The “If Condition” Commands in cmdutils - Technical Appendices

Bob Zawalich June 15, 2024 updated February 1, 2026

These appendices were once part of the document The “If Condition” Commands in cmdutils and Execute Commands, but I decided they were really only of interest to me and have split them into a separate document.

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: Condition Descriptions

All conditions are applied to the currently active selection. These conditions names and descrptions are stored in _lstConditionNamesTildeDescriptions in Evaluate Plugin Conditions. I have here replaces the single tilde separator with 6 undescores to be a bit more human readable.

The Help button in Evaluate Plugin Conditions woll display a list of names and the description for the currently selected condition will be displayed in a text box.

_lstConditionNamesTildeDescriptions
{
"is_full_score______The current active part is the full score."
"is_dynamic_part______The current active part is a dynamic part."
"is_score_subset______The current active part is a score subset."
"-------______separator"
"voice_1_objects_selected______At least one selected object is in voice 1."
"voice_2_objects_selected______At least one selected object is in voice 2."
"voice_3_objects_selected______At least one selected object is in voice 3."
"voice_4_objects_selected______At least one selected object is in voice 4."
"only_voice_1_objects_selected______All selected objects are in voice 1."
"only_voice_2_objects_selected______All selected objects are in voice 2."
"only_voice_3_objects_selected______All selected objects are in voice 3."
"only_voice_4_objects_selected______All selected objects are in voice 4."
"-------______separator"
"notes_or_rests_selected______One or more notes, chords, or rests (not bar rests) were found."
"notes_selected______One or more non-rest notes or chords were found."
"rests_selected______One or more rests (not bar rests) were found."
"rests_or_bar_rests_selected______One or more rests or bar rests were found."
"bar_rests_selected______One or more bar rests were found."
"-------______separator"
"tuplet_objects_selected______One or more Tuplet objects were found. Child notes and tuplets are not considered."
"incomplete_tuplets_selected______One or more Tuplet objects were selected, but some of their child notes or tuplets were not selected."
"tuplet_child_notes_selected_no_parent______Notes that are children of Tuplet objects were selected, but their parent Tuplets are not selected."
"-------______separator"
"selection_is_empty______Nothing is selected (as after pressing Esc twice). System staff is ignored."
"selection_is_empty_system_ok______Nothing is selected (as after pressing Esc twice). System staff is included."
"selection_is_passage______The selection is a passage or system passage selection."
"selection_is_system_passage______The selection is a system passage selection."
"-------______separator"
"passage_selection_bars_fully_selected______There is a passage selection and all objects in the selected bars are selected."
"-------______separator"
"top_staff_selected______The top staff in the score is selected."
"bottom_staff_selected______The bottom staff in the score is selected."
"first_bar_in_score_selected______Objects from the first bar in the score are selected."
"last_bar_in_score_selected______Objects from the last bar in the score are selected."
"one_staff_only_selected______All selected objects are from the same staff."
"-------______separator"
"selection_contains_only_staves_in_one_grand_staff______All the staves in the selection belong to a single grand staff."
"selection_contains_only_all_staves_of_one_grand_staff______The staves selected are all the staves of a single grand staff, and nothing else."
"only_top_staff_of_grand_staff_selected______A single staff is selected and it is the top staff of a grand staff instrument, which may be an ossia or extra staff."
"only_bottom_staff_of_grand_staff selected______A single staff is selected and it is the bottom staff of a grand staff instrument, which may be an ossia or extra staff."
}

A passage selection is required for the following conditions. If we start without a passage selection, the selection is temporarily converted to a passage selection, and then restored.

"bottom_staff_selected"

- The bottom staff in the score is selected

"top_staff_selected"

- The top staff in the score is selected

"first_bar_in_score_selected"

- Objects from the first bar in the score are selected

"last_bar_in_score_selected"

- Objects from the last bar in the score are selected

"one_staff_only_selected"

- All selected objects are from the same staff

"passage_selection_bars_fully_selected"

- There is a passage selection and all objects in the selected bars are selected

Appendix 3: Exactly what the conditions test

Here is the actual code in the plugin **Evaluate Plugin Conditions** that evaluates conditions as of this writing (February 1, 2026)

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 4: Mapping ExitIf commands to use conditions

