that has just been traced.

You can also add the command line to the Command List used by the plugin Execute Commands. The added command line will be at the bottom of the Command List the next time Execute Commands is run.

If you generate a plugin you will be given a chance to review and edit the new plugin location and name. You can just press OK/Enter in that dialog to accept the default name and location.

If you generate a plugin you will need to close and restart Sibelius before you can use it.

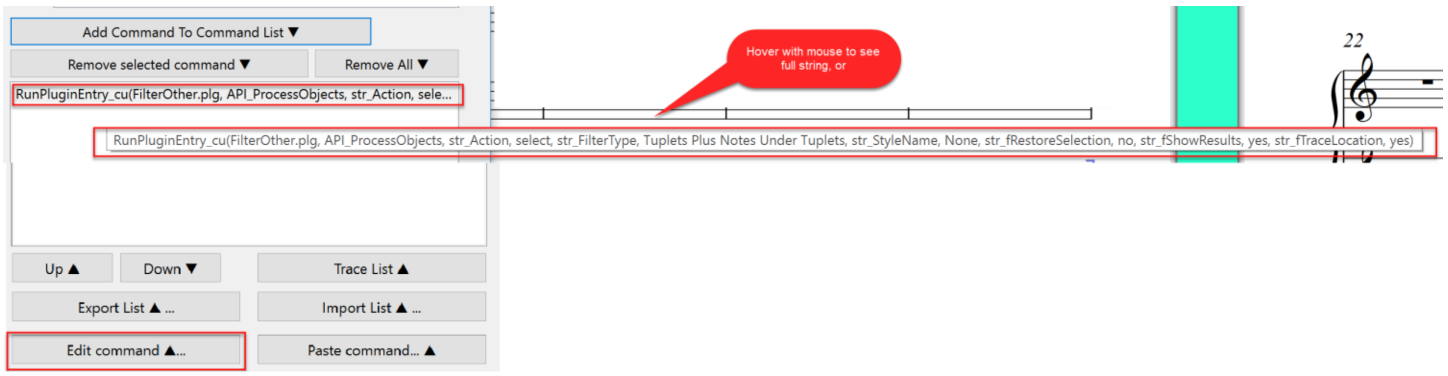
Current command line:

RunPluginEntry_cu(FilterOther.plg, API_ProcessObjects, str_Action, select, str_FilterType, Tuplets Plus Notes Under Tuplets, str_StyleName, None, str_fRestoreSelection, no, str_fShowResults, yes, str_fTraceLocation, yes)

☐ Generate a new plugin from the command line

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

Here is what the Command List in **Execute Commands** would look like.



RunPluginEntry_cu(FilterOther.plg, API_ProcessObjects, str_Action, sele...

RunPluginEntry_cu(FilterOther.plg, API_ProcessObjects, str_Action, select, str_FilterType, Tuplets Plus Notes Under Tuplets, str_StyleName, None, str_fRestoreSelection, no, str_fShowResults, yes, str_fTraceLocation, yes)

Let's pause here for a moment and regroup. Here is the plan: we are simulating the following instruction with a slightly different condition, so we can also process selected notes that are children of tuplets:

RunCommandAnd ExitIfConditionTrue_cu(tuplets_selected,<command>,trace_no)

Here we go.

1. Call **SaveSelection_cu** so we can restore the original selection later
2. Run the generated filter command, which will filter tuplets or notes under a tuplet. This will often change the selection. If no tuplets or tuplet notes were selected, the selection will be empty.
    a. Here is the description of the problem to be solved, from above:

"You could then test whether the selection had been empty or not. If empty, no such tuplets or notes would be selected, so we would want to restore the original selection and continue. If the selection were not empty, we would want to restore the original selection, run the specified command, and exit."

