

ManuScript Language Guide
for Sibelius Ultimate Software

## Legal Notices

© 2022 Avid Technology, Inc., ("Avid"), all rights reserved. This guide may not be duplicated in whole or in part without the written consent of Avid
For a current and complete list of Avid trademarks visit: www.avid.com/legal/trademarks-and-other-notices
Bonjour, the Bonjour logo, and the Bonjour symbol are trademarks of Apple Computer, Inc.
Thunderbolt and the Thunderbolt logo are trademarks of Intel Corporation in the U.S. and/or other countries.
This product may be protected by one or more U.S. and non-U.S. patents. Details are available at www.avid.com/patents.
Product features, specifications, system requirements, and availability are subject to change without notice.
Guide Part Number 9329-66445-00 REV A 12/22

## Contents

Introduction ..... 1
Rationale ..... 1
Technical Support ..... 2
System Requirements and Compatibility Information ..... 2
Conventions Used in Sibelius Documentation ..... 2
Resources. ..... 3
Sibelius ManuScript Language Tutorial. ..... 4
Edit Plug-ins ..... 4
Editing the Code. ..... 7
Loops ..... 9
Objects ..... 11
Representation of a Score ..... 12
The "for each" Loop ..... 14
Indirection, Sparse Arrays, and User Properties ..... 15
Dialog Editor. ..... 19
Set Creation Order ..... 21
Debugging Plug-ins ..... 24
Storing and Retrieving Preferences ..... 25
Reference ..... 30
Syntax ..... 30
Expressions ..... 31
Operators ..... 33
Object Reference ..... 34
Hierarchy of Objects ..... 34
All Objects ..... 35
Accessibility ..... 36
Annotationltem ..... 36
Bar. ..... 37
Barline ..... 41
Barlines ..... 42
BarObject ..... 42
BarRest. ..... 45
Bracket ..... 45
Brackets and Braces ..... 45
Clef ..... 46
Comment ..... 46
ComponentList ..... 47
Component ..... 47
ManuScript Language Guide ..... iii
DateTime ..... 47
Dictionary ..... 48
DocumentSetup ..... 48
DynamicPartCollection ..... 50
DynamicPart ..... 50
EngravingRules ..... 51
File ..... 53
Folder ..... 53
GuitarFrame ..... 54
GuitarScaleDiagram ..... 56
HitPointList ..... 57
HitPoint ..... 57
InstrumentChange ..... 57
InstrumentTypeList. ..... 58
InstrumentType ..... 58
KeySignature ..... 59
Line ..... 60
Lyricltem ..... 60
NoteRest ..... 61
Note ..... 63
NoteSpacingRule ..... 65
PageNumberChange. ..... 66
PluginList ..... 67
Plugin ..... 67
RehearsalMark ..... 67
Score ..... 68
Selection ..... 73
Sibelius ..... 76
Soundlnfo ..... 83
SparseArray ..... 83
SpecialBarline ..... 84
Staff ..... 84
Syllabifier ..... 87
Symbolltem and SystemSymbolltem ..... 88
SystemObjectPositions ..... 88
SystemStaff, Staff, Selection, Bar and, all BarObject-derived Objects ..... 88
SystemStaff ..... 89
Text and SystemTextltem ..... 89
TimeSignature ..... 90
TreeNode ..... 90
Tuplet ..... 90
Utils ..... 91
VersionHistory ..... 94
Version ..... 94
VersionComment ..... 95
Command IDs ..... 96
Global Constants ..... 105
Global Constants ..... 105

## Introduction

ManuScript is a simple, music-based programming language used to write plug-ins for Sibelius Ultimate. ManuScript is based on Simkin, an embedded scripting language developed by Simon Whiteside, and has been extended by him and the rest of the Sibelius team ever since. (Simkin is a spooky pet name for Simon sometimes found in Victorian novels.) For more information on Simkin, and additional help on the language and syntax, visit the Simkin website at www.simkin.co.uk.
? Throughout this guide, "Sibelius" refers to Sibelius Ultimate for the sake of readability.

## Rationale

Providing a plug-in language for Sibelius Ultimate addresses several different issues:

- Music notation is complex and infinitely extensible, so some users will sometimes want to add to a music notation program to expand its possibilities with these new extensions.
- It is useful to allow frequently repeated operations (for example, opening a MIDI file and saving it as a score) to be automated, using a system of scripts or macros.

Certain more complex techniques used in composing or arranging music can be partly automated, but there are too many to include as standard features in Sibelius.

There were several conditions that we wanted to meet in deciding what language to use:

- The language had to be simple, as we want normal users (not just seasoned programmers) to be able to use it.
- We wanted plug-ins to be usable on any computer, as the use of computers running both Windows and macOS is widespread in the music world.
- We wanted the tools to program in the language to be supplied with Sibelius.
- We wanted musical concepts (pitch, notes, bars) to be easily expressed in the language.
- We wanted programs to be able to talk to Sibelius easily (to insert and retrieve information from scores).
- We wanted simple dialog boxes and other user interface elements to be easily programmed.

C/C++, the world's "standard" programming language(s), were unsuitable as they are not easy for the non-specialist to use, they would need a separate compiler, and you would have to recompile for each different platform you wanted to support (and thus create multiple versions of each plug-in).

The language Java was more promising as it is relatively simple and can run on any platform without recompilation. However, we would still need to supply a compiler for people to use, and we could not express musical concepts in Java as directly as we could with a new language.

So we decided to create our own language that is interpreted so it can run on different platforms, integrated into Sibelius without any need for separate tools, and can be extended with new musical concepts at any time.

The ManuScript language that resulted is very simple. The syntax and many of the concepts will be familiar to programmers of C/C++ or Java. Built into the language are musical concepts (Score, Staff, Bar, Clef, NoteRest) that are instantly comprehensible.

## Technical Support

Since the ManuScript language is more the province of our programmers than our technical support team (who are not, in the main, programmers), we can't provide detailed technical help on it, any more than Oracle will help you with Java programming. This document and the sample plug-ins should give you a good idea of how to do some simple programming fairly quickly.

We would welcome any useful plug-ins you write - please contact us at www.sibelius.com/plugins and we may put them on our web site; if we want to distribute the plug-in with Sibelius itself, we'll pay you for it.

Mailing list for plug-in developers
There is a growing community of plug-in developers working with ManuScript, and they can be an invaluable source of help when writing new plug-ins. To subscribe, go to http://avid-listsrv1.avid.com/mailman/listinfo/plugin-dev.

## System Requirements and Compatibility Information

Avid can only assure compatibility and provide support for hardware and software it has tested and approved.
For complete system requirements and a list of qualified computers, operating systems, hard drives, and third-party devices, visit: www.avid.com/compatibility

## Conventions Used in Sibelius Documentation

Sibelius documentation uses the following conventions to indicate menu choices, keyboard commands, and mouse commands:

| Convention | Action |
| :--- | :--- |
| File > Save | Choose Save from the File tab |
| Control+N | Hold down the Control key and press the N key <br> ton |
| Control-click | Click with the right mouse button |
| Right-click |  |

The names of Commands, Options, and Settings that appear on-screen are in a different font.
The following symbols are used to highlight important information:
?่ User Tips are helpful hints for getting the most from your Sibelius system.

A Important Notices include information that could affect data or the performance of your Sibelius system.
min Shortcuts show you useful keyboard or mouse shortcuts.

[^0]
## How to Use this PDF Guide

This PDF provides the following useful features:

- The Bookmarks on the left serve as a continuously visible table of contents. Click on a subject heading to jump to that page.
- Click a + symbol to expand that heading to show subheadings. Click the - symbol to collapse a subheading.
- The Table of Contents provides active links to their pages. Select the hand cursor, allow it to hover over the heading until it turns into a finger. Then click to locate to that subject and page.
- All cross references in blue are active links. Click to follow the reference.
- Select Find from the Edit menu to search for a subject.
- When viewing this PDF on an iPad, it is recommended that you open the file using iBooks to take advantage of active links within the document. When viewing the PDF in Safari, touch the screen, then touch Open in "iBooks".


## Resources

The Avid website (www.avid.com) is your best online source for information to help you get the most out of Sibelius.

## Account Activation and Product Registration

Activate your product to access downloads in your Avid account (or quickly create an account if you do not have one). Register your purchase online, download software, updates, documentation, and other resources.
www.avid.com/account

## Support and Downloads

Contact Avid Customer Success (technical support), download software updates and the latest online manuals, browse the Compatibility documents for system requirements, search the online Knowledge Base or join the worldwide Avid user community on the User Conference.
www.avid.com/support

## Training and Education

Study on your own using courses available online, find out how you can learn in a classroom setting at an Avid-certified training center, or view video tutorials and webinars.
www.avid.com/education

## Products and Developers

Learn about Avid products, download demo software, or learn about our Development Partners and their plug-ins, applications, and hardware.
www.avid.com/products

## Sibelius ManuScript Language Tutorial

## Edit Plug－ins

## A Simple Plug－in

Let＇s start a simple plug－in．You are assumed to have some basic experience of programming（such as BASIC or C），so you＇re al－ ready familiar with ideas like variables，loops，and so on．

To create a new Sibelius plug－in：
1 Start Sibelius and open or create a new score．
2 Choose File＞Plug－ins＞Edit Plug－ins．

| File | Home | Note Input | Notations | Text | Play |
| :---: | :---: | :---: | :---: | :---: | :---: |
| In Save |  |  |  |  |  |
| 目 Save As |  | Show info | $\square$ | － |  |
| 4 New |  |  |  |  |  |
| Open |  | Some of the text below may be used in text in the first and subsequent pages of your parts． |  |  |  |
| $\square$ Close |  | Title： |  |  |  |
|  |  | Easy Piece |  |  |  |
| 凰國 Append |  | Subtitle： |  |  |  |
| Info |  | for piano |  |  |  |
|  |  | Composer： |  |  |  |
| Recent |  | B Bennett |  |  |  |
|  |  | Arranger： |  |  |  |
| Print |  |  |  |  |  |
| Share |  | Artist： |  |  |  |
| Export |  | Instrument changes： |  |  |  |
|  |  | Piano |  |  |  |
| Teaching |  |  |  |  |  |
| Help |  |  |  |  |  |
|  | ins－ |  |  |  |  |
| Install Plug－ins |  |  |  |  |  |
| Edit Plug－ins |  |  |  |  |  |
|  | Show Plug－in Trace Window 2017 |  |  |  |  |
| Selecting Edit Plug－ins in the File tab |  |  |  |  |  |

3 The following dialog appears:


## Edit Plug-ins dialog

4 Click New.


## New ManuScript Plug-in dialog

5 You are asked to type the internal name of your plug-in (used as the plug-in's filename), the name that should appear on the menu and the name of the category in which the plug-in should appear, which will determine which ribbon tab it appears on.

6 Type Test as the internal name, Test plug-in as the menu name and Tests as the category name, then click OK.
7 You'll see Test (user copy) added to the list in the Edit Plug-ins dialog under a new Tests branch of the tree view. Click Close. This shows the folder in which the plug-in is located (Tests, which Sibelius has created for you), the filename of the plug-in (minus the standard .plg file extension), and (user copy) tells you that this plug-in is located in your user application data folder, not the Sibelius program folder or application package itself.

8 If you look in the Home > Plug-ins gallery again you'll see a Tests category, with a Test plug-in underneath it.

9 Choose Home > Plug-ins > Tests > Test and the plug-in will run. You may first be prompted that you cannot undo plug-ins, in which case click Yes to continue (and you may wish to switch on the Don't say this again option so that you're not bothered by this warning in future.) What does our new Test plug-in do? It just pops up a dialog which says Test (whenever you start a new plug-in, Sibelius automatically generates in a one-line program to do this). You'll also notice a window appear with a button that says Stop Plug-in, which appears whenever you run any plug-in, and which can be useful if you need to get out of a plug-in you're working on that is (say) trapped in an infinite loop.

10 Click OK on the dialog and the plug-in stops.

## Three Types of Information

Let's look at what's in the plug-in so far. Choose File > Plug-ins > Edit Plug-ins again, then select Tests/Test (user copy) from the list and click Edit (or simply double-click the plug-in's name to edit it). You'll see a dialog showing the three types of information that can make up a plug-in:

Methods Similar to procedures, functions, or routines in some other languages.
Dialogs The layout of any special dialogs you design for your plug-in.
Data Variables whose value is remembered between running the plug-in. You can only store strings in these variables, so they're useful for things like user-visible strings that can be displayed when the plug-in runs. For a more sophisticated approach to global variables, ManuScript provides custom user properties for all objects-see Edit Plug-ins.


## Example: Test plug-in

## Methods

The actual program consists of the methods. As you can see, plug-ins normally have at least two methods, which are created automatically for you when you create a new plug-in:

## Initialize

This method is called automatically whenever you start up Sibelius. Normally it does nothing more than add the name of the plug-in to the Plug-ins menu, although if you look at some of the supplied plug-ins you'll notice that it's sometimes also used to set default values for data variables.

Sibelius 2020.9 (and later) adds new functionality that lets ManuScript treat a single character as a string, rather than a number, and provides support for quarter-tone accidentals and pitches. Include the following code example with the Initialize () method to enable this functionality:

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

Plug-ins that use this new functionality are not supported with earlier versions of Sibelius.

## Run

This is called when you run the plug-in, you'll be startled to hear (it's like main () in C/C++ and Java). In other words, when you choose Home > Plug-ins > Tests > Test, the plug-in's Run method is called. If you write any other methods, you have to call them from the Run method-otherwise how can they ever do anything?

Click on Run, then click Edit (or you can just double-click Run to edit it). This shows a dialog where you can edit the Run method:


## ManuScript Method dialog

In the top field you can edit the name; in the next field you can edit the parameters (the variables where values passed to the method are stored); and below is the code itself:

```
Sibelius.MessageBox("Test");
```

This calls a method MessageBox which pops up the dialog box that says Test when you run the plug-in. Notice that the method name is followed by a list of parameters in parentheses. In this case there's only one parameter: because it is a string (that is, text) it is in double quotes. Notice also that the statement ends in a semicolon, as in C/C++ and Java. If you forget to type a semicolon, you'll get an error when the plug-in runs.

What is the role of the word Sibelius in Sibelius.MessageBox? In fact it's a variable representing the Sibelius program; the statement is telling Sibelius to pop up the message box (C++ and Java programmers will recognize that this variable refers to an "object"). If this hurts your brain, we'll go into it later.

## Editing the Code

Now try amending the code slightly. You can edit the code just like in a word processor, using the mouse and arrow keys, and you can also use Command $+\mathrm{X} / \mathrm{C} / \mathrm{V}$ (Mac) or Control $+\mathrm{X} / \mathrm{C} / \mathrm{V}$ (Windows) for cut, copy and paste respectively. If you right-click, you get a menu with these basic editing operations on them as well.

Change the code to this:

```
x = 1;
    x = x + 1;
        Sibelius.MessageBox("1 + 1 = " & x);
```

You can check this makes sense (or, at least, some kind of sense) by clicking the Check Syntax button. If there are any blatant mistakes (e.g. missing semicolons) you will be notified where they are.

Then close the dialogs by clicking OK, OK again then Close. Run your amended plug-in from the Plug-ins menu and a message box with the answer $1+1=2$ should appear.

How does it work? The first two lines should be obvious. The last line uses $\&$ to stick two strings together. You cannot use + as this works only for numbers (if you try it in the example above, you will get an interesting answer!).

One pitfall: try changing the second line to:

$$
\mathbf{x}+=1 ;
$$

Then click Check syntax. You will encounter an error: this syntax (and the syntax $\mathbf{x + +}$ ) is allowed in various languages but not in ManuScript. You have to do $\mathbf{x}=\mathbf{x + 1}$;

## Where Plug-ins are Stored

Plug-ins supplied with Sibelius are stored in folders buried deep within the Sibelius program folder on Windows, and inside the application package (or "bundle") on Mac. It is not intended that end users should add extra plug-ins to these locations themselves, as we have provided a per-user location for plug-ins to be installed instead. When you create a new plug-in or edit an existing one, the new or modified plug-in will be saved into the per-user location (rather than modifying or adding to the plug-ins in the program folder or bundle):

- On Windows, additional plug-ins are stored at $\mathrm{C}: \backslash$ Users $<$ username $>\backslash$ AppData\Roaming\Avid\Sibelius\Plugins.
- On Mac, additional plug-ins are stored in subfolders at/Users/<username>/Library/Application Support/Avid/Sibelius/Plugins.

This is worth knowing if you want to give a plug-in to someone else. The plug-ins appear in subfolders which correspond to the categories in which they appear in the various Plug-ins galleries. The filename of the plug-in itself is the plug-in's internal name plus the .plg extension, such as Test.plg.
(Sibelius includes an automatic plug-in installer, which you can access via File > Plug-ins > Install Plug-ins. This makes it easy to download and install plug-ins from the Avid website.)

## Line Breaks and Comments

As with C/C++ and Java, you can put new lines wherever you like (except in the middle of words), as long as you remember to put a semicolon after every statement. You can put several statements on one line, or put one statement on several lines.

You can add comments to your program, again like C/C++ and Java. Anything after // is ignored to the end of the line. Anything between /* and */ is ignored, whether just part of a line or several lines:
// comment lasts to the end of the line
/* you can put
several lines of comments here
*/
For instance:
Sibelius.MessageBox("Hi!"); // print the active score
or:

```
Sibelius /* this contains the application */ .MessageBox("Hi!");
```


## Variables

$\mathbf{x}$ in the Test plug-in is a variable. In ManuScript a variable can be any sequence of letters, digits or _ (underscore), as long as it does not start with a digit.

A variable can contain an integer (whole number), a floating point number, a string (text) or an object (such as a note) -more about objects in a moment. Unlike most languages, in ManuScript a variable can contain any type of data-you do not have to declare what type you want. Thus you can store a number in a variable, then store some text instead, then an object.

Try this:

```
x = 56; x = x+1;
Sibelius.MessageBox(x); // prints '57' in a dialog box
x = "now this is text"; // the number it held is lost
Sibelius.MessageBox(x); // prints 'now this is text' in a dialog
x = Sibelius.ActiveScore; // now it contains a score
Sibelius.MessageBox(x); // prints nothing in a dialog
```

Variables that are declared within a ManuScript method are local to that method; in other words, they cannot be used by other methods in the same plug-in. Global Data variables defined using the plug-in editor can be accessed by all methods in the plug-in, and their values are preserved over successive uses of the plug-in.

A quick aside about strings in ManuScript is in order at this point. Like many programming languages, ManuScript strings uses the back-slash \as an "escape character" to represent certain special things. To include a single quote character in your strings, use $\backslash$ ', and to include a new line you should use $\backslash \mathrm{n}$. Because of this, to include the backslash itself in a ManuScript string one has to write $\backslash \backslash$.

## Converting Between Numbers, Text, and Objects

Notice that the method MessageBox is expecting to be sent some text to display. If you give it a number instead (as in the first call to MessageBox above) the number is converted to text. If you give it an object (such as a score), no text is produced.

Similarly, if a calculation is expecting a number but is given some text, the text will be converted to a number:

```
x = 1 + "1"; // the + means numbers are expected
    Sibelius.MessageBox(x); // displays '2'
```

If the text doesn't start with a number (or if the variable contains an object instead of text), it is treated as 0 :

```
x = 1 + "fred";
    Sibelius.MessageBox(x); // displays '1'
```


## Loops

## "for" and "while"

ManuScript has a while loop which repeatedly executes a block of code until a certain expression becomes True. Create a new plug-in called Potato. This is going to amuse one and all by writing the words of the well-known song " 1 potato, 2 potato, 3 potato, 4." Type in the following for the Run method of the new plug-in:

```
x = 1;
while (x<5)
{
    text = x & " potato,";
    Sibelius.MessageBox (text);
    x = x+1;
}
```

Run it. It should display " 1 potato," " 2 potato," " 3 potato," " 4 potato," which is a start, though annoyingly you have to click OK after each message.

The while statement is followed by a condition in ( ) parentheses, then a block of statements in \{ \} braces (you don't need a semicolon after the final \} brace). While the condition is true, the block is executed. Unlike some other languages, the braces are compulsory (you can't omit them if they only contain one statement). Moreover, each block must contain at least one statement.

In this example you can see that we are testing the value of $\mathbf{x}$ at the start of the loop, and increasing the value at the end. This common construct could be expressed more concisely in ManuScript by using a for loop. The above example could also be written as follows:

```
for x = 1 to 5
{
    text = x & " potato,";
    Sibelius.MessageBox (text);
}
```

Here, the variable $\mathbf{x}$ is stepped from the first value (1) up to the end value (5), stopping one step before the final value. By default, the "step" used is 1 , but we could have used (say) 2 by using the syntax for $\mathbf{x}=1$ to 5 step 2 , which would then print only " 1 potato" and " 3 potato"!

Notice the use of $\&$ to add strings. Because a string is expected on either side, the value of $\mathbf{x}$ is turned into a string.
Notice also we've used the Tab key to indent the statements inside the loop. This is a good habit to get into as it makes the structure clearer. If you have loops inside loops you should indent the inner loops even more.

## The if statement

Now we can add an if statement so that the last phrase is just "4," not " 4 potato":

```
x = 1;
while (x<5)
{
    if(x=4)
    {
        text = x & ".";
    }
    else
    {
        text = x & " potato,";
    }
    Sibelius.MessageBox (text);
    x = x+1;
}
```

The rule for if takes the form if (condition) \{statements\}. You can also optionally add else \{statements \}, which is executed if the condition is false. As with while, the parentheses and braces are compulsory, though you can make the program shorter by putting braces on the same line as other statements:

```
x = 1;
while (x<5)
{
    if(x=4) {
        text = x & ".";
    } else {
        text = x & " potato,";
    }
    Sibelius.MessageBox (text);
    x = x+1;
}
```

The position of braces is entirely a matter of taste.
Now let's make this plug-in really cool. We can build up the four messages in a variable called text, and only display it at the end, saving valuable wear on your mouse button. We can also switch round the if and else blocks to show off the use of not. Finally, we return to the for syntax we looked at earlier.

```
text = ""; // start with no text
for x = 1 to 5
{
    if (not(x=4)) {
        text = text & x & " potato, "; // add some text
} else {
        text = text & x & "."; // add no. 4
    }
}
Sibelius.MessageBox(text); // finally display it
```


## Arithmetic

Here is a complete list of the available arithmetic operators in ManuScript:

| $a+b$ | add |
| :--- | :--- |
| $a-b$ | subtract |
| $a * b$ | multiply |
| $a / b$ | divide |
| $a \%$ b | remainder |
| -a | negate |
| a) | evaluate first |

ManuScript evaluates operators strictly from left-to-right, unlike many other languages; so $2+3 * 4$ evaluates to 20 , not 14 as you might expect. To get the answer 14, you'd have to write $2+(3 * 4)$.

ManuScript supports both integers and floating point numbers. Use at least one floating point value in any arithmetic operation that might result in a floating point number, otherwise the result is rounded to the nearest integer (unless you are using literal strings). For instance, when calculating division using only integer values, the result is truncated; for example, the result of $3 / 2$ is 1 . However, using at least one floating point value in the calculation results in a floating point number (this is true if any or all of the values are a floating point number); for example, the result of $3.0 / 2$ is 1.5 .

