The “If Condition” Commands in cmdutils and Execute Commands

Bob Zawalich June 15, 2024 updated February 1, 2026

Executive Summary

cmdutils.plg contains several new commands 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(tuplet_objects_selected,The selection does not contain tuplets. This plugin will now exit.)
ExitIfConditionTrue_cu(tuplet_objects_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)
```

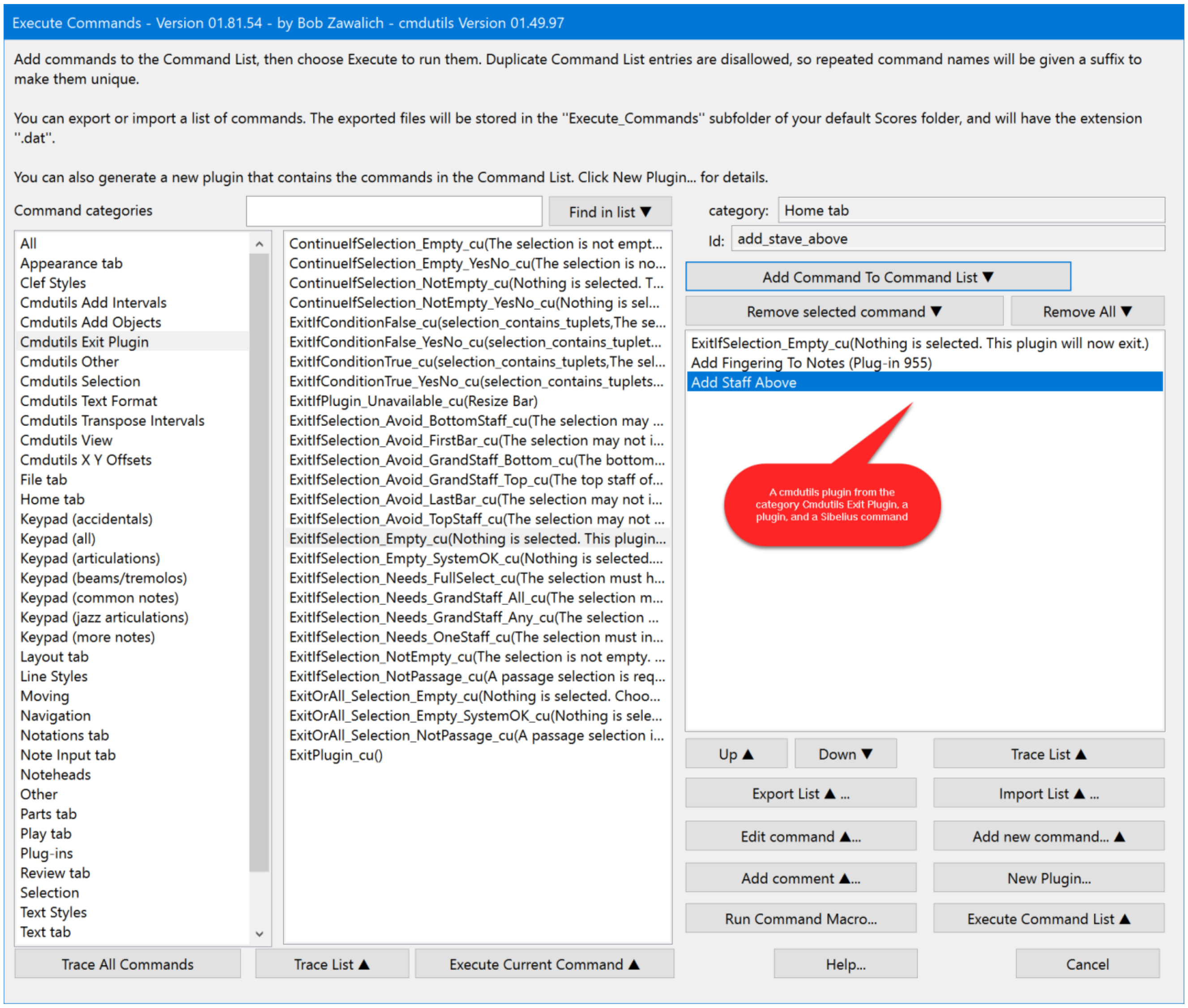
To use these commands, you will need the latest versions of these plugins:

- **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 MessageBoxYesNo_Exit commands

These allow you to put up a message box with a message asking the user if they want to exit or not. This provides some amount of user input that can affect the flow of the macro.

MessageBoxYesNo_Exit_No_cu(Choose No to exit this plugin or macro)
MessageBoxYesNo_Exit_Yes_cu(Choose Yes to exit this plugin or macro)
- 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.

See also the command **GetUserInput_cu(strVariableName)**, which is described in the section **GetUserInput_cu and parameter variables** in the document **The cmdutils library in Execute Commands**

