Write your own simple Sibelius plugin

Bob Zawalich October 13, 2019

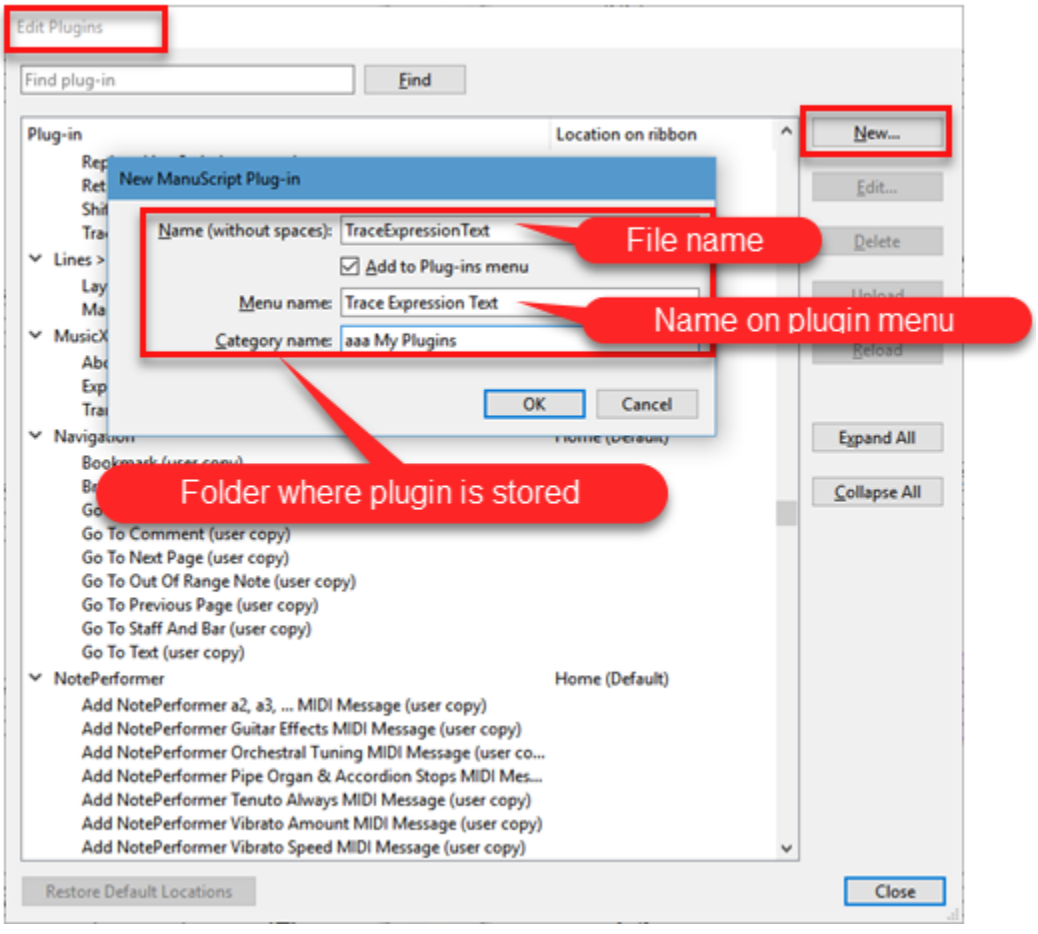
Once you have the general idea of how plugins work, as described in earlier chapters, you can often write very useful plugins with just a few lines of code, most of which will be the same from plugin to plugin. I have published the plugin template **Custom Simple Plugin** (in category **Developers' Tools**), which can be copied and modified easily, but you can also make a new plugin from scratch in very few steps.

Since the plugin editor and ManuScript programming language are contained within Sibelius, you do not need any other software to create, edit, and run a plugin.

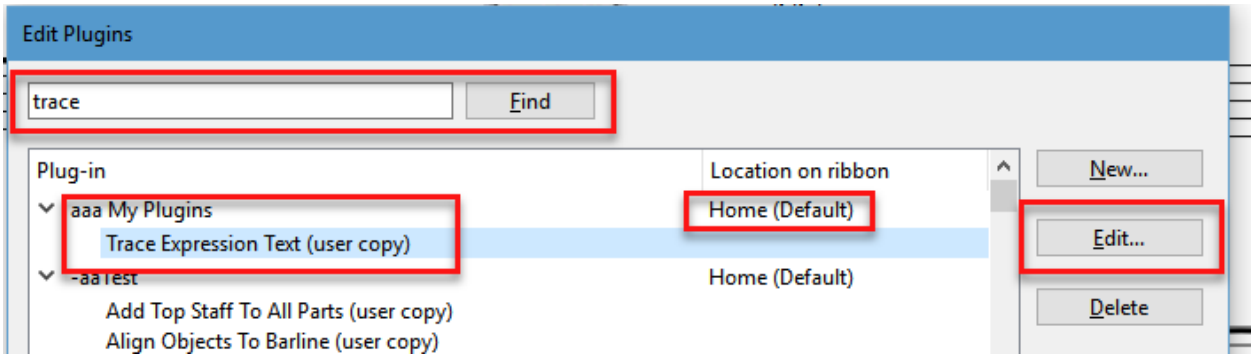
These instructions assume you are using Sibelius version 7.1.3 or later (or any version of Sibelius Ultimate). This will not teach you how to write software; it assumes you know the basics of programming in other languages, such as BASIC, or C++, or JavaScript, or Python.