Conversion from floating point numbers to integers can be achieved with the RoundUp (expr), RoundDown (expr), and Round (expr) functions, which can be applied to any expression.

## Objects

Now we come to the neatest aspect of object-oriented languages like ManuScript, C++ or Java, which sets them apart from traditional languages like BASIC, Fortran and C. Variables in traditional languages can hold only certain types of data: integers, floating point numbers, strings and so on. Each type of data has particular operations you can do to it: numbers can be multiplied and divided, for instance; strings can be added together, converted to and from numbers, searched for in other strings, and so on. But if your program deals with more complex types of data, such as dates (which in principle you could compare using $=,<$ and $>$, convert to and from strings, and even subtract) you are left to fend for yourself.

Object-oriented languages can deal with more complex types of data directly. Thus in the ManuScript language you can set a variable, let's say thischord, to be a chord in your score, and (say) add more notes to it:

```
thischord.AddNote(60); // adds middle C (note no. 60)
thischord.AddNote(64); // adds E (note no. 64)
```

If this seems magic, it's just analogous to the kind of things you can do to strings in BASIC, where there are very special operations which apply to text only:

> A\$ = "1"

A\$ = A\$ + " potato, ": REM add strings
X = ASC (A\$): REM get first letter code
In ManuScript you can set a variable to be a chord, a note in a chord, a bar, a staff or even a whole score, and do things to it. Why would you possibly want to set a variable to be a whole score? So you can save it or add an instrument to it for instance.

## Objects in Action

We'll have a look at how music is represented in ManuScript in a moment, but for a little taster, let's plunge straight in and adapt Potato to create a score:

```
x = 1;
text = ""; // start with no text
while (x<5)
{
    if (not(x=4)) {
        text = text & x & " potato, "; // add some text
    } else {
        text = text & x & "."; // add no. 4
    }
    x = x+1;
}
Sibelius.New(); // create a new score
newscore = Sibelius.ActiveScore; // put it in a variable
newscore.CreateInstrument("Piano");
staff = newscore.NthStaff(1); // get top staff
bar = staff.NthBar(1); // get bar 1 of this staff
bar.AddText(0,text,"Technique"); // use Technique text style
```

This creates a score with a Piano, and types our potato text in bar 1 as Technique text.
The code uses the period (.) several times, always in the form variable.variable or variable.method(). This shows that the variable before the period has to contain an object.

If there's a variable name after the period, we're getting one of the object's sub-variables (called "fields" or "member variables" in some languages). For instance, if $\mathbf{n}$ is a variable containing a note, then $n$.Pitch is a number representing its MIDI pitch ( 60 for middle C ), and n. Name is a string describing its pitch ("C4" for middle C). The variables available for each type of object are listed later.

If there's a method name after the period (followed by () parentheses), one of the methods allowed for this type of object is called. Typically a method called in this way will either change the object or return a value. For instance, if $s$ is a variable containing a score, then s.CreateInstrument("Flute") adds a flute (changing the score), but s.NthStaff (1) returns a value, namely an object containing the first staff.

Let's look at the new code in detail. There is a pre-defined variable called Sibelius, which contains an object representing the Sibelius program itself. We've already seen the method Sibelius. MessageBox (). The method call Sibelius.New () tells Sibelius to create a new score. Now we want to do something to this score, so we have to put it in a variable.

Fortunately, when you create a new score it becomes active (i.e. its title bar highlights and any other scores become inactive), so we can just ask Sibelius for the active score and put it in a variable:
newscore $=$ Sibelius.ActiveScore
Then we can tell the score to create a Piano: newscore. CreateInstrument ("Piano"). But to add some text to the score you have to understand how the layout is represented.

## Representation of a Score

A score is treated as a hierarchy: each score contains 0 or more staves; each staff contains bars (though every staff contains the same number of bars); and each bar contains "bar objects." Clefs, text and chords are all different types of bar objects.

## To add a bar object (i.e. an object which belongs to a bar), such as some text, to a score:

1 Specify which staff you want (and put it in a variable): staff = newscore.NthStaff(1).
2 Specify which bar in that staff you want (and put it in a variable): bar = staff. NthBar (1); finally you tell the bar to add the text: bar.AddText (0, text,"Technique").

3 Specify the name (or index number - see Text styles on page 141) of the text style to use (and it has to be a staff text style, because we're adding the text to a staff).

Notice that bars and staves are numbered from 1 upwards; in the case of bars, this is irrespective of any bar number changes that are in the score, so the numbering is always unambiguous. In the case of staves, the top staff is no.1, and all staves are counted, even if they're hidden. Thus a particular staff has the same number wherever it appears in the score.

The AddText method for bars is documented later, but the first parameter it takes is a rhythmic position in the bar. Each note in a bar has a rhythmic position that indicates where it is (at the start, one quarter after the start, etc.), but the same is true for all other objects in bars. This shows where the object is attached to, which in the case of Technique text is also where the left hand side of the text goes. Thus to put our text at the start of the bar, we used the value 0 . To put the text a quarter note after the start of the bar, use 256 (the units are 1024th notes, so a quarter is 256 units):

```
bar.AddText(256,text,"Technique");
```

To avoid having to use obscure numbers like 256 in your program, there are predefined variables representing different note values (which are listed later), so you could write:

```
bar.AddText(Quarter,text,"Technique");
```

or to be quaint you could use the British equivalent:
bar.AddText (Crotchet, text,"Technique");
For a dotted quarter, instead of using 384 you can use another predefined variable:
bar.AddText(DottedQuarter, text,"Technique");
or add two variables:
bar.AddText (Quarter+Eighth, text,"Technique");
This is much clearer than using numbers.

## The System Staff

As you know from using Sibelius, some objects don't apply to a single staff but to all staves. These include titles, tempo text, rehearsal marks and special barlines; you can tell they apply to all staves because (for instance) they get shown in all the instrumental parts.

All these objects are actually stored in a hidden staff, called the system staff. You can think of it as an invisible staff which is always above the other staves in a system. The system staff is divided into bars in the same way as the normal staves. So to add the title "Potato" to our score we'd need the following code in our plug-in:

```
sys = newscore.SystemStaff; // system staff is a variable
bar = sys.NthBar(1);
bar.AddText(0,"РОТАTO SONG","Subtitle");
```

As you can see, SystemStaff is a variable you can get directly from the score. Remember that you have to use a system text style (here Subtitle is used) when putting text in a bar in the system staff. A staff text style like Technique won't work. Also, you have to specify a bar and position in the bar; this may seem slightly superfluous for text centered on the page as titles are (though in reality even this kind of page-aligned text is always attached to a bar), but for Tempo and Metronome mark text they are obviously required.

## Representation of Notes, Rests, Chords, and Other Musical Items

Sibelius represents rests, notes and chords in a consistent way. A rest has no noteheads, a note has 1 notehead and a chord has 2 or more noteheads. This introduces an extra hierarchy: most of the squiggles you see in a score are actually a special type of Bar object that can contain even smaller things (namely, noteheads). There's no overall name for something which can be a rest, note or chord, so we've invented the pretty name NoteRest. A NoteRest with 0,1 or 2 noteheads is what you normally call a rest, a note or a chord, respectively.

If $\mathbf{n}$ is a variable containing a NoteRest, there is a variable $\boldsymbol{n}$. NoteCount which contains the number of notes, and $\mathbf{n}$. Duration which is the note-value in $1 / 256$ ths of a quarter. You can also get n . Highest and n . Lowest which contain the highest and lowest notes (assuming $n$. NoteCount isn't 0 ). If you set lownote $=\mathrm{n}$. Lowest, you can then find out things about the lowest note, such as lownote. Pitch (a number) and lownote. Name (a string). Complete details about all these methods and variables may be found in Reference.

Other musical objects, such as clefs, lines, lyrics and key signatures have corresponding objects in ManuScript, which again have various variables and methods available. For example, if you have a Line variable ln, then ln.EndPosition gives the rhythmic position at which the line ends.

## The "for each" Loop

It's a common requirement for a loop to do some operation to every staff in a score, or every bar in a staff, or every Bar object in a bar, or every note in a NoteRest. There are other more complex requirements which are still common, such as doing an operation to every Bar object in a score in chronological order, or to every Bar object in a multiple selection. ManuScript has a for each loop that can achieve each of these in a single statement.

The simplest form of for each is like this:

```
thisscore = Sibelius.ActiveScore;
for each s in thisscore // sets s to each staff in turn
{ // ...do something with s
}
```

Here, since thisscore is a variable containing a score, the variable s is set to be each staff in thisscore in turn. This is because staves are the type of object at the next hierarchical level of objects (see Hierarchy of Objects).

For each staff in the score, the statements in \{\} braces are executed.
Score objects contain staves, as we have seen, but they can also contain a Selection object, e.g. if the user has selected a passage of music before running the plug-in. The Selection object is a special case: it is never returned by a for each loop, because there is only a single Selection object; if you use the Selection object in a for each loop, by default it will return Bar objects (not Staves, Bars or anything else!).

Let's take another example, this time for notes in a NoteRest:

```
noterest = bar.NthBarObject(1);
for each n in noterest // sets n to each note in turn
{
    Sibelius.MessageBox("Pitch is " & n.Name);
}
```

$\mathbf{n}$ is set to each note of the chord in turn, and its note name is displayed. This works because Notes are the next object down the hierarchy after NoteRests. If the NoteRest is, in fact, a rest (rather than a note or chord), the loop will never be executed-you don't have to check this separately.

The same form of loop will get the bars from a staff or system staff, and the Bar objects from a bar. These loops are often nested, so you can, for instance, get several bars from several staves.

This first form of the for each loop got a sequence of objects from an object in the next level of the hierarchy of objects. The second form of the for each loop lets you skip levels of the hierarchy, by specifying what type of object you want to get. This saves a lot of nested loops:

```
thisscore = Sibelius.ActiveScore;
for each NoteRest n in thisscore
{
    n.AddNote(60); // add middle C
}
```

By specifying NoteRest after for each, Sibelius knows to produce each NoteRest in each bar in each staff in the score; otherwise it would just produce each staff in the score, because a Staff object is the type of object at the next hierarchical level of objects. The NoteRests are produced in a useful order, namely from the top to the bottom staff, then from left to right through the bars. This is chronological order. If you want a different order (say, all the NoteRests in the first bar in every staff, then all the NoteRests in the second bar in every staff, and so on) you'll have to use nested loops.

So here's some useful code that doubles every note in the score in octaves:

```
score = Sibelius.ActiveScore;
for each NoteRest chord in score
{
    if(not(chord.NoteCount = 0)) // ignore rests
    {
        note = chord.Highest; // add above the top note
        chord.AddNote(note.Pitch+12); // 12 is no. of half-steps (semitones)
    }
}
```

It could easily be amended to double in octaves only in certain bars or staves, only if the notes have a certain pitch or duration, and so on.
This kind of loop is also very useful in conjunction with the user's current selection. This selection can be obtained from a variable containing a Score object as follows:

```
selection = score.Selection;
```

We can then test whether it's a passage selection, and if so we can look at (say) all the bars in the selection by means of a for each loop:

```
if (selection.IsPassage)
{
    for each Bar b in selection
    {
        // do something with this bar
    }
}
```

Be aware that you can not add or remove items from bars during iterating. The example of adding notes to chords above is fine because you are modifying an existing item (in this case a NoteRest), but it's not safe to add or remove entire items, and if you try to do so, your plug-in will abort with an error. However, it's very useful to add or remove items from bars, so you need to do that in a separate for loop, after first collecting the items you want to operate on into a ManuScript array, something like this:

```
num = 0;
for each obj in selection
{
    if (IsObject(obj))
    {
        n = "obj" & num;
        @n = obj;
        num = num + 1;
    }
}
selection.Clear();
for i = 0 to num
{
    n = "obj" & i;
    obj = @n; // get an object from the pseudo array
    obj.Select();
}
```

The $@ \mathrm{n}$ in this example is the array.

## Indirection, Sparse Arrays, and User Properties

## Indirection

If you put the @ character before a string variable name, then the value of the variable is used as the name of a variable or method. For instance:

```
var="Name";
x = @var; // sets x to the contents of the variable Name
mymethod="Show";
@mymethod(); // calls the method Show
```

This has many advanced uses, though if taken to excess it can cause the brain to hurt. For instance, you can use @ to simulate "unlimited" arrays. If name is a variable containing the string " $\mathbf{x} 1$ ", then @name is equivalent to using the variable $\mathbf{x} 1$ directly. Thus:

```
i = 10;
name = "x" & i;
@name = 0;
```

sets variable $\mathbf{x 1 0}$ to $\mathbf{0}$. The last two lines are equivalent to $\mathbf{x}[i]=0$; in the C language. This has many uses; however, you'll also want to consider using the built-in arrays (and hash tables), which are documented below.

## Sparse Arrays

The method described above can be used to create "fake" arrays through indirection, though this is a little fiddly. ManuScript also provides Javascript-style sparse arrays, which can store anything that can be stored in a ManuScript variable, including references to objects. Like a variable, storing a reference to an object in a sparse array will preserve the lifetime of that object (because objects are reference counted), but the underlying object in Sibelius may become invalid if (say) a Score is modified.

To create a sparse array in ManuScript, use the built-in method CreateSparseArray (al,a2,a3,a4...an). You can create an empty array simply by passing in no variables to the CreateSpareArray method.

Sparse arrays provide a read/write variable called Length that returns or sets the length of the array: when you set Length to a number greater than the present size of the array, the array is padded with null values; if you set Length to a number smaller than the present size of the array, any values beyond this number are removed.

To push one or more values to the end of the array, use the method $\operatorname{Push}(a 1, a 2, \ldots a n)$. To remove and return the last element of an array, use the method Pop().

An example of how to use a sparse array:

```
array = CreateSparseArray (4,5,6);
array[10] = 19; // creates 11th element of array, intervening elements are null
array.Length = 20; // extends array to 20 elements, new elements are all null
```

Sparse arrays by their nature may not have values in every array element. To return a new sparse array containing only the populated indices of the original sparse array (those that are not null), use the array's ValidIndices variable. For example, using the above sparse array:

```
array2 = array.ValidIndices; // will contain values 0, 1, 2, 10 and 19
return array[array2[0]]; // returns the first populated element of array
```

You can compare two sparse arrays for equality, for example:

```
if (array = array2) {
    // do something
}
```

To access the end of an array, it's convenient to use negative indices; e.g. array [-1] returns the last element, array [-2] returns the penultimate element, and so on. It's not possible to access elements before the start of the array, so if you do e.g. array [-100] on a six element array, you will get array [0] returned.

## Some things to remember when using sparse arrays:

- Sparse arrays use a zero-based index.
- Elements that have not been initialized are null, and do not cause an error when referenced.
- Assigning to an index beyond the current length increases the Length to one greater than the index is assigned to.
- If an array contains references to objects, whether the arrays are equal or not depends on the implementation of equality for those objects.


## User Properties

All ManuScript objects other than those listed below, including objects created by Sibelius, can have user properties attached to them, allowing for convenient storage of extra data, encapsulation of several items of data within a single object, and returning more than one value from a method, among other things.

To create a new user property, use the following syntax:
object._property:property_name = value;
where object is the name of the object, property_name is the desired user property name, and value is the value to be assigned to the new user property. User properties are read/write and can be accessed as object.property_name.

To get a sparse array containing the names of all the user properties belonging to an object:
names = object._propertyNames;

Here is an example of creating a user property:

```
nr = bar.NoteRest;
nr._property:original = true;
if (nr.original = true) {
    // do something
}
```

Some things to remember when using user properties:

- If you attempt to get or set a user property that has not yet been created, your plug-in will exit with a run-time error.
- To check whether or not a user property has been created without causing a run-time error, use the notation object._property:property_name, which will be null if no matching user property has been created yet.
- User properties cannot be created or accessed for normal data types (e.g. strings, integers, etc.), the global Sibelius object, old-style ManuScript arrays created by CreateArray (), old-style hashes created by CreateHash (), and null.
- User properties that conflict with an existing property name cannot be accessed as object .property_name (though they can be accessed using the ._property : notation).
- User properties belong to a particular ManuScript object and disappear when that object's lifetime ends. To stop an object dying, you can (for example) store it in a sparse array, but be aware that its contents may become invalid if (say) the underlying score changes.


## Dictionary

Dictionary is a programmer extensible object, simply allowing the use of user properties as above with convenient construction. It also has methods allowing the use of arbitrarily named user properties, and can also have methods in plug-ins attached to it allowing the creation of encapsulated user objects (i.e. objects with variables and methods attached to them).

To create a dictionary, use the built-in function CreateDictionary (name1, value1, name2, value $2, \ldots$ nameN, valueN). This creates a dictionary containing user properties called name1, name 2 , name $N$ with values value1, value 2 , value $N$ respectively.

A dictionary can contain named data items (like a struct in languages like $\mathrm{C}++$ ), or data that is indexed by string, so that you can use strings to look items up within it.

The values in a dictionary can be accessed using square bracket notation, so you can use a dictionary like a hash table. For example:

```
test = CreateDictionary("fruit",apple,"vegetable",potato);
test["fruit"] = banana;
test["meat"] = lamb;
```

You can even put other objects, such as sparse arrays, inside dictionaries. For example:

```
test2 = CreateDictionary("fruit",
CreateSparseArray (apple,banana,orange));
```

You can access the user properties within a dictionary using the ._property: notation. For example:
return test2._property:fruit;
which would return the array specified above. Even more direct, you can access user properties in a dictionary as if they were variables or methods, like this:
test2.fruit;
which would also return the array specified above. You can also return more than one value from any ManuScript method using a dictionary, such as:

```
getChord()
value = CreateDictionary("a", aNote, "b", anotherNote);
return value;
//... in another method somewhere
chord = getChord();
trace (chord.a);
trace (chord.b);
```

which returns two values, $\mathbf{a}$ and $\mathbf{b}$, which you can access via e.g. chord. $\mathbf{a}$ and chord. $\mathbf{b}$.

You can compare two dictionaries for equality. For example:

```
if (test2 = test3) {
// do something
}
```

Whether or not dictionaries containing objects evaluate as equal depends on the implementation of equality for those objects.
If you're comfortable with programming in general, you may find it useful to be able to add methods to dictionaries, particularly if you are writing code designed to act as a library for other methods or plug-ins to call. Writing code in this way provides a degree of encapsulation and can make it easy for client code to use your library.

To add a method to a dictionary, call the dictionary's SetMethod () method. For example:

```
pluginmethod "(obj,x,y) {
// a method that does something to obj
}"
test4 = CreateDictionary();
test4.SetMethod("doSomething",Self,"pluginmethod");
test4.doSomething(3,4);
    // call pluginmethod within the current plug-in, passing in
    // test4 (obj in the method above) and 3 (x in the method
    // above) and 4 (y in the method above)
```

In the example above, doSomething is the name of the method belonging to the dictionary, Self tells the plug-in that the method is defined in the same plug-in, and pluginmethod is the name of a method elsewhere in the plug-in (shown at the top of the example).

To return a sparse array containing the names of the methods belonging to a dictionary, use the dictionary's GetMethodNames () method. You can also check the existence of a particular method using the dictionary's MethodExists () method. Use the dictionary's CallMethod () method to call a specific method, where the name of the method is the first parameter, and any parameters to be passed to the specified method follow.

For example:

```
array = test4.GetMethodNames(); // create sparse array containing method names
first_method_name = array[0]; // sets first_method_name to name of first method
metho\overline{dfound = test4.MethodExists("doSomething"); // returns True in this case;}
test4.CallMethod("doSomething",5,6);
```

Everything you put into a dictionary is a user property, so all of the methods outlined in User properties above can be used on data in dictionaries too.

## Using User Properties as Global Variables

You can store SparseArray and Dictionary objects, and indeed any other object, as user properties of the Plugin object itself. In the example below, Self is the object that corresponds to the running plug-in, and a user property globalData is assigned to the plug-in, containing a Dictionary:

```
Self._property:globalData = CreateDictionary(1,2,3,4);
// globalData and Self.globalData can be used interchangeably
trace (globalData);
trace(Self.globalData) ;
```

User properties assigned to the plug-in are persistent between invocations. Take care to ensure that these user properties are created before you attempt to use them, otherwise your plug-in will abort with a run-time error. Using the _property:property_name syntax never causes run-time errors, but direct references to property_name force a runtime error if property_name hasn't been created yet.

The example below shows how to test the existence of a specific user property, globalCounter, initialize it to 0 if it is not found, then increment it by 1 every time the plug-in runs:

```
// Test the persistence of user properties
if (Self.,property:globalCounter = null) {
    Self._property:globalCounter = 0;
}
globalCounter = globalCounter + 1;
// this number increases by one every time the plug-in is run
trace (globalCounter);
trace(Self.globalCounter);
```

If you store a reference to a musical object in a user property that is assigned to the plug-in, there is an increased danger of that reference becoming invalid due to the score being closed or edited, etc. Use the IsValid() method to validate such data before using it.

User properties of plug-ins will be inaccessible (except by using the _property : property_name syntax) if there is an existing global variable of the same name.

## Watch Out for Recursive Cycles!

Be careful not to create recursive cycles using arrays, user properties and dictionaries. When you use, say, an array in a dictionary, you are not creating a copy of the array or its values, but a reference to the original array: dictionaries and arrays are objects, not values. As a result, you could write something where an array contains a dictionary that itself refers to the original array: this will lead to Sibelius crashing. So be careful!

## Other Things to Look Out For

The Parallel 5ths and 8ves plug-in illustrates having several methods in a plug-in, which we haven't needed so far. The Proof-read plug-in illustrates that one plug-in can call another - it doesn't do much itself except call the CheckPizzicato, CheckSuspectClefs, CheckRepeats and CheckHarpPedaling plug-ins. Thus you can build up meta-plug-ins that use libraries of others. Cool!
(You object-oriented programmers should be informed that this works because, of course, each plug-in is an object with the same powers as the objects in a score, so each one can use the methods and variables of the others.)

## Dialog Editor

For more complicated plug-ins than the ones we've been looking at so far, it can be useful to prompt the user for various settings and options. This may be achieved by using ManuScript's simple built-in dialog editor. Dialogs can be created in the same way as methods and data variables in the plug-in editor.

## Showing a Dialog in a Plug-In

To show a dialog from a ManuScript method, we use the built-in call
Sibelius.ShowDialog (dialogName, Self);
where dialogName is the name of the dialog we wish to show, and Self is a "special" variable referring to this plug-in (telling Sibelius to whom the dialog belongs). Control will only be returned to the method once the dialog has been closed by the user.

## Creating or Editing a Dialog

To create a new dialog, choose the Dialog radio button at the bottom of the window that lists methods, data and dialogs, and click Add. To edit an existing dialog, select it from the Dialogs list box at the top right-hand corner of the window, and click Edit.

The dialog form will then appear, along with a long thin "palette" of available controls, as follows:

| $\bigcirc 1$ |  |
| :---: | :---: |
| ( | Radio button |
| ® | Checkbox |
| $\square$ | Button |
| A | Static text |
| ab | Editable text |
| E気 | Combo box |
|  | List box |
| E | Group box |

Control palette

To create a new control, just drag and drop it from the palette onto the dialog.

