Condition Plugins for the “If Condition” Commands in cmdutils and Execute Commands

Bob Zawalich June 15, 2024 updated February 23, 2026

Executive Summary – how to write a condition plugin

- Make a copy of an existing condition plugin such as **Custom Condition Plugin**.
    - In the copy, change the file name and the menu name (in the global variable _PluginMenuName)

- Do not change any code in the methods **Run()** and **API_GetListOfEvaluationMethods**

- Create one or more **Evaluation Methods** in your new plugin
    - The parameters for an Evaluation Method must always be "score, selection"
    - Evaluation Methods must only return "yes" for True and "no" for False. Any other return value is treated as an error return.
    - After writing a method, you must add the method name and a description of the method in the global array **g_lstPlugins_Methods_Descriptions**.

For information about conditions in **Evaluate Plugin Condition** and **cmdutils**, see the document **The “If Condition” Commands in cmdutils and Execute Commands**

Condition Plugins for adding additional conditions

You can add new conditions by creating and using new **Condition Plugins**. Such plugins can be distributed to other users, among other benefits. You will need to be able to create a **ManuScript** plugin or find someone who can write one for you. As of this writing I have not seen an AI -written plugin that actually works, but if you can get one to work, congratulations and good luck.

As of February 2026 (**Evaluate Plugin Condition** version 01.06.00), **cmdutils** commands that take a **condition** as a parameter can specify a **Condition Plugin** name and a method(routine) to act as a condition.

A **Condition Plugin** is a specially-constructed ManuScript plugin that **Evaluate Plugin Condition** can recognize. Details will follow. A **Condition Plugin** will contain one or more **Evaluation Methods**, each of which defines and evaluates one condition.

Using Condition Plugins in Execute Commands

The name of a **Condition Plugin** and its **Evaluation Method** can be specified in **cmdutils** commands that accept conditions. You must specify the condition as <plugin file name with no path or extension><dot/period . as separator><evaluation method name>. Here is an example:

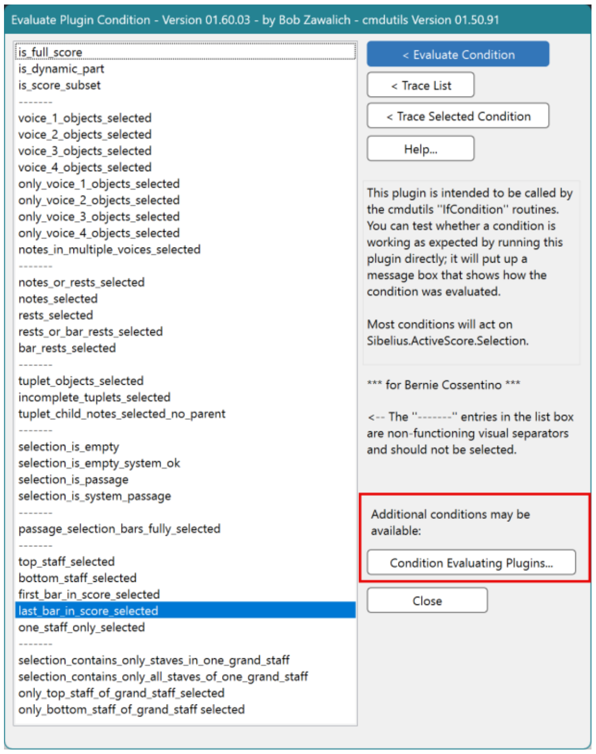
RunCommandIfConditionTrue_cu(**ExtraConditions.WholeNotesInSelection**, MessageBox_cu(Selection includes whole notes),trace_no)

In this example above, the plugin file name is **ExtraConditions** and the **Evaluation Method** that is called is **WholeNotesInSelection.** The 2 names must be separated by a period.

Use no other punctuation, especially periods and commas in either the plugin name or the method name, which must be spelled exactly the same as they are defined.

Evaluate Plugin Condition will confirm that the plugin is installed on your computer, and that the method is defined within the plugin.

You can find and test existing **Condition Plugins** in **Evaluate Plugin Conditions**, using the **Condition Evaluating Plugins…** button. This will be described in more detail later.


Evaluate Plugin Condition - Version 01.60.03 - by Bob Zawalich - cmdutils Version 01.50.91

```
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
```

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.

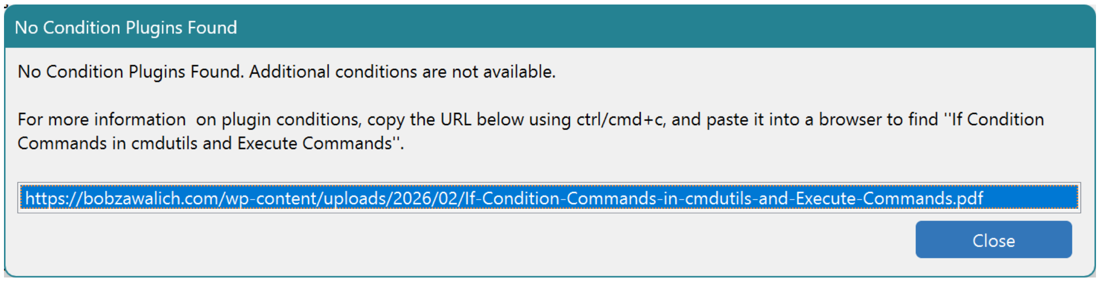
Most conditions will act on Sibelius.ActiveScore.Selection.

*** for Bernie Cossentino ***

<-- The "-------" entries in the list box are non-functioning visual separators and should not be selected.

Additional conditions may be available:

If no **Condition Plugins** are installed, you will see a message like this:

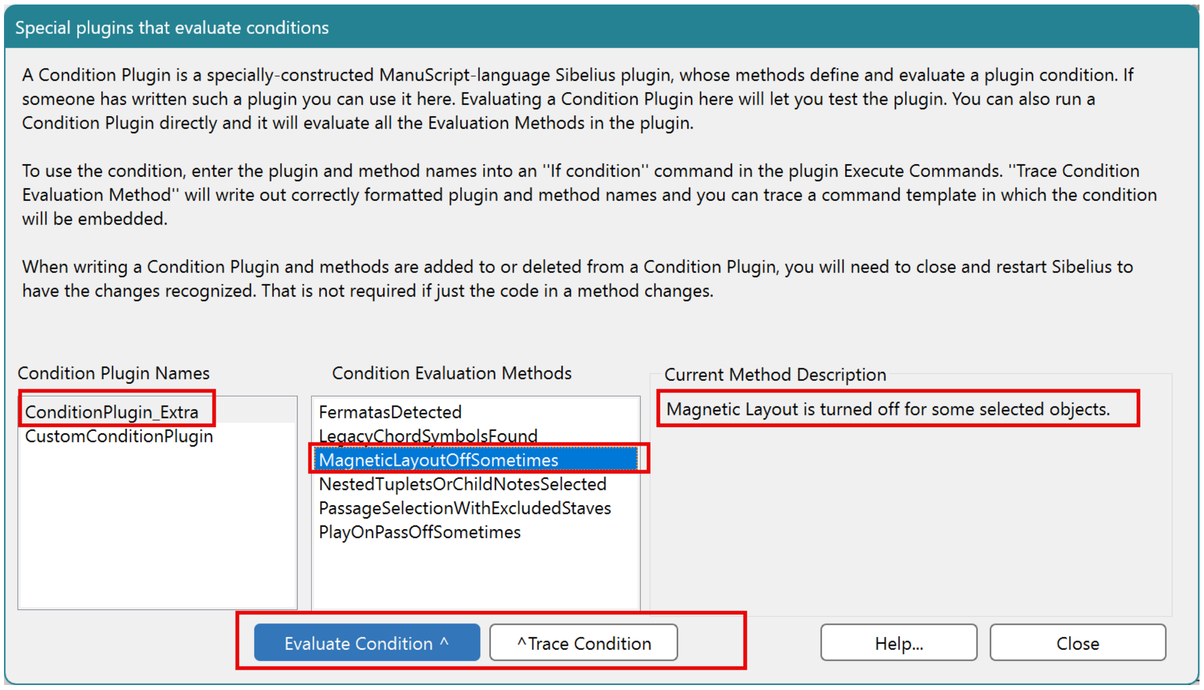

No Condition Plugins Found

No Condition Plugins Found. Additional conditions are not available.

For more information on plugin conditions, copy the URL below using ctrl/cmd+c, and paste it into a browser to find "If Condition Commands in cmdutils and Execute Commands".

https://bobzawalich.com/wp-content/uploads/2026/02/If-Condition-Commands-in-cmdutils-and-Execute-Commands.pdf

If there are available **Condition Plugins**, you will see this dialog, and you can test/evaluate and/or trace the **Evaluation Methods** of the **Condition Plugins**.


Special plugins that evaluate conditions

A Condition Plugin is a specially-constructed ManuScript-language Sibelius plugin, whose methods define and evaluate a plugin condition. If someone has written such a plugin you can use it here. Evaluating a Condition Plugin here will let you test the plugin. You can also run a Condition Plugin directly and it will evaluate all the Evaluation Methods in the plugin.

