ManuScript Quirks (Hans-Christoph Wirth)

- last line of a method cannot contain // comments – causes syntax error
- /* */ comments mess up line numbers in syntax error messages
- deleting notes in a noterest
- ""&, 0+ for global and array values
- globals for dialogs
- double quote issues
- no comments in last line of a method
- delete objects in reverse score order
- non breaking space "for" & chr(160) & "bar";
- clipboard ids/paste to position
- syntax errors in called dialog cause crash
- dictionary null is zero
- n.Deselect deselects entire nr. n.Select selects a single note.
- no multi line edit controls

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 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.

- arrays will not work with Booleans
- Do not try to store Boolean true or False in an array.
-
- arr[0] = True;
- f = 0 + arr[0];
- if (f = True)...
-
- won't work. It works fine to use 0 and 1, though.

- Arithmetical evaluation order is left to right always. Parenthesize any expressions with mixed operators.

This is a serious problem in Manuscript. Prior to Sib 3, it was possible to define a global data variable that looked like

g_doubleq """ (3 double quotes).

But in Sib3, the file format was changed to text, and the conversion to text would cause such a variable to trash the plugin source file.
So don't do that, and in fact be very careful that no global variable accidentaly has a double quote in it (such as a string typed from a dialog edit control). If you are debugging and so saving the source file, such a variable could cause the file to lose data.

I had come up with a mechanism that allowed me to convert a character to a numeric value that I could compare to a constant, so I could recognize a DQ. But this no longer works in Sib 4, and there was still no way I knew of to output a DQ.

In Sib4, this is doable using the Chr() method. Chr(34) is a double quote, and you can say

if (ch = Chr(34))

or

str = Chr(34) & text in quotes & Chr(34);

Still not as convenient as being able to escape it, but way better than in Sib 3. Obviously using this limits the plugin to Sib4, however.

Escaping single quotes in global variables

For a single quote in a data variable, do not use the \ escape character, just say

g_strSQ "this ' is a single quote"

If you build a string in a method, you do need to use the esc character, however:

str = "this" & "\'" & :is a single quote";

- to use floating point, at least one of the numbers has to have a decimal point. Or at least, it will evaluate as floating point that way. 2.0 / 3 will evaluate to a floating point number, but 2/3 will not.

- Just FYI, I have determined that you can put at most 254 characters (not 255!) into a Mac message box before it overflows and loses data.

For Windows the limit is larger, if there is one at all. But if you are to make plugins run on both systems, it is wise to limit the text to a total of

254 characters.

- The plugin trace window is limited to 64K characters in both Windows and Mac up to Sibelius 7. So expect that tracing function calls will cause it to overflow – use the clear button to try to capture the contents further into the output.
-
- In general, if you have to replace a note for some reason (say to change a pitch), and the notes is the same duration as the previous note, it is better to use the Noterest methods (removenote, addnote), rather than the bar or staff methods.

One advantage (of several), is that removing a note but retaining the noterest keeps the articulations, which appear to be a noterest-level property, whereas of you remove the note and add it with bar.AddNote, all articulations are lost.

Of course, if you are changing the duration of a note, you must use the bar-level methods.

- Yesterday I had the following code in my plugin

```
gap = 10;

// some further lines

t2 = 34;

// some further lines

t2 = t2 + 1.5*gap;
```

And ended up with t2's value of "355" rather than the expected "49". This means, that the expression is evaluated as

```
(t2 + 1.5)*gap
```

rather than

```
t2 + (1.5*gap)
```

as I would expect. (Needless to say that it took me half an hour to figure out that extremly stupid extremly unexpected behaviour. If I can't rely on arithmetics, what can I rely on at all?)

Why does the normal precedence rule ("multiplication before addition") not work here? If I replace the last line by

```
t2 = t2 + (1.5*gap);
```

then everything works. Do I really have to put brackets around all multiplications in ManuScript?

For reference, the doc suggests

> The normal precedence for these operators applies; in other words,
> 2+3*4 is 14, not 20, because * is evaluated before +. To make this
> clearer you could write 2+(3*4). To get the answer 20, you'd have to
> write (2+3)*4.

It says nothing special about operator precedence with both floating point and integer operands. I guess there's an automatic integer to floating point conversion happening, but apparently at the wrong place?

Hans-Christoph

- So, as far as I can tell, the only identifier that tells a note is a
> quartertone is the name, and that is read only. Sib can transpose
> quartertones. Do they use something that
> is not exposed to Manuscript? (James?)

Well, yes. Internally the pitch of a note is specified by a diatonic pitch and a chromatic pitch, but the chromatic pitch is twice as large as that exposed to ManuScript; thus there is one unit of chromatic pitch per quartertone, as opposed to one unit per semitone in the "MIDI pitch". I imagine when ManuScript was originally developed, the decision was made to convert all the internal pitch values to MIDI pitches for the sake of simplicity.

- I don't know for sure, but I vaguely remember asking about garbage collection once, and being told that the memory was cleaned up when the variables go out of scope.I did some checks of memory when I allocated a large number of arrays and did not see spikes in memory use, so I assume that this is true.

I tend to reuse arrays by keeping track of the number of entries currently in use, rather than Creating them every time I need one. For hash table, being non global, I usually just create a new one. It might be possible to clear one using the same kind of assignments as works with arrays.

For global arrays, as in dialogs, I have found that you can copy an empty

array (or a pre-initialized array) to an array name, and this replaces the original allocated memory with a copy of the initialized memory. This works if you copy to the top level array name. If you copy an array to an array entry, as in "arr[1] = arr2" , then arr[1] gets a copy of the full array arr2. Thus you can build arrays of arrays.

The one place that this is funny is if you pass a global array as a parameter to a method, then assign to that variable. When this happens, the connection between the global variable and the array is lost. Be careful of this! Example:

```
(define in data area)
lst1
{
"1"
"2"
}

lst2
{
"3"
"4"
}

lst3
{
"4"
"5"
}

In Run():

lst1 = lst2;
lst1.WriteToString() shows 3, 4, and variable name lst1

ChangeArray(lst1);

ChangeArray(array)
{
   array = lst3;
   array.WriteToString() shows 4, 5, but the name is array, not lst1
   lst1.WriteToString()  shows 3, 4 (unchanged)
}
```

In this case, the array copy wipes away the connection between the parameter and the global variable. If you need to do something like this, copy directly to the global variable, or return array from

ChangeArray and assign that value to lst1.

- Does anyone know of a straightforward way to enumerate the keys in a hash

> table?

Yes, there is a way: iterate over the hash table as normal, but use the 'Label' method on each item to retrieve the key.

Example:

```
hash = CreateHash();
hash["key one"] = "data one";
hash["key two"] = "data two";
for each thing in hash {
   trace(thing.Label & " = " & thing);
}
```

This outputs the following to the trace window:

```
key one = data one
key two = data two
```

This works because each element of a hash table is itself a TreeNode and therefore supports the methods Label, NumChildren and WriteToString.

- noncontiguous passage selections