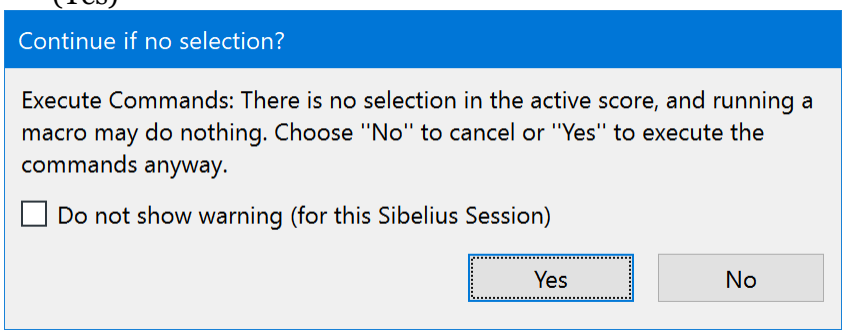
I mentioned above that I set up conditions so they could do the equivalent of the existing ExitIf commands. It is a bit tricky to map the ExitIf name to the appropriate condition and ExitIfConditionTrue/False command. You need to choose both the condition and the appropriate True/False form of the command.

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)



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 5: 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.

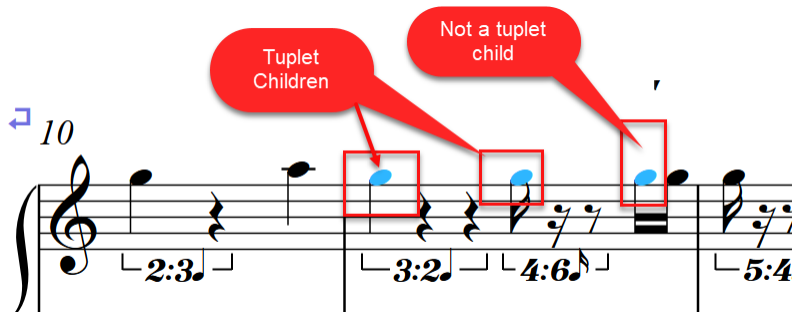
The details of this are tricky, so pause a moment to be sure it makes sense.

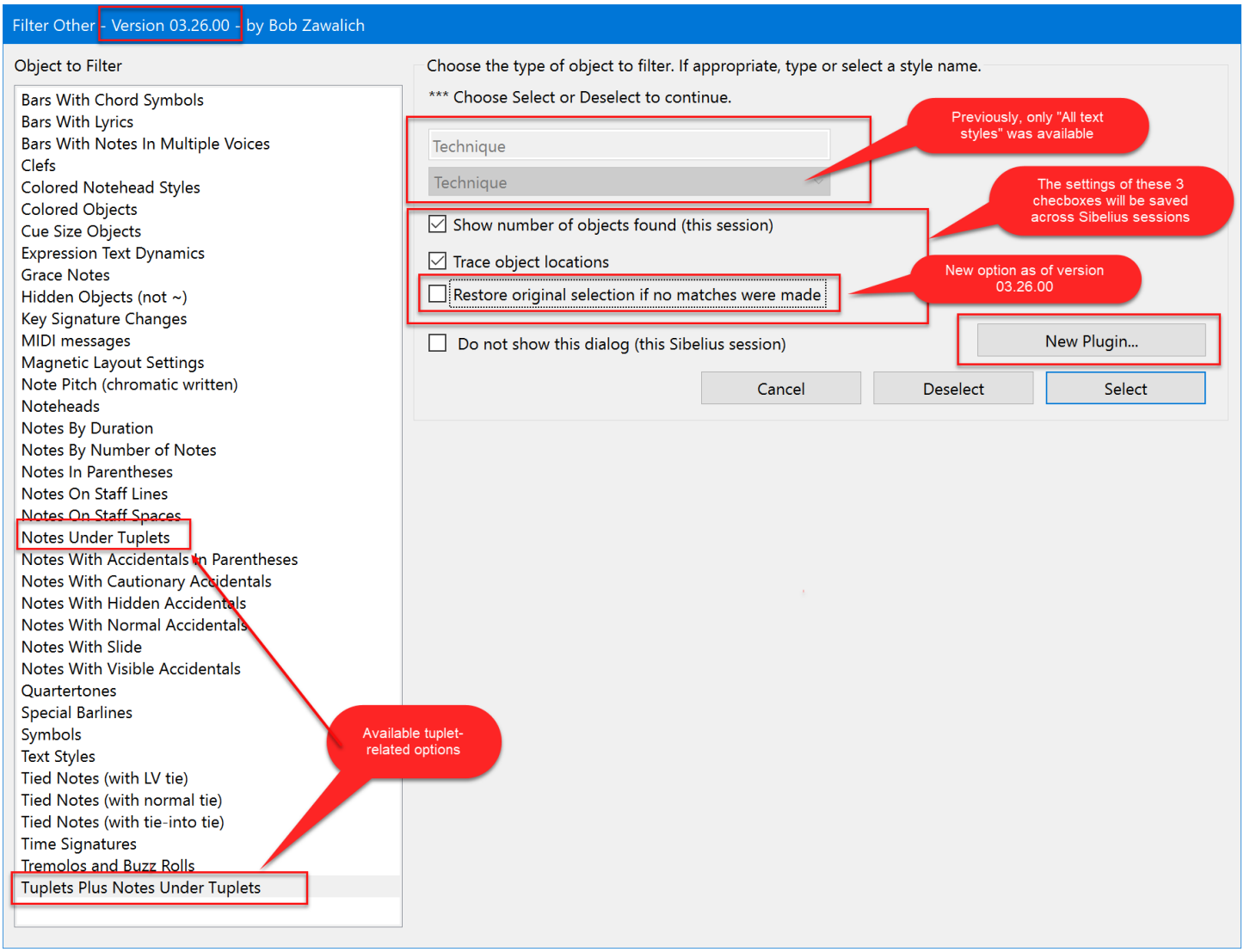
Note that **Saving** and **Restoring** a selection works well for a passage selection but will not restore the selection in a non-passage selection if any of the selected objects were changed or added or deleted. You really need to understand what is happening to the selection to make this work.

