Manuscript Do's and Don't's 2021

Bob Zawalich ~~December 2019~~ June 2021

This is an updating of Gunnar Hellquist's classic document and my 2019 update

~~1. Character literals:~~
~~str = "a";~~
~~ch = CharAt(expression, position);~~

~~These can fail when used in comparisons or in switch statements~~

~~if (str = "abc")~~

~~switch(str)~~
~~{~~
~~case("x")~~

~~This may be the worst bug in ManuScript. Sometimes you pass a variable that was a single character literal and you run into this. This is AWFUL. I can provide more information about this.~~

1. There were 2 major ManuScript improvements made in Sibelius Ultimate 2020.6. They are optional features that can be activate by using this code in Initialize:

```
if (Sibelius.ProgramVersion > 20200600)
{
   SetInterpreterOption(TreatSingleCharacterAsString);
   SetInterpreterOption(SupportHalfSemitonePitchValues);
}
```

- The first fixes the problem I have struck out above. Single character literals are no longer turned into the special Character type, but are left as strings. This means that if the leftmost argument of an expression is a single character literal it no longer forces the right side to be a single character literal as well, so "a" is now not the same as "aaaa". Which is of course what you would expect but it has not been that way.
- The second allows you to create quartertones for the first time. Such notes have a non-integer Pitch value (n.5) that can be passed to AddNote and similar routines.
- These features are described in detail in **Plugin Research and Analysis Notes**

2. Mathematical expressions are always evaluated left to right. **You must always parenthesize expressions with mixed operators**. Parenthesize AND and OR expressions as well. I can't see that this could be changed.
    - Y = a + b /c – d * 3 evaluates as (((a + b)/c)- d) * 3

3. Make floating point operations explicit.
    - To ensure the result will be floating point be sure there is at least one operand that is a floating point number. If you are not sure, multiply an operand by 1.0.
    - Division of integers will not produce a floating point result
        - The rules for this have been changed and can be different in different versions of Sibelius, so being explicit is good. If you want integer results, use RoundDown.

    - Division test examples
        i. divide integer literals 2/5, (2*5)/7: 0, 1
        ii. divide integer variables 2/5, (2*5)/7: 0, 1
        iii. divide mixed integer and fp literals (2.0)/5, (2.0 *5)/7: 0.4, 1.4286
        iv. divide mixed integer and fp variables (2 * 1.0)/5, ((2 * 1.0) *5)/7: 0.4, 1.4286

- **To ensure you get what you want when you divide, especially when using variables you are not sure of:**
    i. **Multiply an operand by 1.0 to ensure floating point**
    ii. **Round/RoundDown/RoundUp the result to ensure an integer result.**

4. If you don't know if a variable is an integer or floating point number, you can force it to be one or the other by multiplying by 1.0 or rounding. You can also call utils.IsNumeric (str, fIntegerOnly). If fIntegerOnly is true, only a valid integer will return True, so a floating point number will fail.

5. **Arrays and global variables return pointers (addresses) rather than values**. In Sibelius 6 or later use Sparse arrays instead of arrays, and Dictionaries instead of hashes. Sparse arrays and Dictionaries can hold other objects, not just strings and numbers. If you want to get the value of an array entry or a global variable (and maybe for a hash as well) you should force the object to return a scalar value not an address. If you expect a number, add zero (0) to the array or global variable; to get a string, concatenate a blank string (""). Some examples:
    - int = 0 + arrOfInts[1];
    - str = "" & arrOfStrings[arrOfStrings.NumChildren];
    - int2 = 0 + g_numEntries;
    - str2 = "" & dlg_strEdit;

    Sometimes you can use a global or array without having to force it to be a value. There are lots of examples of global variables being passed directly as parameters to called routines that seem to work fine. I have also spent afternoons debugging code that fails because a variable was interpreted as an address. In practice I always force arrays and globals to be values, except in rare cases of actually needing arrays of arrays. If that comes up these days I just use Sparse arrays for that purpose.

6. Arrays cannot hold Boolean values. Convert to 0 (false) or 1 (true) when adding to an array (See the Method TrueFalseAsNumber in many of my plugins).

7. The last line in a Method cannot include a // comment (syntax error), Horrible time consuming and stupid. I tend to put a return(True)/return (False) linre at the end of all routines to avoid this, and never put a command on such lines.

8. Hashes and arrays are slow. They use literal searches. Dictionaries and Sparse Arrays are faster (though everything is really slow anyway), and I recommend using them when you can instead of arrays and hashes. **For listboxes in dialogs, you will be forced to use global arrays**, but I often fill a dictionary with the values, and read the entries into an array to get a sorted list. Finding an entry in a dictionary is more efficient than looking it up in an array. You can use utils.GetArrayIndex to get the index of a string in an array.

9. Dictionaries may be the best feature in ManuScript. Sibelius 6 was the pivot point for plugin development. I don't write plugins that run prior to Sibelius 6 anymore.

10. Always use score.Redraw = False; This is the best and cheapest way to speed up plugins in Sibelius.

11. Writing single entries to text files and trace window is very slow. Write them in batches. It will change your life. See the use of TraceBuffered(), in, for example, List Plugins. It is used to copy text files in Install New Plugin.

12. /* */ comments mess up line numbers of error reporting. If you use these comments in the middle of a method and get an error it will double the time it takes to find where the error is. Don't use them!