## Dialog Properties

With no controls selected, either double-click on a blank part of the dialog (or right-click, and then choose Properties) to access the dialog's Properties dialog, which allows you to specify:

- Name: the value of dialogName for the Sibelius.ShowDialog() method call (see Showing a dialog in a plug-in above).
- Title: the name of the dialog as it appears in its title bar.
- Size: the Width and Height (measured in somewhat arbitrary dialog units); you can also set the size of the dialog by resizing it directly when editing it.
- Position: the X and Y position that the dialog should open at by default.


## Laying Out Controls

The dialog editor includes a number of simple options for producing a pleasing layout:

- To select a control, either click it or hit Tab to select the next control in the creation order (Shift-Tab selects the previous control).
- To nudge a selected control, use the arrow keys.
- To align controls, select them using Command-click (Mac) or Control-click (Windows), then use Command+Left Arrow (Mac) or Control + Left Arrow (Windows) to align all of the selected controls with the left-hand edge of the left-most control, or Command +Up Arrow (Mac) or Control+Up Arrow (Windows) to align all of the selected controls with the top edge of the top-most control.
- To space controls evenly, select them using Command-click (Mac) or Control-click (Windows), then use Command + Option+Option + Down Arrow (Mac) or Control+Alt + Shift + Down Arrow (Windows) to space the controls evenly in the distance between the top edge of the top-most and the bottom edge of the bottom-most controls, or Command + Option+Option + Left Arrow (Mac) or Control + Alt + Shift + Left Arrow (Windows) to space the controls evenly in the distance between the left-hand edge of the left-most and the right-hand edge of the right-most controls. Once controls are spaced evenly, you can increase or decrease the space between them proportionally by typing Command+Option+Option+Up, Down, Right, Left Arrow keys (Mac) or Control+Alt+Shift+Up, Down, Right, Left Arrow keys (Windows) as appropriate.

You can optionally display a grid to aid with alignment. Right-click on a blank part of the dialog and choose Grid from the context menu to see a dialog with settings for the grid:


## Grid Settings dialog

Switch on Show grid to show the grid in the editor. Choose between Dots or Lines, and specify the Opacity of the grid display by adjusting the slider. Switch on Snap to grid to enable control snapping as you drag them with the mouse. Although a control that you nudge with the keyboard will not snap to the grid, one side of its selection outline will flash when it comes into alignment with the grid in either the horizontal or vertical directions.

## Undo and Redo

You can undo and redo everything you have done while editing a dialog using Command +Z (Mac) or Control +Z (Windows) to undo and Command +Y (Mac) or Control+Y (Windows) to redo.

## Testing the Dialog

To test the dialog within the editor, right-click a blank part of the dialog and choose Test from the context menu, or type the shortcut Command +T (Mac) or Control +T (Windows). To finish testing and return to the editor, press Esc or click any control whose properties are set to close the dialog (e.g. an OK or Cancel button, if you have created one).

## Saving Changes

To save the changes to the dialog, click the close button in the dialog's title bar. If there are any unsaved changes, Sibelius prompts you to save the changes.

## Set Creation Order

If you have done any programming in other languages that allow you to edit dialogs, you will probably be familiar with the concept of tab order, which refers to the order in which controls are given the focus when the user repeatedly hits the Tab key to cycle through them. ManuScript has a similar concept called creation order, so named because the order in which the controls in a dialog are created affects not only the tab order but also some other subtle things (including radio button grouping-see Radio Buttons).

To set the creation order of controls in your plug-in's dialog, right-click on a blank part of the dialog and choose Set Creation Order from the context menu. A special display appears overlaid on the controls in your dialog, like this:


Plug-in dialog with creation order overlay

To set the creation order, simply click on each control in order. If you make a mistake, press Command (Mac) or Control (Windows) and click on the last control whose order is correct to restart the sequence from that point, then release Command (Mac) or Control (Windows) and resume clicking on the remaining controls. Once you're done, press Esc to finish editing the creation order.

## Control Properties

Every control that you create also has a Properties dialog, which can be accessed by double-clicking a selected control, by right-clicking and choosing Properties from the context menu, or by pressing Command+Return (Mac) or Control+Return (Windows). The dialog for a radio button control, for example, is shown below:


[^1]With a control selected, the properties window varies depending on the type of the control, but most of the options are common to all controls, and these are as follows:

- ID: an internal string that identifies the control; Sibelius generates this for you automatically, but you can change if you like.
- Text: the text appearing in the control.
- Position (X, Y): where the control appears in the dialog, in coordinates relative to the top left-hand corner.
- Size (width, height): the size of the control.
- Variable storing control's value: the ManuScript Data variable that will correspond to the value of this control when the plug-in is run.
- Method called when clicked: the ManuScript method that should be called whenever the user clicks on this control (leave blank if you don't need to know about users clicking on the control).
- Click closes dialog: select this option if you want the dialog to be closed whenever the user clicks on this control. The additional options Returning True / False specify the value that the Sibelius. ShowDialog method should return when the window is closed in this way.
- Give this control focus: select this option if the "input focus" should be given to this control when the dialog is opened (such as whether this should be the control to which the user's keyboard applies when the dialog is opened). This is mainly useful for editable text controls.

Other options vary according to the type of control selected.

## Combo Boxes and List Boxes

Combo boxes and list boxes have an additional property; you can set a variable from which the control's list of values should be taken. Like the value storing the control's current value, this should be a global Data variable. However, in this instance they have a rather special format, to specify a list of strings rather than simply a single string. Look at the variable _ComboItems in Add String Fingering for an example - it looks like this:

```
_ComboItems
{
    "1"
    "2"
    "3"
    "4"
    "1 and 3"
    "2 and 4"
}
```

List boxes have one further property, which is to determine whether they should allow a single selection or multiple selections. The return value from a combo box or a single-selection list box is a single string. If a list box is set to allow multiple selections, the selection is returned as an array of strings.

## Radio Buttons

Radio buttons also have an additional property that allows one to specify groups of radio buttons in plug-in dialogs. When the user clicks on a radio button in a group, only the other radio buttons belonging to that groups are deselected; any others in the dialog are left as they are. This is extremely useful for more complicated dialogs.

To specify a radio group, pick one control from each group that represents the first button of the group, and for these controls ensure that the checkbox Start a new radio group is selected in the control's Properties dialog. Then set the creation order of the controls (see Set Creation Order). A radio button group is defined as being all the radio buttons created between two buttons that have the Start a new radio group flag set (or between one of these buttons and the end of the dialog). So to make the radio groups work properly, ensure that each group is created sequentially in order, with the button at the start of the group created first, and then all the rest of the radios in that group. To finish, click the Set Creation Order menu item again to deactivate this mode.

## Static Text

Static text controls additionally allow you to determine whether the text should be aligned to the Left (useful for explanatory text) or to the Right (useful for text associated with a specific control to its right, such as an edit control, checkbox or combo box).

## Buttons

In most plug-in dialogs, you will want the OK button to be the default button for the dialog, such that if the user presses Return or Enter on their keyboard, the dialog is confirmed, and closes. Likewise, you will want the Cancel button to respond to the user hitting Esc on their keyboard, closing the dialog without making any changes.

For OK buttons, or other buttons that should confirm the dialog, switch on the Default button for dialog checkbox in the button's Properties dialog. Each dialog should only have one default button. You will also normally set Click closes dialog, returning to True. Depending on the other controls in your dialog, you may additionally want to check Give this control focus; if you have one or more edit controls in the dialog, you should probably set Give this control focus on the first of those controls instead.

Cancel buttons, by contrast, should normally only have Click closes dialog, returning set to False.

## Debugging Plug-ins

When developing any computer program, it's all too easy to introduce minor (and not so minor!) mistakes, or bugs. ManuScript performs its own internal error checking at all times, and you'll find that if you try to access a non-existent method or variable on an object, or make a syntax error, or attempt to add or remove bars or items from bars while iterating over them, the plug-in will throw an error and open the plug-in editor window at the line that generated the error.

As ManuScript is a simple, lightweight system, there is no special purpose debugger, but there are a handful of tools provided to help you debug your plug-ins.

## Undo

One good technique for finding problems in your plug-ins is to set Sibelius's undo buffer to a very small size, or to disable it altogether (by dragging the slider on the Other page of File > Preferences to its leftmost position). In the unlikely event that ManuScript does not throw an error when you perform an illegal operation (e.g. adding or deleting an object while iterating over a bar), reducing the undo buffer to its smallest possible size will expose the problem right away - though be warned, the result of such a problem may be that Sibelius will crash.

## Plug-in Trace Window

The trace window can be shown by choosing Plug-ins > Plug-in Trace Window. A special ManuScript command, trace (string), will print the specified string in the trace window. This is useful to keep an eye on what your plug-in is doing at particular points. These commands can then be removed when you've finished debugging. Another useful feature of the trace window is function call tracing. When this is turned on, the log will show which functions are being called by plug-ins.

One potential pitfall with the trace (string) approach to debugging is that the built-in hash table and array objects discussed earlier aren't strings, and so can't be output to the trace window. To avoid this problem, both of these objects have a corresponding method called WriteToString (), which returns a string representing the whole structure of the array or hash at that point. So we could trace the current value of an array variable as follows:
trace ("array variable = " \& array.WriteToString());

## Checking the Validity of Objects

One of the common problems that you might encounter when writing complex plug-ins is that the object you are trying to operate on is no longer valid (e.g. it has already been deleted). You can enable error checking - either for all objects, or for individual objects - that will cause your plug-in to throw an error if an object is no longer valid.

To enable error checking, use the ManuScript command ValidationChecking (enable[, objectl[, object2]...]), and set the Boolean parameter enable to true. If enable is the only parameter, validation checking is enabled for all types of objects, and all plug-ins. If you supply one or more object parameters (e.g. Tuplet, Score, BarObject, etc.), only those objects will be checked, and only in the currently running plug-in. You should ensure ValidationChecking is set to false before you give your plug-ins to anybody else to use.

You can also use the special method IsValid() to determine whether an object is valid: it will return false if the object in question no longer exists. GetValidationError (object) returns an empty string if there is no error, or returns a string if an error has occurred, use trace (GetValidationError (score)) ; to trace any validation error returned by a Score object to the trace window.

## Stopping the Plug-in

If you want to force your plug-in to stop on a particular error condition, use StopPlugin ([message]), which will stop your plug-in, display the optional message in an alert box, and open the plug-in editor at the line of code reached.

You can also use ExitPlugin (), which exits the plug-in cleanly without dropping into the plug-in editor.

## Storing and Retrieving Preferences

In Sibelius 4 or later, you can use Preferences.plg, contributed by Hans-Christoph Wirth, to store and retrieve user-set preferences for your plug-ins.

## How Does it Work?

Preferences.plg stores its data in a text file in the user's application data folder. Strings are accessed as associated pairs of a key (the name of the string) and a value (the contents of the string). The value can also be an array of strings, if required.

## Initializing the Database

errorcode $=$ Open (pluginname,featureset);
Open the library and lock for exclusive access by the calling plug-in. The calling plug-in is identified with the string pluginname. It is recommended that this string equals the unique Sibelius menu name of the calling plug-in.

Parameter featureset is the version of the feature set requested by the calling plug-in. The version of the feature set is currently 020000 . Each library release shows in its initial dialog a list of supported feature sets. The call to Open () will fail and show a user message if you request an unsupported feature set. If you should want to prevent this user information (and probably setup your own information dialog), use CheckFeatureSet () first.

After Open () the scope is undefined, such that you can access only global variables until the first call to SwitchScope ().
Return value: Open () returns zero or a positive value on success. A negative result indicates that there was a fatal error and the database has not been opened.

- -2 other error
- -1 library does not support requested feature set
- 0 no common preferences database found
- 1 no preferences found for current plug-in
- 2 preferences for current plug-in loaded

In case of errors (e.g. if the database file is unreadable), open () offers the user an option to recover from the error condition. Only if this fails too will an error code be returned to the calling plug-in.

```
errorcode = CheckFeatureSet(featureset);
```

Check silently if the library supports the requested feature set.
Return value: CheckFeatureSet () returns zero or a positive value on success. A negative value indicates that the requested feature set is not supported by this version.

```
errorcode = Close();
```

Release the exclusive access lock to the library. If there were any changes since the last call to Open () or Write (), dump the data changes back to disk (probably creating a new score, if there was none present).

Return value: Close () returns zero or a positive value on success. A negative result indicates that there was a fatal error and the database has not been written.

```
errorcode = CloseWithoutWrite();
```

Release the exclusive access lock to the library, discarding any changes performed since last call to Open () or Write ().
Return value: CloseWithoutWrite () returns zero or a positive value on success. A negative result indicates that there was a fatal error, namely that the database was not open at the moment.

```
errorcode = Write(dirty);
```

Force writing the data back to disk immediately. Keep library locked and open. If dirty equals 0 , the write only takes place if the data has been changed. If dirty is positive, the common preferences score is unconditionally forced to be rewritten from scratch.

Return value: Write () returns zero or a positive value on success. A negative result indicates that there was a fatal error and the database has not been written.

## Accessing Data

```
index = SetKey(keyname, value);
```

Store a string value under the name keyname in the database, overwriting any previously stored keys or arrays of the same keyname.
If keyname has been declared as a local key, the key is stored within the current scope and does not affect similar keys in other scopes. It is an error to call SetKey () for local keys if the scope is undefined.

Return value: SetKey () returns zero or a positive value on success, and a negative value upon error.
errorcode $=$ SetArray (keyname, array, size) ;
Store an array of strings under the name keyname in the database, overwriting any previously stored keys or arrays of the same keyname. size specifies the number of elements in the array. A size of -1 is replaced with the natural size of the array, i.e., array. NumChildren.

If keyname has been declared as a local key, the array is stored within the current scope and does not affect similar keys in other scopes. It is an error to call SetArray () for local keys if the scope is undefined.

Return value: SetArray () returns zero or a positive value on success, and a negative value upon error.

```
value = GetKey(keyname);
```

Retrieve the value of key keyname from the database. It is an error to call GetKey () on an identifier which had been stored the last time using SetArray (). For local keys, the value is retrieved from the current scope which must not be undefined.

Return value: The value of the key or Preferences.VOID if no key of that name found.

```
size = GetArray(keyname, myarray);
```

Retrieve the string array stored under name keyname from the database. It is an error to call GetArray () on an identifier which has been stored the last time by SetKey (). For local arrays, the value is retrieved from the current scope which must not be undefined.

You must ensure before the call that myarray is of ManuScript's array type (i.e., created with CreateArray ()).
Return value: size equals the number of retrieved elements or -1 if the array was not found. Note that size might be smaller than myarray. NumChildren, because there is currently no way to reduce the size of an already defined array.

```
size = GetListOfIds(myarray);
```

Fill the array myarray with a list of all known Ids in the current score (or in the global scope, if undefined). Before you call this method, ensure that myarray is of ManuScript's array type (i.e. created with CreateArray ()).

Return value: returns the size of the list, which might be smaller than the natural size of the array, myarray.Numchildren.
index $=$ UnsetId (keyname);
Erase the contents stored with an identifier (there is no distinction between keys and arrays here). If the key is declared as local, it is erased only from the local scope which must not be undefined.

Return value: The return value is zero or positive if the key has been unset. A negative return value means that a key of that name has not been found (which is not an error condition).

## RemoveId (keyname) ;

Erase all contents stored in the database with an identifier (there is no distinction between keys and arrays here). If the key is declared as local, it is erased from all local scopes.

Return value: The return value is always zero.

```
RemoveAllIds();
```

Erase everything related to the current plug-in.
Return value: the return value is always zero.

## Commands for Local Variables

```
errorcode = DeclareIdAsLocal(keyname);
```

Declare an identifier as a local key. Subsequent calls to Set. . . and Get. . . operations will be performed in the scope which is set at that time. The local state is stored in the database and can be undone by a call to DeclareIdAsGlobal or RemoveId.

Return value: Non-negative on success, negative on error.

```
size = GetListOfLocalIds(myarray);
```

Fill the array myarray with a list of all Ids declared as local. Before you call this method, ensure that myarray is of ManuScript's array type (i.e. created with CreateArray ()).

Return value: Returns the size of the list, which might be smaller than the natural size of the array, myarray. NumChildren.

```
errorcode = SwitchScope(scopename);
```

Select scope scopename. If scope scopename has never been selected before, it is newly created and initialized with no local variables. Subsequent Set . . . and Get . . . operations for keys declared as local will be performed in scope scopename, while access to global keys is still possible.

The call SwitchScope (" ") selects the undefined scope which does not allow access of any local variables.
Return value: Non-negative on success, negative on error.

```
errorcode = RemoveScope();
```

Erase all local keys and arrays from the current scope and delete the current scope from the list of known scopes. It is an error to call RemoveScope () if the current scope is undefined. After the call, the database remains in the undefined scope.

```
errorcode = RemoveAllScopes();
```

Erase all local keys and arrays from all scopes and delete all scopes from the list of known scopes. After the call, the database remains in the undefined scope. Note that this call does retain the information which Ids are local (see DeclareIdAsLocal ()).

Return value: Non-negative on success.
string $=$ GetCurrentScope ();
Retrieve the name of the currently active scope, or the empty string if the database is in undefined scope.
Return value: Returns a string.

```
size = GetListOfScopes(myarray);
```

Fill the array myarray with a list of all known scope names. You must ensure before the call that myarray is of ManuScript's array type (i.e., created with CreateArray ()).

Return value: Returns the size of the list, which might be smaller than the natural size of the array, myarray.NumChildren.

## Miscellaneous

## Trace (tracelevel) ;

Select level of tracing for the library. Useful levels are: 0 for no trace, 10 for sparse trace, 20 for medium trace, 30 for full trace. This command can also be run when the library is not open, to specify the tracing level for the open () call itself.

## TraceData();

Writes a full dump of the data stored currently in ThisData array to the trace window. This is the full data belonging to the current plug-in. TraceData () always traces the data, regardless of the current trace level selected.

```
filename = GetFilename();
```

Return the full filename of the preferences database (including path).

## Editor();

Invoke the interactive plug-in editor. This method must not be called while the database is open. Direct calls to Editor () from plug-ins are deprecated, since the end-user of your plug-in will probably not expect to be able to edit (and destroy) the saved preferences of all plug-ins at this stage.

## Basic Example

Suppose you have a plug-in called myplugin and would like to save some dialog settings in a preferences file such that these settings are persistent over several Sibelius sessions and computer reboots. Your dialog may contain two checkboxes and a list box. Let DialogDontAskAgain and DialogSpeedMode be the global variables holding the status of the checkboxes, respectively, and let DialogJobList hold the contents of the list box item.

## The work with the database can be reduced to four steps:

1 Open the database and retrieve initial data. At begin of your plug-in, e.g. right at top of your Run () method, you have to add some code to initialize the database. You probably also want to initialize your global keys based on the information currently stored in the database. See below for a detailed example. (Depending on your program, you might have to define prefOpen as a global variable in order to prevent trying to access an unopened database in future.)

```
// At first define hard coded plug-in defaults, in case that the plug-in
// is called for the first time. If anything else fails, these defaults
// will be in effect.
DialogDontAskAgain = 0;
DialogSpeedMode = 0;
DialogJobList = CreateArray();
DialogJobList[0] = "first job";
DialogJobList[1] = "second job";
// Attempt to open the database
prefOpen = Preferences.Open( "myplugin", "020000" );
if( prefOpen >= 0 ) {
    // Database successfully opened. So we can try to load the
    // information stored last time.
    // It's a good idea to work with a private version scheme, in order
    // to avoid problems in the future when the plug-in is developed
    // further, but the database still contains the old keys. In our
    // example, we propose that the above mentioned keys are present
    // if "version" key is present and has a value of "1".
    version = Preferences.GetKey( "Version" );
```

```
    switch( version ) {
    case( "1" ) {
    // Now overwrite the above set defaults with the information stored
    // in the database.
    DialogDontAskAgain = Preferences.Getkey( "DontAskAgain" );
    DialogSpeedMode = Preferences.Getkey( "SpeedMode" );
    Preferences.GetArray( "JobList", DialogJobList );
    }
    default {
    // handle other versions/unset version gracefully here ...
        }
    }
}
```

2 Work with the data. After the initialization step, you can and should work with global variables DialogDontAskAgain, DialogSpeedMode, and DialogJobList as you are used to: read from them to base control flow decisions on their setting, write to them (mostly from within your own dialogs) to set new user preferences.

3 Write data back to the database. To make any changes persistent, you must tell the database the new values to be written to the hard disk. See below for a detailed example. According to taste, you can execute these lines each time the settings are changed, or only once, at the end of your plug-in.

```
if( prefOpen >= 0 ) {
    Preferences.SetKey( "Version", "1" );
    Preferences.SetKey( "DontAskAgain", DialogDontAskAgain );
    Preferences.SetKey( "SpeedMode", DialogSpeedMode );
    Preferences.SetArray( "JobList", DialogJobList, -1 );
}
```

4 Close the database. In any case, you must release the lock to the library on exit of your plug-in. This writes data actually back to disk, and enables other plug-ins to access the shared database later. To do this, use:

```
Preferences.Close();
```


## Reference

## Syntax

Here is an informal run-down of the syntax of ManuScript.
A method consists of a list of statements of the following kinds:

| Block | \{ statem ents \} <br> for example: ```{ a = 4; }``` |
| :---: | :---: |
| While | while \{ expression \} block for example: ```while (i < 3) { Sibelius.MessageBox(i); i = i + 1; }``` |
| Switch | ```switch (test-expression) { case (case-expression-1) block [case (case-expression-2) block] [default block]``` <br> The switch statement consists of a "test" expression, multiple case statements and an optional default statement. If the value of test-expression matches one of the case-expressions, then the statement block following the matching case statement will be executed. If none of the case statements match, then the statement block following the default statement will be executed. For example: ```switch (note.Accidental) { case (DoubleSharp) { Sibelius.MessageBox("Double sharp"); } case (DoubleFlat) { Sibelius.MessageBox("Double flat"); } default { Sibelius.MessageBox("No double"); } }``` |
| if else | ```if (expression) block[else block] for example: if (found) { Application.ShowFindResults (found); } else { Application.NotFindResults(); }``` |


| for each | for each variable in expression block <br> This sets variable to each of the sub-objects within the object given by the expression. <br> Normally there is only one type of sub-object that the object can contain. For instance, a Note <br> Rest (such as a chord) can only contain Note objects. However, if more than one type of sub-object is possible you can specify the type: <br> for each Type variable in expression <br> block <br> for example: ```for each NoteRest n in thisstaff { n.AddNote(60); // add middle C }``` |
| :---: | :---: |
| for | for variable = value to value [ step value] <br> block <br> The variable is stepped from the first value up to or down to the end value by the step value. It stops one step before the final value. <br> So, for example: ```for x=1 to note.NoteCount { ... }``` <br> works correctly. |
| assignment | variable $=$ expression; <br> for example: <br> value $=$ value +1 ; <br> or <br> variable. variable $=$ expression; <br> for example: <br> Question.CurrentAnswer=True; |
| method call | variable .identifier (comma-separated expressions) ; <br> for example: <br> thisbar.AddText(0,"Mozart","text.system.composer"); |
| selfmethod call | identifier (comma-separated expressions) ; <br> Calls a method in this plug-in, for example: CheckIntervals(); |
| return | return expression; <br> Returns a value from a plug-in method, given by the expression. If a method doesn't contain a return statement, then a "null" value is returned (either the number zero, an empty string, or the null object described below). |

## Expressions

Here are the operators, literals and other beasts you're allowed in expressions.

| Self | This is a keyword referring to the plug-in owning the method. You can pass yourself to other methods, for <br> example: <br> other.Introduce(Self); |
| :--- | :--- |
| null | This is a literal object meaning "nothing." |
| Identifier | This is the name of a variable or method (letters, digits or underscore, not starting with a digit) you can pre- <br> cede the identifier with @ to provide indirection; the identifier is then taken to be a string variable whose value <br> is used as the name of a variable or method. |
| vember variable | variable. variable |
| This accesses a variable in another object. |  |
| for example: |  |

Text in double quotes, for example: "some text." For strings that are rendered by Sibelius as part of the score, i.e. the contents of some text object, there is a small but useful formatting language allowing one to specify how the text should appear. These "styled strings" contain commands to control the text style. All commands start and end with a backslash ( $\backslash$ ) The full list of available styling commands is as follows:

| $\backslash n \backslash$ | New paragraph |
| :--- | :--- |
| $\backslash N \backslash$ | New line |
| $\backslash B \backslash$ | Bold on |
| $\backslash b \backslash$ | Bold off |
| $\backslash I \backslash$ | Italic on |
| $\backslash i \backslash$ | Italic off |
| $\backslash U \backslash$ | Underline on |
| \u\} $&{\text { Underline off }} \\ {\text { \fArial Black }} &{\text { Font change to Arial Black (for example) }} \\ {\text { Ictext. character. musictext }}$ |  |

\ctext.character.musictext\}
$\backslash f \_\backslash \quad$ Font change to text style's default font, including removing any active character
styles
$\backslash$ s123\Size change to 123 (units are $1 / 32$ nds of a space, not points)
$\mathrm{v} \backslash \quad$ Vertical scale in percent
$\backslash h \backslash \quad H o r i z o n t a l ~ s c a l e ~ i n ~ p e r c e n t ~$
$\backslash t \backslash \quad$ Tracking (absolute) in 1/32nds of a space
$\backslash p \ \quad$ Baseline adjustment: use normal, subscript, or superscript
<br>\$keyword $\backslash \quad$ Substitutes a string from the Score Info dialog (see below)
A consequence of this syntax is that backslashes themselves are represented by $\backslash \backslash$, to avoid conflicting with the above commands.
The substitution command $\backslash \$$ keywor $\backslash \backslash$ supports the following keywords:
Title, Composer, Arranger, Lyricist, MoreInfo, Artist, Copyright, Publisher and PartName.
Each of these correspond to a field in the File > Score Info dialog.
not expression
not
Logically negates an expression, for example:
not ( $x=0$ )
expression and expression
and
or
equality
subtract
add
minus
concatenation
subexpression
Logical and, for example:
FoxFound and BadgerFound
expression or expression
Logical or, for example:
FoxFound or BadgerFound
expression $=$ expression
Equality test, for example:
Name="Clock"
expression - expression
Subtraction, for example:
12-1
expression + expression
Addition, for example:
$12+1$

- expression

Inversion, for example:
-1
expression \& expression
Add two strings, for example:

$$
x=" 2 "+22 " ; / / \text { same as } x=4
$$

(expression)
For grouping expressions and enforcing precedence, e.g.