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

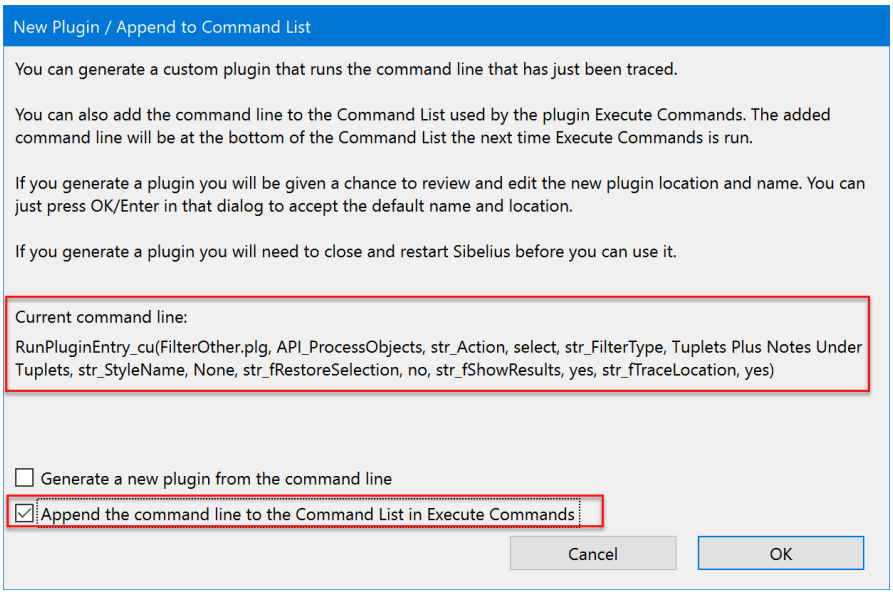
Let us deal with the worst case and start with a non passage selection. For a filter, we could run any Sibelius filtering command, such as **filter_nonspecial_barlines.** This gives you access to all the filters on **Home>Filters**. You could also run any filtering plugin, ideally one that does not bring 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.

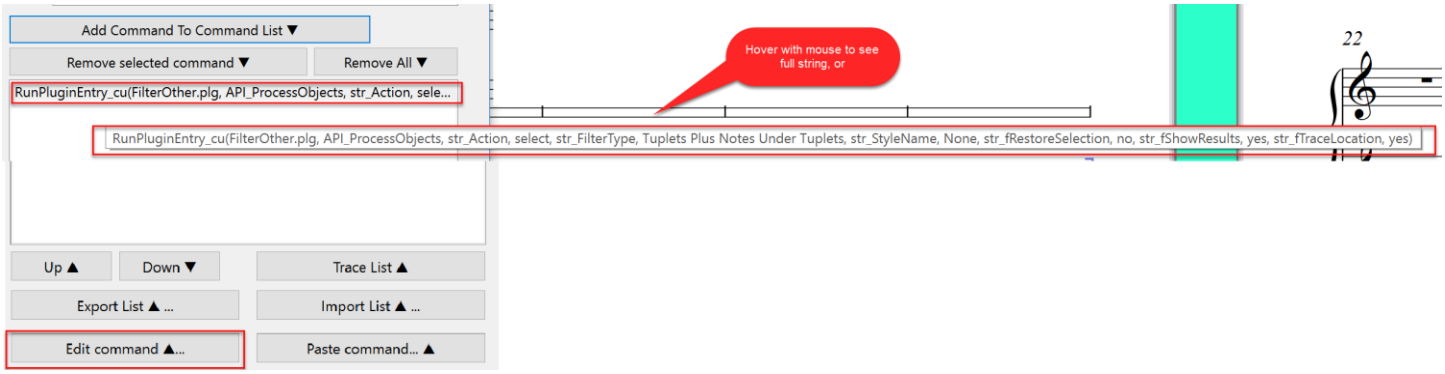




Here is the dialog that appears when you press **New plugin...** and tell it to append the command line to the **Execute Command** Command List:



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



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.

