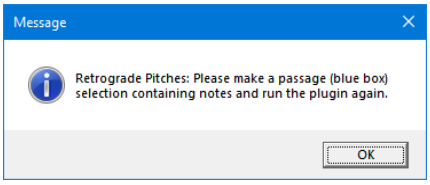
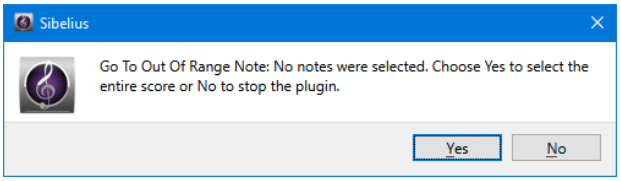
Plugin Techniques: Do Not Show This Dialog (this Sibelius session)

Bob Zawalich November 14, 2020

Why bring up a dialog?

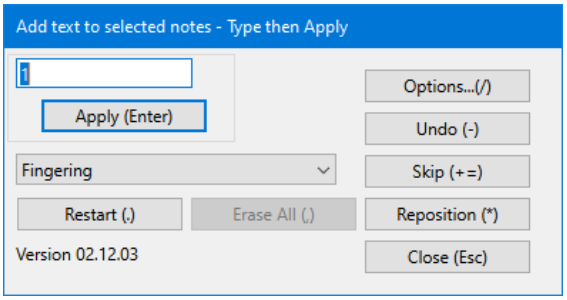
Some plugins are very simple, with no options, and they just run and do what they do, the way many Sibelius commands work. They will typically not bring up an option. Even some complex plugins, like the ones in the Transformation category like **Rotate Pitches** and **Shuffle Pitches** just do their work on the selection. If there is no selection, you might see a warning like this:



or


and you would either need to make a selection and start again, or have the plugin process the entire score, but if you did have an appropriate selection, the plugin would just do its thing.

The point of some plugins, though, is to let you specify things to do. These always bring up dialogs and you click on radio buttons or checkboxes or choose from list boxes or type in edit boxes, and they will *always* show a dialog.



In some cases, a plugin may put up a dialog to let you choose some options that you would likely reuse without changes many times in a Sibelius session after first checking and changing them. In that case, you might want the dialog to come up once or twice, but then it would be nice to not see if any more.

(A Sibelius session is the time between the time Sibelius is first started until it is completely closed).

This is what this document is about.

(Personal opinion note:) When *I* make a plugin that does this, I nearly always make the dialog come up at least the first time the plugin is run in a session. Downloadable dialogs often have no

easily accessible documentation except what appear in the dialog, so I don't like it when a plugin runs and you have no real idea what it did.

I set up such plugins to always show a dialog the first time the plugin is run in a Sibelius session, and you have an option to hide the dialog until the next time Sibelius is restarted. The first time you run the plugin in the new session I have it show up again, partly because it may have been years since you last ran it, and you would have no idea what settings you chose. So it will appear, and you can hide it again.

This is what I typically do. In **Initialize**, I set the global flag variable **g_fFirstTime** to True.

```
Initialize () {
AddToPluginsMenu(_PluginMenuName, 'Run');
g_fFirstTime = True;
}
```

At the start of **Run**(), I check **g_ fFirstTime**, and turn it off, as well as turning **dlg_fDoNotShowDialog** off. This code will only be executed once per session.

```
if (g_fFirstTime)
{
   g_fFirstTime = False;
   dlg_fDoNotShowDialog = False;
}
```

While I could have just left **dlg_fDoNotShowDialog** defined *True* in the global data, you have to remember to reset it every time you publish a plugin, because it immediately gets turned off when you are testing, and I really want this to be *True* the first time the plugin is run is a session. I don't like to add code to **Initialize**, because **Initialize** is called for each plugin when Sibelius starts up, so if the code is something that can wait for the first time the plugin is run, I just set the **g_fFirstTime** flag, and do the work in **Run**() if the plugin is called.

