How to create your own filtering plugin for Sibelius 6.2 or later

Bob Zawalich October 23, 2019

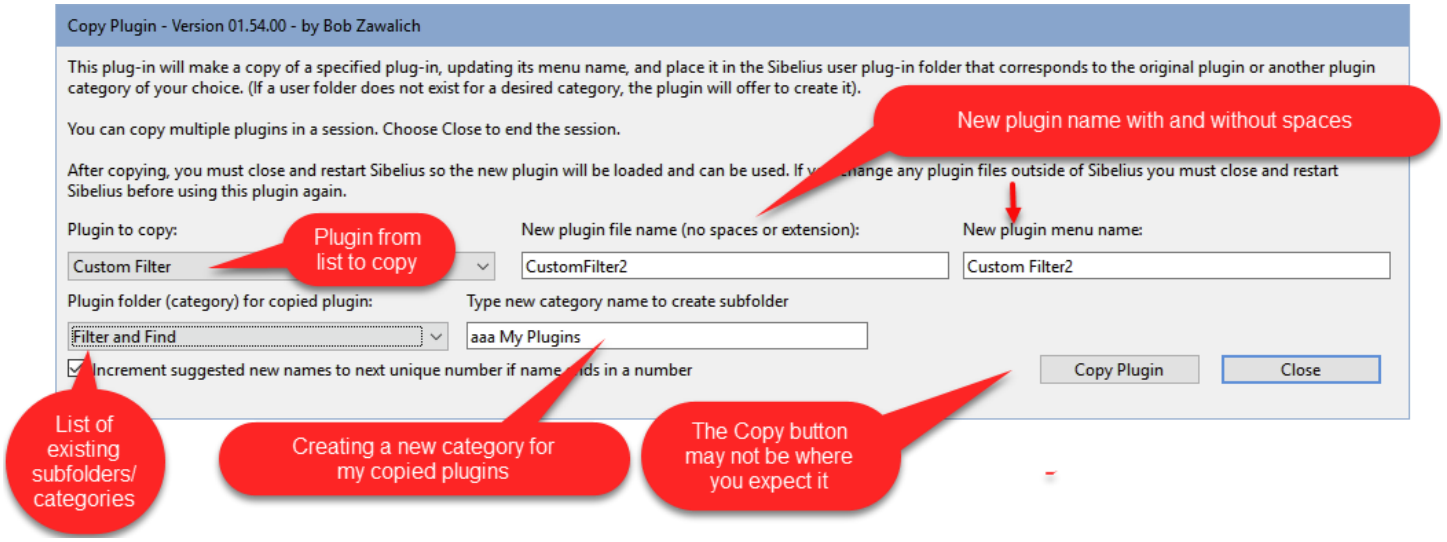
The downloadable plugin **Custom Filter** (category **Filter and Find**) is a user-modifiable plugin intended to be used for creating custom filters. The intention of the plugins is that you should make a copy of the plugin, change the internal plugin name, and install it, then change the code in the Method **IsDesiredObject** so that it chooses which objects you wish to filter.

The code in **Custom Filter** (as of version 01.50.00) is in the public domain (unlike most downloadable plugins whose code is copyrighted by the author), so you may change code as you like and reuse the code for any use you desire. There is no support for using this code, and you use it at your own risk, but it is free to use.

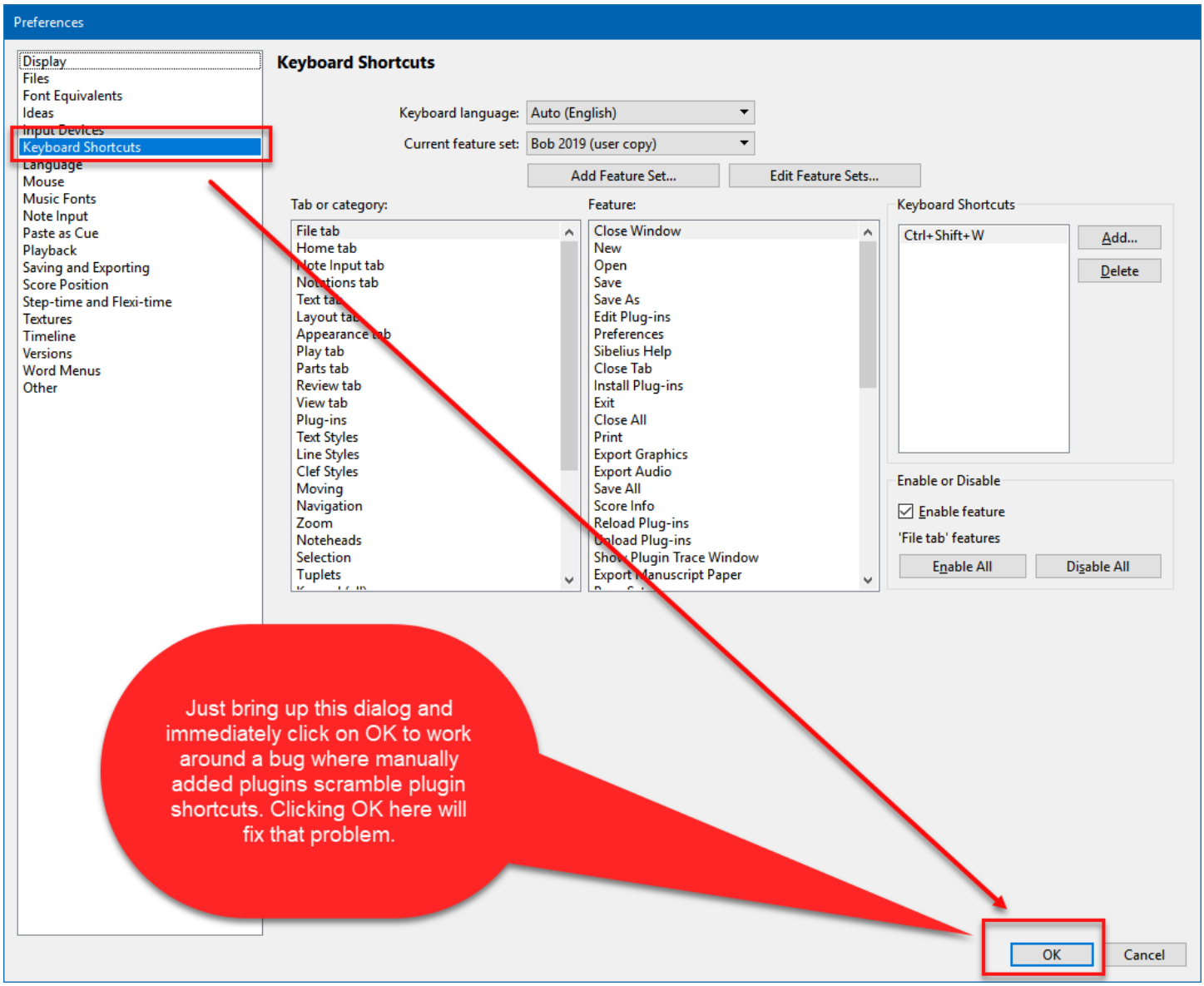
This plugin as written can only be run in Sibelius 6.2 or later. The instructions that follow reference the plugin Installer, which is only available in Sibelius 7.1.3 or later.