Creating a plugin from scratch

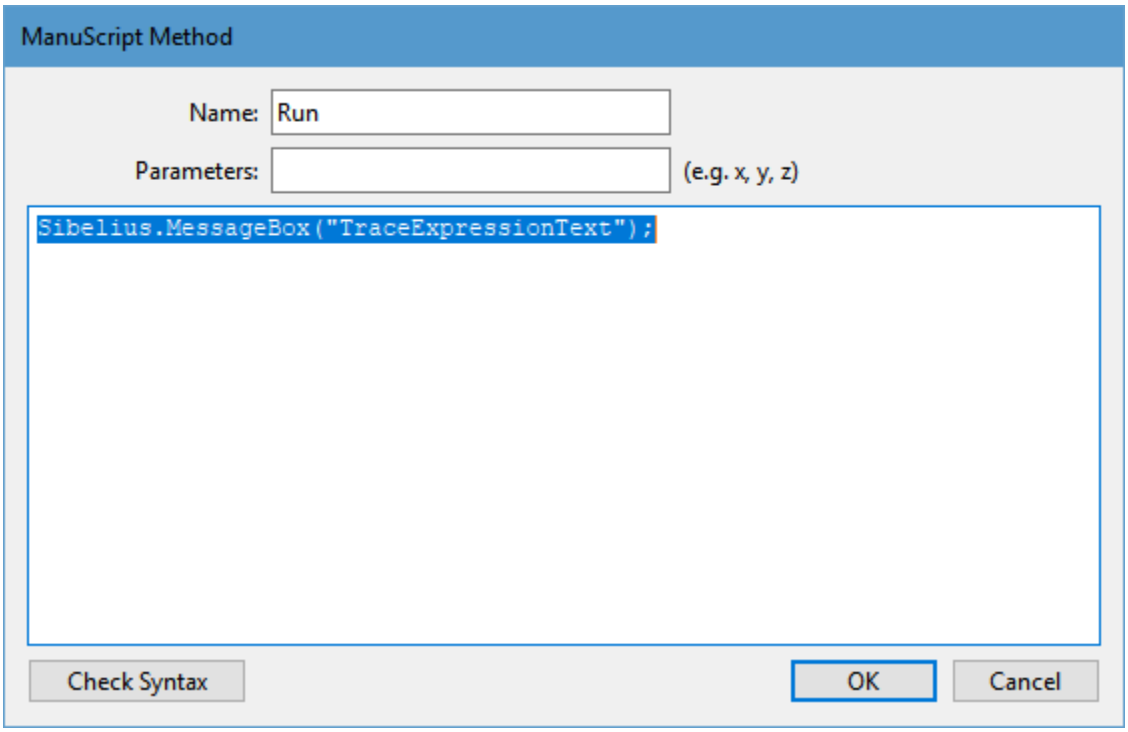
- Go to **File>Plug-ins>Edit Plug-ins**
- Click on **New**, and choose a name and location for your plugin.
    - The location could be an existing plugin category/folder, but I find it useful to have a separate category for private plugins.
    - I like to name my private folder so that it will appear at the start of the alphabetical list in **Edit Plugins**, so **aaa My Plugins** could be a reasonable choice. The first time you choose a new **Category name** the folder will be created for you in the correct location.
    - I recommend always using the same name for the **Name** and **Menu name**, with spaces added between words for the **Menu name**.



- Click OK in the new dialog, then you will need to find your plugin so you can edit it. Type the start of the menu name in the **Find** box, and click **Find** until you have selected your plugin. Now click on **Edit**...



- This will bring up the plugin editor with your plugin loaded. It will have 2 methods:
  - **Initialize**, which adds your plugins to the plugin menus when Sibelius first loads up. It has a single line and you will not need to touch it unless you want to change the menu name. In this example the line is:
  - AddToPluginsMenu("Trace Expression Text","Run");
  - **Run,** which is called when you start a plugin, and is where most of your code lives. Choose **Run**, then click **Edit**. When the editor comes up, click the **Delete** key to remove the original sample line and we are ready to begin.