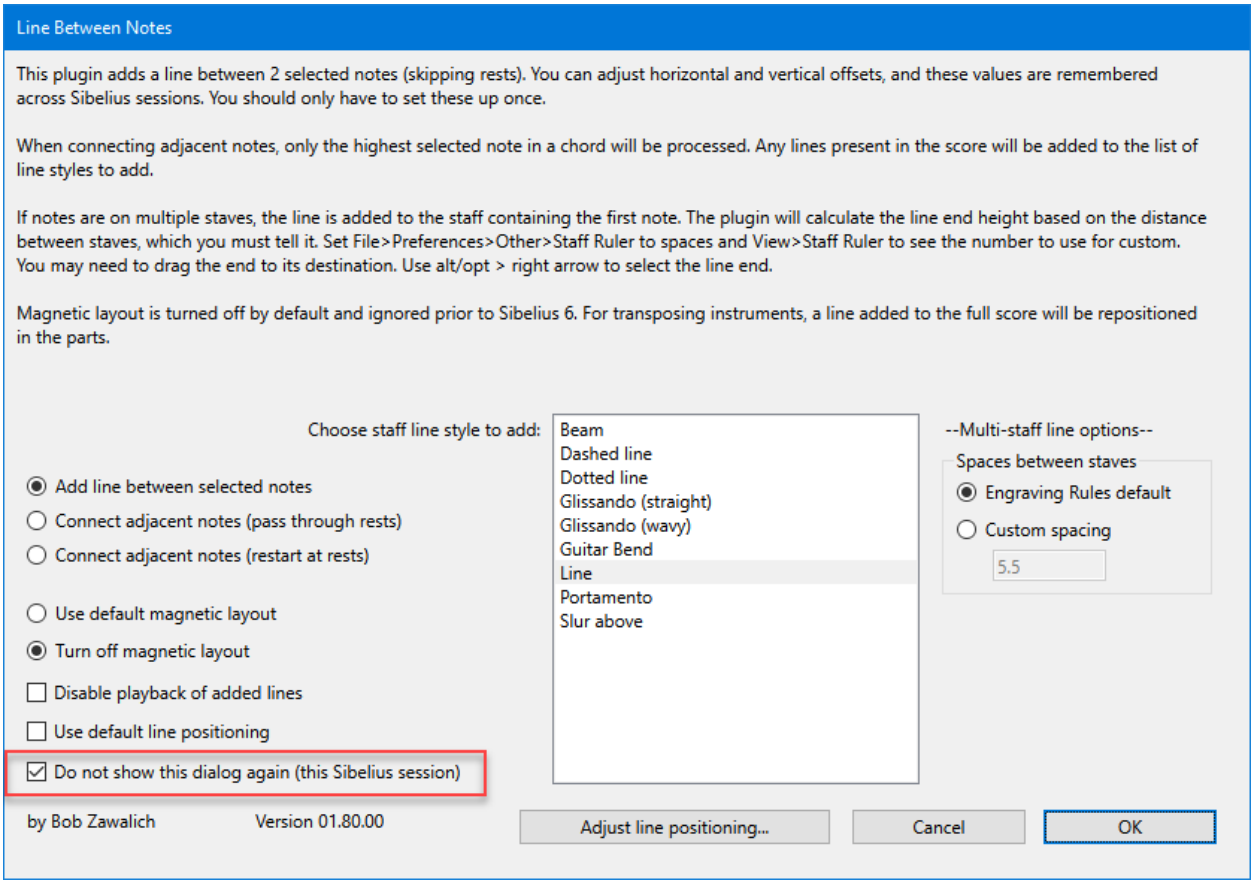
Now you need to add a checkbox to your dialog, use a global variable for its value (I use **dlg_fDoNotShowDialog**), and test it before you bring up a dialog.


```
if (dlg_fDoNotShowDialog = False)
{
   ok = DoDialog(DisplayDialog);
}
```

And there you have code to hide a dialog but have it appear the first time you run a plugin in a Sibelius session.

When to hide the dialog

Here is an example of a fairly complex plugin that will let you hide the dialog.



The dialog provides an explanation about what it is going to do, and it has a number of options. My intention is that when you first run the plugin in a session you would check that the settings are what you want, and make any desired changes, but since you may well want to add a bunch of **Gliss** lines to your score, you really don't need to see the dialog every time you run it, so there is an option to hide the dialog.

If you hide the dialog and rerun the dialog, the plugin will run as if the dialog had come up and you had chosen the options active the last time the plugin was run. The previous options will almost always be active (in plugins I have written) for the rest of the Sibelius session.