Here is a set of steps I recommend using for setting up a custom plugin:

1. Using the Installer, install the plugins **Custom Filter** (category **Filter and Find**) and **Copy Plugin** (category **Developers' Tools**).
2. Run **Copy Plugin**, choose **Custom Filter** from the list of installed plugins as the plugin to copy, and choose a unique name for your new plugin. Be sure to click on the **Copy Plugin** button, not **Close**. Choose a convenient category/plugin subfolder for your new plugin. In this case I created a new subfolder called **aaa My Plugins**, which will appear at the top of the list of categories so it will be easy to find. You could copy multiple plugins in the same plugin session, but we only need 1, so after copying, click on **Close**.

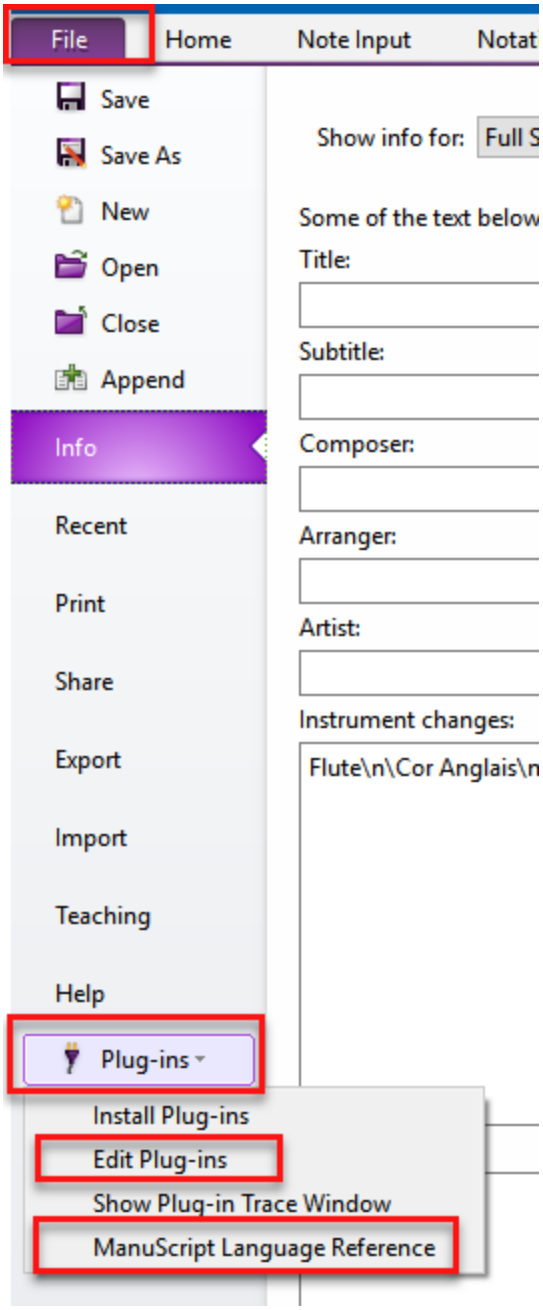


3. This will make a copy of **CustomFilter.plg**, changing the name of the file and the internal plugin menu name within the file, placing the new plugin file in the specified plugin subfolder.
4. Now you need to take 2 additional steps before editing your new plugin:
    a. Close and restart Sibelius, which will cause your plugin to be loaded into memory and made available to Sibelius.
    b. Click on the **File** tab, choose **Preferences>Keyboard Shortcuts**, and immediately choose **OK**. This fixes a Sibelius bug that scrambles plugin shortcuts when new plugins are added manually (or in our case, via a plugin).