To use the condition, enter the plugin and method names into an "If condition" command in the plugin Execute Commands. "Trace Condition Evaluation Method" will write out correctly formatted plugin and method names and you can trace a command template in which the condition will be embedded.

When writing a Condition Plugin and methods are added to or deleted from a Condition Plugin, you will need to close and restart Sibelius to have the changes recognized. That is not required if just the code in a method changes.

Condition Plugin Names
- ConditionPlugin_Extra
- CustomConditionPlugin

Condition Evaluation Methods
- FermatasDetected
- LegacyChordSymbolsFound
- MagneticLayoutOffSometimes
- NestedTupletsOrChildNotesSelected
- PassageSelectionWithExcludedStaves
- PlayOnPassOffSometimes

Current Method Description

Magnetic Layout is turned off for some selected objects.

Tutorial: Writing a Condition Plugin

This section assumes that you have programming experience. You will be writing a plugin in the ManuScript programming language. Its reference manual can be found at **File>Plugins>ManuScript Language Reference**. This is not for the faint of heart, but experienced ManuScript programmers should find this straightforward.

The easiest way to write a **Condition Plugin** is to make a copy of a published template plugin such as **Custom Condition Plugin**, make some changes, and then write new **Evaluation Methods** to define and implement new conditions that can be used with **cmdutils "If commands"**.

A **Condition Plugin** must include these components

- A method named **API_GetListOfEvaluationMethods**. This will be called by **Evaluate Plugin Condition**, which uses the existence of this method as a signal that this is a **Condition Plugin**.
    - The code in the methods **API_GetListOfEvaluationMethods** and **Run()** should not be changed. These routines should always be copies of the code in a published **Condition Plugin**.

- A global array named **g_lstPlugins_Methods_Descriptions**, which provides a list of method names and descriptions that **Evaluate Plugin Condition** will interpret as a condition. You will find this array in the **Data** block in the **Sibelius Plugin Editor**. If you are starting from a template, there will be placeholder entries in the array for any template **Evaluation Methods**. Use these as a guide for creating entries for your own methods, then remove the placeholder entries from the array. You can also remove the template **Evaluation Method** code if you like after writing your own methods.

- The global variables **_msgRunEvaluationTesting** and **_PluginMenuName** must be defined because they are used in the **Initialize()** and **Run()** methods. **_msgRunEvaluationTesting** never needs to be changed. **_PluginMenuName** must be the menu name of the plugin and should be similar to the plugin file name. It should be changed every time you make a copy of a template plugin.

- There must be one or more **Evaluation Methods**. Each of these defines and implements a condition.
    - Each Evaluation Method must have "score, selection" as its parameter list.
    - Evaluation Methods must return "yes" for True and "no" for False.

Examples of **conditions** can be found in the method **How_To_Add_New_Conditions** in **Evaluate Plugin Condition,** which can be seen by editing the plugin file for **Evaluate Plugin Condition.**

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).

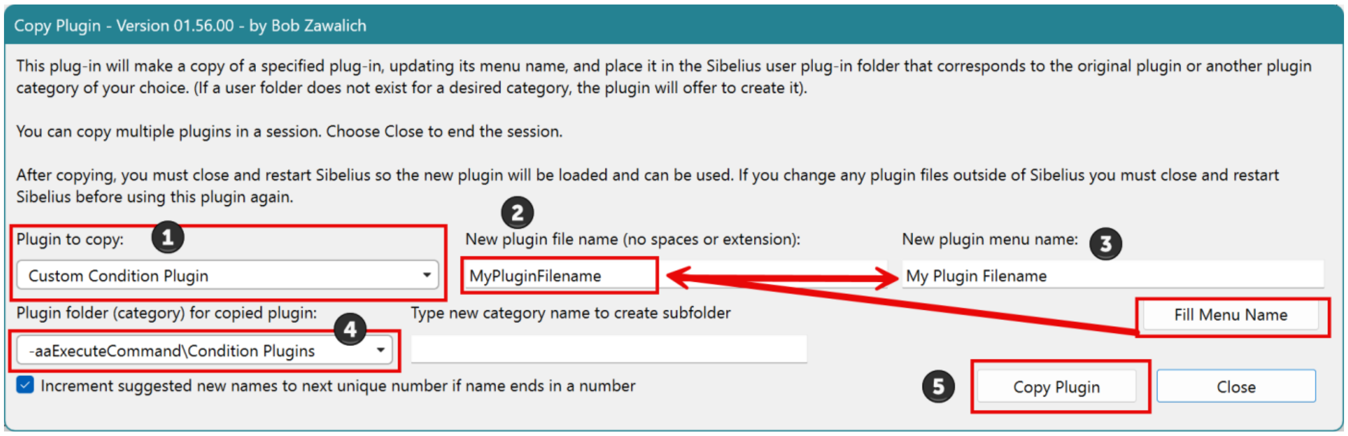
There is a documentation-only method **HOW_TO_SET_UP_A_CONDITION_PLUGIN** in the published **Condition Plugins** which goes into detail on writing such plugins.

Steps in writing a Condition Plugin

The first step is to decide what you want the condition to do, and to give it a name.
These instructions assume you will be starting by making a clone of the published plugin **Custom Condition Plugin**.

1. Use **File>Plug-ins>Install Plug-ins** to install the template plugin **Custom Condition Plugin** which should be copied but not modified. Also install **Copy Plugin**.
2. You will need to install **Execute Commands**, **cmdutils**, and **Evaluate Plugin Condition**. Be sure you have the most recent versions of these plugins.

3. Make a copy of **Custom Condition Plugin**, which you will rename and modify. If you use the plugin **Copy Plugin**, run it and have it make a copy of **Custom Condition Plugin** and store it in a valid user plugin subfolder.
    a. **Copy Plugin** makes a copy of the template plugin, and gives the copy a new file name, and changes the name in the global variable **_PluginMenuName**, which defines the plugin **Menu Name**, then moves the copy to an appropriate plugin folder.


Copy Plugin - Version 01.56.00 - by Bob Zawalich

This plug-in will make a copy of a specified plug-in, updating its menu name, and place it in the Sibelius user plug-in folder that corresponds to the original plugin or another plugin category of your choice. (If a user folder does not exist for a desired category, the plugin will offer to create it).

You can copy multiple plugins in a session. Choose Close to end the session.

After copying, you must close and restart Sibelius so the new plugin will be loaded and can be used. If you change any plugin files outside of Sibelius you must close and restart Sibelius before using this plugin again.

1 Plugin to copy: Custom Condition Plugin

2 New plugin file name (no spaces or extension): MyPluginFilename

3 New plugin menu name: My Plugin Filename

4 Plugin folder (category) for copied plugin: -aaExecuteCommand\Condition Plugins

Type new category name to create subfolder

✓ Increment suggested new names to next unique number if name ends in a number

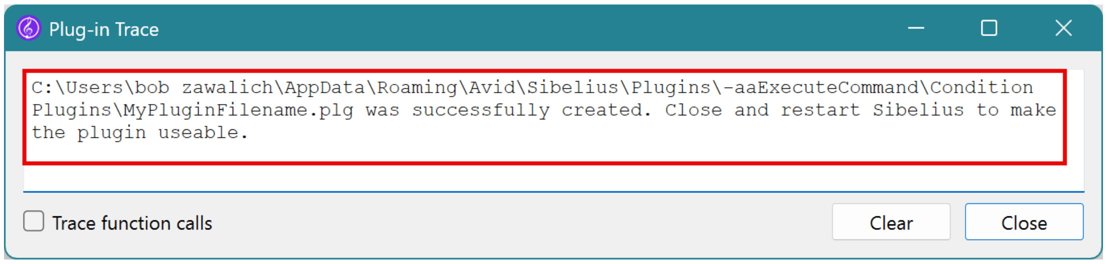
In this example, **Step 1** is to choose the name of the **plugin to copy** from the drop-down list box. If **Custom Condition Plugin** is not in the list you will need to close this plugin, install **Custom Condition Plugin**, and then run **Copy Plugin** again.

Step 2 is to give your plugin a name, which in this example is **MyPluginFilename**. You can surely do better than that! The name should not contain spaces or periods, and I usually capitalize the first letter of each word in the name.

Step 3 is to create a plugin **Menu Name**, which is the name that you will see in plugin menus. If you have filled in the file name and press **Fill Menu Name**, it will make a copy of the file name with a space between each of the words.

In **Step 4**, choose the plugin subfolder when the copied plugin will be saved. By default the folder will be the same folder where the plugin being copied is stored, but you can drop down the list box and find and choose other suitable folders.