In the next Sibelius session, plugins that save dialog settings across Sibelius sessions will continue to start with the last-used options. For many plugins, though, options are not saved across sessions, and you will see the default options that come up the first time you run the plugin. So it is good to check the options before you decide to hide the dialog.

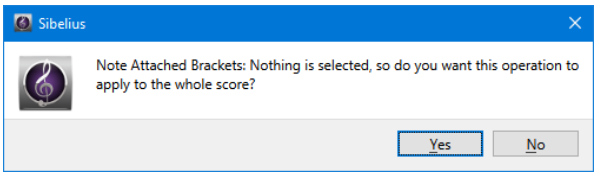
How to get a dialog to show up once hidden

I pretty much made up the concept of hiding dialog in plugins. In plugins I have written, you can always get the dialog to reappear by closing and restarting Sibelius.

That is not always convenient, though. Most plugins tend to work on a selection, so many plugins that hide dialogs can make the dialog appear when there is no selection.

One way to do this:

Most plugins I write that process selections do not let you run with no selection, and you will see a message like this if you try to do it:



If you say *No*, the plugin stops. If you say *Yes*, and the dialog was set to not show, this will be taken as a signal to make the plugin appear. If this happens, I suggest making the full selection of the score before the dialog come up, so it will be visually clearer that OK will process the full score.

If I am thinking this through carefully (which I have not always done), I will have used

selection.StoreCurrentSelection() before I select the full score and bring up the dialog, and if the dialog is cancelled, I will restore the original selection using selection.RestoreSelection();.

Other, less desirable selection options would be to not do the full selection until after the dialog is *OK*ed, which loses the visual indicator, or making the full selection but not resetting it on *Cancel*. You can find examples of both of these in my catalog. I think what I have just described works well in these situations.

The entire block of code around the dialog (for dialogs that can be hidden), might look like this:

```
// Check for an empty selection

fEmptySel = False;
fProcessEntireScore = False;
if (IsEmptySelection(score, False) = True)
{
	fContinue = MyYesNoMessageBox(_msgSelectWholeScore); //True means Yes, continue and process
score
	if (fContinue = False)
	{
		return False; // they said no, so stop the plugin
	}
	fProcessEntireScore = True;
	fEmptySel = True;
}

// Select all, saving the original selection

selection.StoreCurrentSelection();  // so the selection can be restored if the dialog is cancelled

if (fProcessEntireScore = True)
{
	// select all staff items. Do this before the dialog comes up to emphasize that the action will apply to the
entire score.
	selection.SelectPassage(1, score.SystemStaff.BarCount, 1, score.StaffCount);
}

// we will not get here if there was no selection and the user did not choose to select the entire score
```

```
// here we want to show the dialog if the selection HAD been empty, even though now we have made a full selection.

// if dlg_fDoNotShowDialog is False, the dialog will appear regardless of the state of the selection.
// If there had been a selection and dlg_fDoNotShowDialog were True, the dialog will not appear.
// Our special case is that there had been no selection, and we agreed to select the entire score. In that case we are
forcing the dialog to appear
// even if we said we should hide it.

// bring up the dialog, conditioned on dlg_fDoNotShowDialog and fEmptySel

if (fEmptySel or (dlg_fDoNotShowDialog = False))
{
   ok = DoDialog(DisplayDialog);
   if (ok = False)
   {
      selection.RestoreSelection();  // dump the full score selection if we cancel
      return False;
   }
}
```

The first part checks for a selection, by the criteria used in the call to **IsEmptySelection**. If there was an acceptable selection, **fEmptySel** and **fProcessEntireScore** will still be *False*, and the normal rules will apply. The dialog will not appear if **dlg_fDoNotShowDialog** is *False*.

(I apologize for the double negative of **dlg_fDoNotShowDialog** = False, rather than just Having **dlg_fShowDialog** and checking for *True*, but I felt that not showing was an exception and that was when the code was needed. Feel free to disagree on this decision).

If there had been no valid selection, though, the mechanism comes into play. The user is asked to *Cancel* or select the entire score. *Cancel* will close the plugin. If the user agrees to process the entire score, **fEmptySel** and **fProcessEntireScore** will be *True,* and the entire score will be selected.