The “If Condition” commands

These new commands,

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

ExitIfConditionTrue_cu(tuplets_objects_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_objects_selected**,The selection does not contain tuplets. This plugin will now exit.)

The condition is **tuplets_objects_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.

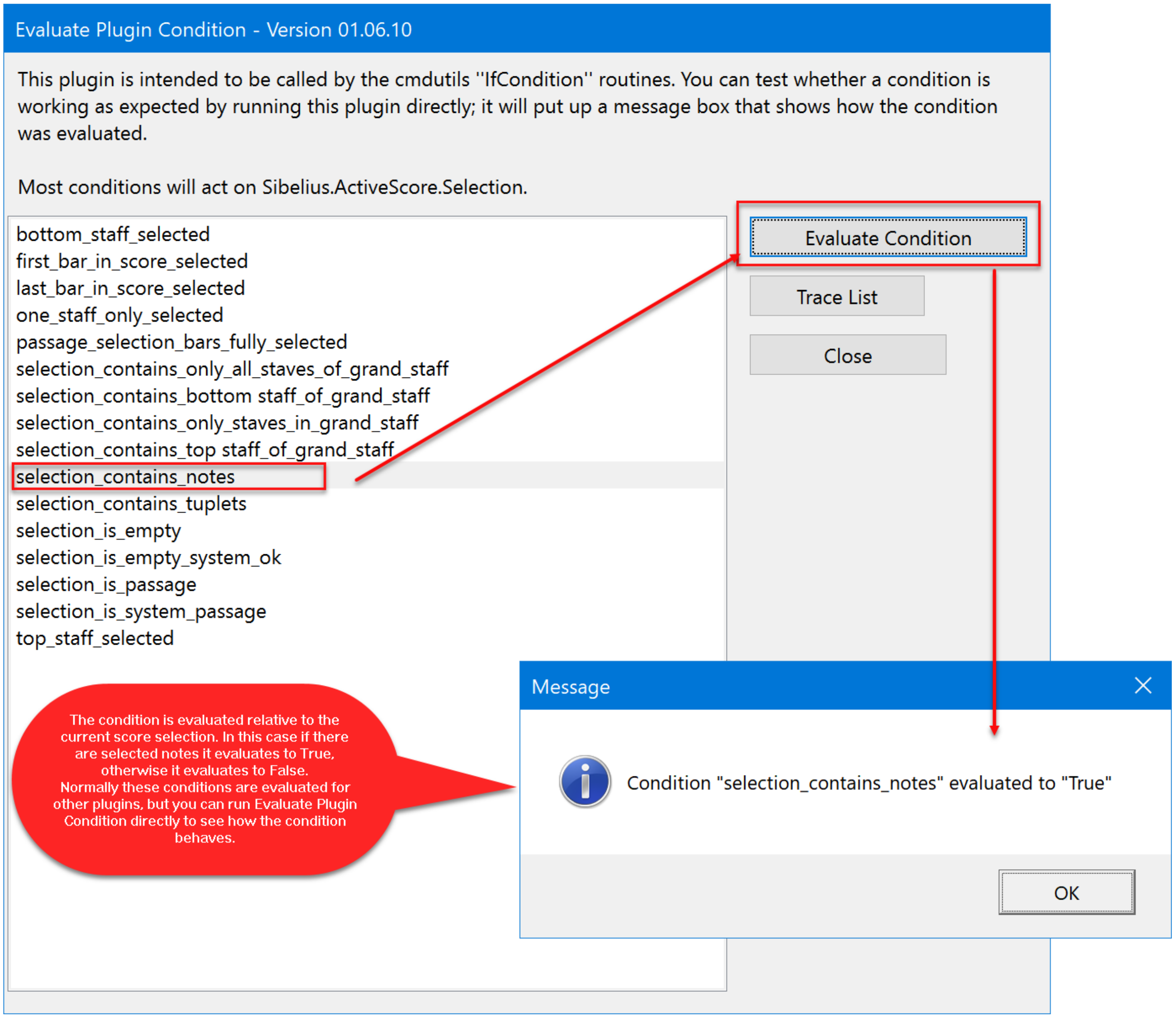
I considered replacing the existing **ExitIf** commands with condition-type commands, but decided not to break existing macros, and that the commands with conditions in the command names were actually easier to work with than the new ones, since you did not need to find the correct condition name.

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

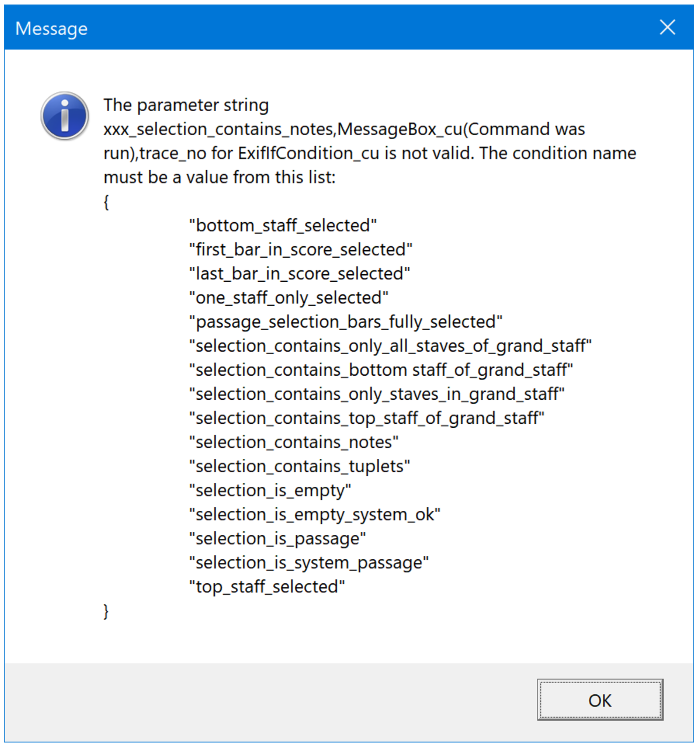
1. Add a new condition name to the array **dlg_lstConditionNames**, preferably in alphabetical order.
2. Add a new case statement to code in **API_TestCondition** to evaluate the new condition. Return 1 if the condition is satisfied, 0 if not.

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 10, 2025, the available conditions are (“-----” are visual separators in the list)