Step 5 copies the plugin. When the copy is finished you will see a message in the Plugin Trace window that looks like this. It will give you the full path of the new plugin. **It will tell you to close and restart Sibelius. Be sure you close and save any open scores before you close Sibelius.**


Plug-in Trace

```
C:\Users\bob zawalich\AppData\Roaming\Avid\Sibelius\Plugins\-aaExecuteCommand\Condition
Plugins\MyPluginFilename.plg was successfully created. Close and restart Sibelius to make
the plugin useable.
```

☐ Trace function calls

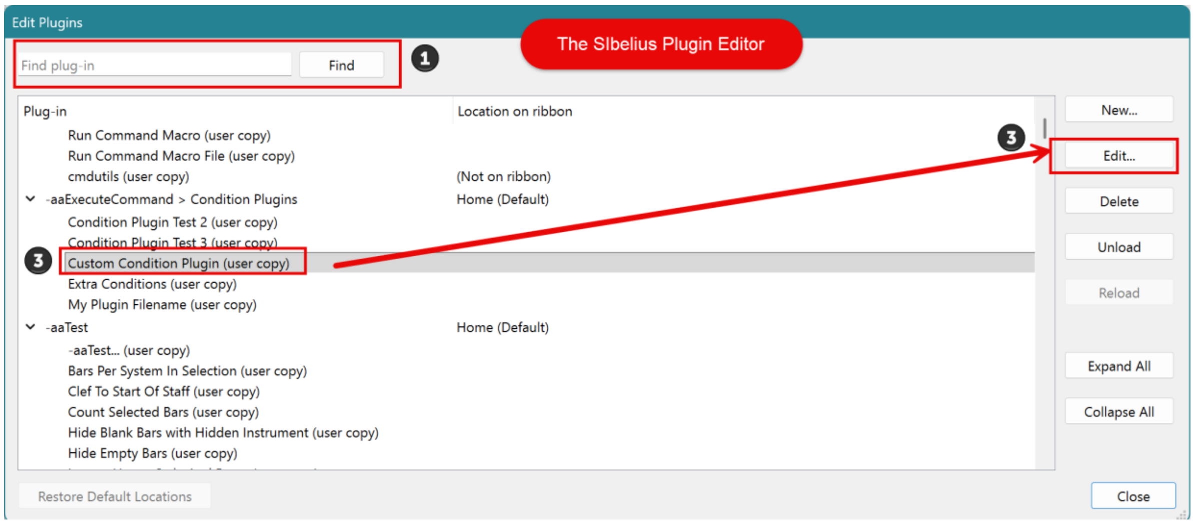
Now we need to make changes to the new plugin. I recommend using the editor in **File>Plug-ins>Edit Plug-ins**, though you can use any text editor. The advantages of the plugin editor include making it less likely you will trash the plugin file by using unbalanced brackets or parentheses, or inappropriate double quotes, which can make the file unreadable in Sibelius. The plugin editor also has a **Check Syntax** button that makes it easier to find misspellings and other syntax errors, so use a different editor carefully.

1. Using the Sibelius Plugin Editor

This discussion assumes that you have made a copy of the plugin **Custom Condition Plugin** and are going to edit it with the **Sibelius Plugin Editor**. You are welcome to accomplish the desired result by different means, but you will be on your own.

This also assumes that you have saved closed and restarted Sibelius since you created your **Condition Plugin**. If not, Sibelius will not find your plugin.

Edit the file by going to **File>Plug-ins>Edit Plug-ins**. This will bring up the **Edit Plugins** dialog shown below. To find and edit your plugin you can either scroll in the listbox or enter the start of the plugin menu name (often the same as the file name but with spaces between the words) into the Find box. You may need to press Find multiple times to find your plugin. Press **Edit** to open your plugin.



Plug-in	Location on ribbon
Run Command Macro (user copy)	
Run Command Macro File (user copy)	
cmdutils (user copy)	(Not on ribbon)
-aaExecuteCommand > Condition Plugins	Home (Default)
Condition Plugin Test 2 (user copy)	
Condition Plugin Test 3 (user copy)	
Custom Condition Plugin (user copy)	
Extra Conditions (user copy)	
My Plugin Filename (user copy)	
-aaTest	Home (Default)
-aaTest... (user copy)	
Bars Per System In Selection (user copy)	
Clef To Start Of Staff (user copy)	
Count Selected Bars (user copy)	
Hide Blank Bars with Hidden Instrument (user copy)	
Hide Empty Bars (user copy)	

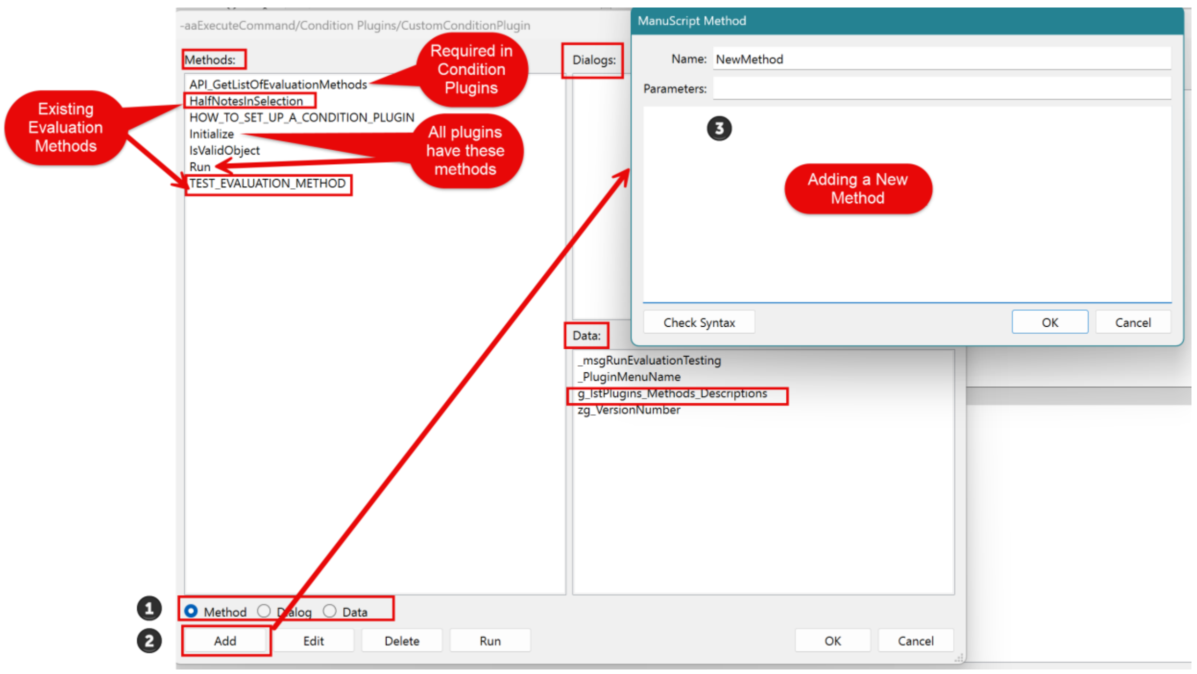
2. Editing a Plugin

There are 2 places where we will need to make changes.

- You need to create or edit one or more **Evaluation Methods**. The template should have one or more examples of **Evaluation Methods** you can use as a model.
- When you create a new **Evaluation Method**, you need to add an entry into the global array named **g_lstPlugins_Methods_Descriptions** (in the Data area in the plugin editor)

So now let's create an **Evaluation Method**. Here is what you would do to create a new method from scratch:

After you click **Edit,** you will get a dialog that lets you edit the chosen plugin, **Custom Condition Plugin.**


~aaExecuteCommand/Condition Plugins/CustomConditionPlugin

Methods:

```
API_GetListOfEvaluationMethods
HalfNotesInSelection
HOW_TO_SET_UP_A_CONDITION_PLUGIN
Initialize
IsValidObject
Run
TEST_EVALUATION_METHOD
```

Dialogs:

Data:

```
_msgRunEvaluationTesting
_PluginMenuName
g_lstPlugins_Methods_Descriptions
zg_VersionNumber
```

ManuScript Method

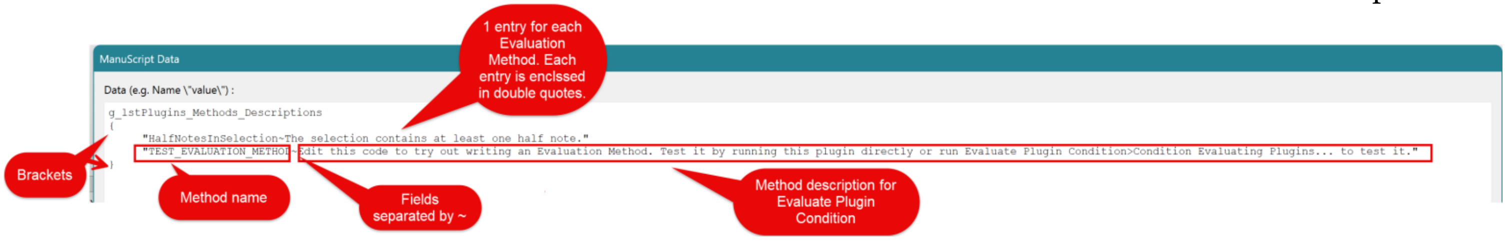
Name: NewMethod

Parameters:

Existing Evaluation Methods
Required in Condition Plugins
All plugins have these methods
Adding a New Method
3
1
Method
Dialog
Data
2
Add
Edit
Delete
Run
OK
Cancel
Check Syntax
OK
Cancel

Observe the different regions in the dialog:

- On the right there are 2 regions
  - **Dialogs** is where dialogs are defined. There are no dialogs in this plugin
  - **Data** is where global data is kept. Global data is accessible to all methods.
    - **_PluginMenuName** is used in most plugins in the **Initialize** method to define the plugin menu name
    - **zg_VersionsNumber** is a 6-digit number that is present in all the plugins I (Bob Zawalich) write. It is a good way to indicate to a user if they have the most recent version of the plugin, and is usually displayed in the main dialog, which we do not have here.
    - **g_lstPlugins_Methods_Descriptions** is required to be defined in **Condition Plugins**. It provides the method name and a description for each **Evaluation Method** in the plugin. Here is what the array looks like for this plugin. Double-click on the data item name to see an edit window. I stretched out the window so the text did not wrap.