Name = "Fred" \& "Bloggs"; // `Fred Bloggs'
You can't use + as this would attempt to add two numbers, and sometimes succeed (!). For instance:
$(4+1)$ *5

```
method call forexample:
    x = monkey.CountBananas();
Identifier (comma-separated expressions);
self method call Calls a method in this plug-in, for example:
    x = CountBananas();
```


## Operators

## Condition Operators

You can put any expressions in parentheses after an if or while statement, but typically they will contain conditions such as $=$ and $<$. The available conditions are very simple:

| $a=b$ | equals (for numbers, text or objects) |
| :--- | :--- |
| $a<b$ | less than (for numbers) |
| $a>b$ | greater than (for numbers) |
| $c$ and d | both are true |
| $c$ or $d$ | either are true |
| not $c$ | inverts a condition, e.g. not $(x=4)$ |
| $<=$ | less than or equal to |
| $>=$ | greater than or equal to |
| $!=$ | not equal to |
| Use = to compare for equality, not == as found in C/C++ and Java. |  |

## Arithmetic

| $\mathrm{a}+\mathrm{b}$ | add |
| :--- | :--- |
| $\mathrm{a}-\mathrm{b}$ | subtract |
| $\mathrm{a} * \mathrm{~b}$ | multiply |
| $\mathrm{a} / \mathrm{b}$ | divide |
| $\mathrm{a} \% \mathrm{~b}$ | remainder |
| -a | negate |
| (a) | evaluate first |

ManuScript will evaluate expressions from left to right, so that $2+3 * 4$ is 20 , not 14 as you might expect. To avoid problems with evaluation order, use parentheses to make the order of evaluation explicit. To get the answer 14 , you'd have to write $2+(3 * 4)$.

ManuScript also now supports floating point numbers, so whereas in previous versions 3/2 would work out as 1 , it now evaluates to 1.5. Conversion from floating point numbers to integers is achieved with the RoundUp (expr), RoundDown (expr) and Round (expr) functions, which can be applied to any expression.

## Object Reference

## Hierarchy of Objects



Text, Clef, Line, TimeSignature, KeySignature, Highlight, Lyric, Barline, Tuplet, GuitarFrame, GuitarScaleDiagram, Comment, Annotationltem, NoteRest (these are all BarObjects)

Note (in NoteRests only)

## All Objects

## Methods

AddToPluginsMenu ("menu text", "function name") Adds a new menu item to the Plug-ins menu. When the menu item is selected the given function is called. This is normally only used by plug-ins themselves. This method may only be called once per plug-in (that is each plug-in may only add one item to the Plug-ins menu); subsequent method calls will be ignored.

Asc (expression) Returns the ASCII value of a given character (the expression should be a string of length 1 ).
CharAt (expression, position) Returns the character from the expression at the given (zero-based) position, for example CharAt ("Potato", 3) would give "a."

Chr (expression) Returns a character (as a string) with the given ASCII value. This can be used for inserting double quotes (") into strings with Chr (34).

Cmd (command name) An alias for the Sibelius.FindCommandID () function. For example:
Cmd("ToGGle ReVieW mODe");
Use Cmd () in conjunction with Sibelius.Execute () to execute commands by name rather than by CommandID. For example: Sibelius.Execute (Cmd("Select All"));

CreateArray () Returns a new array object.
CreateHash () Returns a new hash-table object.
GetInterpreterOption (optionName) Returns a boolean indicating whether the provided interpreter option is enabled or not.
GetValidationError (object) Returns the validation error, if any, of the specified object. Useful to pass validation errors to the plug-in trace window.

ExitPlugin() Exits the plug-in cleanly without dropping into the plug-in editor.
InterpreterOptionExists (optionName) Returns a boolean indicating whether the interpreter option exists in the running version of Sibelius.

IsObject (expression) Returns 1 (or True) if expression evaluates to an object rather than a null, boolean, string, or any number. (Not to be confused with the IsPassage variable of Selection objects!)

IsValid (object) Returns 1 (or True) if the object is valid, returns 0 (or False) if the object no longer exists (that is has been deleted).

JoinStrings (expression, delimiter) Joins together (concatenates) an array of strings into a single string, separated by the string delimiter.

Length (expression) Gives the number of characters in the value of the expression.
Round (expression) Returns the nearest integer to the value of the expression, for example Round (1.5) would be " 2 " and Round (1.3) would be " 1 ."

RoundDown (expression) Returns the nearest integer less than the value of the expression, for example RoundDown (1.5) would be "1."

Roundup (expression) Returns the nearest integer greater than the value of the expression, for example Roundup (1.5) would be " 2 ."
SetInterpreterOption (optionName, [optionValue]) Allows enabling or disabling a particular interpreter option. By default, if optionValue is not provided, it is assumed that the option should be enabled. Note that the scope of the options are per plug-in and that once set, the option value persists as long as the plug-in remains loaded. It should be called early in the execution of the plug-in (it doesn't hurt to call it more than once).

SplitString (expression, [delimiter, ][trimEmpty]) Splits a string into an array of strings, using the given delimiter. The delimiter can be a single character or a string containing several characters-for instance ". " would treat either a comma or full stop as a delimiter. The default delimiter is the space character. If the trimEmpty parameter is True then this will ignore multiple delimiters (which would otherwise produce some empty strings in the array). The default value of trimEmpty is False.

```
s=':a:b:c';
bits=SplitString(s,':', false);
// bits[0] = ''; bits[1] = 'a'; bits[2] = 'b' ...
s='a b c';
bits=SplitString(s,' ', true);
// bits[0] = 'a'; bits[1]='b' ...
```

StopPlugin ([message]) Stops the plug-in, and shows the optional message in an alert box. Opens the plug-in editor at the line of code reached.

StyleCmd (command name) An alias for the Sibelius. FindCommandID () function. For example:
Cmd("ToGGle ReVieW mODe");
Substring (expression, start, [length]) This returns a substring of the expression starting from the given start position (zero-based) up to the end of the expression, for example Substring ("Potato", 2) would give "tato". When used with the optional length parameter, Substring returns a substring of the of the expression starting from the given start position (zero-based) of the given length, for example Substring ("Potato", 2,2) would give "ta".

Trace (expression) Sends a piece of text to be shown in the plug-in trace window, for example Trace ("Here's a trace");
ValidationChecking (enable [, object1[, object2]...]) If enable is the only parameter, validation checking is enabled for all types of objects, and across all plug-ins. If you supply one or more object parameters (such as Tuplet, Score, BarObject, and so on), only those objects will be checked, and only in the currently running plug-in. You should ensure ValidationChecking is set to false before you give your plug-ins to anybody else to use.

## User Properties

All objects (except for the Sibelius object, old-style ManuScript arrays created using CreateArray (), old-style ManuScript hashes created using CreateHash (), and null) can also have user properties assigned to them.

## Accessibility

Accessed from the Sibelius object.

## Methods

None.

## Variables

ScoreDescription Returns true if Sibelius's built-in score description functionality is enabled (read/write).

## Annotationltem

Represents an annotation. Has no methods or variables but may be referenced in other functions, such as for filtering.

## Methods

None.

## Variables

None.

## Bar

A Bar contains BarObject objects.
for each variable in produces the BarObjects in the bar
for each type variable in produces the BarObjects of the specified type in the bar

## Methods

AddBarNumber (new bar number[,format[, extra_text[, prepend[,skip this bar]]]]) Adds a bar number change to the start of this bar. new bar number should be the desired external bar number. The optional format parameter takes one of the three pre-defined constants that define the bar number format; see Global Constants. The optional extra_text parameter takes a string that will be added after the numeric part of the bar number, unless the optional boolean parameter prepend is True, in which case the extra_text is added before the numeric part of the bar number. If the optional skip this bar parameter is True, the bar number change is created with the Don't increment bar number option set. Returns the BarNumber object created.

AddChordSymbolFromPitches (position, pitches[, instrument style]) Adds a chord symbol from the given array of pitches at the specified position. The optional instrument style parameter operates the same as in the AddGuitarFrame method (see above). If the method is unable to create a chord symbol, the method returns null; otherwise it returns the GuitarFrame object created.

AddClef (pos, concert pitch clef[,transposed pitch cleff) Adds a clef to the staff at the specified position. concert pitch clef determines the clef style when Notes > Transposing Score is switched off; the optional transposed pitch clef parameter determines the clef style when this is enabled. Clef styles should be an identifier like "clef.treble"; for a complete list of available clef styles, see Clef Styles. Alternatively you can give the name of a clef style, such as "Treble," but bear in mind that this may not work in non-English versions of Sibelius. Returns the Clef object created.

AddComment (sr,text[, color[, maximized]]) Adds a comment at the specified $s r$ position in the current bar, displaying the specified text. The optional color parameter allows you to specify the color of the comment that is created (if not specified, the comment is created with its default color), and the optional maximized Boolean parameter allows you to set the comment to be minimized (if not specified, the comment is created maximized by default). If you want to specify the maximized parameter without specifying a particular color, set color to $\mathbf{- 1}$.

AddCommentWithName (sr, text, username[, color[,maximized]]) Adds a comment that will display a given username at the specified $s r$ position in the current bar, displaying the specified text. The optional color parameter allows you to specify the color of the comment that is created (if not specified, the comment is created with its default color), and the optional maximized Boolean parameter allows you to set the comment to be minimized (if not specified, the comment is created maximized by default). If you want to specify the maximized parameter without specifying a particular color, set color to $\mathbf{- 1}$.

AddGraphic (file name , pos[, below staff[,$x$ displacement[, $y$ displacement[, size ratio]]]]) Adds a graphic above or below the bar at a given position. If below staff is True, Sibelius will position the graphic below the staff to which it is attached, otherwise it will go above (the default). You may additionally displace the graphic from its default position by specifying $x$ - and $y$ displacements. These should be expressed in millimeters, the latter defining an offset from the top or bottom line of the staff, as appropriate. By default, the graphic will be created 5 mm away from the staff. To adjust the size of the graphic, you may set a floating point number for its size ratio. When set to 1.0 (the default), the graphic will be created with a height equal to that of the staff to which it is attached. A value of 0.5 would therefore halve its size, and 2.0 double it. The graphic may be rescaled to a maximum of five times the height of its parent staff. This function returns True if successful, otherwise False.

AddGraphicToBlankPage (file name, nth page, $x$ offset, $y$ offset [, size ratio]) Adds a graphic to a blank page belonging to the current bar. nth page specifies the particular blank page you would like the graphic to, starting from 1. The $x$ offset and $y$ offset parameters are floating point values relative to the size of the page the graphic is being added to. For example, an $x$ offset of 0.0 would position the graphic at the very left of the page; 0.5 in the center. You may specify the size of the graphic by specifying a value for size ratio. This defaults to 1.0, which has the same effect as creating a graphic in Sibelius manually using Create $>$ Graphic. (As with AddGraphic, 0.5 would halve its size, and 2.0 double it.) The graphic may be rescaled to a maximum of five times its initial size. This function returns True if successful, otherwise False.

AddGuitarFrame (position, chord name[, instrument style[,fingerings]) Adds a chord symbol for the given chord name to the bar. The optional instrument style parameter should refer to an existing instrument type that uses tab, and should be specified by identifier; see Instrument Types. If instrument style is not specified, Sibelius will create a chord symbol that will optionally display a chord diagram using the default tab tuning associated with the instrument type used by the staff to which the chord symbol will be attached. The position is in $1 / 256$ th quarters from the start of the bar. The optional fingerings parameter gives the fingerings string to display above (or below) the guitar frame, if supplied. If the method is unable to create a chord symbol, the method returns null; otherwise it returns the GuitarFrame object created.

AddInstrumentChange (pos, styleID[ , add_clef[ ,show_text[, text_label[, show_warning[ , warning_label,[full_instrument_nam e[,short_instrument_name]]J]]]) Adds an instrument change to the bar at the specified position. styleID is the string representing the instrument type to change to (see Instrument Types for a list). The optional boolean parameter add_clef, True if not specified, determines whether Sibelius will add a clef change at the same position as the instrument change if required (that is if the clef of the new instrument is different to that of the existing instrument). show_text is an optional boolean parameter, True if not specified, determining whether or not the text label attached to the instrument change should be created shown (the default) or hidden. text_label is an optional string parameter; if specified, Sibelius will use this string instead of the default string (the new instrument's long name). show_warning is an optional boolean parameter, True if not specified, determining whether or not Sibelius should create a text object (using the Instrument change staff text style) above the last note preceding the instrument change, announcing the instrument change and giving the player time to pick up the new instrument. warning_label is an optional string parameter; if specified, Sibelius will use this string instead of the default string (the word "To" followed by the new instrument's short name). You can also override the names Sibelius will give the instruments on subsequent systems. If a null string is passed to either full_instrument_name or short_instrument_name (or if the arguments are omitted), the instrument names will remain unchanged. Returns the InstrumentChange object created.

AddKeySignatureFromText (pos, key name, major key[, add double barline[, hidden [, one staff only]]]) Adds a key signature to the bar. The key signature is specified by text name, such as "Cb" or "C\#". The third parameter is a Boolean flag indicating if the key is major (or minor). Unless the fourth parameter is set to False, a double barline will ordinarily be created alongside the key signature change. You may additionally hide the key signature change by setting hidden to True, and make the change of key appear on the bar's parent staff only with the one staff only flag. Returns the KeySignature object created.

AddKeySignature (pos , num sharps, major key[, add double barline[, hidden[, one staff only]]]) Adds a key signature to the bar. The key signature is specified by number of sharps $(+1$ to +7 ), flats $(-1$ to -7$)$, no accidentals $(0)$ or atonal $(-8)$. The third parameter is a Boolean flag indicating if the key is major (or minor). Unless the fourth parameter is set to False, a double barline will ordinarily be created alongside the key signature change. You may additionally hide the key signature change by setting hidden to True, and make the change of key appear on the bar's parent staff only with the one staff only flag. Returns the KeySignature object created.

AddLine (pos, duration, line style[, $d x[, d y[$, voicenumber[,hidden]]]]) Adds a line to the bar. The line style can be an identifier such as "line.staff.hairpin.crescendo" or a name, such as "Crescendo". For a complete list of line style identifiers that can be used in any Sibelius score, see Line Styles. Style identifiers are to be preferred to named line styles as they will work across all language versions of Sibelius. Returns the Line object created, which may be one of a number of types depending on the Line style used.

AddLiveTempoTapPoint (position) Adds a Live Tempo tap point at the rhythmic position specified by position, in 1/256th quarters from the start of the bar.

AddLyric (position, duration, text[, syllable type [, number of notes, voicenum]]]) This method adds a lyric to the bar. The position is in $1 / 256$ th quarters from the start of the bar, and the duration is in $1 / 256$ th quarter units. The two optional parameters allow you to specify whether the lyric is at the end of a word (value is " 1 ", and is the normal value) or at the start or middle of a word (value is " 0 "), and how many notes the lyric extends beneath (default value 1). You can also optionally specify the voice in which the lyric should be created; if voicenum is 0 or not specified, the lyric is created in all voices. Returns the LyricItem object created.
AddNote (pos, sounding pitch,duration, [tied [, voice[, diatonic pitch[,string number]]]]) Adds a note to staff, adding to an existing NoteRest if already at this position (in which case the duration is ignored); otherwise creates a new NoteRest. Will add a new bar if necessary at the end of the staff. The position is in $1 / 256$ th quarters from the start of the bar. The optional tied parameter should be True if you want the note to be tied. Voice 1 is assumed unless the optional voice parameter (with a value of $1,2,3$ or 4 ) is specified. You can also set the diatonic pitch, that is the number of the "note name" to which this note corresponds, 7 per octave ( $35=$ middle C, $36=\mathrm{D}, 37=\mathrm{E}$ and so on). If a diatonic pitch of zero is given, a suitable diatonic pitch will be calculated from the MIDI pitch. The optional string number parameter gives a string number for this note, which is only meaningful if the note is on a tablature stave. If this parameter is not supplied then a default string number is calculated based on the current tablature stave type and the guitar tab fingering options (specified on the Notes page of File > Preferences). Returns the Note object created (to get the NoteRest containing the note, use Note.ParentNoteRest).

AddPageNumber ([blank page offset $]$ ) Creates and returns a page number change at the end of the bar. Due to the nature of adding a page number change, a page break will also be created at the end of the bar. Therefore, the page number change will actually be placed at the start of the next bar. The desired properties of the page number change can be set by calling the appropriate methods on the Page Number Change object returned.

The blank page offset flag allows you to create page number changes on blank pages. If a BarObject is followed by one or more blank pages, each blank page may also have a page number change of its own. If unspecified, the page number change will be created on the next available page (whether it contains music or not) after the bar, otherwise the user may specify a 1-based offset which refers to the $n$th blank page after the bar itself.

AddPageNumberAtStartOfBar () Creates and returns a page number change at the start of the bar. This is useful for adding a page number change at the very start of the score, that is to change the initial page number, by using this method on the first bar of the score. If used on a bar later in the score, it will create the page number change at the end of the previous bar, but unlike the AddPageNumber method, it will not force a page break, so in general the AddPageNumber method is recommended.

AddRehearsalMark ([consecutive [,mark[, new prefix and suffix[, prefix[,suffx[,override defaults]]]]]) Adds a rehearsal mark above the bar. If no parameters have been specified, the rehearsal mark will inherit the properties of the previous rehearsal mark in the score, incrementing accordingly. Optionally, the appearance of the rehearsal mark may be overridden. If consecutive is False, Sibelius will not continue the numbering of the new rehearsal marks consecutively, but allow the user to set a new mark. A mark may be expressed as a number of a string. For example both 5 and "e" are both valid and equivalent values. If new prefix and suffix is True, the values set for prefix and suffix will be applied to the new rehearsal mark. The final parameter, override defaults, is a Boolean defaulting to False whose purpose it is to mimic the behavior of the option with the same name in the Rehearsal Mark dialog in Sibelius.

AddSpecialBarline (barline type[, pos]) Adds a special barline to a given position in a bar; see Global Constants. If no position has been specified, start repeat barlines will snap to the start of the bar by default. All other special barline types will snap to the end.

AddSymbol (pos,symbol index or name) Adds a symbol to the bar at the specified position. If the second parameter is a number, this is taken to be an index into the global list of symbols, corresponding to the symbol's position in the Create $>$ Symbol dialog in Sibelius (counting left-to-right, top-to-bottom from zero in the top-left hand corner). Some useful symbols have pre-defined constants; see Global Constants. There are also constants defined for the start of each group in the Create > Symbol dialog, so that to access the 8va symbol, for example, you can use the index OctaveSymbols +2 .

It's better to use indices rather than names, because the names will be different across the various language versions of Sibelius. Returns the Symbol object created, or null if no symbol can be added to the score.

AddText (pos, text, style[, voicenum]) Adds the text at the given position, using the given text style. A staff text style must be used for a normal staff, and a system text style for a system staff. The styles should be an identifier of the form "text.staff.expression"; for a complete list of text styles present in all scores, see Text Styles. Alternatively you can give the name of a text style, such as "Expression", but be aware that this may not work in non-English versions of Sibelius. You can also optionally specify the voice in which the lyric should be created; if voicenum is 0 or not specified, the text object is created in all voices. Returns the Text object created.

AddTextToBlankPage ( $x$ Pos, $y$ Pos, text, style, pageOffset) Adds the text at the given position, using the given text style. A blank page text style must be used; you cannot add staff text or system text to a blank page. style takes a style ID, using the form "text.blankpage.title"; for a complete list of text styles present in all scores, see Text Styles. $x P$ os and $y P o s$ are the absolute position on the page. pageOffset takes a positive number for a blank page following a special page break (the first blank page is 1 ), and negative for a blank page preceding the first bar of the score (the blank page immediately before the first bar is $\mathbf{- 1}$, the one before that $\mathbf{- 2}$, and so on). Returns the Text object created.

To add text to a blank page, first create the special page break using the Bar. BreakType variable, and set the number of blank pages using Bar. NumBlankPages or Bar. NumBlankPagesBefore. Then use Bar.AddTextToBlankPage.

AddTimeSignature (top, bottom, allow cautionary, rewrite music[, use symbol]) Returns an error string (which will be empty if there was no error) which if not empty should be shown to the user. The first two parameters are the top and bottom of the new time signature. The third tells Sibelius whether to display cautionary time signatures from this time signature. If rewrite music is True then all the bars after the inserted the time signature will be rewritten. You can also create common time and alla breve time signatures. If you're creating a time signature in $4 / 4$ or $2 / 2$, set use symbol to True and Sibelius will replace the numbers of the time signature with their symbolic equivalent.