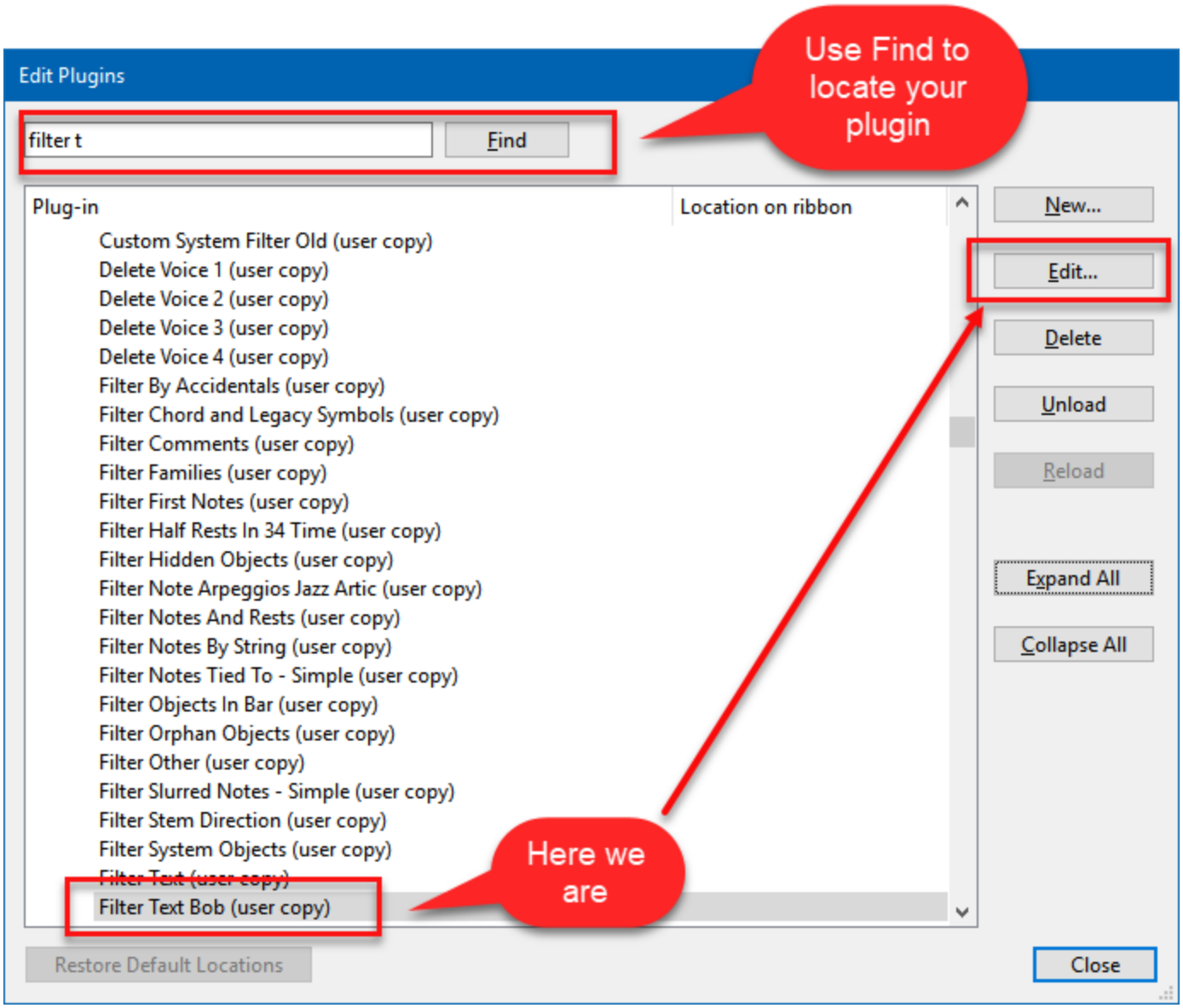


At this point the new plugin is equivalent, in Sibelius' eyes, to any other installed plugin.

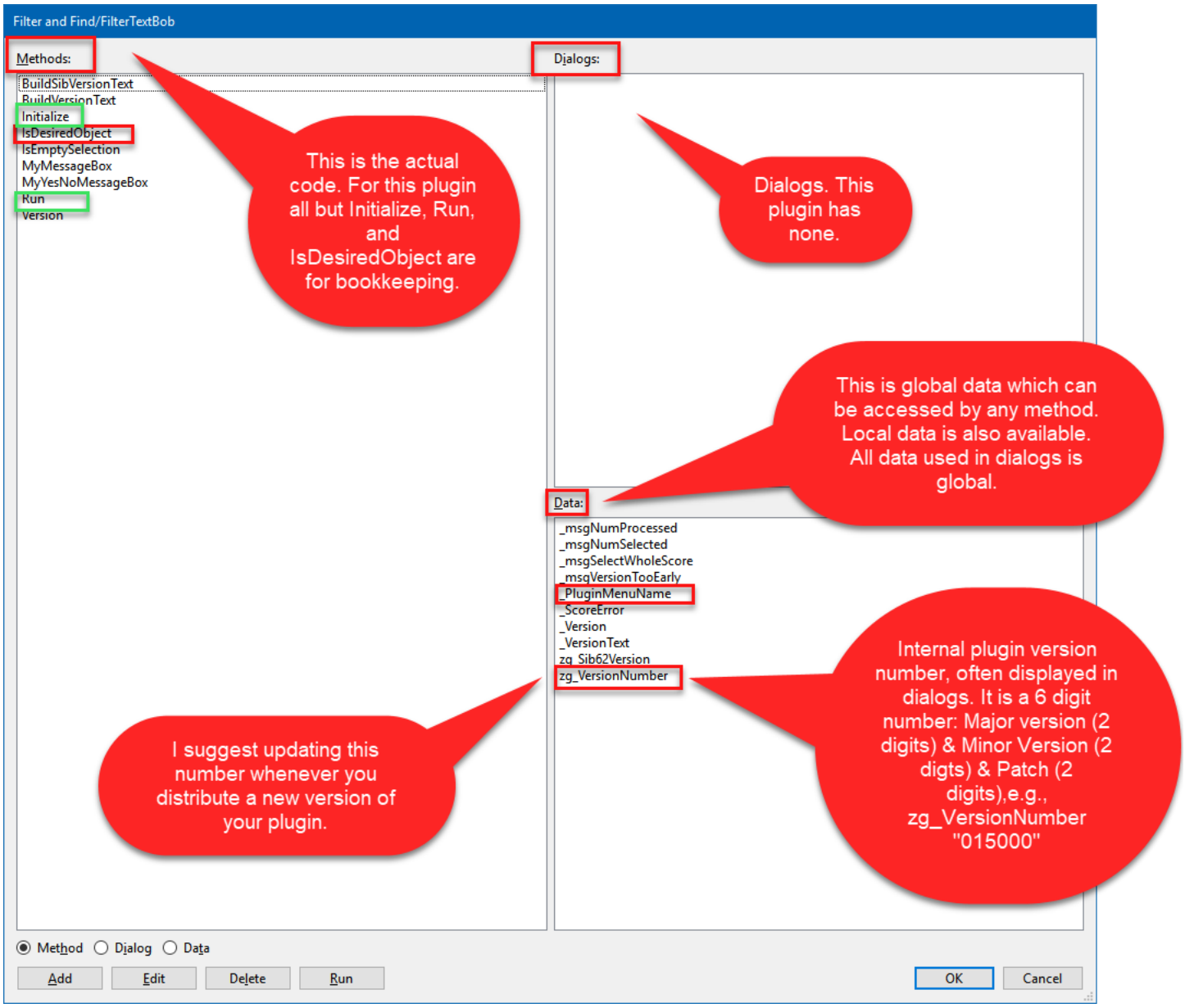
Now we can edit the new plugin. Go to **File >Plug-ins>Edit Plug-in**, and you will see the plugin editor. Type your new plugin menu name (the one with the spaces) into the **Find** box, click on Find, and then click on **Edit**. (Note that the Reference manual for **ManuScript**, the language plugins are written in, is also accessed from this menu).