13. **Progress bar updates are very slow**. If you need to use one update only every 10 or 100 times through the loop. See, for example Calculate Statistics:

```
for barnum = firstbarnum to lastbarnum
{
```

```
if ((progress % 100 ) = 0) // update every 100 bars for speed
{
        continue = Sibelius.UpdateProgressDialog(progress, _Bar & ' ' & progress & ' ' & _of & ' ' &
numBarsTotal);
```

14. In Sib 6 or later, use Dictionaries and Sparse Arrays whenever you can.

15. Use Dictionaries for sorting

16. Use **global** Dictionaries and Sparse Arrays
    - The only problem with these is that if you have created one, and then reinstall, update, or edit the plugin so as to change the global structure, you really need to close and restart Sibelius to flush it out of memory.
    - I have spent a fruitless time lot of time debugging after making changes until I realized that my global allocation was not cleared.
    - This is especially problematic if you have a library routine that only allocates a structure when it is needed and has not been allocated, so you cannot just force allocation in Run(). See cmdutils.plg for examples of this.
    - Use Sparse Arrays to hold objects, rather than a pseudo array (see Indirection in the ManuScript Reference).

17. If code is not working as expected and you cannot figure out why, first try closing and restarting Sibelius, or better, reboot your machine. I often see a messages about a trashed memory locations while rebooting, and afterward things work better.

18. Naming conventions
    - Use lower case names for variables and upper case for Methods.
    - Use a underscore (_) prefix on text variables that need to be translated
    - Use a prefix such as g_ for global variables, as their behavior is different than locals.
    - I have my own set of naming conventions I always use. These are not official but they save me a lot of time

19. Do not add or delete objects being searched in a for each loop (crashes)

        - Collect objects in a sparse array, then process the sparse array.

20. Sibelius hangs on syntax and runtime errors when code is called from within a dialog.

21. A Listbox with initial focus will get a Listbox selection change message before the listbox comes up, preventing it from being used to respond to the first click (see document ManuScript: problems with List box called back called immediately after the dialog comes up (but not always))). See MyPlugins or Run Command Macro.
        - I have used a special 0-th entry in listboxes to avoid this problem when I wanted listboxes to respond whenever the sselection changes and a callback routine is called.

22. Pasting a system object into the system staff (using paste to position) does not work. Paste to the first staff in the score instead.

23. **"for each Note note in noterest" does not work**. You must say "for each note in noterest". As of Sibelius 6.2 you can say "for each Note note in selection". Previously you could only access NoteRests, and then had to look inside the NoteRest to get Notes.

24. Adding and especially deleting notes within a chord is problematic because if Note object are being stored outside the NoteRest object, they will change if you add or delete a note that is lower in pitch than it. Typically, I add or delete from the highest note down to avoid problems. See RemoveAllNotes from Divide Durations

```
RemoveAllNotes (nr)
{
    while (nr.NoteCount > 0)
    {
        n = nr.Highest;
        //trace('RemoveAllNotes removing note with pitch ' & n.Pitch);
        nr.RemoveNote(n);
        }
```

25. Stemweight for cross staff notes returns erroneous values. You cannot use it to detect cross staff notes.

26. Quartertones: you can recognize one by its name, and you can transpose one, but you cannot create one from scratch. The pitch and diatonic pitch values are often not what you would expect. They mostly still get skipped.

27. Selected default barlines can cause a crash. A user can select non-special barlines, though a plugin cannot. Recent versions of Sibelius will filter these barlines in Home>Filters.. If you are looking at "for each obj in selection", and get a selected default barline, the object returned is actually a Bar object. Looking at obj.Type will crash a plugin, since a Bar is not a Bar Object. I avoid this by always checking IsObject(obj) in such loops, as IsObject(bar) will return False.

28. Use IsValidObject() instead of IsObject. There is a version in recent versions of utils.plg, but you can easily write your own.

```
IsValidObject(obj)
{
    ok = ((obj != null) and (IsObject(obj)) and (IsValid(obj)));
    return ok;
}
```

29. Sibelius.Close (fSilent) fails as of Sibelius 6. It also only closes the active part of a score.
    - You should mostly use Sibelius.CloseAllWindows, or even better if available, CloseAllWindowsForScore.
    - Look at utils.CloseScore in recent versions for a good way to deal with this.
        - **Close(***show dialogs***)** Closes the current score or part view; if the current view is the last tab in the current window, the window will
        - therefore also be closed. If the optional Boolean parameter is **True** then warning dialogs may be shown about saving the active score,
        - and if it is **False** then no warnings are shown (and the score will not be saved).
        - **CloseAllWindows(***show dialogs***)** Closes all open document windows. If the optional Boolean parameter is **True** then warning dialogs
        - may be shown about saving any unsaved scores, and if it is **False** then no warnings are shown (and the scores will not be saved).
        - **CloseAllWindowsForScore(***score, showDialogs***)** Closes all of the windows associated with the specified score. The second parameter,
        - showDialogs, is an optional Boolean.
        - **CloseDialog(***dialogName***,***pluginName***,***returnValue***)** Closes the dialog *dialogName* belonging to the plug-in *pluginName* (normally
        - this should be set to **self**), returning the Boolean value *returnValue*, which can be set to **True** (**1**) or **False** (**0**). Normally you
        - do not need to use this method to close a dialog, as you can set buttons (typically with labels like OK or Cancel) to close the dialog and
        - return a value, but if you want greater control over when a dialog is closed, this method provides it.
        - **CloseWindow(***show dialogs***)** Closes the current window (that closes all of the open tabs in the current window). If the optional Boolean
        - parameter is **True** then warning dialogs may be shown about saving the score, and if it is **False** then no warnings are shown (and
        - the score will not be saved).