```
dlg_lstConditionNames
{
"is_full_score"
"is_dynamic_part"
"is_score_subset"
"-------"
"voice_1_objects_selected"
"voice_2_objects_selected"
"voice_3_objects_selected"
"voice_4_objects_selected"
"only_voice_1_objects_selected"
"only_voice_2_objects_selected"
"only_voice_3_objects_selected"
"only_voice_4_objects_selected"
"notes_in_multiple_voices_selected"
"-------"
"notes_or_rests_selected"
"notes_selected"
"rests_selected"
"rests_or_bar_rests_selected"
"bar_rests_selected"
"-------"
"tuplet_objects_selected"
"incomplete_tuplets_selected"
"tuplet_child_notes_selected_no_parent"
"-------"
"selection_is_empty"
"selection_is_empty_system_ok"
"selection_is_passage"
"selection_is_system_passage"
"-------"
"passage_selection_bars_fully_selected"
"-------"
"top_staff_selected"
"bottom_staff_selected"
"first_bar_in_score_selected"
"last_bar_in_score_selected"
"one_staff_only_selected"
"-------"
"selection_contains_only_staves_in_one_grand_staff"
"selection_contains_only_all_staves_of_one_grand_staff"
"only_top_staff_of_grand_staff_selected"
"only_bottom_staff_of_grand_staff 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)

ExitIfConditionTrue_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_objects_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.