In this block,

```
selection.StoreCurrentSelection();  // so the selection can be restored if the dialog is cancelled

if (fProcessEntireScore = True)
{
        // select all staff items. Do this before the dialog comes up to emphasize that the action will apply to the
entire score.
        selection.SelectPassage(1, score.SystemStaff.BarCount, 1, score.StaffCount);
}
```

the original selection is saved (*StoreCurrentSelection* in case the dialog is cancelled, and then the entire score is selected. This is done now so that when the dialog comes up the user will see the score fully selected behind the dialog as a visual reminder of what will be processed on OK.

Now, when it is time to show the dialog, we see

```
if (fEmptySel or (dlg_fDoNotShowDialog = False))
```

The dialog will appear if the *Do Not Show Dialog* flag had not been set, but will also appear if the dialog was to be hidden but there had been no selection. This is the mechanism that allows the dialog to come up with an empty selection.

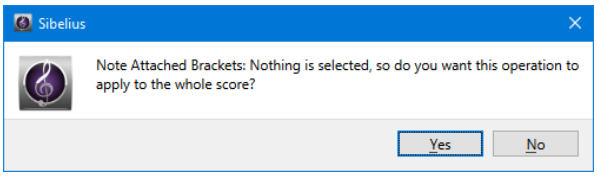
If the dialog is cancelled, the user would have been able to change dialog settings without changing the score. This block

```
ok = DoDialog(DisplayDialog);
if (ok = False)
{
    selection.RestoreSelection();  // dump the full score selection if we cancel
    return False;
}
```

will restore the original selection if the user cancels out of the dialog.

Whew. This is harder to explain than it is to do, though, and once it is set up in a template, you don't need to think about it much. The current versions of *Minimum Plugin*, a downloadable and free-to-use plugin template, are set up this way.

Before I went to the model of treating no selection as either *Cancel* or select all,



I would sometimes let the code go through and bring up the dialog so the user could change options. In that case, I would make it clear that the only thing you could do was to change the dialog settings, and then the plugin would stop. I would typically disable the *OK* button and change the text of *Cancel* to *Close* (if I was able to do that – it is a bit tricky to change the button properties). One can still do that, but I find the current approach works better in general.

Another way to do it:

The previous model is pretty good, but people are sometimes confused that the dialog comes up, and they might want to change the settings but are not sure that *Cancel* will retain the settings.

I have fairly recently started using a different approach, to make it a bit more obvious that the dialog will appear even though hidden.

If there is no selection and the dialog has been hidden, you first get a message saying "There is no selection and the plugin dialog is set to not be shown. Choose Yes to have the dialog appear the next time this plugin runs with a selection, or No to keep the dialog hidden."

Then you will get a second message: "Nothing is selected, so do you want this operation to apply to the whole score?"

This gives you several options, all of which leave you in a pretty good state. My plan is to be using this method for future plugins, despite it bringing up 2 messages in the worst case.

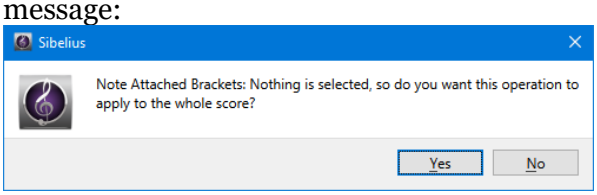
If you say *Yes* to the first message, saying *Yes* at the second message will bring up the dialog with the entire score selected, and you can change options and either choose *OK* to process the score, or *Cancel*, which will usually put the dialog down, retaining any changes, and end the plugin.

If you say *Yes* to the first message but *No* to the second, then the plugin ends, and the dialog will not appear. But its "*Do not show*" checkbox is turned off, so the next time you run the plugin with a selection, the dialog will come up.

If you say *No* to the first dialog, the dialog will keep its hidden state. If you say *Yes* to processing the entire score, the plugin will run on the score with no dialog appearing.