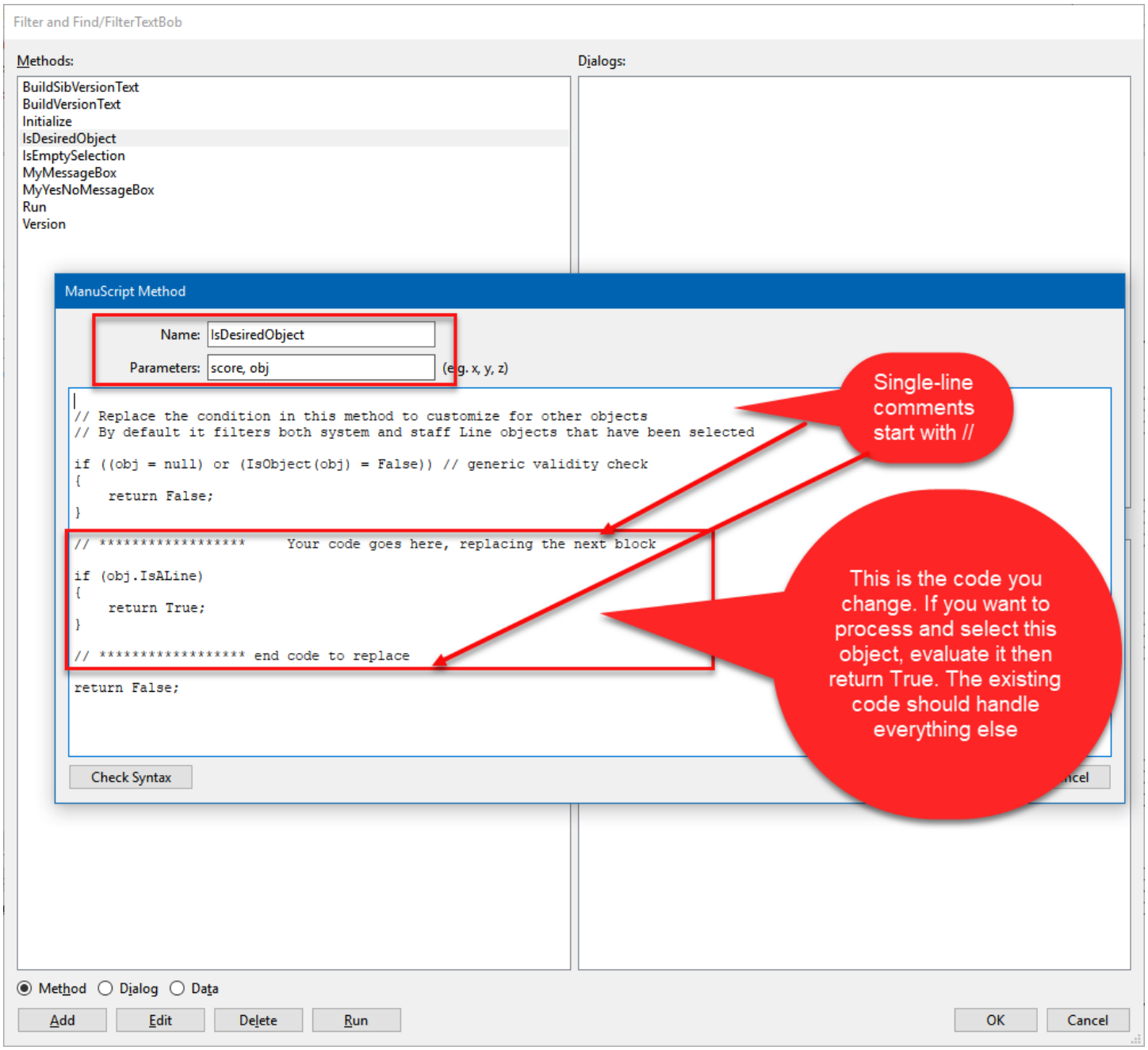
Here is the **Edit Plugins** dialog:



After clicking on Edit you should see something like this:



This is a small plugin, and our focus is even smaller. We only want to change the method **IsDesiredObject**, so choose it in the list and then choose **Edit**.



