# Manuscript language do´s and dont´s

Gunnar Hellquist

## Version History

2002-12-01 Updated version. With tips on version handling and passing in/out parameters to procedures.
*2002-09-05 or around that date.* Original versions started being distributet.

## About this document

This is my personal compilation of good and bad ideas and experiences from Manuscript in Sibelius. Most of the material is compiled from various sources, and sadly I have forgotten who gave what part.



## My conventions

I use a set of conventions for naming things. All of this is not quite stable yet, but some of it has been proved to work over time.

- Local variables start with a lowercase character, example: localData
- Global variables start with a lowercase character, example: GlobalData
- Variable names starting with _ is used only for things that should be translated between different languages, example: _ListOfClefs

In addition to these conventions I have a set that is not fully fixed yet, but has been working rather well in at least one plug-in. This set of conventions is in flux right now.:

- In combo boxes I use a strong set of conventions. The base is the need for the Sibelius people to be able to translate the plugin to for example German. In another place in this document I have a small procedure helping in this. The naming convention is like this, assuming we want the user to be able to select a Clef.
    - M_Clef will contain the selection the user did in the Combo Box.
    - _ClefItems is the list the user may select from. The translator may change the words here, but not the ordering.
    - M_ClefInfo holds the internal “translation” of the word that is independent from what language the user sees.
- All the variables a user may select from a dialog is on the form M_Data . (This is a good idea, but can be improve by having different names for Buttons, Text, List boxes and so on).

## Keeping backups

I´ve lost countless hours on not having kept my old version of plug-ins. So this is how I do it today.

1. I open the windows explorer and point to the plug-in file.
2. A quick copy followed by paste (Ctrl-C followed by Ctrl-V)
3. Point to the new copy and press right button and Rename file
4. I rename the file to xxxx.plgmmdd where mmdd is month and date. It seems like Sibelius has no problems with these extra files laying around in the plug-in directory.

# Tips and tricks

## *In general*

- Use the trace ( ) procedure call. It is documented.

## *Passing in/out parameters to a method.*

This is not supported in the language itself, but can be simulated using Hash arrays. I do it as follows.

In the calling method:

```
Pars = CreateHash();a
Pars[“CurrentBar”] = 1;    // just an example
Xyz(Pars);                 // calling the procedure
Bar = Pars[“CurrentBar”];
```

In the procedure xyz (Pars), just a stupid example:

```
Pars[“CurrentBar”] = Pars[“CurrentBar”] + 1;
```

## *Dialog design good ideas*

- Always select snap to grid in the properties box. This helps greatly in aligning things in the dialog. I tend to have show grid on as well. (Sadly enough you have to click this every time you open a dialog for editing).
- Start by designing a rough sketch on paper. Write in pencil and add in variable names. If you have a convention on variable and itemlist for combo and list boxes, things becomes easier.

# Code stuff

## *Are the staves in a selection contigous?*

Q: Can I find out if a selection is on for example staff 1,2,3 and not on 1,3?

A: IsPassage is true for any range selection and will not help.

To find out whether or not the selection is over a non-contiguous set of staves, you could count the total number of staves in the selection (by using for each Stave s in selection) and

then see whether or not this differs from selection.BottomStaff - selection.TopStaff + 1. If it doesn't, then all the staves in that range are in the selection (and it's contiguous); if it does differ, then some staves are absent (and it's non-contiguous).

## *Translating stuff*

I got a description from Daniel and James on how to write translatable text strings. In order to support this I adopted the following conventions, and also wrote the following little method.

Convention:

* menu items are in _VarItems
* the result from a menu is in M_Var
* the internal string of the selection is in VarInfo

Of course i change the "Var" text to whatever the variable really means

I´ll describe this with selection of a clef type (not that I know if this should be translated, but anyway). In this case I´ve written the user texts in Swedish and the internal string in the names used by Sibelius in creating a clef change. Of course the internal string could be anything that is useful for your application, it might be a coded string with lots of characters.

```
_ClefItems  !  ClefInfo
----------------------
Alt           !  Alto
Bas           !  Bass
Diskant       !  Treble
Tenor         !  Tenor
```

One important notion here is that the ordering of items is of no concern, as long as they are the same in the two Data items. This would allow a translator to reorder the items in alphabetical order.