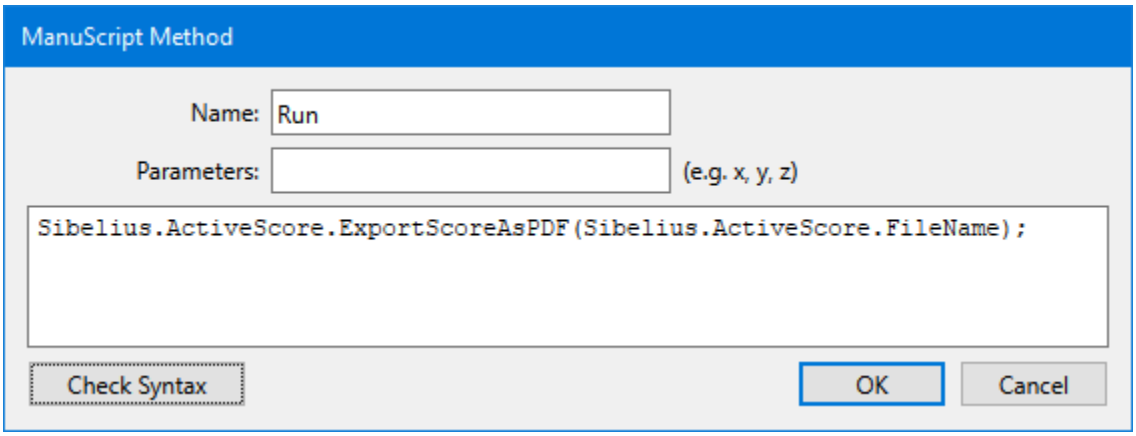


Writing a plugin that does something simple but useful

Sometimes you can accomplish something useful with very little code. If you replace the contents of the Run method with this single line of code:

```
Sibelius.ActiveScore.ExportScoreAsPDF(Sibelius.ActiveScore.FileName);
```

and assign the plugin to a shortcut, then typing the shortcut will export the full score that you are working on as a PDF file, even if you are currently working on a part. If you export your score to PDF frequently, this will accomplish the task, overwriting any previously exported file, with no dialog and very little interruption. There are existing plugins that can give you more options, but sometimes this is exactly what you want to do, and it is very easy to do it.



Writing a plugin that examines or modifies (but does not add or delete) selected objects.

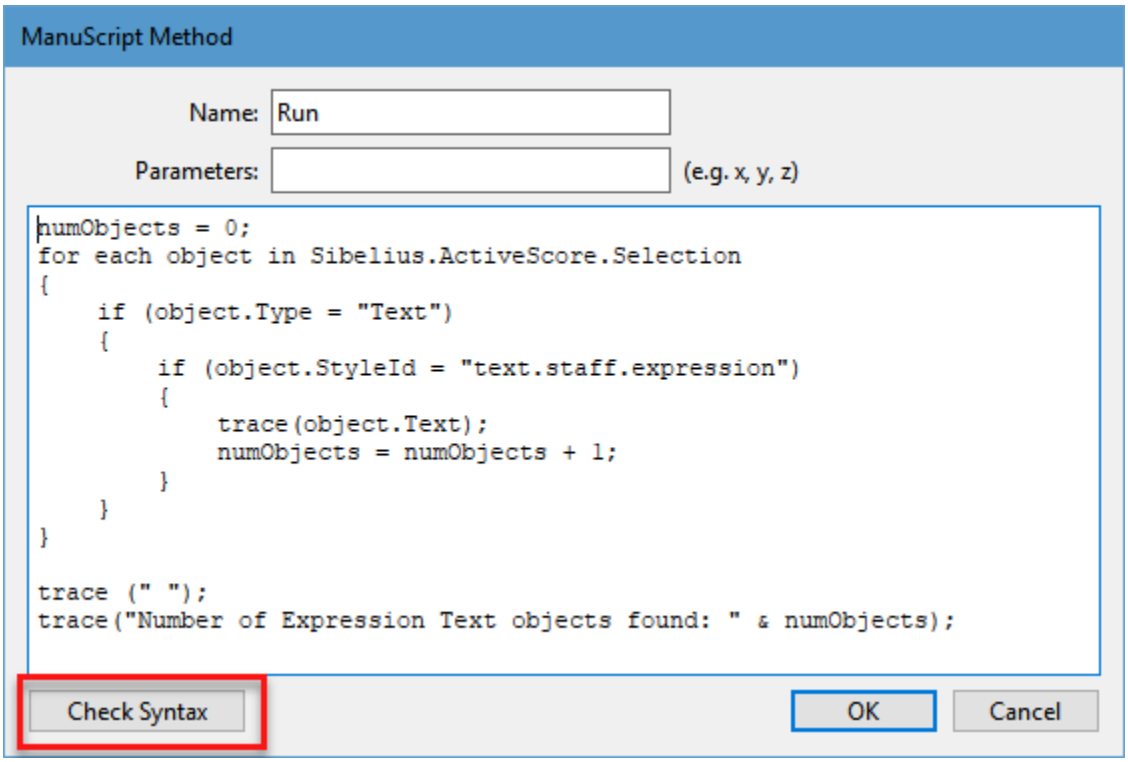
How you determine the code needed to implement a plugin is discussed elsewhere, but to see examples of working code, you can use **Edit Plugins** to look at the code of any installed plugin. The **ManuScript Language Reference,** found at **File>Plug-ins> ManuScript Language Reference** describes the syntax of the language, and provides some tutorials.