ManuScript Data

Data (e.g. Name \"value\") :

```
g_lstPlugins_Methods_Descriptions
{
    "HalfNotesInSelection~The selection contains at least one half note."
    "TEST_EVALUATION_METHOD~Edit this code to try out writing an Evaluation Method. Test it by running this plugin directly or run Evaluate Plugin Condition>Condition Evaluating Plugins... to test it."
}
```

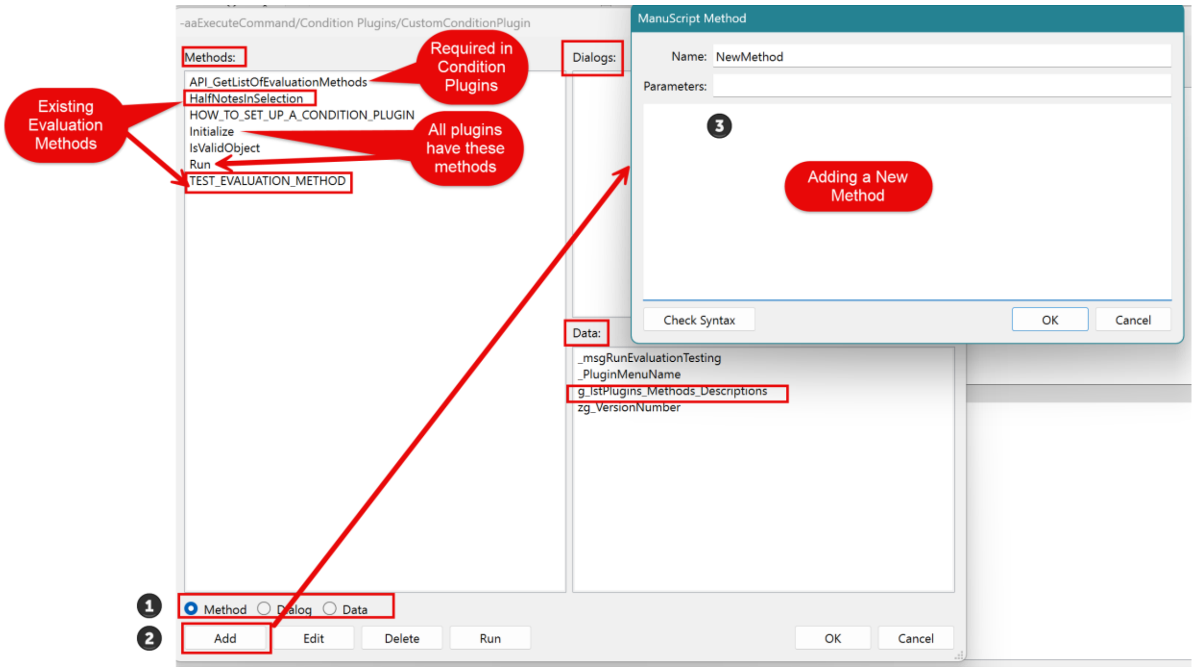
1 entry for each Evaluation Method. Each entry is enclssed in double quotes.
Brackets
Method name
Fields separated by ~
Method description for Evaluate Plugin Condition

- An array will have a name (this array is **g_lstPlugins_Methods_Descriptions)**, and its data will be enclosed in { } brackets.
- Each entry in the array is enclosed in double quotes and can only be a text string.
- For this array, each entry has 2 parts, separated by a tilde (~)
  - The **Evaluation Method name**, exactly as it is defined.
  - **A concise description** of what the method does, to be displayed in the plugin Evaluate Plugin Condition. The name should be reasonably descriptive as well.

- On the Left is the list all the current methods.
  - **Standard methods**
    - Every plugin will have **Initialize()** and **Run()** in the list.
    - Any **Condition Plugin** needs to have the method **API_GetListOfEvaluationMethods**.

        - You should not have to change any of these methods.
    - **Evaluation Methods**
        - In this plugin there are 2 Evaluation Methods,
            - **HalfNotesInSelection** and
            - **TEST_EVALUATION_METHOD**
            - If you are not sure which routines are Evaluation Methods, look at the array
            - **g_lstPlugins_Methods_Descriptions.**

    - **Helper Methods**
        - You can have any number of helper methods, which are not made public. In this plugin the helper methods are
            - **IsValidObject** and
            - **HOW_TO_SET_UP_A_CONDITION_PLUGIN** , which is strictly for documentation
- Here is the Edit window:



API_GetListOfEvaluationMethods
HalfNotesInSelection
HOW_TO_SET_UP_A_CONDITION_PLUGIN
Initialize
IsValidObject
Run
TEST_EVALUATION_METHOD

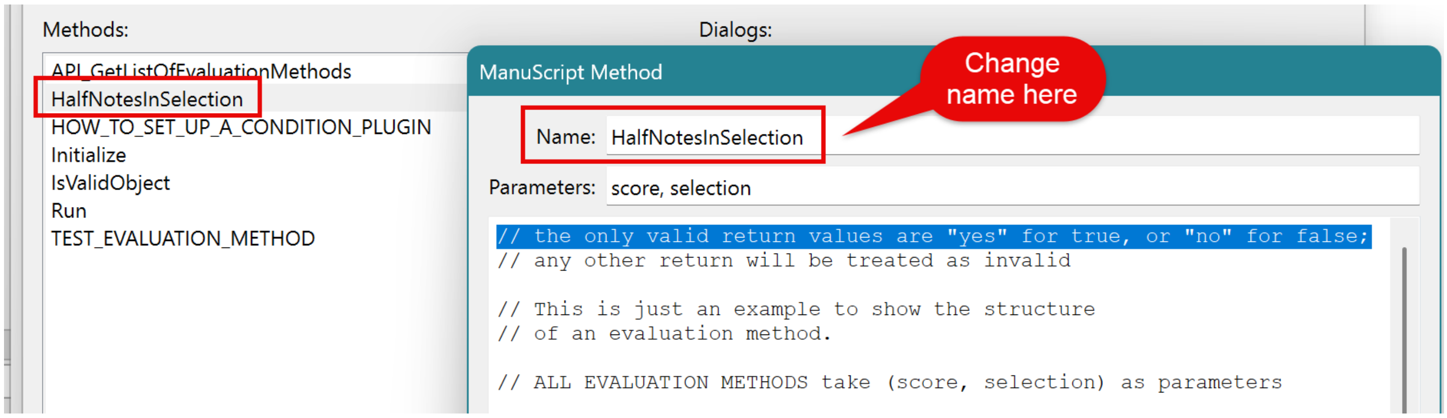
_msgRunEvaluationTesting
_PluginMenuName
g_lstPlugins_Methods_Descriptions
zg_VersionNumber

Adding a new Evaluation Method

- To add a new method from scratch, make sure **Method** is chosen in the radio buttons on the bottom left, then press **Add**. You will see a **ManuScript Method** window, with the name **NewMethod**.

- While this is often a good way to add a method, if I am creating a method that is similar to the existing ones, I will usually edit an existing method and change its name. Changing a method name creates a copy of the method being edited, and you will be editing the copy, with the original untouched.

- Let's demonstrate this by editing **HalfNotesInSelection,** changing the name to **IsDynamicPart**, and pressing OK.