One needs to think carefully about this. The appropriate condition we have is "**selection_is_empty**". If the selection is empty, then no tuplets or notes under tuplets were found. **We want to exit if such objects were found,** so we will use the **False** form of the **RunCommandIfCondition** command

As in the example above, we might want to put up a messages box explaining why we are exiting, so add this command, which tests the same condition as the **Exit** command, but if the condition returns False if will just put up a message box.

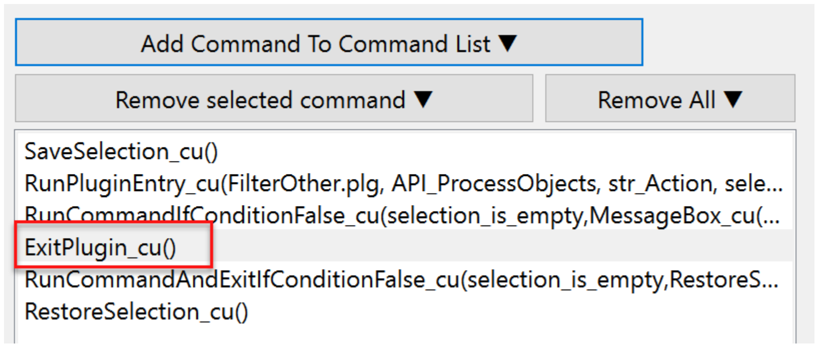
3. **RunCommandIfConditionFalse_cu(selection_is_empty,MessageBox_cu(Exiting because selection contains tuplets or notes inside tuplets),trace_no)**

4. **RunCommandAndExitIfConditionFalse_cu(selection_is_empty,RestoreSelection_cu(),trace_no)**
    a. we will want to restore the selection and exit if the filtered selection is not empty, so we run the instruction above. If we do not exit, we want to restore the selection and continue.
5. **RestoreSelection_cu()**
6. So here is the set of instructions we want:

```
SaveSelection_cu()
RunPluginEntry_cu(FilterOther.plg, API_ProcessObjects, str_Action, select, str_FilterType, Tuplets Plus Notes Under Tuplets,
str_StyleName, None, str_fRestoreSelection, no, str_fShowResults, yes, str_fTraceLocation, yes)
RunCommandIfConditionFalse_cu(selection_is_empty,MessageBox_cu(Exiting because selection contained tuplets or notes inside
tuplets),trace_no)
RunCommandAndExitIfConditionFalse_cu(selection_is_empty,RestoreSelection_cu(),trace_no)
RestoreSelection_cu()
```

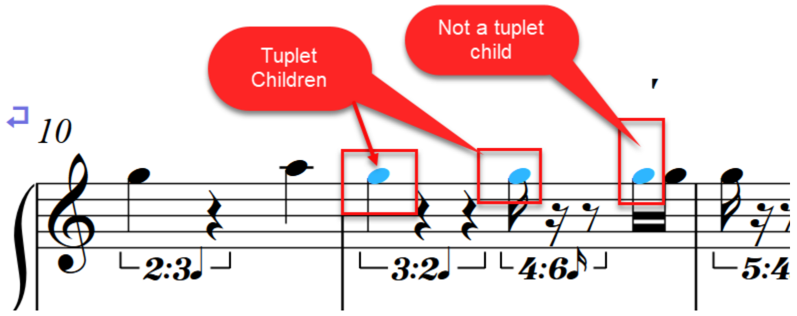
To be sure it is actually working correctly, I often add an ExitPlugin() command into the Command List, and slide it up and down so I can see the results at various points in the command execution.