A common task is to see what is in your score, so let's write a plugin that looks for **Expression** text in a selection and writes the text to the plugin **Trace Window**, along with the number of objects found. We will want to use a **for each** loop for this. This will look in the selection and pass back each object that passes a test. We cannot add or delete objects in a **for each** loop, but examining objects work fine.

The minimum code for such a plugin would look like this:

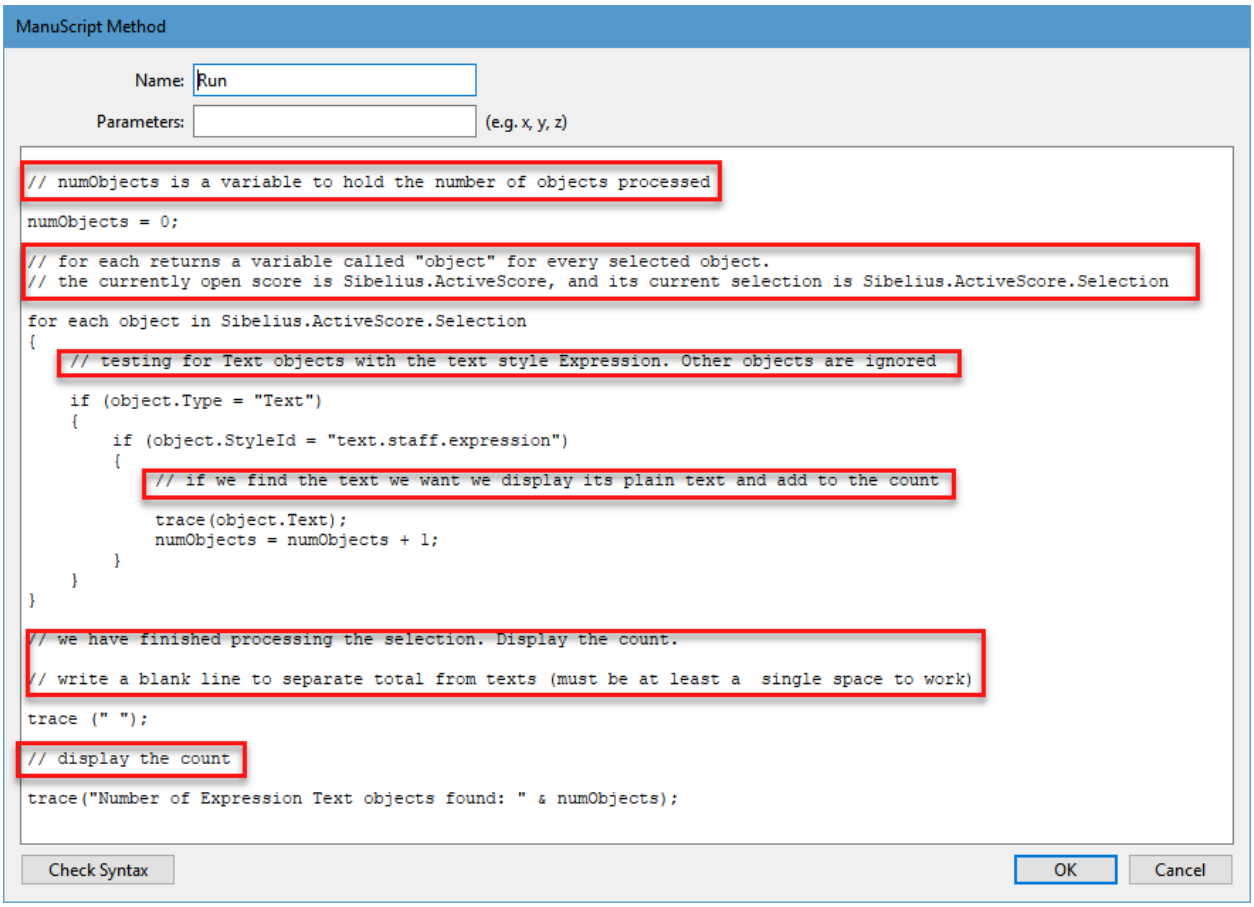
```
for each object in Sibelius.ActiveScore.Selection
{
	// do something…
}
```

