Manuscript Do's and Don't's 2019

Bob Zawalich December 2019

This is an updating of Gunnar Hellquist's classic document

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.

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
    - 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.

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. 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.

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

10. 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.

11. /* */ 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!

12. Progress bar updates are very slow. If you need to use one update only every 10 or 100 times through the loop

13. Use Dictionaries for sorting

14. Use global Dictionaries and Sparse Arrays

15. Use Sparse Arrays to hold objects

16. 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

17. 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.

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

19. 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 firs click (see document ManuScript: problems with List box called back called immediately after the dialog comes up (but not always)))

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

21. "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.

22. 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);
}
```

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

24. 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.

25. 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.