There is not a lot here. You see the method name, **IsDesiredObject**, and 2 parameters. The **score** parameter is the currently active score, which is not being used in the example, and **obj** is the currently selected object we are investigating. If you are curious, look in the **Run** method and you will find a bit of code that calls **IsDesiredObject**:

```
for each obj in selection // collect selected items in list so we can clear selection
{
    if (IsDesiredObject(score, obj)) // REVIEW this line may need to pass in more data
    {
            arsObjects.Push(obj);
    }
}
```

I don't really want to get into this too much, but briefly, "for each" successively returns each object, such as a note, line, or piece of text from the current selection. We use **IsDesiredObject** to determine if this object is one we want to select, and if so, **IsDesiredObject** will return **True** (the line " if (IsDesiredObject(score, obj))" is a shorthand for " if (IsDesiredObject(score, obj) = True) "), and we will store the current objects in **arsObjects**, a data structure called a **Sparse Array**. "Push" adds the object to the end of the sparse array. When the "for each" loop is done, the plugin will change the selection to include only the objects we had stored.

But you don't really need to know that; you just need to know how to choose your object in **IsDesiredObject**, so let's return there. We want to replace the lines

```
if ((obj.Type = "Text") or (obj.Type = "SystemTextItem") or (obj.Type = LyricItem))
{
   return True;
}
```

(which returns **True** if the object is any of the 3 kinds of text objects , and **False** otherwise which selects only the text in the original selection), with something that filters what we want.

At this point we are where you sometimes find yourself in a music lesson where your teacher says "Here are the notes for this scale. Now make a solo from those notes..."

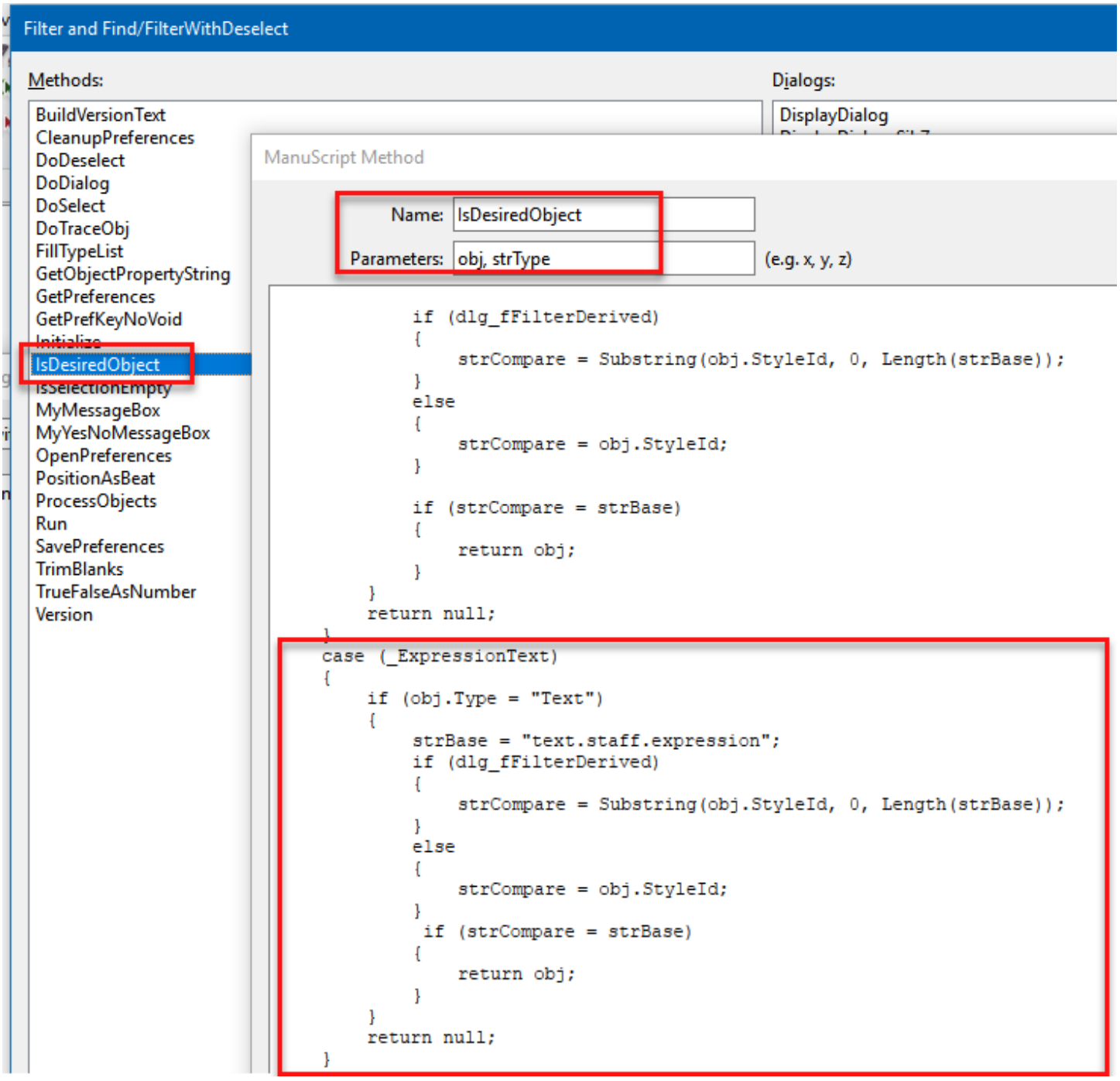
You can do anything you want here as long as it is legal in the **ManuScript** language. In a "for each" loop you cannot add or delete objects, but you can change the properties of an object, or just analyze an object as we are doing here. You may find the language cannot do what you want and will need to look for a different approach. You can look in other plugins whose names might sound similar to see what they do, and you can look at the ManuScript Language Reference in **File>Plug-ins**.