AddTimeSignatureReturnObject (top, bottom, allow cautionary, rewrite music[, use symbol]) As above, but returns the time signature object created, or null if unsuccessful.

AddTuplet (pos, voice, left, right, unit[, style[, bracket[, fullDuration]]]) Adds a tuplet to a bar at a given position. The left and right parameters specify the ratio of the tuplet, for example 3 (left) in the time of 2 (right). The unit parameter specifies the note value (in $1 / 256$ th quarters) on which the tuplet should be based. For example, if you wish to create an eighth note (quaver) triplet group, you would use the value 128. The optional style and bracket parameters take one of the pre-defined constants that affect the visual appearance of the created tuplet; see Global Constants. If fullDuration is true, the bracket of the tuplet will span the entire duration of the tuplet. Returns the Tuplet object created.

If AddTuplet () has been given illegal parameters, it will not be able to create a valid Tuplet object. Therefore, you should test for inequality of the returned Tuplet object with null before attempting to use it.

Bar [array element] Returns the nth item in the bar (counting from 0) for example Bar [0]
Clear ([voice number]) Clears a bar of all its items, leaving only a bar rest. If a particular voice number is specified, only the items in that voice will be removed.

ClearNotesAndModifiers ([voice number]) Clears a bar of all its notes, rests, tuplets and slurs, replacing them with a single bar rest. If a particular voice number is specified, only the items in that voice will be removed.

Delete () Deletes and removes an entire bar from a score. This, by definition, will affect all the staves in the score.
DeletePageNumber ([blank page offset]) Deletes the page number change at the end of the bar, or if there are one or more blank pages after the bar, any page number change that occurs on any of those blank pages. If blank page offset is unspecified, the page number change on the first page after the bar will be deleted.

GetClefAt (pos) Returns a Clef object corresponding to the current clef at the specified rhythmic position.
GetKeySignatureAt (pos) Returns a KeySignature object corresponding to the current clef at the specified rhythmic position.
GetInstrumentTypeat (pos) Returns an InstrumentType object representing the instrument type used by the bar at the specified rhythmic position.

GetPageNumber ([blank page offset]) Returns the page number change object at the end of the bar, or if the bar contains no page number change, null. As with AddPageNumber, you may get the page number change from any of the blank pages that follow the bar by specifying a valid blank page offset.

InsertBarRest (voice number[, rest type]) Inserts a bar rest into the bar, but only if the bar is void of any NoteRests (or an existing bar rest) using the same voice number. The optional rest type parameter allows you to specify the type of bar rest or repeat bar to be created, defined by the constants WholeBarRest (the default if rest type is not specified), BreveBarRest, OneBarRepeat, TwoBarRepeat and FourBarRepeat. Returns True if successful.

NthBarObject ( $n$ ) Returns the nth object in the bar, counting from 0 .
RemoveLiveTempoTapPoint (position) Removes a Live Tempo tap point at the rhythmic position specified by position, in $1 / 256$ th quarters from the start of the bar.

ResetSpaceAroundBar (above, below) Does the equivalent of Layout > Reset Space Above Staff and/or Reset Space Below Staff for the given bar. Set above to True to reset the space above the staff, and below to True to reset the space below the staff.

Respace() Respaces the notes in this bar.

## Variables

BarNumber The bar number of this bar. This is the internal bar number, which always runs consecutively from 1 (read only).
BarObjectCount The number of objects in the bar (read only).
BreakType The break at the end of this bar, given by the constants MiddleOfSystem, EndOfSystem, MiddleOfPage, EndOfPage, NotEndOfSystem, EndOfSystemOrPage or SpecialPageBreak. To learn the correspondence between these constants and the menu in the Bars panel of the Properties window, see the discussion in Global Constants.

When you set the break at the end of a bar to be SpecialPageBreak, Sibelius will add one blank page after the break. You can then adjust the number of pages by setting the value of either Bar. NumBlankPages or Bar. NumBlankPagesBefore, or tell Sibelius to restart the music on the next left or right page with Bar.MusicRestartsOnPage.

ExternalBarNumber This has been deprecated as of Sibelius 5, because it can only return a number, and bar numbers that appear in the score may now include text. Use ExternalBarNumberString instead, which returns the external bar number of this bar, taking into account bar number changes in the score (read only). Note that you cannot pass this bar number to any of the other ManuScript accessors; they all operate with the internal bar number instead.

ExternalBarNumberString The external bar number of this bar as a string, taking into account bar number changes and bar number format changes (read only). Note that you cannot pass this bar number to any of the other ManuScript accessors; they all operate with the internal bar number instead.

GapAfter Sets the gap (in spaces) after the bar (read/write)
GapBefore Sets the gap (in spaces) before the bar (read/write).
InMultirest Returns one of four global constants describing if and/or where the bar falls in a multirest (read only). The constants are NoMultirest, StartsMultirest, EndsMultirest and MidMultirest; see Global Constants.

Length The rhythmic length (read only).
MusicRestartsOnPage Tells Sibelius to restart the music on the next left or right page after a special page break, and can only be set if BreakType is SpecialPageBreak. This variable may be set to only two of the global special page break constants:
MusicRestartsOnNextLeftPage or MusicRestartsOnNextRightPage (write only).
NthBarInSystem Returns the position of the bar in the system, relative to the first bar on the system (bar 0) (read only).
NumBlankPages The number of blank pages following the bar containing a special page break.
NumBlankPagesBefore The number of blank pages preceding the bar containing a special page break. This value only has an effect if a special page break exists in bar 1.

OnHiddenStave Returns True if the bar is currently hidden by way of Hide Empty Staves (read only).
OnNthPage Returns the zero-based page number on which the bar occurs in the current part (read only).
OnNthPageExternal Returns a string containing the external page number (which is the page number displayed in the score) of the page in which the bar occurs.

OnNthSystem Returns the zero-based system number (relative to its parent page) in which the bar occurs (read only).
ParentStaff The staff containing this bar (read only).
SectionEnd Corresponds to the Section end checkbox on the Bars panel of Properties (read/write).
Selected Returns True if the entire bar is currently selected (read only).
SpecialPageBreakType Returns the type of the special page break; see the documentation for the Special page break types in Global Constants (read only).

SplitMultirest When True, a multirest intersecting the bar in question will be split (read/write).
Time The time at which the bar starts in the score in milliseconds (read only).

## Barline

Accessed from a Barlines object.

## Methods

None.

## Variables

BottomStave Returns the Staff object at which the barline ends, relative to the current part.
BottomStaveNum Returns the number of the bottom staff included in the barline, relative to the current part.
TopStave Returns the Staff object at which the barline starts, relative to the current part.
TopStaveNum Returns the number of the top staff included in the barline, relative to the current part.

## Barlines

Accessed from a Score Object. Corresponds to the barline groupings in the score.
for each barline in iterates through all the barlines in the list, for example:
s = Sibelius.ActiveScore;
barlines = s.Barlines;
for each barline in barlines \{
// do something with barlines here
\}
Array access [int $n$ ] returns the $n$th barline in the list, or null if the barline does not exist.

## Methods

AddBarline (top staff number, bottom staff number) Creates a new bar line inclusively spanning the staff numbers (relative to the current part) supplied. Returns the new Barline object created, or null if it fails.

ClearAll () Removes all the barlines from the score.
DeleteNthBarline (index) Removes a given barline identified by index from the score.

## Variables

NumChildren Returns the number of unique barlines in the score (read only).

## BarObject

BarObjects include Clef, Line, NoteRest, and Text objects. All the methods and variables below apply to all specific types of BarObject - they are listed here instead of separately for each type. (To object-oriented programmers, the NoteRest, Clef, and those types are derived from the base class BarObject.)

## Methods

Delete () Deletes an item from the score. This will completely remove text objects, clefs, lines and so on from the score; however, when a NoteRest is deleted, it will be converted into a rest of similar duration. To delete multiple items from a bar, see Deleting Multiple Objects from a Bar.

Deselect () Removes the object from the selection list of the parent score. If the selection is currently a passage selection, it is first changed to a multiple selection before the object is deselected. Returns True if the object is successfully removed from the selection.

FreezeMagneticLayoutPositions () Does the same as selecting an object and choosing Layout > Freeze Magnetic Layout Positions, that explicitly sets the object's Dx/Dy to the position produced by Magnetic Layout, then disables Magnetic Layout for that object.

GetIsInVoice (voicenum) Returns True if the object is in the voicenum specified.
GetPlayedOnNthPass ( $n$ ) Returns True if the object is set to play back the $n$th time.

NextItem ([voice[, item type]]) Returns the next item in the parent bar of the current item, or null if no item exists. If no arguments have been supplied, the very next item in the bar will be returned, regardless of its voice number and item type. You may additionally specify the voice number of the object you're looking for ( 1 to 4 , or 0 for any voice number), and the item's type. Note that an item will only be returned if it exists in the same bar as the source item. By way of example, to find the next crescendo line in voice 2 , you would type something along the lines of: hairpin = item.NextItem(2, "CrescendoLine");

PreviousItem ([voice[, item type]]) As above, but searches backwards.
RemoveVoice (voicenum) Removes the object from the specified voicenum, leaving the object in all remaining voices.
ResetPosition ([horizontal[, vertical]]) Performs Layout > Reset Position on the object. If you supply no parameters, this method will reset both the horizontal and vertical position of the object. If either or both of the optional Boolean parameters horizontal or vertical is set to True, you can reset the position of the object either horizontally or vertically independently if required.

ResetDesign() Performs Layout > Reset Design on the object.
Select () Appends the object to the selection list of the parent score. A multiple selection consisting of any number of individual objects can be built up by repeatedly calling Select on each object you wish to add to the list. Note that calling Select on a BarObject will first clear any existing passage selection.

SetAllVoices () Sets the object to be in all voices. This has no effect on some types of object, such as NoteRests.
SetVoice (voicenum [, clear other voices]) Sets the object to be in voice voicenum, optionally removing the object from all other voices if the Boolean parameter clear other voices is True.

ShowInAll () Shows the object in the full score, and in all relevant parts; equivalent to Edit > Hide or Show > Show In All.
ShowInParts () Hides the object in the full score, and shows it in all relevant parts; equivalent to Edit > Hide or Show > Show In Parts.

ShowInScore () Hides the object in all relevant parts, and shows it in the full score; equivalent to Edit $>$ Hide or Show $>$ Show In Score.

SetPlayedOnNthPass ( $n$, do play) Tells Sibelius whether or not the object should play back the $n$th time.
TimeOnNthPass ( $n$ ) Returns the time at which the object occurs in the score in milliseconds on the $n$th pass through the score, where $n$ is an integer specifying the pass (specify 1 for the first pass through the score), or returns $\mathbf{- 1}$ in the case of an error (because the specified value of $n$ is out of range).

## Variables

CanBeInMultipleVoices Returns True if the object can be in more than one voice (read-only).
Color The color of this BarObject (read/write). The color value is in 24 -bit RGB format, with bits $0-7$ representing blue, bits 8 15 green, bits 16-23 red and bits 24-31 ignored. Since ManuScript has no bitwise arithmetic, these values can be a little hard to manipulate; you may find the individual accessors for the red, green and blue components to be more useful (see below). In order to write a value to this variable, the object must be selected.

ColorAlpha The alpha channel component of the color of this BarObject, in the range $0-255$ (read/write). In order to write a value to this variable, the object must be selected.

ColorRed The red component of the color of this BarObject, in the range $0-255$ (read/write). In order to write a value to this variable, the object must be selected.

Colorgreen The green component of the color of this BarObject, in the range $0-255$ (read $/ \mathrm{write}$ ). In order to write a value to this variable, the object must be selected.

ColorBlue The blue component of the color of this BarObject, in the range $0-255$ (read/write). In order to write a value to this variable, the object must be selected.

CueSize True if the object is cue-size in the current part or score, and False if the object is normal size (read/write).
CurrentTempo Returns the tempo, in bpm, at the location of the object in the score (read only).

DrawOrder Returns the layer at which the object is currently drawn. When used to set the layer of an object, values from 1 (meaning the bottom layer) to 32 (meaning the highest layer) can be used; 0 is a special value that tells Sibelius to use the default layer for that type of object (read/write).

Dx The horizontal graphic offset of the object from the position implied by the Position field, in units of $1 / 32$ spaces (read/write).
Dy The vertical graphic offset of the object from the center staff line, in units of $1 / 32$ spaces, positive going upwards (read $/ \mathrm{write}$ ).
HasCustomDrawOrder Returns True if the object is set to a layer other than its default layer (read only).
Hidden True if the object is hidden in the current part or score, and False if the object is shown (read/write).
OnNthBlankPage Returns 0 if the object occurs on a page of music, otherwise a number from 1 upwards indicating the $n$th blank page of the bar on which the object occurs (read only).

ParentBar The Bar containing this BarObject (read only).
Position Rhythmic position of the object in the bar (read only).
Selected Returns True if the object is currently selected (read only).
Time The time at which the object occurs in the score in milliseconds; if the score contains repeats, this will always return the time as if for the first pass through the score (read only). Returns -1 in the case of an error.

Type A string describing the type of object, such as "NoteRest," "Clef." This is useful when hunting for a specific type of object in a bar. See GuitarScaleDiagram for the possible values (read only).

UsesMagneticLayout Returns True if the object is positioned by Magnetic Layout. Returns False if the object is set not to be taken into account by Magnetic Layout. To set whether or not an object should use Magnetic Layout, use one of the global constants AlwaysDodge (equivalent to Edit > Magnetic Layout > n), SuppressDodge (Edit > Magnetic Layout > Off) or DefaultDodge (Edit > Magnetic Layout > Default) (read/write).

UsesMagneticLayoutSettingOverridden Returns True if the object has had its Magnetic Layout settings overridden; otherwise False.

VoiceNumber Is 0 if the item belongs to more than one voice (a lot of items belong to more than one voice) and $\mathbf{1}$ to $\mathbf{4}$ for items that belong to voices 1 to 4 (read only).

Voices Returns or sets Sibelius's internal bit field that represents the voices to which an object belongs; useful for copying the voices used by a given object (read/write).

## Deleting Multiple Objects from a Bar

If you wish to delete multiple objects from a bar, you should first build up a list of items to delete, then iterate over the list deleting each object in turn. It is not sufficient to simply delete the objects from the bar as you iterate over them, as this may cause the iterator to go out of sync.

Therefore, code to delete all tuplets from a bar should look something like this:

```
counter = 0;
for each Tuplet tup in bar {
    name = "tuplet" & counter;
    @name = tup;
    counter = counter + 1;
}
// Delete objects in reverse order
while(counter > 0) {
    counter = counter - 1;
    name = "tuplet" & counter;
    tup = @name;
    tup.Delete();
}
```


## BarRest

Derived from a BarObject object.

## Methods

None.

## Variables

PauseType Returns the type of fermata (pause), if any, on the bar rest. Returns one of the constants PauseTypeNone (0), PauseTypeSquare (1), PauseTypeRound (2), PauseTypeTriangular (3) (read/write).

RestType Returns the type of bar rest via one of the constants WholeBarRest (0), BreveBarRest (1), OneBarRepeat (2), TwoBarRepeat (3), FourBarRepeat (4) (read only). To create a bar rest of a particular type, use bar.InsertBarRest () (see above).

## Bracket

Accessed from a BracketsAndBraces object.

## Methods

None.

## Variables

BottomStave Returns the Staff object at which the bracket ends, relative to the current part.
BottomStaveNum Returns the number of the bottom staff included in the bracket, relative to the current part.
BracketType Returns the type of the bracket: BracketFull, BracketBrace or BracketSub.
TopStave Returns the Staff object at which the bracket starts, relative to the current part.
TopStaveNum Returns the number of the top staff included in the bracket, relative to the current part.

## Brackets and Braces

Accessed from a Score object. Describes the brackets (which may be brackets, sub-brackets or braces) present in the score. for each bracket in iterates through all the brackets in the list. Array access [int $n$ ] returns the $n$th bracket in the list, or null if the bracket does not exist.

## Methods

AddBracket (type, top staff number, bottom staff number) Creates a bracket of a given type, spanning the range of staves specified between top staff number and bottom staff number inclusive, and returns the new Bracket object. The staff numbers are relative to the current part view. Values for type are BracketFull (0), BracketBrace (1) and BracketSub (2).

ClearAll () Removes all existing brackets, sub-brackets and braces from the current part, and returns the number of brackets removed.

DeleteNthBracket ( $n$ ) Deletes the $n$th bracket from the current part, and returns True if successful.

## Variables

NumChildren Returns the number of child brackets, sub-brackets and braces in the list.

## Clef

Derived from a BarObject.

## Methods

None.

## Variables

ClefStyle The name of this clef, which may be different depending on the state of Notes > Transposing Score (read only).
ConcertClefStyleId The concert pitch identifier of the style of this clef (read only).
ConcertClefStyle The concert pitch name of this clef (read only).
StyleId The identifier of the style of this clef, which may be different depending on whether or not Notes > Transposing Score is enabled. This can be passed to the Bar.AddClef method to create a clef of this style (read only).

TransposingClefStyle The transposing score name of this clef (read only).
TransposingClefStyleId The transposing score identifier of the style of this clef (read only).

## Comment

Derived from a BarObject.

## Methods

None; create via BarObject.

## Variables

Maximized Returns True if the comment is maximized, otherwise returns False (read/write).
Text Returns the text of the comment (read/write).
TextWithFormatting Returns an array containing the various changes of font or style (if any) within the comment's text in a new element (read only). For example, "This text is $\backslash B \backslash b o l d \backslash b \backslash$, and this is $\backslash l i t a l i c l i \mid "$ would return an array with eight elements containing the following data:

```
arr[0] = "This text is "
arr[1] = "\B\"
arr[2] = "bold"
arr[3] = "\b\"
arr[4] = ", and this is "
arr[5] = "\I\"
arr[6] = "italic"
arr[7] = "\i\"
```

TextWithFormattingAsString The comment's text including any changes of font or style (read only).
TimeStamp Returns a DateTime object corresponding to the date the comment was created or last edited (read only).
UserName Returns the username of the user who created or last edited the comment (read only).

## ComponentList

An array that is obtained from Sibelius. HouseStyles or Sibelius.ManuscriptPapers. It can be used in a for each loop or as an array with the [ n ] operator to access each Component object:

## Methods

None.

## Variables

NumChildren Number of plug-ins (read only).

## Component

This represents a Sibelius "component," namely a house style or a manuscript paper. Examples:
// Create a new score using the first manuscript paper
papers=Sibelius.ManuscriptPapers;
score=Sibelius.New (papers[0]);
// Apply the first house style to the new score
styles=Sibelius.HouseStyles;
score.ApplyStyle(styles[0], "ALLSTYLES");

## Methods

None.

## Variables

Name The name of the component (read only).

## DateTime

This object returns information about the current date and time.

## Methods

None.

## Variables

Seconds Returns the number of seconds from the time in a date (read only).
Minutes Returns the number of minutes from the time in a date (read only).
Hours Returns the number of hours from the time in a date (read only).
DayOfMonth Returns the nth day on the month, 1-based (read only).
Month Returns the nth month of the year, 1-based (read only).
Year Returns the year (read only).
NthDayOfWeek Returns the nth day of the week, 0 -based (read only).
NthDayOfYear Returns the nth day of the year, 0 -based (read only).
LongDate Returns the date in a human-readable format, for example: 1st May 2008 (read only).
ShortDate Returns the date in a human-readable format, for example: 01/05/2008 (read only).

LongDateAndTime Returns the date and time in a human-readable format, for example: 1st May 2008 14:07 (read only).
ShortDateAndTime Returns the date and time in a human-readable format, for example: 01/05/2008 14:07 (read only).
TimeWithSeconds Returns the time in a human-readable format, for example: 14:07 (read only).
TimeWithoutSeconds Returns the time in a human-readable format, for example: 14:07:23 (read only).

## Dictionary

To create a dictionary, use the built-in function CreateDictionary (name1, value1, name2, value2, ... nameN, valueN). This creates a dictionary containing user properties called name1, name2, name $N$ with values value1, value 2 , value $N$ respectively.

## To iterate over dictionaries:

1 To iterate over element values in Dictionary objects, use for each $n$ in Dictionary or for each Value $n$ in Dictionary.

2 To iterate over element names in Dictionary objects, use for each Name $n$ in Dictionary.
3 To iterate over value name pairs in Dictionary objects, use for each Pair $n$ in Dictionary; this returns a new Dictionary object: $n$. Name is the element name, $n$. Value is the element value.

## Methods

CallMethod (methodname, param1,param $2, \ldots$ paramN) Calls the specified method methodname in the dictionary, passing in any other values that are required for the method as further parameters.

GetMethodNames () Returns a sparse array containing the names of the methods belonging to a dictionary.
GetPropertyNames () Returns a sparse array of the names of all the user properties in the dictionary (same as _propertyNames).
MethodExists (methodname) Returns True if the specified method methodname exists in the dictionary.
PropertyExists (propertyname) Returns True if the specified user property propertyname exists in the dictionary.
SetMethod (methodname, Self,method) Binds a method to the dictionary. methodname is the name by which you want to access the method via the dictionary, Self refers to the plug-in in which the method is found, and method is the name of the method itself, found elsewhere in the plug-in.

## Variables

None.

## Converting Old-Style Hash Tables to Dictionaries

The Dictionary object is, among other things, a replacement for the old Hash object, which was a simple hash table object. You are recommended to use the new Dictionary object instead of the old Hash object in your plug-ins, but if you have an existing plug-in in which old-style hashes are used, you can convert them to Dictionaries as follows:

Hash.ConvertToDictionary () Returns a new Dictionary object, populated with strings converted from the old-style Hash.

## DocumentSetup

Accessed from a Score object, DocumentSetup corresponds to the settings in Layout > Document Setup.
When you first access the DocumentSetup object, the units default to millimeters; if you want to use another unit of measurement, set DocumentSetup. Units before you set any of the other values. This will not, however, change the units displayed in Layout > Document Setup; to do that, set DocumentSetup. UnitsInDocumentSetupDialog.

Be careful also that if you set DocumentSetup. PageSize after setting DocumentSetup. PageWidth or
DocumentSetup. PageHeight, the page size specified will override any custom height/width you may have just set: so set the page size before you then adjust the width or height of the page.

## Methods

None.

## Variables

AboveTopStaveGap Returns or sets the top staff margin on each page in the units specified by the Units variable (read/write).
AboveTopStaveGapAfterFirstPage Returns or sets the top staff margin on pages after the first page in the units specified by the Units variable (read/write). To set this, first set FirstPageHasUniqueVerticalStaveMargins to True.

BelowBottomStaveGap Returns or sets the bottom staff margin on each page in the units specified by the Units variable (read/write). To set this, first set FirstPageHasUniqueVerticalStaveMargins to True.

BelowBottomStaveGapAfterFirstPage Returns or sets the bottom staff margin on each page after the first page in the units specified by the Units variable (read/write).

FirstPageHasUniqueVerticalStaveMargins Returns True if the After first page checkbox is enabled in Document Setup, specifying that the first page of the score has different top and bottom staff margins to subsequent pages; otherwise returns False (read/write).