The process works as follows, assuming that M_Clef has the selection the user did.

```
internal = TranslateUI (M_Clef, _ClefItems, ClefInfo);
internal = ClefInfo  ( i );
```

The method is given below.

```
// Name: Translate UI (short for translate user input
// variable, itemlist

// looks up the value of a variable in and itemlist and returns
// the corresponding entry in varinfo.
// returns first item for safety if not found.

i = 0;
n = itemlist.NumChildren;
while (i < n)
{
   if (variable = itemlist[i])
    {
```

```
            return varinfo[i];
          };
          i = i +1 ;
        };

        // code should never reach here
        Sibelius.MessageBox (_ErrorText & " GetIndex " & variable & " " &
itemlist[0]);
        return varinfo[0] ;
```

# Undocumented stuff

## *Switch*

The switch statement is not documented (I guess that will be added in a future version). It works as follows:

```
Warning = "dog"; // or "fish" or any other string
switch (Warning)
{
  case "dog"
    { Sibelius.MessageBox("dog");}
  case "fish"
    { Sibelius.MessageBox("fish");}
  default
    { Sibelius.MessageBox("other");}
}
```

# False friends

This chapter is about things that may make you stumble, simply because they nearly behave the way you expect them to. Some of this is designed behaviour, some of it is perhaps “features” that where not quite intended.

## *The for statement, one to few*

The for statement

```
  for I = 1 to 5
      {  trace (i);   }
```

will print 1,2,3,4.

Note that it will NOT print 5. This is different from most other programming languages.

## *The selection object and parentbar*

The selection object behaves a little different, which is documented in the Manuscript manual. But there seems to be an additional twist to things that has been selected by starting from a selection object instead of through a staff object. The problem is that objects seem to forget which bar the they belong to, that is the parentbar variable doesn't have a useable value.

## *The Nth-anything that is different*

There are three Nth-things in the Manuscript language. Two of them behave the same way. NthStaff() and NthBar() both start counting from 1. The one is NthBarObject() that starts counting from 0.

## *Local and global variables*

This "feature" is a real killer unless you know about it. It is a definite false friend because the code might look alright but behaves very differently from what you see.

First I´ll look at the perfectly normal, and totally expected behaviour of assigning to local variables. The following is a code snippet from some code.

```
a= 1 ;  //  a is a local variable
b=2;   // b is also a local variable

a = b;  //  The variable a is assigned the VALUE of the variable b.
trace(a);     // writes 2 as expected
b = "hejsan" ;  // the variable b is given a new value.
trace(a);      // still writes 2 as expected
```

Now I´ll rewrite the code and have the global variable B instead of the local b as before. Global variables are defined in the Data part of plug-in editing.

```
a= 1 ;  //  a is a local variable
B=2;   // B is global variable

a = B;  //  See text below.
trace(a);  // writes 2 as expected
B = "hejsan" ;  // the variable B is given a new value.
trace(a);     // writes hejsan which is not expected
```

It seems that somehow the variable a above didn´t get the value of b, instead it was somehow connected to the current value of b.

The simple remedy to this problem is to rewrite the example above as follows (in bold style is the change):

```
a= 1 ;  //  a is a local variable
```

```
B=2;   // B is global variable

a = B + 0;  //  See text below.
trace(a);  // writes 2 as expected
B = "hejsan" ;  // the variable B is given a new value.
trace(a);     // writes 2 as expected
```

In this later version, we force the evaluation of B+0 which of course is 2. But it also breaks the connection between a and the current value of B.

## *Local and global variables but now for arrays*

As you remember from the discussion above, assigning a local variable to a global variable creates a connection between these two.

This works in a very similar way if the variable on the right side is an element of an array.

```
a= 1 ;  //  a is a local variable
b = CreateArray();  // see note
b[1]=2;

a = b[1];  //  See text below.
trace(a);  // writes 2 as expected
b[1] = "hejsan" ;
trace(a);     // writes hejsan which is not expected.
```

Note: the problem is the same if you do CreateHash instead.

The solution of course is to do a = b[1] + 0  .
When working with strings the corresponding thing is a = b[1] & "" .

## *Arrays and arrays, again*

There is one more problem with arrays. If you assign one member of the array to another member of an array, the value will be undefined.

```
b = CreateArray();  // see note
b[1]=1;
b[2] =2;

b[1] = b[2];  //  See text below.
trace(b[1]);  // writes something totally unexpected
```

The solution here is the same as for the other cases above.

## *The lost last object*

(found by Neil Sands)

There appears to be a strange bug with iterating over staves (which is effectively what the 'for each <X> in score' will do; it will iterate over all staves in the score, and over each of these staves in turn). The result of this is that BarObjects coming at the very end of the bar (ie. after any music in the bar) will not be returned by this iterator. This includes clef changes at the end of the bar and key signature & time signature changes (probably).