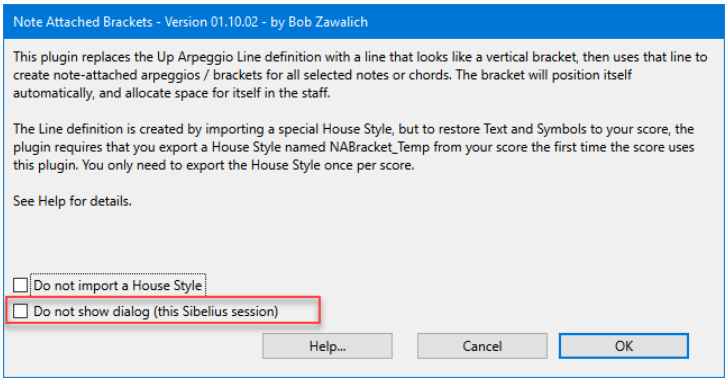
Example: NoteAttachedBrackets.plg

In this plugin, if you run it with no selection and have not hidden the dialog, you will get this message:



If you say *No*, the plugin stops, and you will either need to make a real selection or say *Yes* to continue.

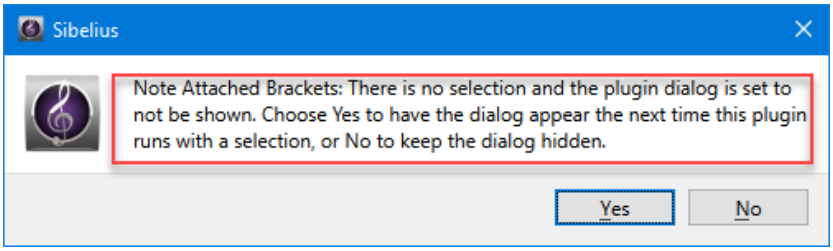
If you say *Yes* and get to the dialog, it looks like this:



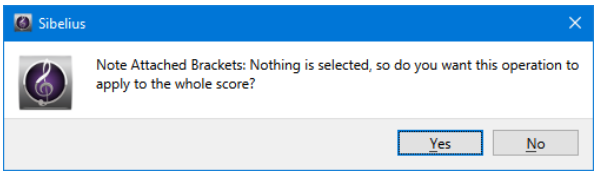
At this point you can choose *OK* to process the entire score or *Cancel*. In either case, let's say that you checked *Do not show dialog* to hide the dialog for future runs.

If you run the plugin again with a selection, it just does its job, using whatever setting you made for *Do not import a House Style*. Where it gets interesting in when you run the plugin with no selection.

You will first see this message:

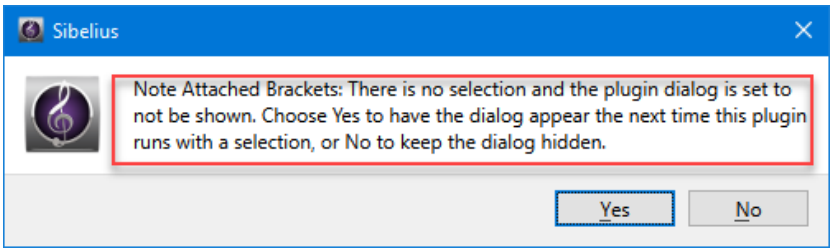


If you say *No*, then the dialog will still have its *DoNoShowDialog* checkbox checked. You will then see the *No Selection* dialog.

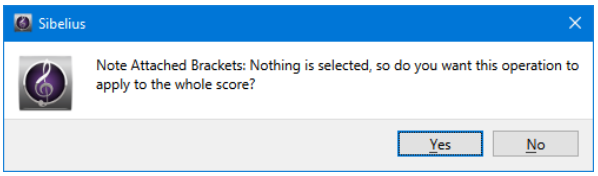


If you respond *No*, the plugin stops, and it is in the same state it was in when you ran it. If you say *Yes*, the plugin will select the entire score and run without the dialog.

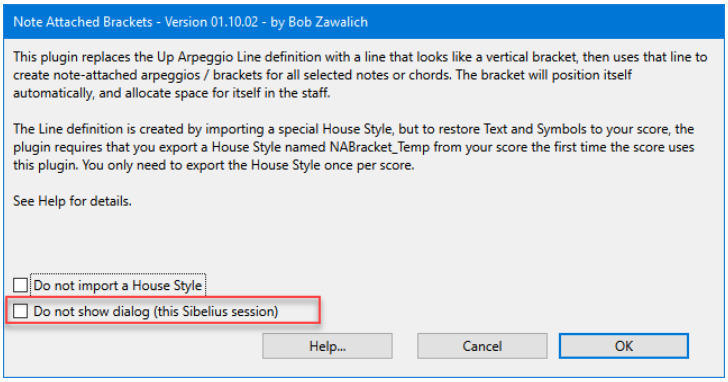
If you get the first message box and reply *Yes*,



you will also get the *No Selection* message.