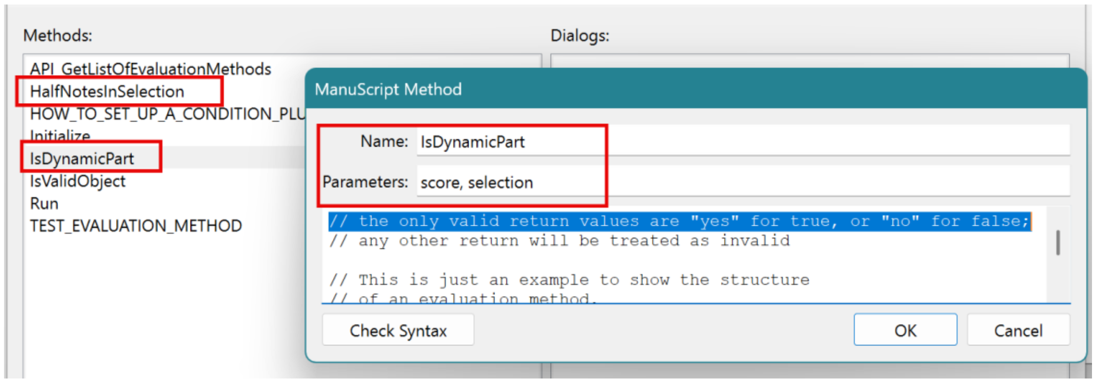


Name: HalfNotesInSelection

Parameters: score, selection

```
// the only valid return values are "yes" for true, or "no" for false;
// any other return will be treated as invalid

// This is just an example to show the structure
// of an evaluation method.

// ALL EVALUATION METHODS take (score, selection) as parameters
```

After the name is changed, OK is pressed, and the new method is opened for edition:



Name: IsDynamicPart

Parameters: score, selection

```
// the only valid return values are "yes" for true, or "no" for false;
// any other return will be treated as invalid

// This is just an example to show the structure
// of an evaluation method.
```

We will see both **Method** names in the list. Double-click on **IsDynamicPart** or select it and press **Edit** to start editing it.

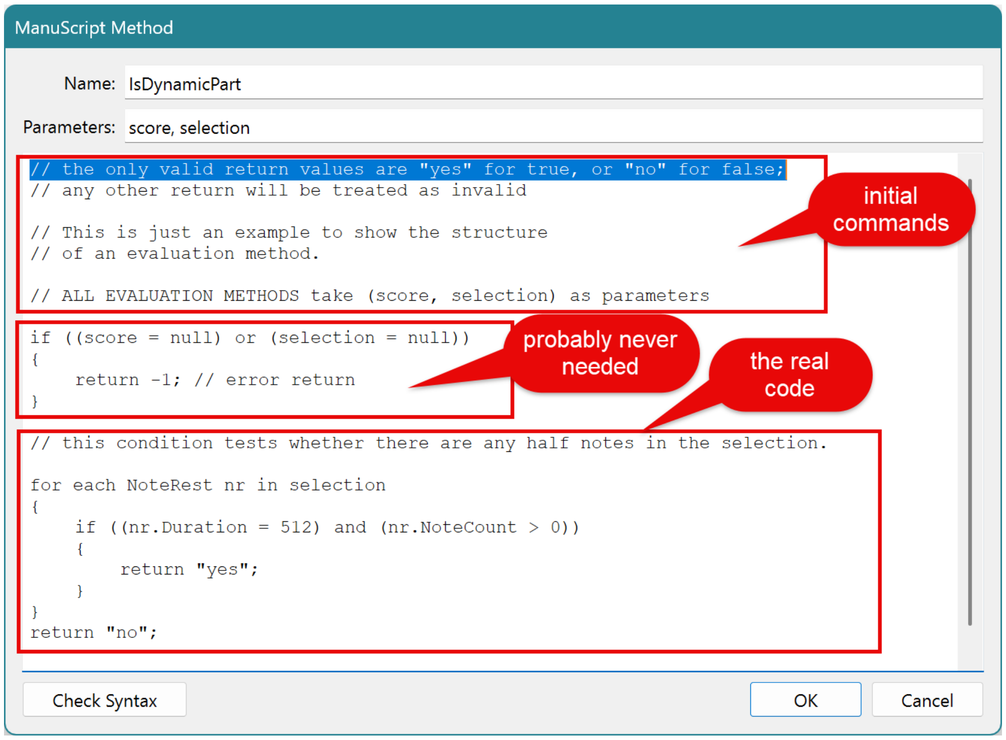
The first thing to note is that the method, as with all **Evaluation Methods**, takes the parameters "score, selection". The score is always going to be the same as **score.ActiveScore**, and selection will pretty much always be **score.Selection**. **Evaluation Methods** can always access the currently active score (or part, or subset) and selection but should not change either of them.

As a precaution, when **Evaluation Methods** are called in **Evaluate Plugin Condition**, the selection is saved before calling the **Evaluation Method** and restored on return.

This lets the method access the current score and selection. Most Sibelius commands are applied to the current selection as well.

We need to look at all the code now to see what code we can reuse and what we should get rid of.

The first 2 blocks are going to be in any of the **Evaluation Methods** that I have written, but you don't really need any of this. The method can only return **"yes"** or **"no"** as valid return values. They need to be in double quotes and lower case. Anything else is treated as an error. If the method fails for some reason, I usually return -1, with or without quotes, but any non-valid return value should work.

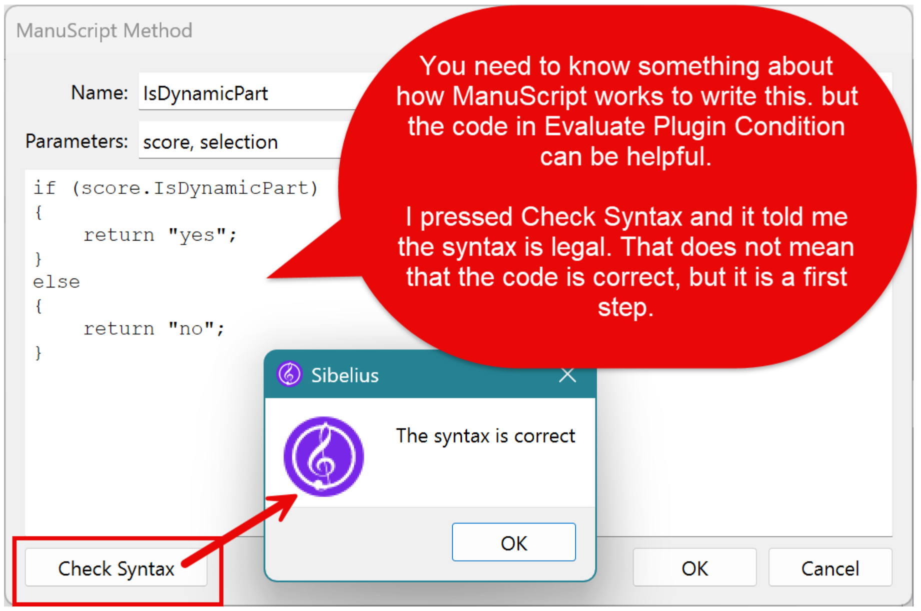


```
// the only valid return values are "yes" for true, or "no" for false;
// any other return will be treated as invalid

// This is just an example to show the structure
// of an evaluation method.

// ALL EVALUATION METHODS take (score, selection) as parameters

if ((score = null) or (selection = null))
{
    return -1; // error return
}

// this condition tests whether there are any half notes in the selection.

for each NoteRest nr in selection
{
    if ((nr.Duration = 512) and (nr.NoteCount > 0))
    {
        return "yes";
    }
}
return "no";
```

In this example, we only need the name and the parameters, so select and delete everything in the body of the window.

Now we need to write the actual code for the condition, and how to do that is beyond the scope of this document. I will take some code from another plugin, which returned **True** or **False.** I need to change these to **"yes"** and **"no"**. I was lucky – this condition can be written very easily, because there is a field in the score object called **IsDynamicPart**.

You can't evaluate the code yet but press Check Syntax to see if there are obvious errors and fix those until Check Syntax finds no errors.


ManuScript Method
Name: IsDynamicPart
Parameters: score, selection
You need to know something about how ManuScript works to write this. but the code in Evaluate Plugin Condition can be helpful.
I pressed Check Syntax and it told me the syntax is legal. That does not mean that the code is correct, but it is a first step.
Sibelius
The syntax is correct
OK
Check Syntax
OK
Cancel

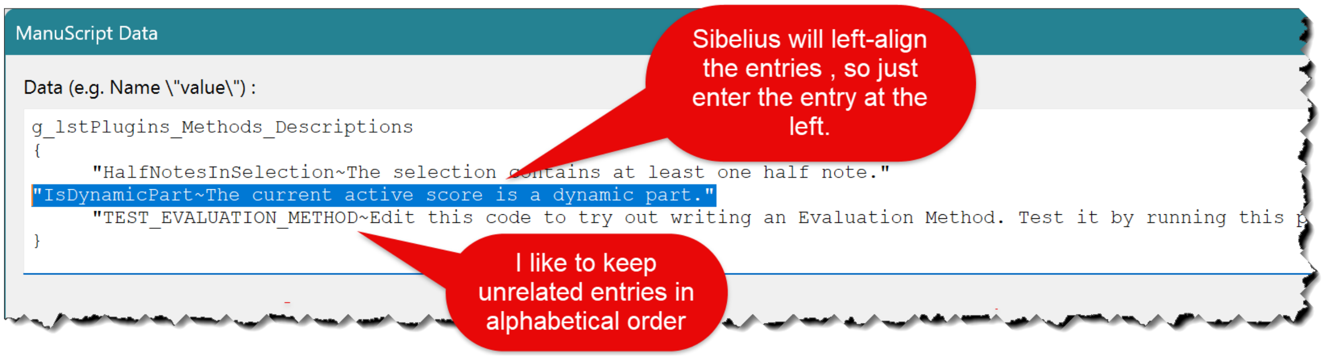
```
if (score.IsDynamicPart)
{
    return "yes";
}
else
{
    return "no";
}
```

Some syntax rules to observe:

- **if** statements put parentheses around conditions and there must always be a matched open and closed parenthesis.
- Statements end with **semicolons**(;).
- Single and double quotes must always be paired.
- Use // to comment out a line of code.
- Blocks of code (here just single lines) are in closed in pairs of braces ({ }).
    - I put the braces on separate lines so I can visualize the entire block at once. It is also common practice to put the open **{** at the end of the first line and the closing **}** at the end of the final block, which saves vertical space. Either design is fine.
- One quirky **ManuScript** thing to note. Do not use a comment (such as // I am a comment) as the last line of a method. This causes errors and the error messages do not help.