So let us say that we want to look at **Text** objects, and maybe **Expression Text** objects only, and then add a few more conditions. For this example we will only go as far as filtering **Expression** text, and the rest will be an exercise for the reader.

I might think, well, the plugins **Filter With Deselect** or **Filter Other** might handle **Expression** text, and I might check out their code. I run **Filter With Deselect**, and see that indeed **Expression** text is in the list. So I open **Filter With Deselect** in the plugin editor, and if I am lucky I might be able to figure out how it does the filter. I am going to be **very careful** not to change anything in this plugin because I can easily break it. When I am done looking at it, I will choose Cancel, never OK, unless I really know what I am doing.

In the dialog editor, I find, to my delight, the method **IsDesiredObject**, which is a clue to me that this plugin was likely derived from a template similar to **Custom Filter**. Remember **Custom Filter**?

I look through the method code and the names used are reasonably clear, to me at least, and I find this block:



Now this is more complex that you might need, but I wrote this for publication and I wanted to handle cases like having a used-defined text style derived from **Expression** text. Let's skip that part. Let's also ignore the fact that this version of **IsDesiredObject** returns either the object or null, rather than True or False.

Here are the useful lines of code:

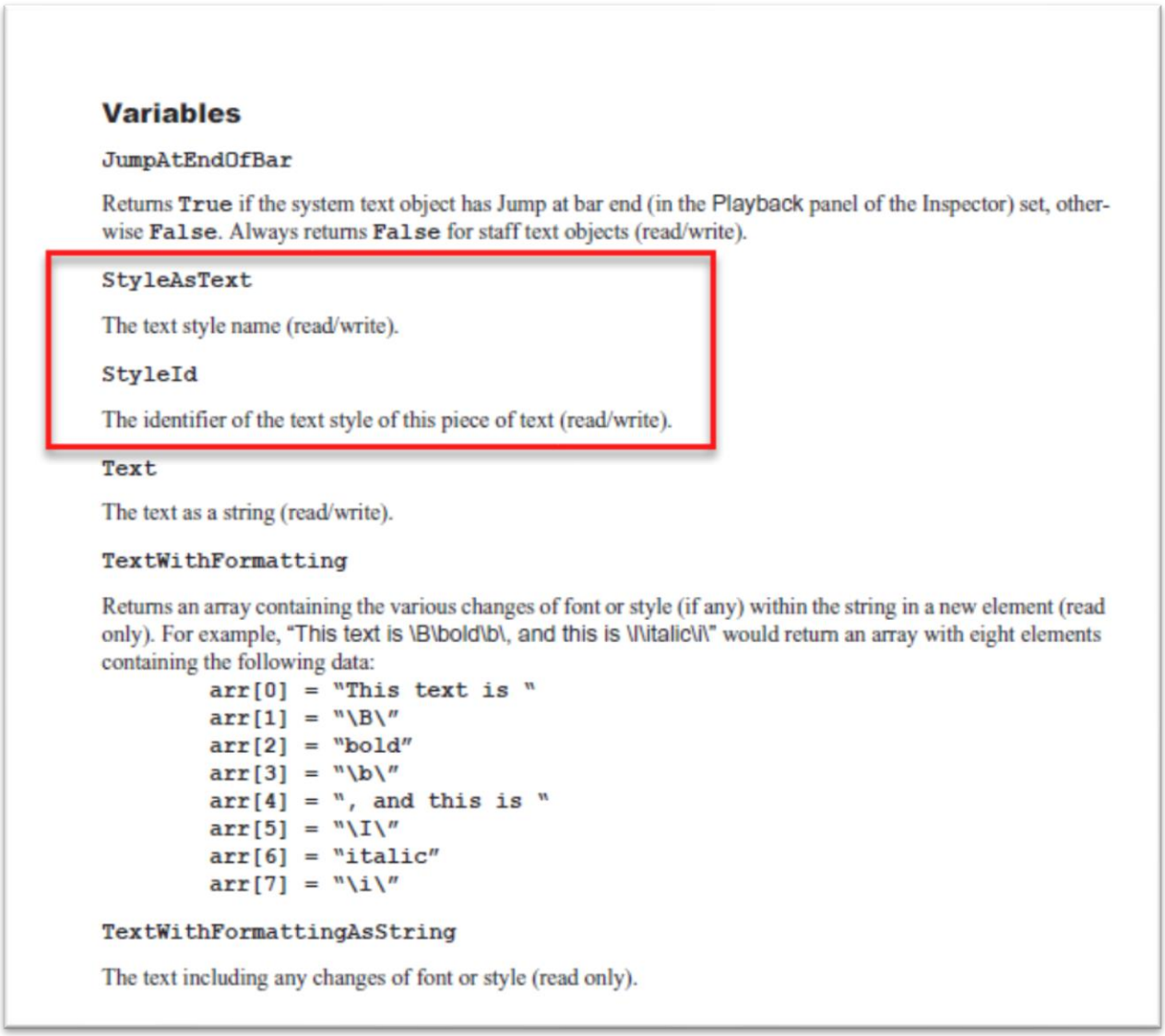
```
if (obj.Type = "Text")
strBase = "text.staff.expression";
strCompare = obj.StyleId;
```

We need to know, from experience, or reading the manual, or looking at other plugins, that "obj" is a **Bar Objec**t, and all **Bar Objects** have a **Type** field. At the end of the ManuScript Reference is a list of types:

"Types of Objects in a Bar
The **Type** field for objects in a bar can return one of the following values:
Clef, **SpecialBarline**, **TimeSignature**, **KeySignature**
Line, **ArpeggioLine**, **Bend**, **CrescendoLine**, **DiminuendoLine**, **GlissandoLine**, **OctavaLine**, **PedalLine**, **RepeatTimeLine**, **Slur**, **Trill**, **Box**, **BeamLine**, **Tuplet**, **RitardLine**, **HighLight**
LyricItem, **Text**, **SystemTextItem**, **GuitarFrame**, **GuitarScaleDiagram**, **RehearsalMark**, **InstrumentChange**
BarRest, **NoteRest**, **Graphic**, **Comment**, **Bracket**, **BarNumber**
SymbolItem, **SystemSymbolItem"**

We need to know that we have to put the type name in quotes when using it, and that a staff text object has the type "Text". (A system text object has type "SystemTextItem", and a lyric has type "LyricItem", and the **Type** field tells them apart).