Here is some code that looks for and counts **Expression Text** objects, and writes out their text and the number of objects found:

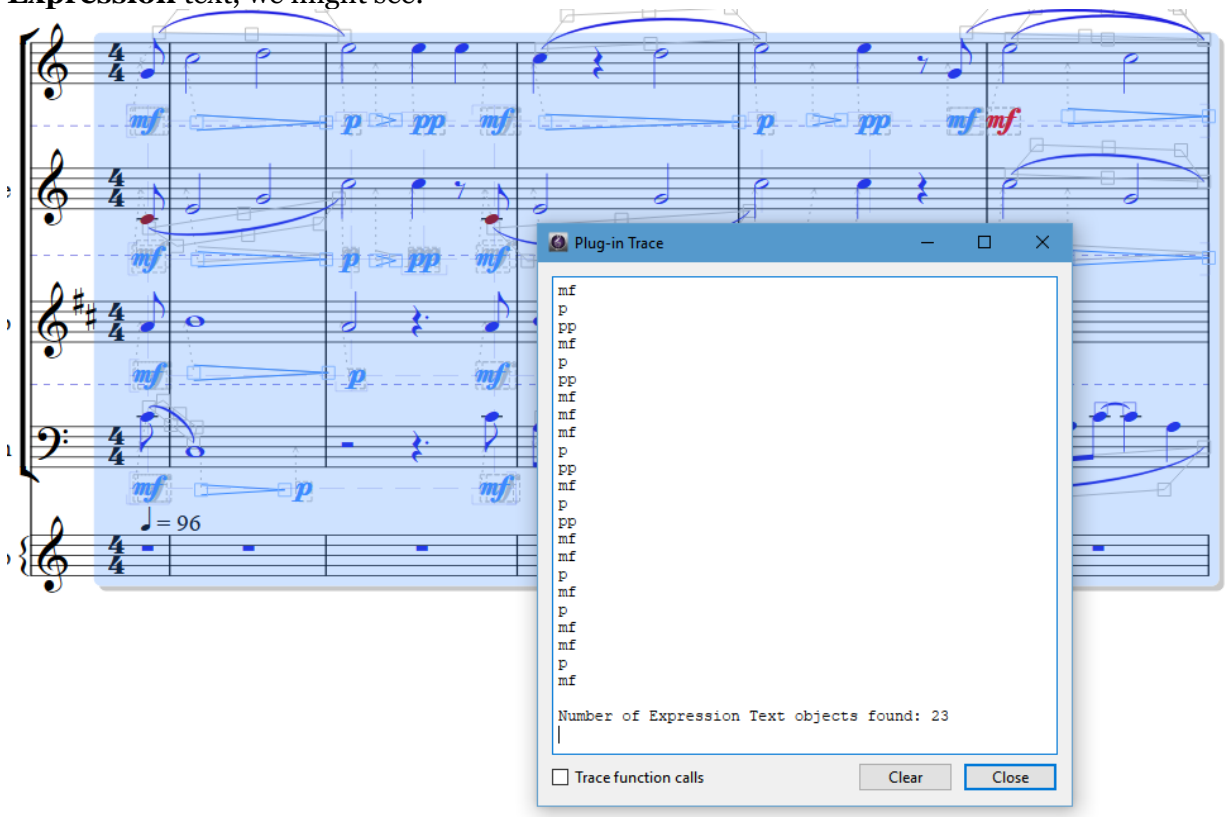


Be sure to use the **Check Syntax** button to find simple errors before you try running the code.