Often, just because the final result is correct, it does not prove that the processing was correct. This is the life of a programmer…

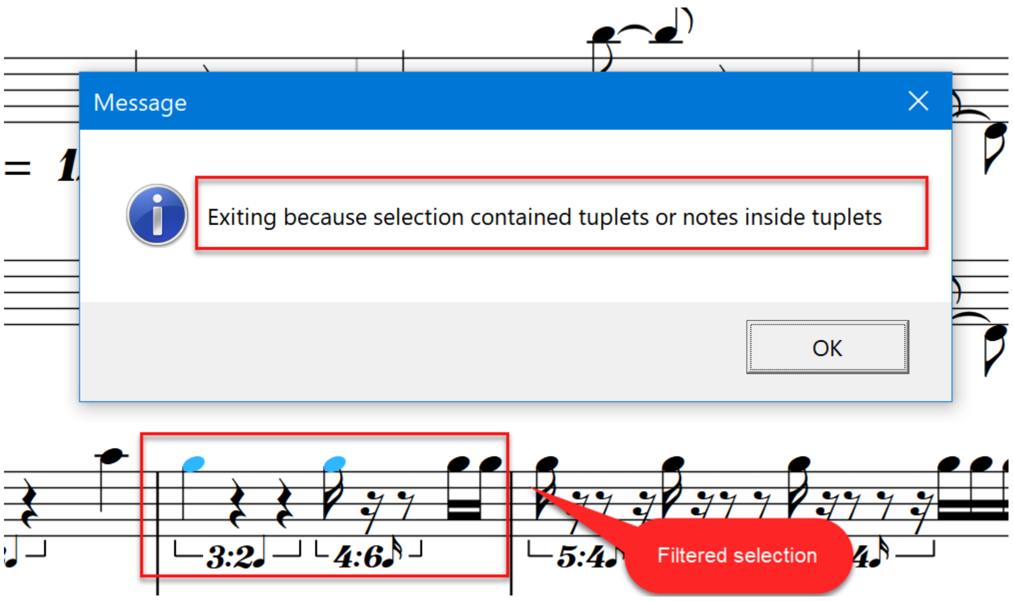


```
SaveSelection_cu()
RunPluginEntry_cu(FilterOther.plg, API_ProcessObjects, str_Action, sele…
RunCommandIfConditionFalse_cu(selection_is_empty,MessageBox_cu(…
ExitPlugin_cu()
RunCommandAndExitIfConditionFalse_cu(selection_is_empty,RestoreS…
RestoreSelection_cu()
```

Here is the original selection again. *Assume the **ExitPlugin_cu()** command is not present.*

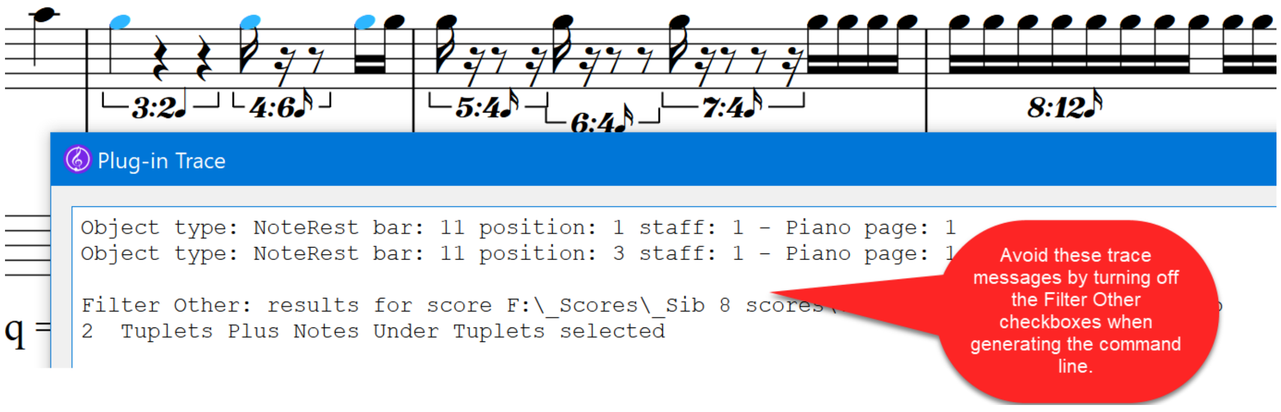


We see this message box after the filter. Note that the selection at this point includes only the notes in the tuplets, as expected.