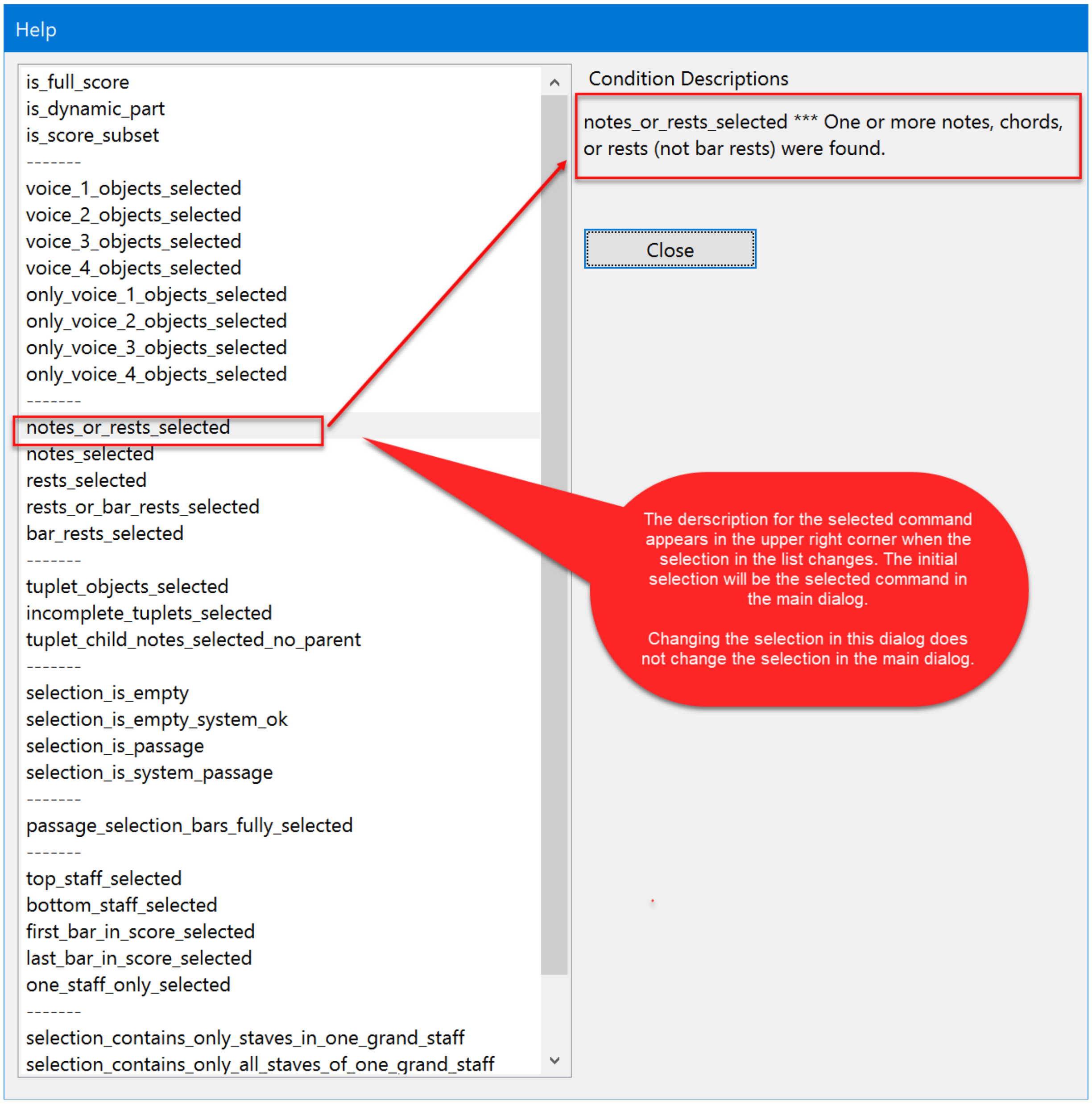
To use these commands, you will need the latest versions of these plugins:

- **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.

Condition descriptions

Use the Help button in Evaluate Plugin Conditions to see descriptions for each of the available conditions in the condition list. (This list box is out of date, but it is close enough to be usable here)



Help

is_full_score
is_dynamic_part
is_score_subset
----
voice_1_objects_selected
voice_2_objects_selected
voice_3_objects_selected
voice_4_objects_selected
only_voice_1_objects_selected
only_voice_2_objects_selected
only_voice_3_objects_selected
only_voice_4_objects_selected
----
notes_or_rests_selected
notes_selected
rests_selected
rests_or_bar_rests_selected
bar_rests_selected
----
tuplet_objects_selected
incomplete_tuplets_selected
tuplet_child_notes_selected_no_parent
----
selection_is_empty
selection_is_empty_system_ok
selection_is_passage
selection_is_system_passage
----
passage_selection_bars_fully_selected
----
top_staff_selected
bottom_staff_selected
first_bar_in_score_selected
last_bar_in_score_selected
one_staff_only_selected
----
selection_contains_only_staves_in_one_grand_staff
selection_contains_only_all_staves_of_one_grand_staff

Condition Descriptions

notes_or_rests_selected *** One or more notes, chords, or rests (not bar rests) were found.

Close

The derscription for the selected command appears in the upper right corner when the selection in the list changes. The initial selection will be the selected command in the main dialog.

Changing the selection in this dialog does not change the selection in the main dialog.

Technical Appendices

Some quite technical appendices describing what **Evaluate Plugin Conditions** does and how to emulate **ExitIf** commands using conditions can be found in the document **If Condition Commands in cmdutils Technical Appendices.** Most users should never need this, but the document is available on request.