Respond *Yes* here and the plugin will bring up the dialog, with the entire score selected, with its *DoNoShowDialog* checkbox *unchecked*. You can now safely reply *OK*, because the entire score has been selected, due to saying *Yes* to the second message box. The plugin will process the entire score. (As a user you should still be sure that is what you want)

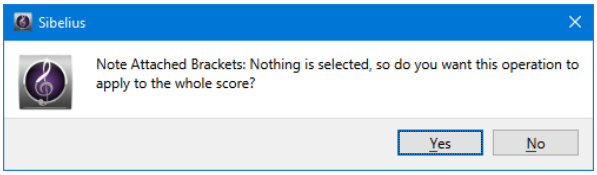


A significant gain for the programmer in having the plugin select the entire score is that you do not need to have a special version of the plugin that exists only to set the options. *OK* will process the entire score, and *Cancel* will have let you change the options without processing

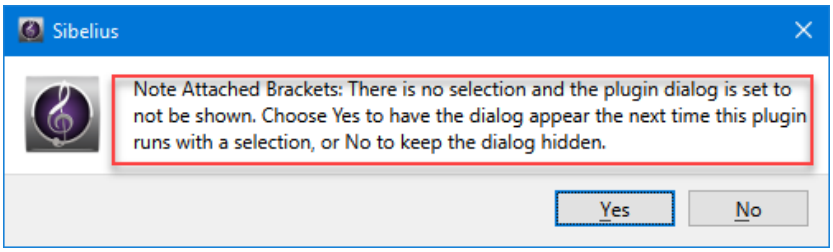
anything. (In almost all the plugins I write, *Cancel* does not reset the dialog state back to the options in place when the dialog first came up).

Whatever you reply, the *DoNoShowDialog* checkbox is *unchecked*, until you check it again.

If you had responded *No* to



The plugin will terminate without the dialog appearing, but the dialog's *DoNoShowDialog* checkbox has been *unchecked*.



The next time you run the plugin and it reaches the point in the code where the dialog would appear, it will appear. To hide it in the future, you will need to check the *DoNoShowDialog* checkbox again.

Code in the Run method of NoteAttachedBrackets.plg

(the empty selection block)

```
fProcessEntireScore = False;
fEmptySel = IsEmptySelection(score, True);

if (fEmptySel and (dlg_fDoNotShowDialog))
{
  fChangeFlag = MyYesNoMessageBox(_msgChangeDoNoShowDialog); //True means Yes, continue and process
score
  if (fChangeFlag)
  {
    dlg_fDoNotShowDialog = False;
  }
}

if (fEmptySel)
{
  fContinue = MyYesNoMessageBox(_msgSelectWholeScore); //True means Yes, continue and process score
  if (fContinue = False)
  {
    return False; // they said no, so stop the plugin
  }
  fProcessEntireScore = True;
}
```

...(when it is time to show the dialog)

```
selection.StoreCurrentSelection();
if (fProcessEntireScore = True)
{
	selection.SelectPassage(1, score.SystemStaff.BarCount, 1, score.StaffCount); // select all staff items
}

if (fEmptySel or (dlg_fDoNotShowDialog = False)) // set up empty sel special if you don't want to select all if no sel
//if (dlg_fDoNotShowDialog = False)
{
	ok = DoDialog(InfoDialog, indexSymbol);
	if (ok = False)
	{
		//CleanupPreferences();
		selection.RestoreSelection();  // in case we did a full selection but are cancelling
		return False;
	}

	//SavePreferences();
}

*****************
Message text:
	_msgChangeDoNoShowDialog "There is no selection and the plugin dialog is set to not be shown. Choose Yes to have the dialog appear the next time this plugin runs with a selection, or No to keep the dialog hidden."
	_msgSelectWholeScore "Nothing is selected, so do you want this operation to apply to the whole score?"
```

Some plugins do not process a selection, and if the dialog can be hidden, the only way to get it back will be to close and restart Sibelius. I have not written many of these, though, so I might just give a warning when the dialog is hidden that you will need to restart Sibelius to see it again.

Conclusion

You do not need to do this at all, of course, but should you choose to allow dialogs to be hidden, these are mechanisms that have worked well for me.