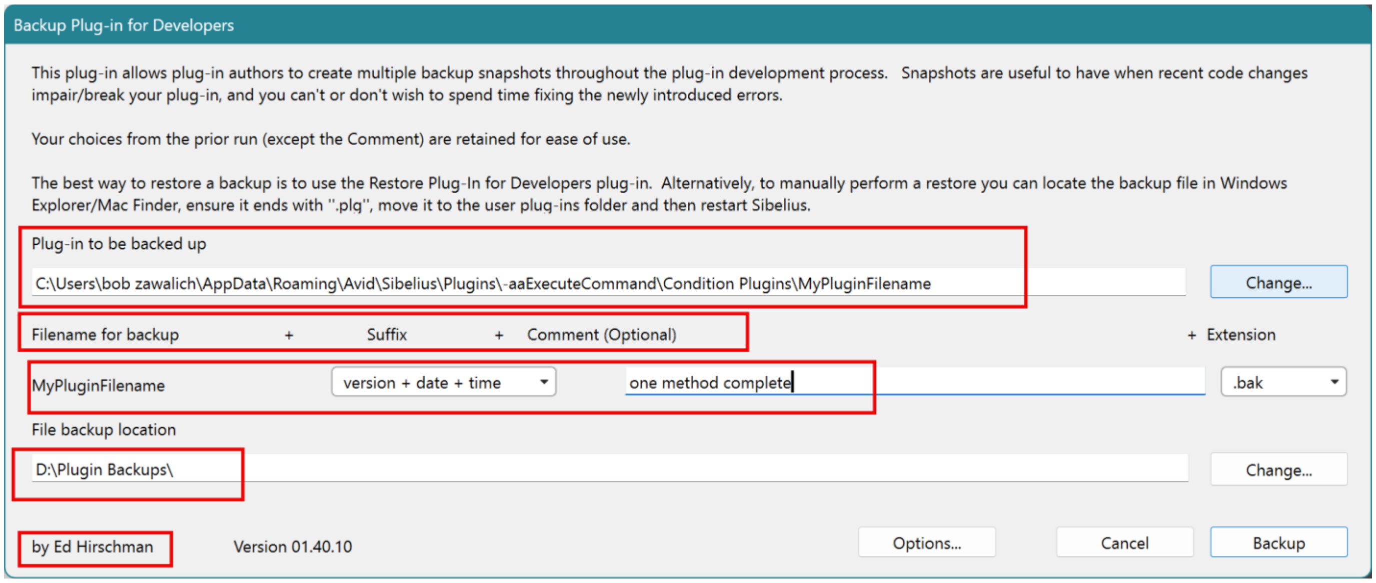
Adding a method name and description in **g_lstPlugins_Methods_Descriptions**

Now we need to add an entry for this method in **g_lstPlugins_Methods_Descriptions.** Close the **Method** edit window, and double-click on **g_lstPlugins_Methods_Descriptions** in the **Data** panel. Press **Enter** at the end of a line to open a spot for the new entry, choose a description, and type the entry.


ManuScript Data
Data (e.g. Name \"value\") :
Sibelius will left-align the entries , so just enter the entry at the left.
I like to keep unrelated entries in alphabetical order

```
g_lstPlugins_Methods_Descriptions
{
     "HalfNotesInSelection~The selection contains at least one half note."
"IsDynamicPart~The current active score is a dynamic part."
     "TEST_EVALUATION_METHOD~Edit this code to try out writing an Evaluation Method. Test it by running this p
}
```

So, it looks good. Now would be a good time to make a backup of the plugin. I always use Ed Hirschman's installable plugin **Backup Plug-in for Developers** to make a copy of the plugin with the date and time appended to the filename, saved to a different drive. Use any backup method you want but please, back it up somewhere.



This plug-in allows plug-in authors to create multiple backup snapshots throughout the plug-in development process. Snapshots are useful to have when recent code changes impair/break your plug-in, and you can't or don't wish to spend time fixing the newly introduced errors.

Your choices from the prior run (except the Comment) are retained for ease of use.

The best way to restore a backup is to use the Restore Plug-In for Developers plug-in. Alternatively, to manually perform a restore you can locate the backup file in Windows Explorer/Mac Finder, ensure it ends with ".plg", move it to the user plug-ins folder and then restart Sibelius.

Plug-in to be backed up: C:\Users\bob zawalich\AppData\Roaming\Avid\Sibelius\Plugins\-aaExecuteCommand\Condition Plugins\MyPluginFilename

Filename for backup + Suffix + Comment (Optional) + Extension

MyPluginFilename | version + date + time | one method complete | .bak

File backup location: D:\Plugin Backups\

by Ed Hirschman   Version 01.40.10

Testing your plugin

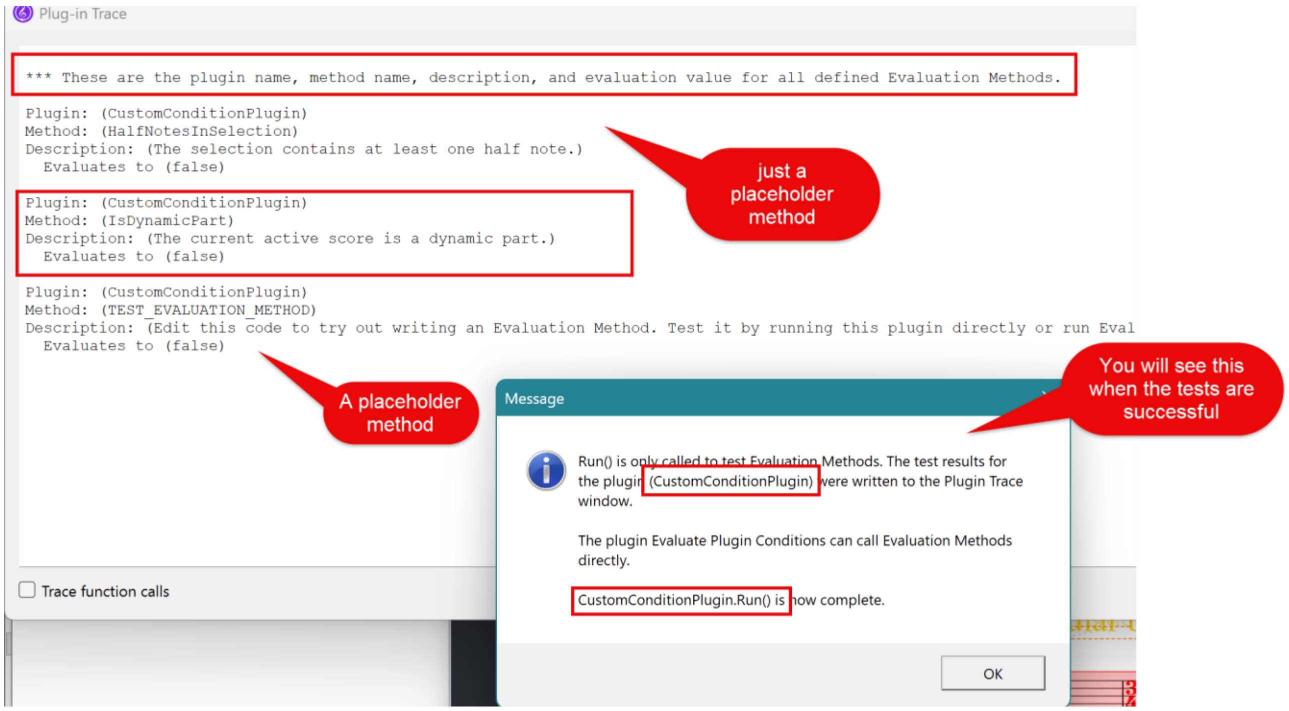
Here are 3 ways to test a plugin

1. Run the Condition Plugin.

The **Run()** method of **Condition Plugins** is set up so that it will run the code for each **Evaluation Method** in **g_lstPlugins_Methods_Descriptions. Run()** will be called if you run the plugin from a menu, shortcut, or plugin. This is really the best place to test the code because the other mechanisms run the **Condition Plugin** while a dialog box is up, and if there are syntax errors in such a called plugin, Sibelius can hang.

This is what the traced output might look like. You will see the **Evaluation Method** name and description and how it evaluates the condition using the current selection.