Orientation Returns or sets the current page orientation. Values are OrientationPortrait (0) and OrientationLandscape (1). If you change the orientation, this will swap the PageTopMargin and PageBottomMargin values with the PageLeftMargin and PageRightMargin values, to reflect the change in orientation (read/write).

PageHeight Returns or sets the height of a page in the units specified by the Units variable (read $/ \mathrm{write}$ ).
PageSize Returns or sets the current page size. Values are listed in PageSize Values. If you attempt to set PageSize to PageSizeCustom, Sibelius will do nothing; to set a custom page size, set PageWidth and PageHeight individually. Setting any default PageSize value will also change the PageWidth and PageHeight values (read/write).

PageWidth Returns or sets the width of a page in the units specified by the Units variable (read/write).
MarginType Returns or sets the current page margin type. Values are PageMarginsSame (0), PageMarginsMirrored (1), PageMarginsDifferent (2) (read/write).

PageBottomMargin Returns or sets the bottom page margin in the units specified by the Units variable (read/write).
PageLeftMargin Returns or sets the left page margin in the units specified by the Units variable (read/write).
PageRightMargin Returns or sets the right page margin in the units specified by the Units variable (read/write).
PageTopMargin Returns or sets the top page margin in the units specified by the Units variable (read/write).
RightPageLeftMargin Returns or sets the left page margin for right-hand pages in the units specified by the Units variable (read/write). Setting this value automatically sets MarginType to PageMarginsDifferent.

RightPageRightMargin Returns or sets the right page margin for right-hand pages in the units specified by the Units variable (read/write). Setting this value automatically sets MarginType to PageMarginsDifferent.

StaffleftMarginFullNames Returns or sets the margin to the left of staves showing full instrument names in the units specified by the Units variable (read/write).

StaffLeftMarginNoNames Returns or sets the margin to the left of staves showing no instrument names in the units specified by the Units variable (read/write).

StaffLeftMarginShortNames Returns or sets the margin to the left of staves showing short instrument names in the units specified by the Units variable (read/write).

StaffSize Returns or sets the staff size in the units specified by the Units variable (read/write).

Units Returns or sets the units of measurement for all of the relevant variables of the DocumentSetup object. Always returns 0 (millimeters). Values are DocumentSetupUnitsmm (0), DocumentSetupUnitsInches (1), DocumentSetupUnitsPoints (2) (read/write).

UnitsInDocumentSetupDialog Returns or sets the units of measurement currently shown in the Layout ODocument Setup dialog. Values are as for Units.

## DynamicPartCollection

Accessed from a Score object. DynamicPartCollection contains DynamicPart objects.
The DynamicPartCollection object always contains the full score as the first entry, whether or not any dynamic parts exist. The DynamicPart objects are returned in the order in which they were created (the last part returned is the most-recently created one). For scores in which dynamic parts were generated automatically, the parts will normally be returned in top to bottom score order.

The edit context for ManuScript is stored in the score itself which means that ManuScript can only ever access one part at a time - the "current" DynamicPart for that Score object. This is irrespective of the number of score windows open for a score, which dynamic parts are open, and even if the user has managed to create two different ManuScript Score objects referring to the same Sibelius score.

It is inadvisable to modify Staves, Bars, or any BarObjects that do not exist on Staves in Score. CurrentDynamicPart. Doing so will create part overrides for part-specific properties of these objects which will be invisible until those Staves are added to the part. DynamicPart. IncludesStaff () can be used to test if a DynamicPart contains a particular Staff object.

Both DynamicPartCollection and DynamicPart refer to an underlying Score and part(s) and will generate errors if the Score and/or part(s) are no longer valid (for example, if a DynamicPart has been deleted). DynamicPart are never "re-used." For example, if you delete a DynamicPart and create a new DynamicPart, the old ManuScript DynamicPart object will not refer to the newly-created DynamicPart.
for each variable in iterates through all valid DynamicPart objects for the Score, always starting first with the full score. Adding or deleting parts while iterating will have undefined results, and is not recommended.

Array access [int $n$ ] returns the $n$th part ( 0 is always the full score), or null if the part does not exist.

## Methods

CreateDefaultParts () Creates the default set of dynamic parts, as created automatically by Sibelius when clicking the New Part button in the Parts window. This method does nothing and returns False if the Score has no staves.

CreatePartFromStaff (staff) Creates a dynamic part from the specified Staff object, if valid. Returns the new DynamicPart object for success, or null for failure.

DeletePart (dynamic part) Deletes the specified part, if it's valid. Returns True for success, False for failure. This method fails is the specified dynamic part is the currently active part for the Score, or is the full score, or refers to a different Score.

## Variables

NumChildren Returns the number of DynamicPart objects for the Score returned by iteration (read only).

## DynamicPart

Accessed from a DynamicPartCollection object.
for each variable in returns the Staff objects in the dynamic part, in top to bottom order.
A This can return a Staff that is not included in Score. CurrentDynamicPart.

## Methods

AddStaffToPart (staff) Adds the specified staff to the bottom of the dynamic part. Returns False for failure. This method will cause an error if it is called on the full score, or if attempting to add a staff that is already present in the part or a staff from a different score.

DeleteStaffFromPart (staff) Deletes the specified staff from the dynamic part. Returns False for failure. This method will cause an error if called on the full score, or if attempting to delete a staff that is not present in the part, or if deleting the last staff in a part, or attempting to delete a part from a different score.

IncludesStaff (staff) Returns True if the specified staff is contained in this dynamic part.

## Variables

IsFullScore Returns True if this is the full score (read only).
IsSelectedInPartsWindow Returns True if the part is selected in the Parts window (read only).
StaveCount Returns the number of staves in the part (read only).
ParentScore Returns the Score object containing this dynamic part (read only).

## EngravingRules

Accessed via the Score object. Corresponds to selected settings in the House Style $>$ Engraving Rules dialog.

## Methods

BarNumberFrequency (BarNumberFrequencyCategory, [customFrequency) Equivalent to setting the Bar Number Frequency setting in the Engraving Rules Dialog. Valid BarNumberFrequencyCategory values include: EverySystem (0), NoBarNumbers (1), EveryNthBar (2). The optional customFrequency argument is used in conjunction with the EveryNthBar (2) BarNumberFrequencyCategory, and sets how often (in bars) Bar Numbers will appear in the score.

## Variables

AdjustTranspositionIfKeySigWraps Returns True if Sibelius will adjust note spelling for transposing instruments in extreme keys, False otherwise; corresponds to the Adjust note spelling in transposing instruments in remote keys option on the Clefs and Key Signatures page (read/write).

BarlineJoinSystemEnd Returns True if Sibelius will draw complete barlines at the end of a system regardless of the barline breaks within a system; False otherwise, corresponding to the option Join barlines at end of systems on the Barlines page (read/write).

BarlineSingleStaves Returns True if Sibelius will draw barlines on a single staff; False otherwise, corresponding to the option Barline at start of single staves on the Barlines page (read/write).

BeamThickness Returns or sets the thickness of beams in spaces, from the Beams and Stems page (read/write).
BarlineWidth Returns or sets the width of normal barlines in spaces, from the Barlines page (read/write).
CautionaryNaturalsInKeySignatures Returns True if key changes will show cautionary naturals; False otherwise, from the Clefs and Key Signatures page (read/write).

CueNoteScale Returns or sets the percentage by which cue-sized notes are scaled relative to normal-sized notes, from the Notes and Tremolos page (read/write).

DashedBarlineGap Returns or sets the gap between dashes of dashed barlines in spaces, from the Barlines page (read/write).
DashedBarlineLength Returns or sets the length of the dashes of dashed barlines in spaces, from the Barlines page (read/write).
DashedBarlineWidth Returns or sets the width of dashed barlines in spaces, from the Barlines page (read/write).
DottedBarlineGap Returns or sets the gap between dots of dotted barlines in spaces, from the Barlines page (read/write).
DottedBarlineWidth Returns or sets the width of dotted barlines in spaces, from the Barlines page (read/write).

DoubleBarlineSeparation Returns or sets the distance between the lines in double or triple barlines in spaces, from the Barlines page (read/write).

DoubleBarlineWidth Returns or sets the width of the lines in double or triple barlines in spaces, from the Barlines page (read/write).

DoubleTremoloStyle Returns or sets the style used for double tremolos in the score, from the Notes and Tremolos page; values are DoubleTremolosTouchingStems (0), DoubleTremolosBetweenStems (1), DoubleTremolosOuterTremoloTouchingStems (2) (read/write).

ExtraSpacesAboveForSystemObjectPositions Returns or sets the $n$ extra spaces above for System Object Positions value on the Staves page (read/write).

ExtraSpacesBelowVocalStaves Returns or sets the $n$ extra spaces below vocal staves (for lyrics) value on the Staves page (read/write).

ExtraSpaceBetweenGroupsOfStaves Returns or sets the $n$ extra spaces between groups of staves value on the Staves page (read/write).

FinalBarlineSeparation Returns or sets the distance between the two lines in final or repeat barlines in spaces, from the Barlines page (read/write).

FinalBarlineThinWidth Returns or sets the width of the thin line in final or repeat barlines in spaces, from the Barlines page (read/write).

FinalBarlineWidth Returns or sets the width of the thick line in final or repeat barlines in spaces, from the Barlines page (read/write).

GraceNoteScale Returns or sets the percentage by which grace notes are scaled relative to normal notes, from the Notes and Tremolos page (read/write).

InstrumentNamesFirstSystem Corresponding to the option for instrument names on the first system on the Instruments page; values are InstrumentNamesFull (0), InstrumentNamesShort (1), InstrumentNamesNone (2) (read/write).

InstrumentNamesNewSections Corresponding to the option for instrument names at the start of new sections on the Instruments page; values are InstrumentNamesFull (0), InstrumentNamesShort (1), InstrumentNamesNone (2) (read/write).

InstrumentNamesSubsequentSystems Corresponding to the option for instrument names on subsequent systems on the Instruments page; values are InstrumentNamesFull (0), InstrumentNamesShort (1), InstrumentNamesNone (2) (read/write).

JustifyGrandStaveInstruments Returns True if Justify both staves of grand staff instruments on the Staves page is enabled, otherwise False (read/write).

JustifyMultiStaveInstruments Returns True if Justify all staves of multi-staff instruments on the Staves page is enabled, otherwise False (read/write).

LegerLineThickness Returns or sets the thickness of leger lines in spaces, from the Notes and Tremolos page (read/write).
RepeatBarlineAllDots Returns True if Sibelius will draw dots in repeat barlines on in every staff space; False otherwise, corresponding to the option Repeats with dots in all staff spaces on the Barlines page (read/write).

RepeatBarlineDoubleThick Returns True if Sibelius will draw coincident repeat barlines as two thick lines rather than a three thin, thick, thin lines; False otherwise, corresponding to the option Double thick lines for coincident repeats on the Barlines page (read/write).

RepeatBarlineWings Returns True if Sibelius will draw wings on repeat barlines; False otherwise, corresponding to the option Wings on repeat barlines on the Barlines page (read/write).

RespellRemoteKeysInTransposingScore Returns True if Sibelius will choose the equivalent key signature with one fewer flat or sharp for transposing instruments; False otherwise, corresponding to the option Respell remote key signatures in transposing score on the Clefs and Key Signatures page (read/write).

ShowNameOfPrevailingInstrumentChangeAtStartOfSystems Returns True if Sibelius will update the instrument name at the start of each system to reflect the current instrument change, Fal se otherwise; corresponds to the Change instrument names at start of system after instrument changes option on the Instruments page (read/write).

SlurMiddleThickness Returns or sets the default thickness of the middle of slurs in spaces, from the Slurs page (read/write).
SlurOutlineWidth Returns or sets the thickness of slur ends in spaces, from the Slurs page (read/write).
SmallStaffSizeScale Returns or sets the percentage by which small staves are scaled relative to normal-sized staves, from the Staves page (read/write).

SpacesBetweenStaves Returns or sets the $n$ spaces between staves value on the Staves page (read/write).
SpacesBetweenSystems Returns or sets the $n$ spaces between systems value on the Staves page (read/write).
StaffJustificationPercentage Returns or sets the Justify staves when page is at least $n \%$ full value on the Staves page (read/write).

StaffLineWidth Returns or sets the width of a staff line in spaces, from the Staves page (read/write).
StemThickness Returns or sets the thickness of stems in spaces, from the Beams and Stems page (read/write).
ThickBarlineWidth Returns or sets the width of thick barlines in spaces, from the Barlines page (read/write).
TieMiddleThickness Returns or sets the thickness of the middle of ties in spaces, from the Ties 1 page (read/write).
TieOutlineWidth Returns or sets the thickness of tie ends in spaces, from the Ties 1 page (read/write).

## File

Retrievable using for each on a folder.

## Methods

Delete () Deletes a file, returning True if successful.
Rename (newFileName) Renames a file, returning True if successful.

## Variables

CreationDate Returns the file's creation date and time as a DateTime object, in local time (read only).
CreationDateAndTime A string giving the date and time the file was last modified in GMT (read only).
ModificationDate Returns the file's modification date and time as a DateTime object, in local time (read only).
Name The complete pathname of the file, no extension (read only).
NameWithExt The complete pathname of the file, with extension (read only).
NameNoPath Just the name of the file, no extension (read only).
Path Returns just the path to the file (read only).
Type A string giving the name of the type of the object; File for file objects (read only).

## Folder

Retrievable from methods of the Sibelius object.
for each variable in produces the Sibelius files in the folder, as File objects
for each type variable in produces the files of type in the folder, where type is a Windows extension.
Useful values are SIB (Sibelius files), MID (MIDI files) or OPT (PhotoScore files), because they can all be opened directly by Sibelius. On the Macintosh files of the corresponding Mac OS Type are also returned (so, for example, for each MID $\mathbf{f}$ will return all files whose names end in .MID, and all files of type "Midi").

Both these statements return subfolders recursively.

## Methods

FileCount (Type) Returns the number of files of type Type in the folder. As above, useful values are SIB, MID, or OPT.

## Variables

FileCount The number of Sibelius files in the folder (read only).
FileCountAlltypes The number of files of all types in the folder (read only).
Name The name of the folder (read only).
Type A string giving the name of the type of the object; Folder for folder objects (read only)

## GuitarFrame

Derived from a BarObject. This refers to chord symbols as created by Create > Chord Symbol, whether or not they show a guitar chord diagram (guitar frame), but is called GuitarFrame in ManuScript for historical reasons.

## Methods

CopyOutSuffixes () Returns an array containing a list of the suffix elements present in the chord. If the chord symbol is an unrecognised chord type, the array returned will be empty. The values that can be returned in the array are as follows:

| halfdim | dim |
| :--- | :--- |
| add6/9 | $6 / 9$ |
| sus2/4 | aug |
| omit5 | alt |
| omit3 | b13 |
| maj13 | $\# 11$ |
| add13 | 13 |
| maj11 | 11 |
| dim13 | $\# 9$ |
| dim11 | b9 |
| maj9 | b6 |
| add9 | $\# 5$ |
| maj7 | b5 |
| dim9 | $\# 4$ |
| dim7 | nc |
| sus9 | 9 |
| sus4 | 7 |
| add4 | 6 |
| sus2 | 5 |
| add2 | m |
| maj | $/$ |

GetChromaticPitchesOfChordInClosePosition (consider root) Returns an array containing the chromatic pitches of the notes in the chord, assuming a voicing in close position. If consider root is True (it defaults to False), the pitches returned will be offset according to the chromatic value of the root note on which the chord is based.

GetEndStringForNthBarre (barreNum) Returns the string number on which the nth barré ends.
GetPitchOfNthString (stringNum) ] Returns the pitch of the given (open) string stringNum, as a MIDI pitch.
GetPositionOfFingerForNthBarre (barreNum) Returns the fret position that the nth barre occupies.
GetPositionOfFingerOnNthString (stringNum) Returns the position of the black dot representing the finger position on a given string stringNum, relative to the top of the frame. A return value of 0 means the string is open (that is a hollow circle appears at the top of the diagram), and -1 means that the string is not played (that is an $X$ appears at the top of the diagram). Used in conjunction with GetPitchOfNthString (), you can calculate the resulting pitch of each string.

GetStartStringForNthBarre (barreNum) Returns the string number from which the nth barré begins.

IsNthStringPartOfBarre (stringNum) Returns True if the given string is part of a barré.
NthStringHasClosedMarkingAtNut (nth string) Returns True if there's an X marking at the top or left of the specified string.
NthStringHasOpenMarkingAtNut (nth string) Returns True if there's an O marking at the top or left of the specified string.

## Variables

BassAsString The note name of the chord symbol's altered bass note (for example: "F").
ChordNameAsStyledString The name of the chord represented by this chord symbol as it appears in the score, for example: "Cm7" (read only).

ChordNameAsPlainText The name of the chord represented by this chord symbol as it appears when editing the chord symbol, so that in its plain text representation, for example: "Cmmaj7" (read only).

ChromaticRoot The chromatic pitch $(\mathrm{C}=0, \mathrm{~B}=11)$ of the chord symbol's root note (read only).
ChromaticBass The chromatic pitch $(\mathrm{C}=0, \mathrm{~B}=11)$ of the chord symbol's altered bass note (read only).
DiatonicRoot The diatonic pitch, that is the number of the "note name" to which this note corresponds, 7 per octave $(0=\mathrm{C}, 1=\mathrm{D}$, $2=\mathrm{E}$ and so on), of the chord symbol's root note (read only).

DiatonicBass The diatonic pitch, that is the number of the "note name" to which this note corresponds, 7 per octave $(0=\mathrm{C}, 1=\mathrm{D}$, $2=\mathrm{E}$ and so on), of the chord symbol's altered bass note (read only).

Fingerings The fingerings string for this chord. This is a textual string with as many characters as the guitar frame has strings (for example, six for standard guitars). Each character corresponds to a guitar string. Use to denote that a string has no fingering.

FrameIsVisible True if the chord symbol is currently showing a guitar chord diagram (read only).
Horizontal True if the guitar chord diagram is horizontally orientated, False if it is vertically orientated (read/write).
LowestVisibleFret The number of the top fret shown in the guitar chord diagram; setting the special value $\mathbf{- 1}$ resets the lowest visible fret to the default for that chord diagram (read/write).

NumBarresInChord The number of unique barrés in the guitar chord diagram (read only).
NumberOfFrets The number of frets in the guitar chord diagram, that is the number of horizontal lines; setting the special value -1 resets the number of frets to the default for that chord diagram (read/write).

NumberOfStrings The number of strings in the guitar chord diagram, for example,. the number of vertical lines (read only).
NumPitchesInClosePosition The number of unique pitches in the chord, assuming a voicing in close position with no duplicates.

Recognized Returns True if the chord symbol is a specific recognized chord type, and Fal se otherwise, that is if the chord symbol is shown in red in the score because Sibelius is unable to parse the user's input (read only).

RootAsString The note name of the chord symbol's root (for example, "C\#").
ScaleFactor The scale factor of the guitar chord diagram (as adjustable via the Scale parameter on the General panel of Properties), expressed as a percentage (read/write).

ShowFingerings Set to True if the fingerings string should be displayed, False otherwise (read only).
SuffixText The suffix part of the chord symbol as it appears in the score, or an empty string if the chord isn't recognized (read only).
TextIsVisible True if the chord symbol is currently showing a text chord symbol (read only).
TransposingChromaticRoot Returns the chromatic pitch of the root note for the specified chord symbol as if the score is shown at transposed pitch, but regardless of whether or not Notes > Transposing Score is enabled.

TransposingChromaticBass Returns the chromatic pitch of the altered bass note for the specified chord symbol, if present, as if the score is shown at transposed pitch, but regardless of whether or not Notes > Transposing Score is enabled.

TransposingDiatonicRoot Returns the diatonic pitch of the root note for the specified chord symbol as if the score is shown at transposed pitch, but regardless of whether or not Notes > Transposing Score is enabled.

TransposingDiatonicBass Returns the diatonic pitch of the altered bass note for the specified chord symbol, if present, as if the score is shown at transposed pitch, but regardless of whether or not Notes > Transposing Score is enabled.

TransposingRootAsString Returns a string representing the pitch of the root note for the specified chord symbol as if the score is shown at transposed pitch, but regardless of whether or not Notes > Transposing Score is enabled.

TransposingBassAsString Returns a string representing the pitch of the altered bass note for the specified chord symbol, if present, as if the score is shown at transposed pitch, but regardless of whether or not Notes > Transposing Score is enabled.

VisibleComponents The visible parts of the chord symbol, that is whether it displays a text chord symbol only (TextOnly), a guitar chord diagram only (FrameOnly), both a text chord symbol and a guitar chord diagram (FrameAndText), or whether or not the chord symbol shows a guitar chord diagram based on the type of instrument to which it is attached
(InstrumentDependent) (read/write).

## GuitarScaleDiagram

Derived from a BarObject. This refers to guitar scale diagrams as created by Create > Guitar Scale Diagram.

## Methods

GetDotFingeringsOnNthString (nth string) Returns an array of strings containing the text that has been entered on the dots on a given string.

GetDotSymbolsOnNthString ( $n$th string) Returns an array of values describing the appearance of each of the dots on a given string. The possible values are DotStyleCircle, DotStyleFilledCircle, DotStyleSquare, DotStyleFilledSquare, DotStyleDiamond, and DotStyleFilledDiamond.

GetPitchesOfDotsOnNthString ( $n$th string) Returns an array containing the pitches of all the dots on a given string, in ascending order of pitch.

GetPitchOfNthString (stringNum) Returns the pitch of the given (open) string stringNum, as a MIDI pitch.

## Variables

Fingerings The fingerings string for this scale diagram. This is a textual string with as many characters as the scale diagram has strings (for example, six for standard guitars). Each character corresponds to a guitar string. Use - to denote that a string has no fingering.

Horizontal True if the guitar scale diagram is horizontally orientated, False if it is vertically orientated (read/write).
LowestVisibleFret The number of the top fret shown in the guitar scale diagram; setting the special value $\mathbf{- 1}$ resets the lowest visible fret to the default for that scale diagram (read/write).

NumberOfFrets The number of frets in the guitar scale diagram, for example, the number of horizontal lines; setting the special value -1 resets the number of frets to the default for that scale diagram (read/write).

NumberOfStrings The number of strings in the guitar scale diagram, for example, the number of vertical lines (read only).
Root Returns the chromatic pitch $(\mathrm{C}=0)$ of the scale's root note (read only).
ScaleFactor The scale factor of the guitar scale diagram (as adjustable via the Scale parameter on the General panel of Properties), expressed as a percentage (read/write).

Scaleтype Returns the type of the guitar scale diagram, as specified in the list of GuitarScaleDiagram Type Values (read only).
ShowFingerings Set to True if the fingerings string should be displayed, False otherwise (read only).

## HitPointList

Retrievable as the HitPoints variable of a score. It can be used in a for each loop or as an array with the [n] operator-this gives access to a HitPoint object. The HitPoint objects are stored in time order, so be careful if you remove or modify the time of the objects inside a loop. If you want to change the times of all the hit points by the same value then use the Shiftrimes function.

## Methods

Clear () Removes all hit points from the score.
CreateHitPoint (timeMs, label) Creates a hit point in the score at the given time (specified in milliseconds) with a specified string label. Returns the index in the HitPointList at which the new hit point was created.