If we look in the ManuScript Reference for a **Text** object, we see some variables, and the ones we are interested in are **StyleAsText** or **StyleId**.



Variables

`JumpAtEndOfBar`

Returns `True` if the system text object has Jump at bar end (in the Playback panel of the Inspector) set, otherwise `False`. Always returns `False` for staff text objects (read/write).

`StyleAsText`

The text style name (read/write).

`StyleId`

The identifier of the text style of this piece of text (read/write).

`Text`

The text as a string (read/write).

`TextWithFormatting`

Returns an array containing the various changes of font or style (if any) within the string in a new element (read only). For example, "This text is \B\bold\b\, and this is \I\italic\i\" would return an array with eight elements containing the following data:

```
arr[0] = "This text is "
arr[1] = "\B\"
arr[2] = "bold"
arr[3] = "\b\"
arr[4] = ", and this is "
arr[5] = "\I\"
arr[6] = "italic"
arr[7] = "\i\"
```

`TextWithFormattingAsString`

The text including any changes of font or style (read only).

Because I have been writing plugins since about 2001, I know that the **StyleAsText** value for **Expression** text is "Expression" but that is not in the documentation, at least not any more. It has the disadvantage of only working in English, so the **StyleId** field was added to have a language-independent identifier, and you can find these in the Reference under **Global Constants**. **Expression** text uses the **StyleId**

"text.staff.expression"

So back to **IsDesiredObject**. We want to select only **Expression** text (which will pick up both normal expressions and dynamics like ***mp***, but that is for another time).

Instead of

```
if (obj.IsALine)
{
    return True;
}
```

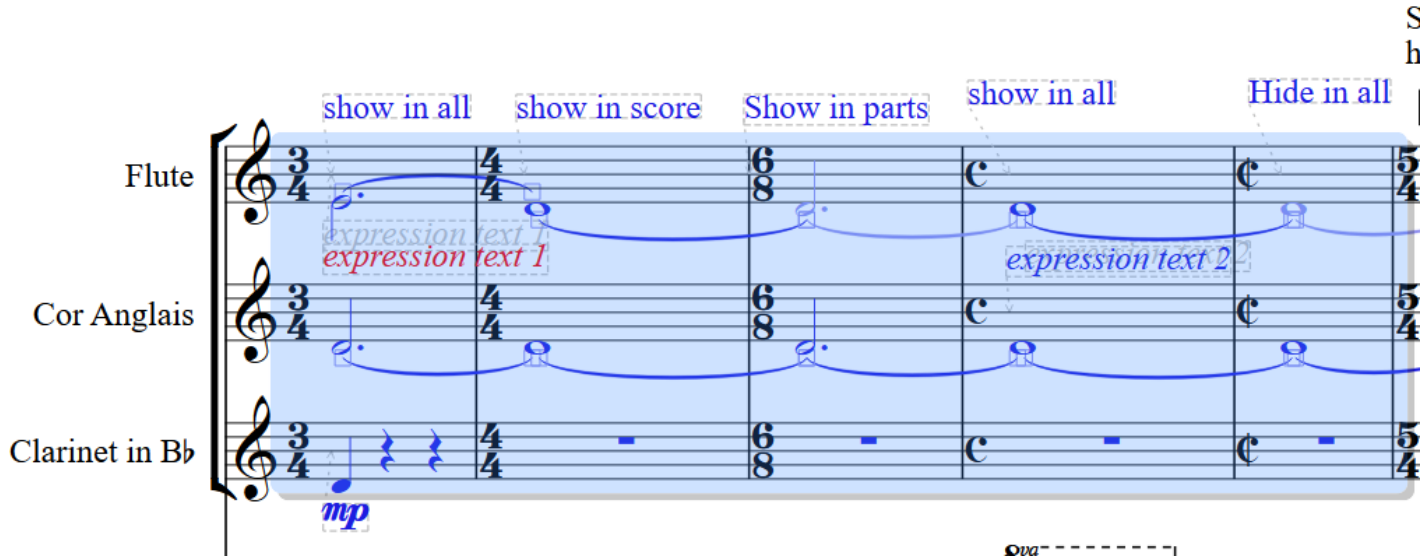
we might have something like

```
if ((obj.Type = "Text") and (obj.StyleId = "text.staff.expression"))
{
    return True;
}
```