For now, let's briefly discuss what this code is doing. Here is the same routine with comment lines (starting with //) added:



The plugin is in the folder **aaa My Plugins**, which was added to the **Home> Plug-ins** menu by default. If we run **Home>Plugins>aaa My Plugins> Trace Expression Text** on a selection in a score that contains **Expression** text, we might see:

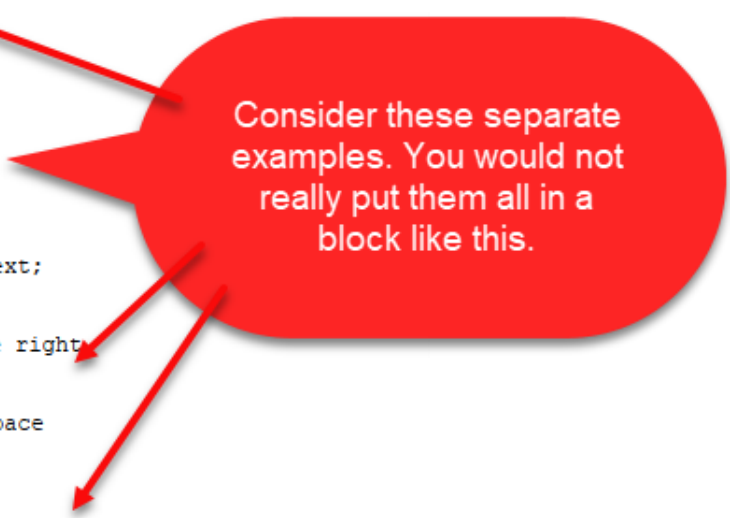


It is not elegant output, but it could give you what you needed without a great deal of work.

Changing properties of selected objects

Instead of just tracing the **Text** field of the chosen objects, we might want to change some of its properties. Here are a few examples of things you might want to do:

```
if (object.Type = "Text")
{

    // some examples of things you can do at this point

    // Color desired objects red
    if (object.StyleId = "text.staff.expression")
    {
        object.ColorRed = 255;
        object.ColorGreen = 0;
        object.ColorBlue = 0;
    }

    // change the text to include the number
    if (object.StyleId = "text.staff.expression")
    {
        numObjects = numObjects + 1;
        object.Text = numObjects & " - " & object.Text;
    }

    // shift the horizontal position 2 spaces to the right
    if (object.StyleId = "text.staff.expression")
    {
        dxOneSpace = 32; // one unit is 1/32 of a space
        object.Dx  = object.Dx + (2 * dxOneSpace);
    }

    // change the text style to be Technique and reposition the text vertically
    if (object.StyleId = "text.staff.expression")
    {
        object.StyleId = "text.staff.technique";
        object.ResetPosition(False, True);
    }
}
```



A simple general-purpose plugin

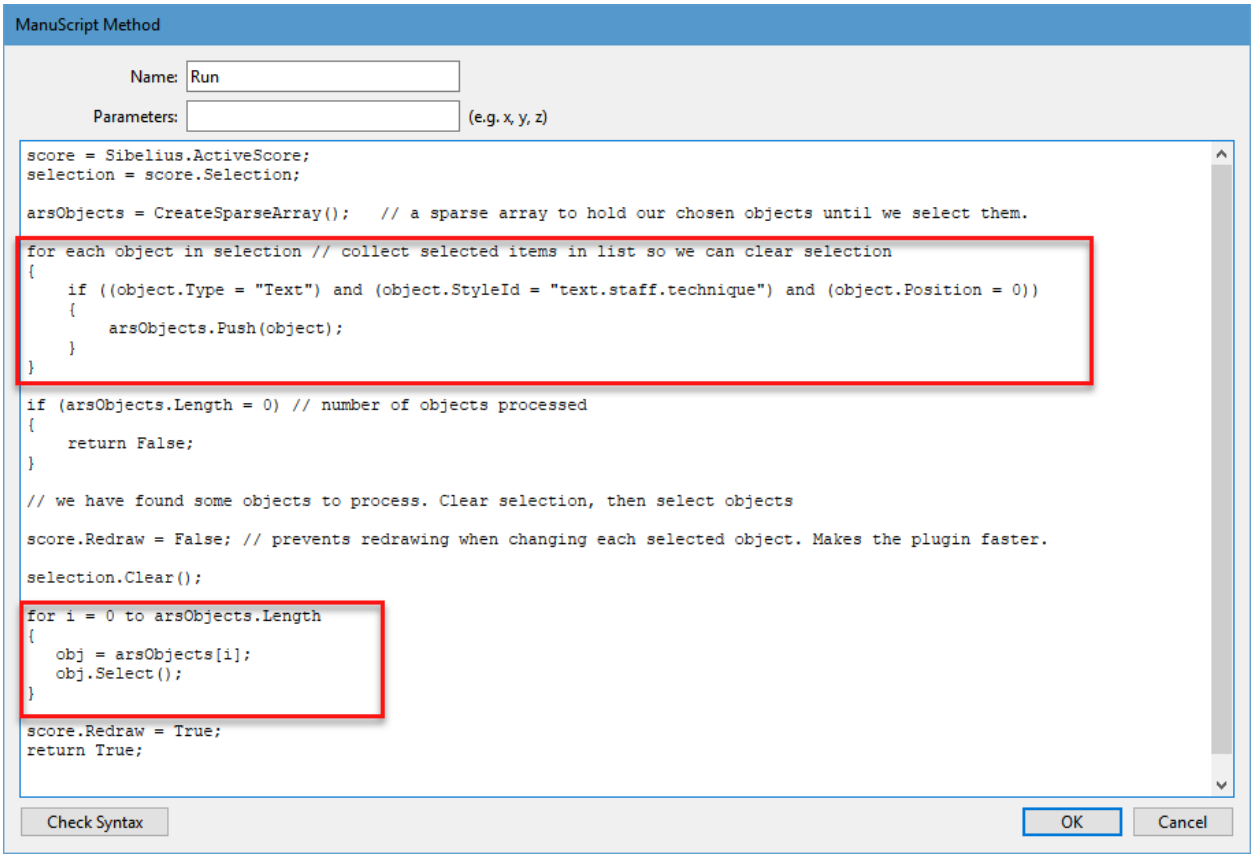
If we want to find certain objects and then add or delete new objects, or want to change the selection, we can't do that inside a **for each** loop, because that would disrupt the pool of objects that **for each** is looking at. In such cases a useful technique is to have 2 loops:

- A **for each** loop that finds the objects you want to process, and stores those objects in an array
- A separate **for** loop that processes the stored objects.

If you are a programmer you may think this sounds inefficient, but in practice a **for each** loop that makes no changes can be quite fast, and the second loop will have to look at fewer objects, so such plugins are fast enough except for very large scores, where you may need to optimize.

Here is an example of a plugin that will filter only the selected **Technique** text that is at the start of a bar. Since a filter needs to change the selection, and the **for each** loop processes the selection, we gather up the text first, and then clear the original selection and select the objects we found.

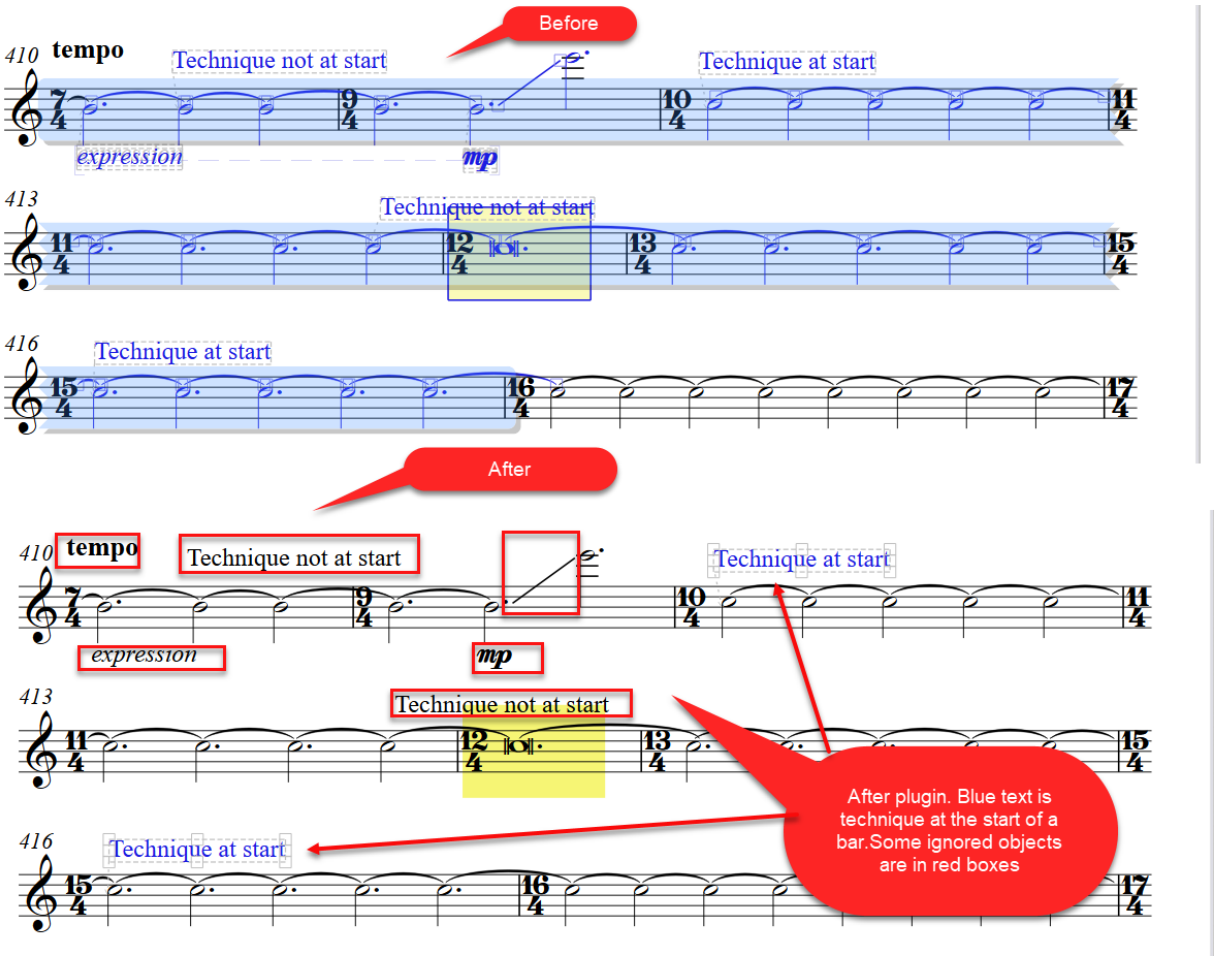
Create a new plugin as before, and make this the **Run** method:



The 2 loops are shown in red. The first loop chooses the objects to select, and if they pass the test, they are stored in the **Sparse Array** *arsObjects*. A **Sparse Array** (only available in Sibelius 6 or later) can hold any kind of object, and so is a good storage choice compared to an **Array**, which can only hold numbers or strings.

Here is a brief description of what this does.

- We create the **score** and **selection** variables to avoid some typing when we use them, and create the sparse array
- We have a **for each** loop which looks for 3 things
  - Is it a **Text** object?
  - Is the text style **Technique**?
  - Is the **Text** object located at the start of the bar?
- If the conditions are all met, the object is added to the end of the **Sparse Array** using the **Push** command.
- (Instead of the "nested if statements" I used in the first example, this **if statement** has 3 conditions separated by the word **and**. It will only succeed if all 3 conditions are true. We need to carefully separate the conditions with parentheses so they evaluate correctly. When I need to write such a statement, I often lay out the parentheses first, as in: if (() and () and ()), and fill in the tests within their appropriate parentheses. Getting these things to use parentheses correctly is quite tricky).
- If an object is put in a **Sparse Array**, the variable **Length** gets incremented, so we test to see if we found anything by checking that *arsObjects.Length* is not zero.
- Since we know we have found something, we set **score.SetRedraw** to **False** to speed things up by telling Sibelius not to redraw the screen until we are done. We clear the selection so that when we add the found objects to the selection there is nothing there except what we want.
- The second **for** loop gets each stored **Text** object from *arsObjects* and selects it. We reset **score.Redraw**, and we are done.
- Some testing shows we get what we wanted. Be sure to test with real scores as well as simple tests, and always include things you want to ignore as well as what you want to find in a test score.
- In the second loop we can do almost anything allowable and we will not mess up the for **each loop**.



Running the plugin and using keyboard shortcuts

If you use **Edit > Plugins > New** to create the plugin, you should be able to run it immediately. If you have copied and changed an existing file, you will typically need to close and restart Sibelius in order to edit it. Also, adding plugin files outside of Sibelius can mess up keyboard shortcuts to other plugins. Fortunately, there is an easy way to fix that. Go to **File>Preferences>Keyboard Shortcuts**, and immediately click on OK. This is a Sibelius bug that was present until at least Sibelius Ultimate 2019.9.

Any plugin can be assigned its own keyboard shortcut. You can also use plugins like **Run Plugin By Name** or **My Plugins** to make it easier to find and run other plugins.

Debugging a plugin

There are few tools available for debugging; there is no source level debugger, for example. The only effective tools are the **trace()** command and **Sibelius.MessageBox()**, which can be used to display the current values of variables, and the **Trace function calls** checkbox in the **Plugin Trace Window**. In plugins I have written, I often comment out **trace** commands, rather than removing them, so I can reactivate them if I have to return to code and see what is going on.

This is a simple plugin

If you compare the code in published plugins to the code we are using here, you will see that this is much simpler. We are not checking for errors or warning if you don't select anything. We don't care about having the plugin be easily translatable. The code presents no explanation of what you need to do to run the plugin, because you write it for your own use and it is not likely to be used by anyone else. Personal plugins can do just enough to get you what you need, and once you are familiar with the **ManuScript** language, you can write or modify a plugin that can do exactly what you need to do without needing someone to write it for you.

A lot of published plugins are quite complex and handle strange conditions that you can ignore if they are not something you will encounter, but which are needed for general use. There is a lot you can do with a plugin that only requires a few lines of code, and it can be well worth learning the basics of writing such plugins.

Happy programming!