In this example, the 2 placeholder methods are included, and at some point you should delete them and remove their entries from **g_lstPlugins_Methods_Descriptions.**



Plug-in Trace
just a placeholder method
A placeholder method
You will see this when the tests are successful
Message
Trace function calls
OK

```
*** These are the plugin name, method name, description, and evaluation value for all defined Evaluation Methods.

Plugin: (CustomConditionPlugin)
Method: (HalfNotesInSelection)
Description: (The selection contains at least one half note.)
  Evaluates to (false)

Plugin: (CustomConditionPlugin)
Method: (IsDynamicPart)
Description: (The current active score is a dynamic part.)
  Evaluates to (false)

Plugin: (CustomConditionPlugin)
Method: (TEST_EVALUATION_METHOD)
Description: (Edit this code to try out writing an Evaluation Method. Test it by running this plugin directly or run Eval
  Evaluates to (false)
```

Run() is only called to test Evaluation Methods. The test results for the plugin (CustomConditionPlugin) were written to the Plugin Trace window.

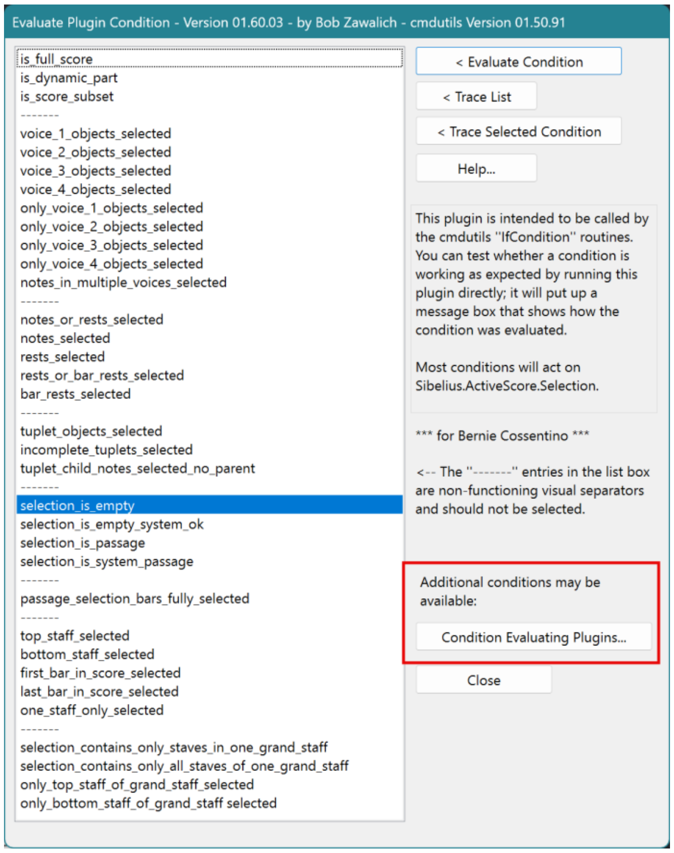
The plugin Evaluate Plugin Conditions can call Evaluation Methods directly.

CustomConditionPlugin.Run() is now complete.

I ran it on a full score, so it returns "no", which will be translated to **true** or **false** for this listing, which matches the way **Evaluate Plugin Condition** reports a condition. Try changing the open score to be a part or score subset. This routine should return false for full score and subsets, and true for dynamic parts.

2. Testing in *Evaluate Plugin Condition*

When you run **Evaluate Plugin Condition**, a big dialog comes up.


Evaluate Plugin Condition - Version 01.60.03 - by Bob Zawalich - cmdutils Version 01.50.91

```
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
```

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.

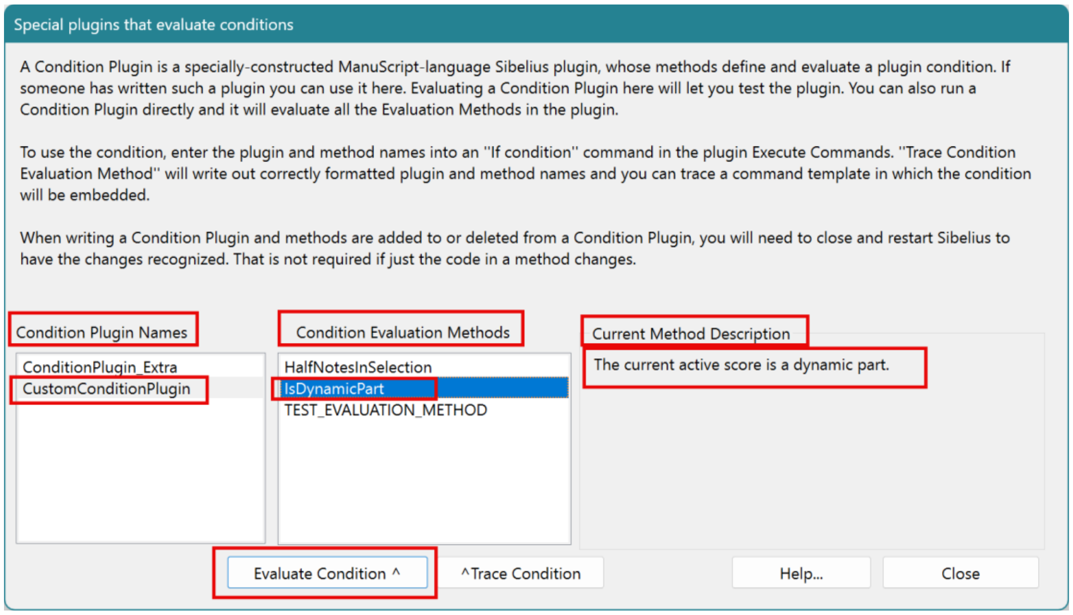
*** for Bernie Cossentino ***

<-- The "-------" entries in the list box are non-functioning visual separators and should not be selected.

Additional conditions may be available:

Assuming you have the most recent version, you will see a **Condition Evaluating Plugins...** button on the bottom right. This brings up another dialog, **which will likely be slow the first time you run it in a Sibelius session**. It will get faster. Find your plugin in the leftmost list box and choose an **Evaluation Method** in the second listbox. You should see the correct description on the right.

Press **Evaluate Condition** to test the method:


Special plugins that evaluate conditions

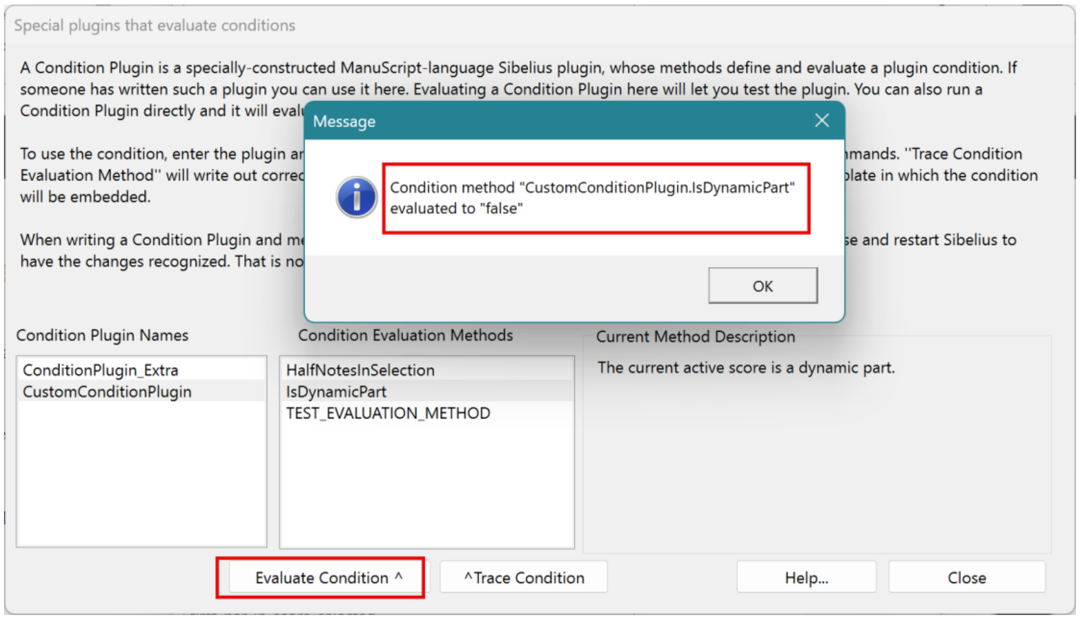
A Condition Plugin is a specially-constructed ManuScript-language Sibelius plugin, whose methods define and evaluate a plugin condition. If someone has written such a plugin you can use it here. Evaluating a Condition Plugin here will let you test the plugin. You can also run a Condition Plugin directly and it will evaluate all the Evaluation Methods in the plugin.

To use the condition, enter the plugin and method names into an "If condition" command in the plugin Execute Commands. "Trace Condition Evaluation Method" will write out correctly formatted plugin and method names and you can trace a command template in which the condition will be embedded.

When writing a Condition Plugin and methods are added to or deleted from a Condition Plugin, you will need to close and restart Sibelius to have the changes recognized. That is not required if just the code in a method changes.