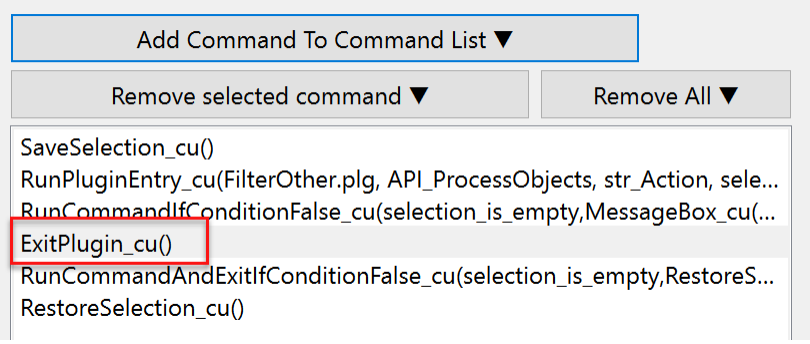
3. **RunCommandIfConditionFalse_cu(selection_is_empty,MessageBox_cu(Exiting because selection contains tuplets or notes inside tuplets),trace_no)**

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:

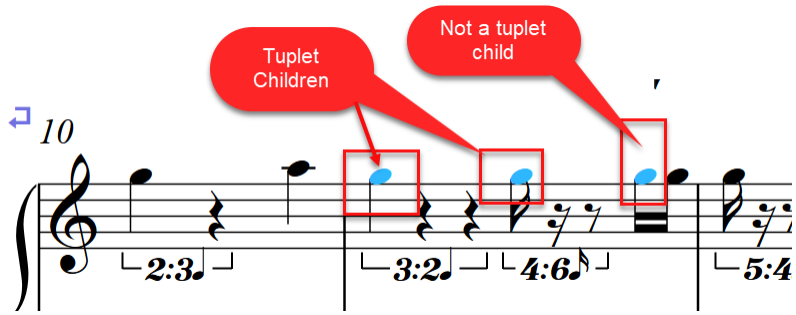
```
SaveSelection_cu()
RunPluginEntry_cu(FilterOther.plg, API_ProcessObjects, str_Action, select, str_FilterType, Tuplets Plus Notes Under Tuplets,
str_StyleName, None, str_fRestoreSelection, no, str_fShowResults, yes, str_fTraceLocation, yes)
RunCommandIfConditionFalse_cu(selection_is_empty,MessageBox_cu(Exiting 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.

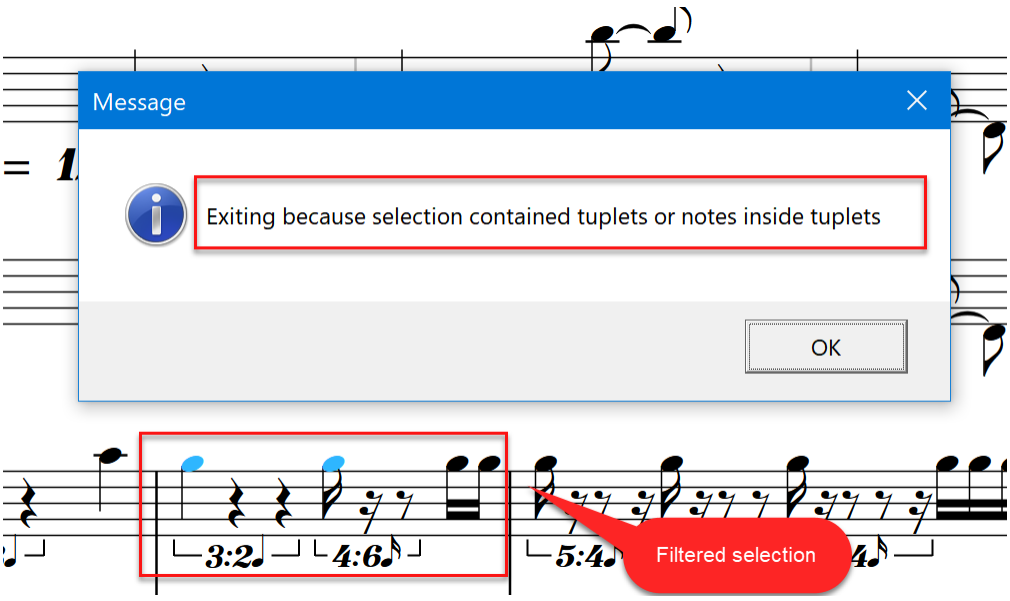
Often, just because the final result is correct, it does not prove that the processing was correct. This is the life of a programmer…