Remove (index) Removes the given hit point number.
ShiftTimes (timeMs) Adds the given time (in milliseconds) onto all the hit points. If the time is negative then this is subtracted from all the hit points.

## Variables

NumChildren Number of hit points (read only).

## HitPoint

Individual element of the HitPointList object.

## Methods

None.

## Variables

Bar The bar in which this hit point occurs (read only).
Label The name of the hit point (read/write).
Position The position within the bar at which this hit point occurs (read only).
Time The time of the hit point in milliseconds. Note that changing this value may change the position of the hit point in the HitPointList (read/write).

## InstrumentChange

Derived from a BarObject. Provides information about any instrument changes that may exist in the score.

## Methods

None.

## Variables

FullInstrumentName Returns the full instrument name associated with this instrument change (read/write).
FullinstrumentNameWithFormatting Returns the full instrument name associated with this instrument change including any changes of font or style, if any (read/write).

FullStaffName Returns the full staff name associated with this instrument change (read/write).
FullStaffNameWithFormatting Returns the full staff name associated with this instrument change including any changes of font or style, if any (read/write).

ShortInstrumentName Returns the short instrument name associated with this instrument change (read/write).
ShortInstrumentNameWithFormatting Returns the short instrument name associated with this instrument change including any changes of font or style, if any (read/write).

ShortStaffName Returns the short staff name associated with this instrument change (read/write).
ShortStaffNameWithFormatting Returns the short staff name associated with this instrument change including any changes of font or style, if any (read/write).

StyleIdword Returns the style ID of the new instrument; see Instrument Types (read only).
TextLabel Returns the text that appears above the staff containing the instrument change in the score (read only).

## InstrumentTypeList

Contains a list of InstrumentType objects common to a given score.
for each type variable in returns each instrument type in the list, in alphabetical order by the instrument type's style ID.
Array access [int $n$ ] returns the $n$th instrument type, in the same order as using a for each iterator, or null if the instrument type does not exist.

## Methods

None.

## Variables

NumChildren Returns the number of unique instrument types in the list (read only).

## InstrumentType

Provides information about an individual instrument type.

## Methods

Clone() Makes an exact copy of an existing instrument type.
PitchOfNthString (string num) Returns the pitch of a given string in a tablature staff, with string number 0 being the lowest string on the instrument.

## Variables

Balance Returns the instrument's default balance, in the range 0-100 (read only).
Category Returns an index representing the category of the staff type belonging to this instrument type; $\mathbf{0}=$ pitched; $1=$ percussion; 2 = tablature (read only).

ChromaticTransposition Returns the number of half-steps (semitones) describing the transposition of transposing instruments; such as for B-flat Clarinet, this returns -2 (read/write).

ChromaticTranspositionInScore Returns the number of half-steps (semitones) describing the transposition of transposing instruments in a score shown at concert pitch. Typically this is only used by instruments that transpose by octaves, so this will return, for example, 12 for piccolo or $\mathbf{- 1 2}$ for guitars (read only).

ComfortableRangeHigh Returns the highest comfortable note (MIDI pitch) of the instrument (read only).
ComfortableRangeLow Returns the lowest comfortable note (MIDI pitch) of the instrument (read only).
ConcertClefStyleId Returns the style ID of the normal clef style of the instrument (read only).
DefaultSoundId Returns the default sound ID used by the instrument (read only).

Diatonic'Transposition Returns the number of diatonic steps describing the transposition of transposing instruments; such as for B-flat Clarinet, this returns -1 (read/write).

DiatonicTranspositionInScore Returns the number of diatonic steps describing the transposition of transposing instruments in a score shown at concert pitch (read only).

DialogName Returns the name of the instrument as displayed in the Create > Instruments dialog in Sibelius (read/write).
FullName Returns the name of the instrument as visible on systems showing full instrument names (read only).
HasBracket Returns True if the instrument has a bracket (read only).
HasKeySignatureOrTuning Returns True if the instrument type has the Key signature / Tuning checkbox switched on in the Edit Staff Type dialog.

InstrumentTypeForChordDiagrams Returns the style ID of the tab instrument type that determines the tuning used for chord diagrams shown for this instrument, that is corresponding to the Tab instrument to use for string tunings in the New/Edit Instrument dialogs.

IsVocal Returns True if the instrument type used has the Vocal staff option switched on, meaning that, for example, the default positions of dynamics should be above the staff rather than below (read only).

NumStaveLines Returns the number of staff lines in the staff (read only).
NumStrings Returns the number of strings in a tablature staff (read only).
OtherClefStyleId Returns the style ID of the clef style of the second staff of grand staff instruments, piano for example (read only).

Pan Returns the instrument's default pan setting, in the range - 127 (hard left) to 127 (hard right) (read only).
ProfessionalRangeHigh Returns the highest playable note (MIDI pitch) of the instrument for a professional player (read only).
ProfessionalRangeLow Returns the lowest playable note (MIDI pitch) of the instrument for a professional player (read only).
ShortName Returns the name of the instrument as visible on systems showing short instrument names (read only).
StyleId Returns the style ID of the instrument; see Global Constants (read only).
TransposingClefStyleId Returns the style ID of the clef to be used when Notes > Transposing Score is enabled (read only).

## KeySignature

Derived from a BarObject.

## Methods

None.

## Variables

AsText The name of the key signature as a string (read only).
IsOneStaffonly True if this key signature belongs to one staff only (read only).
Major True if this key signature is a major key (read only).
Sharps The number of sharps (positive) or flats (negative) in this key signature (read only).

## Line

Anything you can create from the Create > Line dialog is a Line object, such as CrescendoLine, DiminuendoLine, and so on. These objects are derived from a BarObject.

## Methods

None.

## Variables

Duration The total duration of the line, in 1/256th quarters (read/write).
EndBarNumber The bar number in which the line ends (read only).
EndPosition The position within the final bar at which the line ends (read only).
$\operatorname{RhDx}$ The horizontal graphic offset of the right-hand side of the line, in units of $1 / 32$ spaces (read/write).
RhDy The vertical graphic offset of the right-hand side of the line from the center staff line, in units of $1 / 32$ spaces, positive going upwards (read/write).

StyleId The identifier of the line style associated with this line (read only).
StyleAsText The name of the line style associated with this line (read only).

## Lyricltem

Derived from a BarObject

## Methods

None.

## Variables

Duration The total duration of the lyric line, in $1 / 256$ th quarters (see Line) (read/write).
NumNotes Gives the number of notes occupied by this lyric item (read/write). Note that changing this value will not automatically change the length of the lyric line; you also need to set the lyric line's Duration variable to the correct length.

StyleAsText The text style name (read/write).
StyleId he identifier of the text style of this lyric (read/write).
SyllableType An integer indicating whether the lyric is the end of a word (EndOfWord) or the start or middle of one (MiddleOfWord) (read/write). This affects how the lyric is jusitifed, and the appearance of hyphens that follow it. EndOfWord and MiddleOfWord are global constants; see SyllableTypes for LyricItems.

Text The text as a string (read/write).

## NoteRest

Derived from a BarObject. A NoteRest contains Note objects, stored in order of increasing diatonic pitch.
for each variable in returns the notes in the NoteRest.

## Methods

AddAcciaccaturaBefore (sounding pitch, [duration[,tied [, voice [, diatonic pitch[, string number[, force stem dir]I]I]I) Adds a grace note with a slash on its stem (acciaccatura) before a given NoteRest. The duration should be specified as normal, for example, 128 would create a grace note with one beam/flag. The optional tied parameter should be True if you want the note to be tied. Voice 1 is assumed unless the optional voice parameter (with a value of $1,2,3$ or 4 ) is specified. If force stem dir is set to True (the default), stems of graces notes in voices 1 and 3 will always point upwards, and stems of notes in voices 2 and 4 , downwards. You can also set the diatonic pitch, that is the number of the "note name" to which this note corresponds, 7 per octave ( $35=$ mid$\mathrm{dle} \mathrm{C}, 36=\mathrm{D}, 37=\mathrm{E}$ and so on). If a diatonic pitch of zero is given then a suitable diatonic pitch will be calculated from the MIDI pitch. The optional string number parameter gives a string number for this note, which is only meaningful if the note is on a tablature stave. If this parameter is not supplied then a default string number is calculated based on the current tablature stave type and the guitar tab fingering options (specified on the Note Input page of File > Preferences). Returns the Note object created (to get the NoteRest containing the note, use Note. ParentNoteRest).

Note that adding a grace note before a NoteRest will always create an additional grace note, just to the left of the note/rest to which it is attached. If you wish to create grace notes with more than one pitch, you should call AddNote on the object returned.

AddAppoggiaturaBefore (sounding pitch, [duration [, tied [, voice [, diatonic pitch[, string number[, force stem dir]]]]]])
Identical to AddAcciaccaturaBefore, only no slash is added to the note's stem.
AddNote (pitch[ , tied[ , diatonic pitch[, string number]]]) Adds a note with the given MIDI pitch ( $60=$ middle C), for example to create a chord. The optional second parameter specifies whether or not this note is tied (True or False). The optional third parameter gives a diatonic pitch, which is the number of the 'note name' to which this note corresponds, 7 per octave $(35=$ middle $\mathrm{C}, 36=\mathrm{D}, 37=\mathrm{E}$ etc.). If this parameter is 0 then a default diatonic pitch will be calculated from the MIDI pitch. The optional fourth parameter gives a string number for this note, which is only meaningful if the note is on a tablature stave. If this parameter is not supplied then a default string number is calculated based on the current tablature stave type and the guitar tab fingering options (specified on the Notes page of File > Preferences). Returns the Note object created.

Delete () Deletes all the notes in the NoteRest, converting the entire chord into a rest of similar duration.
FlipStem () Flips the stem of this NoteRest-this acts as a toggle.
GetArticulation (articulation number) Returns True or False depending on whether the given articulation is currently set on this note. The valid articulation numbers are defined in Articulations.

NoteRest [array element] Returns the nth note in the chord, in order of increasing diatonic pitch (counting from 0 ). For example, NoteRest[0] returns the lowest note (in terms of diatonic pitch-see AddNote below).

RemoveNote (note) Removes the specified Note object.
SetArticulation (articulation number, set) If set is True, turns on the given articulation; otherwise turns it off. The valid articulation numbers are defined in Articulations.

Transpose (degree, interval type[,keep double accs]) Transposes the entire NoteRest up or down by a specified degree and interval type. To transpose up, use positive values for degree; to transpose down, use negative values. Note that degrees are 0 -based, so 0 is equal to a unison, 1 to a second and so on. For descriptions of the various available interval types, see Global Constants. By default, Sibelius will transpose using double sharps and flats where necessary, but this behavior may be suppressed by setting the keep double accs flag to False.

For help in calculating the interval and degree required for a particular transposition, see the documentation for the Sibelius.CalculateInterval and Sibelius.CalculateDegree methods.

## Variables

ArpeggioDx The horizontal offset of the arpeggio line on the NoteRest (read/write), in units of $1 / 32$ nd of a space (the distance between two adjacent staff lines).

ArpeggioType The type of note-attached arpeggio line present on the NoteRest. Values are ArpeggioTypeNone, ArpeggioTypeNormal, ArpeggioTypeUp, ArpeggioTypeDown (read/write).

ArpeggioTopDy The vertical offset of the top of the note-attached arpeggio line on the NoteRest (read/write), in units of $1 / 32 \mathrm{nd}$ of a space.

ArpeggioBottomDy The vertical offset of the bottom of the note-attached arpeggio line on the NoteRest (read/write), in units of $1 / 32$ nd of a space.

ArpeggioHidden Returns True if the note-attached arpeggio line on the NoteRest is hidden (read/write).
Articulations Lets you copy a set of articulations from one NoteRest to another (read/write), for example: destNr.Articulations $=$ sourceNr.Articulations;

Beam Takes values StartBeam, ContinueBeam, NoBeam and SingleBeam. (see Global Constants for details). These correspond to the keys 7, 8 , * (/ on Mac) and / (* on Mac) on the third (F9) Keypad layout.

CrossStaff Returns the following values pertaining to cross-staff beaming: 0 (Not Crossed), 1 (Crossed Above), 2 (Crossed Below). If cross-staff beaming has been applied to a note, but there is no staff available above or below the original staff, CrossStaff returns -1 (Crossed Above, No Upper Staff) or -2 (Crossed Below, No Lower Staff).

DoubleTremolos Gives the number of double tremolo strokes starting at this note, in the range $0-7$. Means nothing for rests. To create a double tremolo between two successive notes, ensure they have the same duration and set the DoubleTremolos of the first one (read/write).

Duration The duration of the note rest (read only).
Falldx The horizontal offset of a fall, if present on the NoteRest (read/write), in units of $1 / 32$ nd of a space.
FallType The type of note-attached fall present on the NoteRest. Values are FallTypeNone, FallTypeNormal and FallTypeDoit (read/write)

FeatheredBeamType Returns one of three values, based on whether a note is set to produce a feathered beam. Values are FeatheredBeamNone (0), FeatheredBeamAccel (1) and FeatheredBeamRit (2) (read/write).

GraceNote True if it's a grace note (read only).
HasStemlet Returns True if the note is showing a stemlet, according either to the state of the Use stemlets on beamed rests option on the Beams and Stems page of Engraving Rules or the stemlet button on the Keypad (read only).

Highest The highest Note object in the chord (read only).
IsAcciaccatura True if it's an acciaccatura, that is. a grace note with a slash through its stem (read only).
IsAppoggiatura True if it's an appoggiatura, that is a grace note without a slash through its stem (read only).
Lowest The lowest Note object in the chord (read only).
NoteCount The number of notes in the chord (read only).
ParentTupletIfAny If the NoteRest intersects a tuplet, the innermost Tuplet object at that point in the score is returned. Otherwise, null is returned (read only).

PositionInTuplet Returns the position of the NoteRest relative to the duration and scale-factor of its parent tuplet. If the NoteRest does not intersect a tuplet, its position within the parent Bar is returned as usual (read only).

RestPosition The vertical position of a rest (read/write).
ScoopDx The horizontal offset of a scoop or plop, if present on the NoteRest (read/write), in units of $1 / 32 \mathrm{nd}$ of a space.
ScoopType The type of note-attached scoop present on the NoteRest. Values are ScoopTypeNone, ScoopTypeNormal, ScoopTypePlop (read/write).

StemFlipped True if the stem is flipped (read only).
StemletType Provides information about whether the NoteRest is set to display a stemlet using the options on the Keypad. Returns either StemletCustomOff (in which case the NoteRest definitely does not show a stemlet), StemletCustomOn (in which case the NoteRest definitely does show a stemlet), or StemletUseDefault (in which case you should use the read-only variable HasStemlet to determine whether the NoteRest currently shows a stemlet) (read/write).

Stemweight Returns the stem weight of a note, taking beams into account (read only). For an unbeamed note, this is the sum of the stave positions of all the notes in the NoteRest, where the stave position of the middle line is zero and the position increases as you move up the stave and decreases as you move downwards. For a beamed note, it is the sum of all the stem weights of the NoteRests under the beam (treated as though they were unbeamed).

There are some special cases. If a note has its stem direction forced due to voicing, then the stem weight will be one of the global constants StemweightUp or StemweightDown. If a note has its stem direction forced due to the "flip" flag being set, the stem weight will be either StemweightFlipUp or StemweightFlipDown.

If the stem weight is less than zero, the stem will point up, otherwise it will point down.
SingleTremolos Gives the number of tremolo strokes on the stem of this note, in the range -1 (for " $z$ on stem") to 7 . Means nothing for rests (read/write).

## Note

Only found in NoteRests. Correspond to individual noteheads.

## Methods

Delete () Removes a single note from a chord.
Transpose (degree, interval type[,keep double accs]) Transposes and returns a single Note object up or down by a specified degree and interval type*. To transpose up, use positive values for degree; to transpose down, use negative values. Note that degrees are 0 -based, so 0 is equal to a unison, 1 to a second and so on. For descriptions of the various available interval types, see Global Constants. By default, Sibelius will transpose using double sharps and flats where necessary, but this behavior may be suppressed by setting the keep double accs flag to False. For help in calculating the interval and degree required for a particular transposition, see the documentation for the Sibelius.CalculateInterval and Sibelius.CalculateDegree methods.

Individual note objects cannot be transposed diatonically.

## Variables

Accidental The accidental, for which global constants such as Sharp, Flat and so on are defined; see Global Constants (read only).

AccidentalStyle The style of the accidental (read/write). This can be any of following four global constants: NormalAcc, HiddenAcc, CautionaryAcc (which forces an accidental to appear always) and BracketedAcc (which forces the accidental to be drawn inside brackets).

Bracketed The bracketed state of the note, as shown on the F9 layout of the Keypad (read/write).
Color The color of this Note (read/write). The color value is in 24 -bit RGB format, with bits $0-7$ representing blue, bits $8-15$ green, bits 16-23 red and bits 24-31 ignored. Since ManuScript has no bitwise arithmetic, these values can be a little hard to manipulate; you may find the individual accessors for the red, green and blue components to be more useful (see below).
> " ${ }^{\prime \prime}$ When all Notes in a given NoteRest are the same color, then that color is also promoted to the parent NoteRest itself. This allows backwards compatibility with versions of Sibelius prior to 8.3 that did not support the individual coloring of Notes. Coloring of NoteRest-attached objects, such as articulations and rhythm dots is not supported.

ColorAlpha The alpha channel component of the color of this Note, in the range $0-255$ (read/write).
ColorRed The red component of the color of this Note, in the range $0-255$ (read/write).
ColorGreen The green component of the color of this Note, in the range $0-255$ (read/write).

ColorBlue The blue component of the color of this Note, in the range $0-255$ (read/write).
DiatonicPitch The diatonic pitch of the note, that is the number of the "note name" to which this note corresponds, 7 per octave ( $35=$ middle C, $36=\mathrm{D}, 37=\mathrm{E}$ and so on). (read/write)
*' If Note.DiatonicPitch is changed from the full score (not a dynamic part), the written pitch and spelling of any accidental is changed in both the full score and the part (where there is no difference in spelling). If changed from a part, Sibelius respells any accidental in the part only, leaving the full score unchanged. In both cases, while there may be a difference in written pitch, Sibelius guarantees that there is never a difference in the sounding pitch of a note between a part and the full score.

IsAccidentalVisible Returns True if the accidental on the note is visible, which is the equivalent of whether or not the corresponding button on the Keypad is illuminated for that note (read only).

LvTie Is True if the note has an L.V. tie (read/write).
Name The pitch of the note as a string (read only).
NoteStyle The index of the notehead style of this Note (read/write). The styles correspond to those accessible from the Notes panel of the Properties window in Sibelius; see Note Style Names for a complete list of the defined NoteStyles.

NoteStyleName The name of the notehead style of this Note (read/write). If an attempt is made to apply a non-existent style name, the note in question will retain its current notehead.

OriginalDeltaSr The Live start position of this notehead (in 1/256th quarters), as shown in the Playback panel of Properties (read/write). This value can be positive or negative, indicating that the note is moved forwards or backwards.

OriginalDuration The Live duration of this notehead (in 1/256th quarters), as shown in the Playback panel of Properties (read/write).

OriginalVelocity The Live velocity of this notehead (in MIDI volume units, 0-127), as shown in the Playback panel of Properties (read/write). Note that the word "original" refers to the fact that this data is preserved from the original performance if the score was imported from a MIDI file or input via Flexi-time. For further details on this value, and the ones following below, read the Live Playback section in Sibelius Reference.

ParentNoteRest The NoteRest object that holds this note (read only).
Pitch The MIDI pitch of the note, in semitones, $60=$ middle C (read only).
Slide Is True if the note has a slide, False otherwise (read/write).
SlideStyleId The slide line style state of the note, allowing you to attach/detach glissandi and other lines to a note (read/write).
The following Line styles are available by default (as seen in the Inspector):

```
line.staff.gliss.straight
line.staff.gliss.wavy
line.staff.plain
line.staff.port.straight
```

You can define and assign additional custom Line styles not based on the available default Line styles. For example:

```
// Add/set a note slide style
note.SlideStyleId = "line.staff.gliss.straight";
// Log a note slide style to the plug-in trace window
Trace (note.SlideStyleId);
// Using a custom line style
note.SlideStyleId = "line.staff.gliss.straight.user.0000001";
```

StringNum The string number of this note, only defined if the note is on a tablature stave. If no string is specified, reading this value will give -1 . Strings are numbered starting at 0 for the bottom string and increasing upwards (read only).

Tied Is True if the note is tied to the following note (read/write).
TiedInto Is True if the note has a tie-into object attached to the left-hand side of the note (read/write).
TieIntoStyle The name of the tie-into object's style for this Note (read/write). Valid tie-into style names include: Solid, Dashed, Dotted. If an attempt is made to apply a non-existent style name, the tie-into object in question retains its current style.

TieStyle The name of the tie style for this Note (read/write). Valid tie style names include: Solid, Dashed, Dotted. If an attempt is made to apply a non-existent style name, the tie in question retains its current style.

WrittenAccidental The accidental, taking transposition into account (read only).
WrittenDiatonicPitch The written diatonic pitch of the note, taking transposition into account if Score.TransposingScore is True ( 35 = middle C).

WrittenName The written pitch of the note as a string (taking transposition into account) (read only).
WrittenPitch The written MIDI pitch of the note, taking transposition into account if Score. TransposingScore is True (60 = middle C) (read only).

UseOriginalDeltaSrForPlayback Is True if the Live start position of this Note should be used for Live Playback. Corresponds to the Live start position checkbox in the Playback panel of the Properties window.

UseOriginalDurationForPlayback Is True if the Live duration of this Note should be used for Live Playback. Corresponds to the Live duration checkbox in the Playback panel of the Properties window.

UseOriginalVelocityForPlayback Is True if the Live velocity of this Note should be used for Live Playback. Corresponds to the Live velocity checkbox in the Playback panel of the Properties window.

## NoteSpacingRule

Provides access to the settings from the Appearance > House Style > Note Spacing Rule dialog. Obtained by way of the Score object, for example:

```
nsr = Sibelius.ActiveScore.NoteSpacingRule;
```


## Methods

None.

## Variables

The following variables are listed in the same order as the options to which they correspond in the Note Spacing Rule dialog.
FixedBarRestWidth The width of an empty bar if the Fixed empty bar width $n$ spaces radio button is chosen (read/write). This value is only used if DetermineEmptyBarWidthBySrLength is False.

DetermineEmptyBarWidthBySrLength Returns True if Empty bar width is determined by time signature is chosen, otherwise False (read/write).

StartOfBarGap The value of Before first note in bar $n$ spaces (read/write).
MinimumDurationSpace The value of Short notes $n$ spaces (read/write).
SpaceForSixteenth The value of 16 th note (semiquaver) $n$ spaces (read/write).
SpaceForEighth The value of 8th note (quaver) $n$ spaces (read/write).
SpaceForQuarter The value of Quarter note (crotchet) $n$ spaces (read/write).
SpaceForHalf The value of Half note (minim) $n$ spaces (read/write).
SpaceForWhole The value of Whole note (semibreve) $n$ spaces (read/write).
SpaceForDoubleWhole The value of Double whole note (breve) $n$ spaces (read/write).
AllowSpaceForVoiceConflicts Returns True if Allow extra space for colliding voices is enabled, otherwise False (read/write).