Condition Plugin Names	Condition Evaluation Methods	Current Method Description
ConditionPlugin_Extra	HalfNotesInSelection	The current active score is a dynamic part.
CustomConditionPlugin	IsDynamicPart	
	TEST_EVALUATION_METHOD	

The plugin and **Evaluation Method** names should be displayed, along with "true" or "false" (The Evaluation Method actually returned "yes" or "no"but the testing routines map those to "true" or "false".


Special plugins that evaluate conditions

A Condition Plugin is a specially-constructed ManuScript-language Sibelius plugin, whose methods define and evaluate a plugin condition. If someone has written such a plugin you can use it here. Evaluating a Condition Plugin here will let you test the plugin. You can also run a Condition Plugin directly and it will eval

To use the condition, enter the plugin ar Evaluation Method" will write out corre will be embedded.

mands. "Trace Condition late in which the condition

When writing a Condition Plugin and m have the changes recognized. That is no

e and restart Sibelius to

Message

Condition method "CustomConditionPlugin.IsDynamicPart" evaluated to "false"

OK

Condition Plugin Names	Condition Evaluation Methods	Current Method Description
ConditionPlugin_Extra	HalfNotesInSelection	The current active score is a dynamic part.
CustomConditionPlugin	IsDynamicPart	
	TEST_EVALUATION_METHOD	

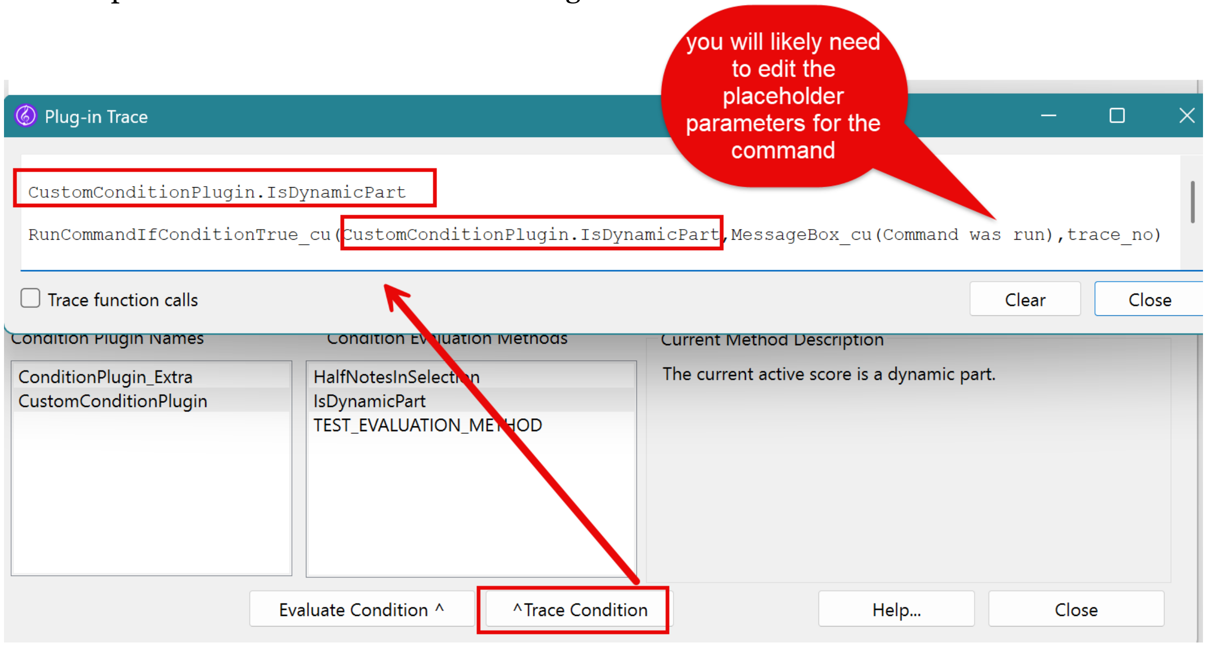
Test each of the Methods you have changed, with different types of selections until you are satisfied that it all works.

3. Testing in Execute Commands

The last way to test a **Condition Plugin** is to use its name and method (<name>.<method>) as a parameter in an "**If condition**" command in **Execute Commands**.

To make it easier to set up the command, you can run **Execute Plugin Condition** and press **Condition Evaluating Plugins…**. You will now see a button labeled **^Trace Condition**.

Pressing this will write out the plugin and method names to the plugin **Trace Window**. It will also put up another dialog giving you a list of **cmdutils** commands that support conditions. Choose the one you want to use and press OK. You should see something like this:

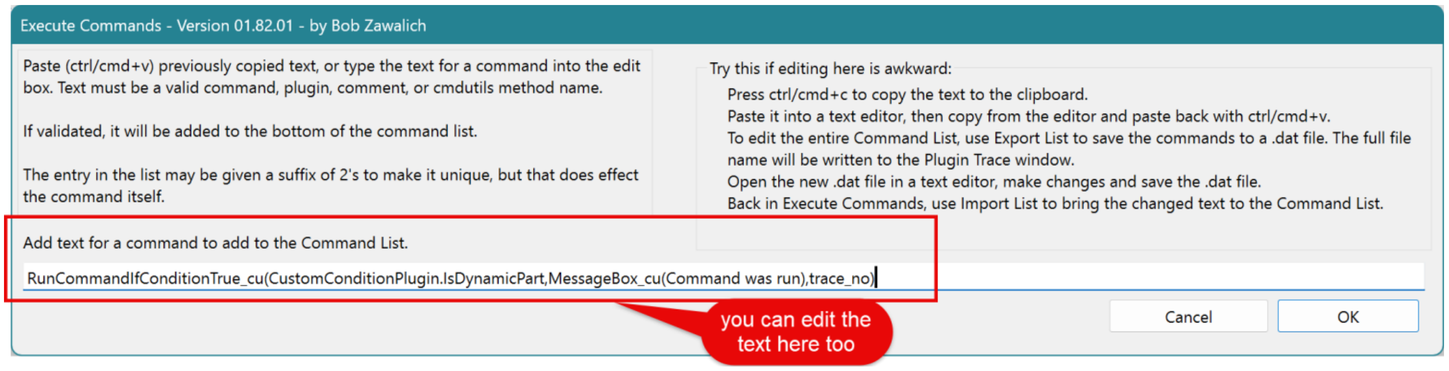

you will likely need to edit the placeholder parameters for the command
Plug-in Trace

```
CustomConditionPlugin.IsDynamicPart

RunCommandIfConditionTrue_cu(CustomConditionPlugin.IsDynamicPart,MessageBox_cu(Command was run),trace_no)
```

Trace function calls
Clear
Close

Condition Plugin Names	Condition Evaluation Methods	Current Method Description
ConditionPlugin_Extra	HalfNotesInSelection	The current active score is a dynamic part.
CustomConditionPlugin	IsDynamicPart	
	TEST_EVALUATION_METHOD	

You can edit the text in the trace window. Copy the command you want to run with ctrl/cmd+c, then run **Execute Commands** and press **Paste Command…** Paste into the edit box with ctrl/cmd+v, and it will go into the **Command List**. Use the **Up** and **down** buttons to position the commands, then run **Execute Command List**.



Execute Commands - Version 01.82.01 - by Bob Zawalich

Paste (ctrl/cmd+v) previously copied text, or type the text for a command into the edit box. Text must be a valid command, plugin, comment, or cmdutils method name.

If validated, it will be added to the bottom of the command list.

The entry in the list may be given a suffix of 2's to make it unique, but that does effect the command itself.

Try this if editing here is awkward:

Press ctrl/cmd+c to copy the text to the clipboard.
Paste it into a text editor, then copy from the editor and paste back with ctrl/cmd+v.
To edit the entire Command List, use Export List to save the commands to a .dat file. The full file name will be written to the Plugin Trace window.
Open the new .dat file in a text editor, make changes and save the .dat file.
Back in Execute Commands, use Import List to bring the changed text to the Command List.

Add text for a command to add to the Command List.

```
RunCommandIfConditionTrue_cu(CustomConditionPlugin.IsDynamicPart,MessageBox_cu(Command was run),trace_no)
```

Test the conditions for real. Good luck with that. Give it to a friend to try out if you can.

Appendix 1: Adding New Conditions By Editing the plugin Evaluate Plugin Conditions

Additional conditions could be added to **Evaluate Plugin Condition** by editing the ManuScript code of the plugin. I recommend using Condition Plugins instead, but I had already written these instructions, so here they are.

Be aware that it is very easy to mess up **Evaluate Plugin Condition** and there is no support for what you might do. You are also not allowed to distribute the modified plugin, since the code is copyrighted.

I would not suggest trying this unless you are an experienced ManuScript programmer, and I think that **Condition Plugins** are a much better mechanism than editing **Evaluate Plugin Condition.**

Nevertheless…

A new condition can be added by editing the plugin **Evaluate Plugin Condition**

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 the plugin file for **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).

Appendix 2: Cmdutils commands 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.)

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

 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)
```

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.