Exiting because selection contained tuplets or notes inside tuplets

When we OK the message box, the selection should be restored to the original, and it is. Here is the final result. The original selection is restored, as expected.

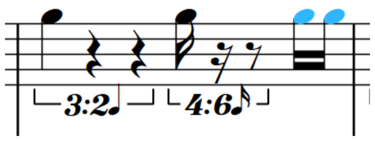


Plug-in Trace

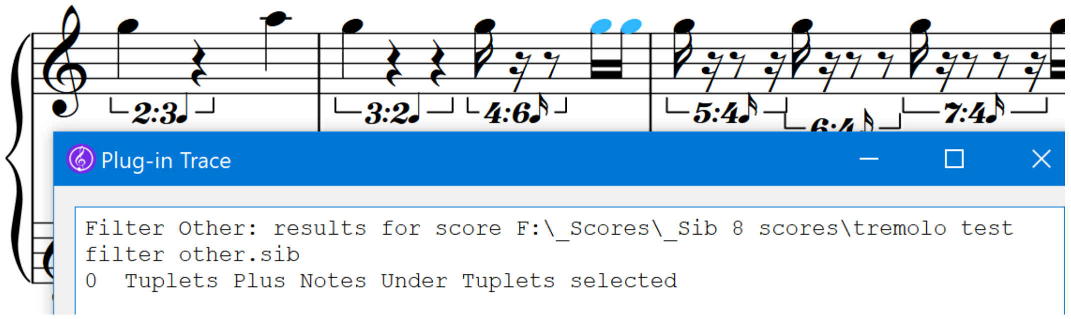
```
Object type: NoteRest bar: 11 position: 1 staff: 1 - Piano page: 1
Object type: NoteRest bar: 11 position: 3 staff: 1 - Piano page: 1

Filter Other: results for score F:\_Scores\_Sib 8 scores\
2  Tuplets Plus Notes Under Tuplets selected
```

Avoid these trace messages by turning off the Filter Other checkboxes when generating the command line.

What if we ran this macro when the selection did not contain any tuplets or children of tuplets, such as this?



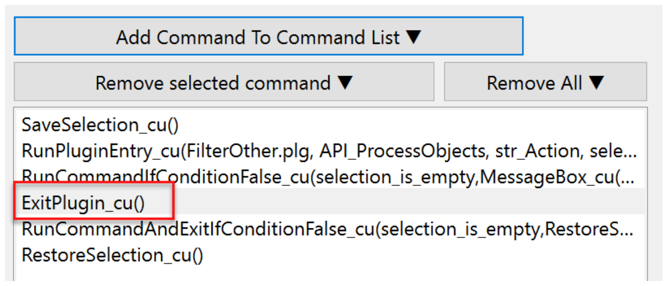
In this case, neither **RunCommandIf** command will run commands or exit. The filter would have left an empty selection, but the last command, **RestoreSelection_cu()** will be run, so the selection appears to be unchanged.



Plug-in Trace

```
Filter Other: results for score F:\_Scores\_Sib 8 scores\tremolo test
filter other.sib
0  Tuplets Plus Notes Under Tuplets selected
```

Since the selection was unchanged, this was a case where I wanted to make sure the score was correct at an intermediate stage, so I added an **ExitPlugin()** command after the filter to show the state of the score after the filter:



```
SaveSelection_cu()
RunPluginEntry_cu(FilterOther.plg, API_ProcessObjects, str_Action, sele...
RunCommandIfConditionFalse_cu(selection_is_empty,MessageBox_cu(...
ExitPlugin_cu()
RunCommandAndExitIfConditionFalse_cu(selection_is_empty,RestoreS...
RestoreSelection_cu()
```

In this example, the selection was empty after the filter, as expected. When I removed the **ExitPlugin_cu()** call, the selection was restored at the end, again as expected.