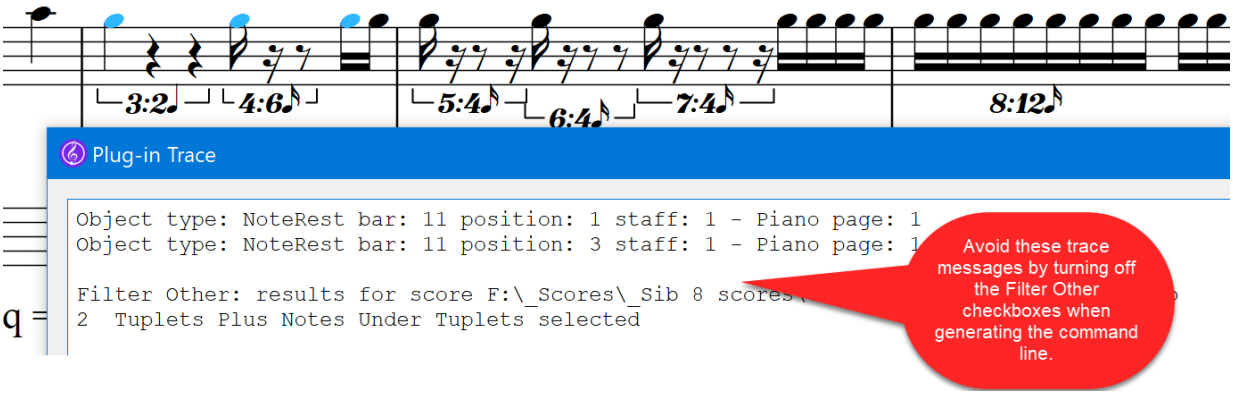
Here is the original selection again. *Assume the **ExitPlugin_cu()** command is not present.*



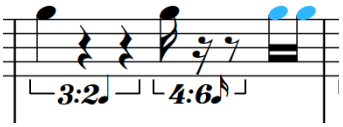
We see this message box after the filter. Note that the selection at this point includes only the notes in the tuplets, as expected.



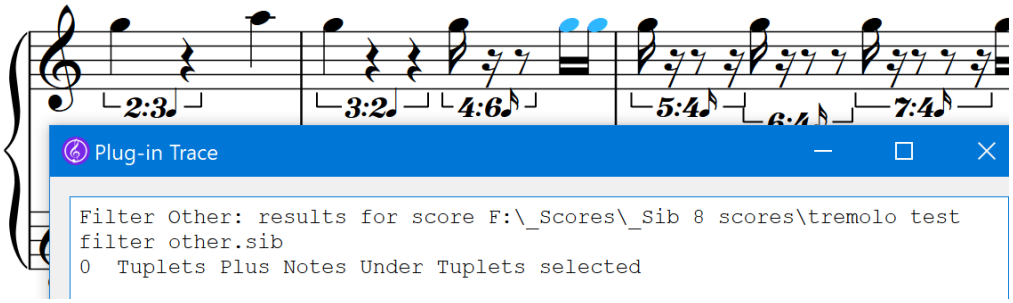
When we OK the message box, the selection should be restored to the original, and it is. Here is the final result. The original selection is restored, as expected.



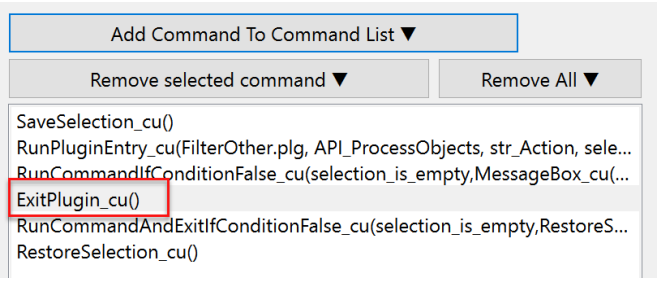
What if we ran this macro when the selection did not contain any tuplets or children of tuplets, such as this?



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.



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:



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.