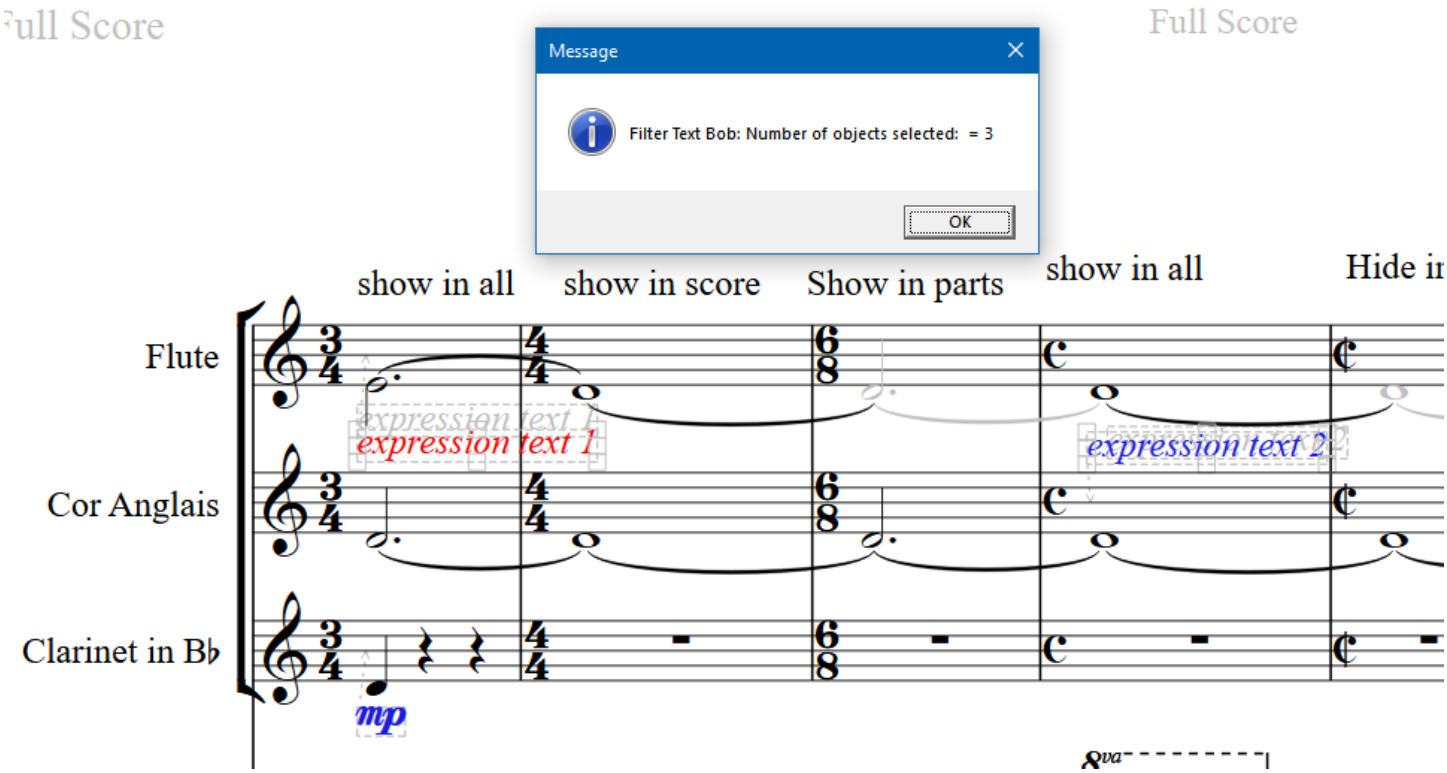
The 2 parts of the expression are carefully parenthesized. In **ManuScript** ***always*** parenthesize any expressions connected by **and** or **or**, or any mathematical expression like *a + b / c* (which should be a + (b / c) to make it behave as expected). Click on **Check Syntax** until it shows no errors before saving.

I hit OK, and then I could run the plugin from the next dialog, but I usually prefer to OK all the way out of the plugin editor and then run the plugin directly. Note that the plugin file is not saved until I hit OK to close the last plugin editor dialog. If I cancel out, all my changes will be lost and the plugin file on my hard disk will be unchanged.

Now I would open a score with appropriate text and see if it works. Start with this:



and we end up with



which looks ok. There will be a lot more testing needed to really be sure it works, but it is looking good right now.

Other plugin templates

There are several downloadable plugins that are set up to be freely modified, most of which are templates for filters. Each has default code that performs some simple action; that default code is meant to be replaced by your code. The places where code needs to be replaced are clearly marked.

These are in the plugin category **Filter and Find**:

- **Custom Filter**
  - This filters objects in the system staff and normal staves. The default code filters Text, SystemTextItem and LyricItem objects.
- **Custom Staff Filter**
  - This filters objects in non-system staves, skipping system objects. The default code filters NoteRest objects that contain a single note only, skipping rests and chords.
- **Custom System Filter**
  - This filters objects in the system staff only. If the original selection is a non-system passage selection it will be converted to a system selection covering the same bar range, and only objects in the system staff will be processed. The default code filters SpecialBarline objects.
- **Custom Note Filter**
  - This filters Note objects, effectively skipping the system staff. The default code filters quarter notes.

These are templates that are not specifically set up as filters:

- **Custom Simple Plugin (category Developers' Tools)**
    - This template has no error checking. It is meant for quick and dirty personal plugins. Some code in the methods **IsDesiredObject()** and **Run()** is meant to be changed. By default, the plugin filters Line objects in the system staff and normal staves.
- **Minimum Plugin (category Developers' Tools)**
    - This is a plugin with full initial error checking. This is the template I use when I am writing most plugins that are not modifications of another plugin Usually you only need to replace the code in the method **Process Selection**. The default code traces the type of selected objects in the system staff and normal staves.

And so…

This was a big article to get a small result, but ultimately here is what I wish to impart: there are a lot of situations where you might want to do something that is not handled by Sibelius or an existing plugin. You might be able to find someone to write one for you, but there are not very many plugin writers out there. In many cases you could start from something that exists (either a public domain plugin or something you have permission to modify), and if you are lucky, a small tweak will get you what you want. And you can do it yourself.

A custom filter can get you what you want with only a few lines of changed or added code.

You can also write a more "bare-bones" version of a filtering plugin that requires fewer steps and is totally suitable for your own use. See the chapter "Write your own Simple plugin" for the details.

There is a plugin developers' mailing list you can sign up for and people there are generally helpful about answering questions.

Good luck. Happy programming!