SpaceAroundGraceNote The value of Space around grace notes $n$ spaces (read/write).
ExtraSpaceAfterLastGraceNote The value of Extra space after last grace note $n$ spaces (read/write).
IncludeChordSymbols Returns True if Allow space for chord symbols is enabled, otherwise False (read/write).

ExtraSpaceBetweenGuitarFrames The value of Minimum gap between chord symbols $n$ spaces (read/write).
MinSpaceAroundNote The value of Around noteheads (and dots) $n$ spaces (read/write).
MinSpaceBeforeAccidental The value of Before accidentals $n$ spaces (read/write).
MinSpaceBeforeArpeggio The value of Before arpeggio $n$ spaces (read/write).
MinSpaceAfterHook The value of After tails with stems up $n$ spaces (read/write).
MinSpaceAroundLegerLine The value of Around leger lines $n$ spaces (read/write).
MinSpaceAtStartOfBar The value of After start of bar $n$ spaces (read/write).
MinSpaceAtEndOfBar The value of Before end of bar $n$ spaces (read/write).
MinTieSpacing The value of Min space (tie above/below note) $n$ spaces (read/write).
MinTieSpacingChords The value of Min space (tie between notes) $n$ spaces (read/write).
IncludeLyrics Returns True if Allow space for lyrics is enabled, otherwise False (read/write).
AllowFirstLyricOverhang Returns True if Allow first lyric to overhang barline is enabled, otherwise False (read/write).
AllowSpaceForHyphen Returns True if Allow extra space for hyphens is enabled, otherwise False (read/write).
SpaceBetweenLyrics The value of Minimum gap between lyrics $n$ spaces (read/write).

## PageNumberChange

Provides access to get and set the attributes of a page number change at the end of a bar or on a blank page.

## Methods

SetFormatChangeOnly (format change only) If format change only is True, this has the same effect as switching off the New page number check box on the Page Number Change dialog in Sibelius. The page numbering will therefore continue counting consecutively, but it's possible to (for example) hide a group of page numbers and restore visibility at a later point on the score without having to keep track of the previous page numbers.

SetHideOrShow (page number visibility) Takes one of the three Page number visibility global constants to determine the visibility of the initial page number change and its subsequent pages; see Global Constants.

SetPageNumber (page number) Takes an integral number specifying the new number you wish to assign to the page.
SetPageNumberFormat (format) Takes one of the four Page number format global constants to change the format used to display the page number change; see Global Constants.

## Variables

BarNumber Returns the bar number expressed as an integer (read only).
HideOrShow Returns one of the three Page number visibility global constants; see Global Constants (read only).
PageNumber Returns the page number expressed as an integer. For example, page $x$ when using Roman numerals would be 10 , or 24 with alphabetics (read only).

PageNumberAsString Returns the page number change as visible on the corresponding page in Sibelius (read only). PageNumberBlankPageOffset

Returns the blank page offset of the page number change, or 0 if there are no blank pages following the bar containing the page number change (read only).

PageNumberFormat Returns one of four Page number format global constants describing the format of the page number change; see Global Constants (read only).

## PluginList

An array that is obtained from Sibelius. Plugins. It can be used in a for each loop or as an array with the [ n ] operator to access each Plugin object.

## Methods

Contains (pluginName) Returns True if a plug-in with the given name is installed. This can be used to query whether a plug-in is installed before you try to call it.

## Variables

NumChildren Number of plug-ins (read only).

## Plugin

This represents an installed plug-in. Typical usage:

```
    for each p in Sibelius.Plugins
    {
        trace("Plugin: " & p.Name);
}
```


## Methods

The following methods are intended to allow you to check the existence of specific methods, data and dialogs in plug-ins, which allows you to check in advance that calling a method in another plug-in will succeed, and fail gracefully if the method is not found:

MethodExists (method) Returns True if the specified method exists in the current Plugin object.
DataExists (data) Returns True if the specified data exists in the current Plugin object.
DialogExists (dialog) Returns True if the specified dialog exists in the current Plugin object.

## Variables

File The File object corresponding to the file that the plug-in was loaded from (read only).
Name The name of the plug-in (read only).

## RehearsalMark

Derived from a BarObject and found in the system staff only. RehearsalMarks have an internal numbering and a visible text representation, both of which can be read from ManuScript.

## Methods

None.

## Variables

Mark The internal number of this rehearsal mark. By default rehearsal marks are consecutive (with the first one numbered zero), but the user can also create marks with specific numbers.

MarkAsText The textual representation of this rehearsal mark as drawn in the score. This is determined by the House Style > Engraving Rules options, and can take various forms (numerical or alphabetical).

## Score

You can obtain the Score object by way of the Sibelius object, for example:
score $=$ Sibelius.ActiveScore;
A Score contains one System Staff and one or more Staff objects.
for each variable in returns each staff in the score or the current dynamic part in turn (not the system staff).
for each type variable in returns the objects in the score in chronological order, from the top staff to the bottom staff (for simultaneous objects) and then from left to right (again, not including the system staff).

## Methods

AddBars ( $n$ ) Adds $n$ bars to the end of the score.
ApplyStyle (style file,"style", ["style"]) Imports named styles from the given house style file (.lib) into the score. The style file parameter can either be a full path to the file, or just the name of one of the styles that appears in the House Style > Import House Style dialog. You can import as many "style" elements as you like in the same method. Style names are as follows:

ENGRAVING RULES, DOCSETUP, HOUSE,TEXT,SYMBOLS,LINES,NOTEHEADS,CLEFS,DICTIONARY, SPACINGRULE,DEFAULTPARTAPPEARANCE,INSTRUMENTSANDENSEMBLES, MAGNETICLAYOUTOPTIONS OR ALLSTYLES .

For instance:

```
score2.ApplyStyle("C:\NewStyle.lib", "HOUSE", "TEXT");
```

To import engraving rules for a a house style, but not the document settings, use the ENGRAVINGRULES constant. To import document settings for a house style, but not the engraving rules, use the DOCSETUP constant. To import both the engraving rules and document setup of a house style, use the HOUSE constant. Note that the constant HOUSE refers, for historical reasons, only to those options in the House Style > Engraving Rules and Layout > Document Setup dialogs, not the entire house style. To import the entire House Style, use the ALLSTYLES constant. The constants: ENGRAVINGRULES, DOCSETUP, HOUSE, and ALLSTYLES are mutually exclusive and should not be used in combination.

When importing Text Styles from a House Style, ApplyStyles () lets you import the "Music Text Font" using the TEXT Style name (or any Style name that depends on TEXT).

ClefStyleId (clef style name) Returns the identifier of the clef style with the given name, or the empty string if there is no such clef style.

CreateInstrument (style ID[, change names,["full name",["short name"]]]) Creates a new instrument, given the style ID of the instrument type required (see Instrument Types). If you want to supply the instrument names to be used in the score, set the optional change names parameter to True, then supply strings for the full name and short name. Returns True if the instrument was created successfully and False if the instrument type could not be found.

CreateInstrumentAtBottom (style ID[, change names, ["full name", ["short name"]]]) Behaves the same way as CreateInstrument, only the new instrument is always created below all other instruments that currently exist in the score. This can be useful when programmatically copying a list of staves/instruments from one score to another, as you can guarantee the ordering of the staves will be the same in both scores.

CreateInstrumentAtBottomReturnStave (style ID[, change names,["full name", ["short name"]]]) As above, but returns the Staff object created, or null if unsuccessful.

CreateInstrumentAtTop (style ID[, change names,["full name",["short name"]]]) Behaves in exactly the same way as CreateInstrumentAtBottom, only the new instrument is always created above all other instruments that currently exist in the score.

CreateInstrumentAtTopReturnStave (style ID[, change names, ["full name", ["short name"]]]) As above, but returns the Staff object created, or null if unsuccessful.

CreateInstrumentReturnStave (style ID[, change names, ["full name", ["short name"]]]) Like CreateInstrument, but returns the Staff object created, or null if unsuccessful. Note that if the instrument being created contains more than one staff (such as piano or harp), the top stave of the instrument in question will be returned.

ExportHouseStyle (filename) Exports the house style of the score to either the default "House Style" folder (if the argument is a filename), or to a user-specified path (if the argument is a valid file path).

ExportPartsAsPDF (filename[, single file[ ,part IDs[, include score]]]) Exports one dynamic part, a selection of dynamic parts, or all dynamic parts in PDF format, either concatenated into a single file, or as separate files. The filename parameter should be a complete path. It may contain the following tokens, which Sibelius will expand automatically to generate a complete filename:

```
%f=Score filename
%t = Score title (as specified in the Title field in File > Info)
%p = Part name (as specified in the Part name field in File > Info)
%n = Part number
%o = Total number of parts
%d = Date (format YYYY-MM-DD)
%h = Time (format HHMM)
```

The Boolean parameter single file specifies whether the chosen parts should be extracted into separate PDF files or concatenated into a single PDF file. This parameter defaults to True if not specified.

To specify which parts to export, create a sparse array of part IDs, and pass this in as the third parameter, part IDs. For example:

```
s = Sibelius.ActiveScore;
partsToExport = CreateSparseArray();
parts = s.DynamicParts;
firstNPartsToExport = 2;
i = 0;
for each part in parts {
    if (i <= firstNPartsToExport) { // <= because the first "part" in the
    //DynamicPartsCollection is the full score.
    partsToExport.Push(part);
    }
    i = i + 1;
}
s.ExportPartsAsPDF("c:\\%f - %p.pdf", true, partsToExport);
```

To export all parts, pass in 0 instead of a sparse array.
The final optional Boolean parameter, include score, defaults to False. If set to True, the full score will also be exported along with the parts.

ExportScoreAsPDF (filename) Exports the full score as a PDF, with the specified filename, which should be a complete path. The filename parameter may use the same tokens as the ExportPartsAsPDF () method-see above.

ExtractParts ([show_dialogs[, parts path[,open parts]]]) Extracts parts from the score. The first optional Boolean parameter can be False, in which case the parts are extracted without showing an options dialog. The second optional parameter specifies a folder into which to extract the parts (must end with a trailing folder separator). The third optional Boolean parameter, which defaults to True, specifies whether the extracted parts should be opened immediately, or simply saved.

FreezeMagneticLayoutPositions() Does the same as selecting the whole score and choosing Layout > Magnetic Layout > Freeze Positions, which explicitly sets the Dx/Dy of every object to the position produced by Magnetic Layout, then disables Magnetic Layout for each object.

GetLocationTime (bar number[,position[,pass]]) Returns the time of a given bar (by passing in its bar number) and optional position within that bar in the score in milliseconds. If the score contains repeats, the value returned will always be the time on the first pass through the score, but you can supply the optional pass parameter to specify a particular pass in the repeat structure. If the bar and position are not valid, the return value will be -1 .

GetVersions() Returns the score's VersionHistory object (see VersionHistory).
HideEmptyStaves (startStaveNum, endStaveNum, startBarNum, endBarNum) Hides any empty staves between startStaveNum and endStaveNum, from startBarNum to endBarNum. Both the staff numbers and bar numbers are 1-based, and refer to the active part.

GetVersions() Returns the score's VersionHistory object (see VersionHistory).

InsertBars ( $n$, barNum[, length]) Inserts $n$ bars before bar number barNum. If no length has been specified, the bar will be created with the correct length according to the current time signature. However, irregular bars may also be created by specifying a value for length.

InternalPageNumToExternalPageNum (pagenum) Returns a string containing the external page number of the given internal page number pagenum.

LineStyleId (line style name) Returns the identifier of the line style with the given name, or the empty string if there is no such line style.

NoteStyleIndex (notehead style name) Returns the index of the note style with the given name, or -1 if there is no such note style.
NthStaff (staff index from 1) Returns the $n$th staff of the score or the current dynamic part.
OptimizeStaffSpacing (from staff number[, to staff number[, from bar[,to bar]]]) Does the equivalent of Layout > Optimize Staff Spacing for the given range of staves or a whole score. from staff number must be specified; if to staff number is not specified, Sibelius will optimize the distances between from staff number and the bottom staff in the score; if from bar is not specified, Sibelius sets it to $\mathbf{1}$; if to bar is not specified, Sibelius sets it to the last bar of the score.

PlayLiveTempo (play) Switches Play > Live Tempo on or off; set play to True to switch it on, or False to switch it off.
RemoveAllHighlights () Removes all highlights in this score.
RemoveVideo () Removes an attached video from the score.
RenameTextStyle ("old name", "new name") Renames a text style to a new name.
Save (filename) Saves the score, overwriting any previous file with the same name.
SaveAs (filename,type[, use_defaults,foldername]) Saves the score in a specified format, overwriting any previous file with the same name. The optional argument use_defaults only applies to graphics files, and specifies whether or not the default settings are to be used. When set to False, the Export Graphics dialog will appear and allow the user to make any necessary adjustments. The optional foldername specifies the folder in which the file is to be saved, and will create the specified folder if it does not exist. The foldername parameter must not end with a path separator (which is " $\backslash \backslash$ " on Windows).

The possible values for type are:

```
SIBL Sibelius format (current version)
    EMF EMF
    BMP Windows bitmap
    PICT PICT format
    PDF PDF format
    PNG PNG format
    Midi MIDI format
    TIFF TIFF format
    MUSICXML Uncompressed MusicXML
    MXL Compressed MusicXML
```

"乌" The file type value for Uncompressed musicXML files was formerly "XML" rather than "MUSICXML." XML still represents a valid file type, but the file will be saved with a .musicxml extension.

So, to save a file using the current Sibelius file format, you would write score.SaveAs ("filename.sib", "SIBL");
SaveAsAudio (filename[, include all staves[, play from start]]) Creates a WAV file (PC) or AIFF file (Mac) of the score, using Sibelius's File 0Export OAudio feature. If include all staves is True (the default), Sibelius will first clear any existing selection from the score so every instrument will be recorded; only selected staves will otherwise be exported. When play from start is True (also the default), Sibelius will record the entire score from beginning to end, otherwise from the current position of the playback line. Note that SaveAsAudio will only have an effect if the user's current playback configuration consists of solely VST and/or AU devices. The functions returns True if successful, otherwise False (including if the user clicks Cancel during export).

SaveAsSibelius2 (filename[, foldername]) Saves the score in Sibelius 2 format, overwriting any previous file with the same name. The optional foldername specifies the folder in which the file is to be saved. Note that saving as Sibelius 2 may alter some aspects of the score; see Sibelius Reference for full details.

SaveAsSibelius3 (filename[,foldername]) Saves the score in Sibelius 3 format. See documentation for SaveAsSibelius2 above.

SaveAsSibelius4 (filename[,foldername]) Saves the score in Sibelius 4 format. See documentation for SaveAsSibelius2 above.

SaveAsSibelius5 (filename[,foldername]) Saves the score in Sibelius 5 format. See documentation for SaveAsSibelius2 above.

SaveAsSibelius6 (filename[,foldername]) Saves the score in Sibelius 6 format. See documentation for SaveAsSibelius2 above.

SaveAsSibelius7 (filename[,foldername]) Saves the score in Sibelius 7 format. See documentation for SaveAsSibelius2 above.

SaveAsSibelius7_5 (filename[,foldername]) Saves the score in Sibelius 7.5 format. See documentation for SaveAsSibelius2 above.

SaveAsSibelius8 (filename[,foldername]) Saves the score in Sibelius 8 format. See documentation for SaveAsSibelius2 above.

SaveAsSibelius8_5 (filename[,foldername]) Saves the score in Sibelius 8.1 format. See documentation for SaveAsSibelius2 above.

SaveAsSibelius8_6 (filename[,foldername]) Saves the score in Sibelius 8.6 format. See documentation for SaveAsSibelius2 above.

SaveAsSibelius2020_1 (filename[,foldername]) Saves the score in Sibelius 2020.1 format. See documentation for SaveAsSibelius2 above.

SaveCopyAs (filename[,foldername]) Saves a copy of the score in the current version's format without updating the existing score's file name in Sibelius.

SetPlaybackPos (bar number, $s r$ ) Sets the position of the playback line to a given bar number and rhythmic ( $s r$ ) position.
ShowEmptyStaves (startStaveNum, endStaveNum, startBarNum, endBarNum) Shows any empty staves currently hidden using Layout > Hiding Staves > Hide Empty Staves between startStaveNum and endStaveNum, from startBarNum to endBarNum. Both the staff numbers and bar numbers are 1-based, and refer to the active part.

StaveTypeId (stave type name) Returns the identifier of the stave type with the given name, or the empty string if there is no such stave type.

SystemCount (page num) The number of systems on a page (the first page of the score is page 1).
SymbolExists (symbol) Returns True if the symbol index or name symbol is found in the score, otherwise False.
SymbolIndex (symbol name) Returns the index of the symbol with the given name, or $\mathbf{- 1}$ if there is no such symbol.
TextStyleId (text style name) Returns the identifier of the text style with the given name, or the empty string if there is no such text style.

ViewLiveTempo (view) Switches View > Live Tempo on or off; set view to True to switch it on, or False to switch it off.

## Variables

Arranger Arranger of score from File > Score Info (read/write).
Artist Artist of score from File > Score Info (read/write)
Barlines Returns a Barlines object containing information about the barline groupings in the score (read only).
BarPlaybackOrder Returns a sparse array containing a list of integers that describes the order in which the bars will be played, according to the repeat structure of the score or the settings in Play > Interpretation > Repeats. To set the order in which bars should be played, pass in a sparse array containing a list of integers describing the order in which bars should be played back. To return to the score's automatically-determined playback order, pass in null (read/write).

BarPlaybackOrderString Returns a string describing the order in which the bars will be played, according to the repeat structure of the score. The string uses the same format as the read-out in Play > Interpretation > Repeats, for example, " $1-8$, $1-5,9-12$ ". To set the order in which bars should be played, pass in a string of the appropriate format. To return the score's automatically-determined playback order, pass in null (read/write).

BracketsAndBraces Returns a BracketsAndBraces object containing information about the brackets and braces in the score (read only).

Composer Composer of score from File > Score Info (read/write).
ComposerDates Value of Composer's dates from File > Score Info (read/write).
Copyist Copyist of score from File > Score Info (read/write).
Copyright Copyright of score from File > Score Info (read/write).
CurrentDynamicPart Returns or sets the current DynamicPart object for the Score (read/write). Sibelius will not automatically display the new part: use Sibelius.ShowDynamicPart() to change the displayed part.

CurrentPlaybackPosBar Returns the bar number in which the playback line is currently located.
CurrentPlaybackPosSr Returns the rhythmic position within the bar at which the playback line is currently located.
Dedication Dedication of score from File > Score Info (read/write).
DocumentSetup Returns a DocumentSetup object representing the settings in Layout > Document Setup (read only).
DynamicParts Returns a DynamicPartCollection object representing the dynamic parts present in the Score. This object will always stay up to date, even if parts are added or deleted (read only).

EditingLocked True when the score is locked, preventing unintentional edits (read/write).
EnableScorchPrinting Corresponds to the Allow printing and saving checkbox in the Export Scorch Web Page dialog (read/write).

EngravingRules Returns an EngravingRules object corresponding to selected settings in the House Style> Engraving Rules dialog (read only).

FileName The filename for the score (read only).
FocusOnStaves Is True if View > Focus on Staves is enabled (read/write). See also Staff. ShowInFocusOnStaves.
HitPoints The HitPointList object for the score (read/write).
InstrumentChanges Value of Instrument changes from File > Score Info (read/write).
InstrumentTypes Returns an InstrumentTypeList containing the score's instrument types, on which one may execute a for each loop to get information about each instrument type within the score.

IsDynamicPart Returns True if the current active score view is a part (read only).
LiveMode Is True (1) if Play > Live Playback is on (read/write).
Lyricist Lyricist of score from File > Score Info (read/write).
MagneticLayoutEnabled Returns True if the current score has Layout > Magnetic Layout switched on (read/write).
MainMusicFontName Returns the name of the font specified as the Main music font (such as "Opus" or "Reprise") in House Style >Edit All Fonts (read/write).

MainTextFontName Returns the name of the font specified as the Main text font (such as "Times New Roman" or "Arial") in House Style > Edit All Fonts (read/write).

MusicTextFontName Returns the name of the font specified as the Music text font (such as "Opus Text" or "Reprise Text") in House Style > Edit All Fonts (read/write).

NumberOfPrintCopies The number of copies to be printed (read/write).
OpusNumber Opus number of score from File > Score Info (read/write).
OriginalProgramVersion The version of Sibelius in which this score was originally created, as an integer in the following format:
$($ major version $) * 1000+($ minor version $) * 100+($ revision $) * 10$. So Sibelius at the time of this writing would be 8.3.1 would be returned as 8310 .

OtherInformation More information concerning the score from File > Score Info (read/write).
PageCount The number of pages in the score (read only).
PartName Value of Part Name from File > Score Info (read/write).
Publisher Publisher of score from File > Score Info (read/write).
Redraw Set this to True (1) to make the score redraw after each change to it, False (0) to disallow redrawing (write only).
ScoreDuration The duration of the score in milliseconds (read only).
ScoreEndTime The duration of the score, plus the score start time (see above), in milliseconds (read only).
ScoreHeight Height of a page in the score, in millimeters (read only).
ScoreStartTime The value of Timecode of first bar, from Play > Video and Time > Timecode and Duration, in milliseconds (read only).

ScoreWidth Width of a page in the score, in millimeters (read only).
Selection The Selection object for the score, which is a list of selected objects (read only).
ShowMultiRests Is True (1) if Layout > Show Multirests is on (read/write).
StaffCount The number of staves in the score (read only).
StaffHeight Staff height, in millimeters (read only).
Subtitle Subtitle of score (read/write).
SystemCount The number of systems in the score (read only).
SystemObjectPositions Returns a SystemObjectPositions object corresponding to the settings in House Style > System Object Positions for the score (read only).

SystemStaff The SystemStaff object for the score (read only).
Title Title of score from File > Score Info (read/write).
TransposingScore Is True (1) if Notes > Transposing Score is on (read/write).
UsingManualBarPlayOrder Returns True if Manual repeats playback is chosen in Play > Interpretation > Repeats, otherwise False (read only).

YearOfComposition Value of Year of composition from File > Score Info (read/write).

## Selection

for each variable in returns every BarObject (which is an object within a bar) in the selection.
for each type variable in produces each object of type in the selection. Note that if the selection is a system selection (which is surrounded by a double purple box in Sibelius) then objects in the system staff will be returned in such a loop.

## Methods

Clear () Removes any existing selection(s) from the current active score.
ClipboardContainsData ([clipboard Id]) Returns True if the given clipboard contains data. As with the Copy and Paste methods, 0 (or no arguments) refers to Sibelius's internal clipboard, and all other numeric values will interrogate the temporary clipboard with the matching ID.

Copy ([clipboard Id]) Copies the music within the current selection to Sibelius's internal clipboard or a ManuScript-specific temporary clipboard, which goes out of scope along with the Selection object itself. If no clipboard Id is specified, or if it is set to 0 , the selection will be copied to Sibelius's internal clipboard. Any other numeric value you pass in will store the data in a temporary clipboard adopting the ID you specify. Used in conjunction with Paste or PasteToPosition (see below).

Delete ([remove staves]) Deletes the music currently selected in the active score. Akin to making a selection manually in Sibelius and hitting Delete. If remove staves is omitted or set to True, Sibelius will completely remove any wholly selected staves from the score. If you wish Sibelius to simply hide such staves instead, set this flag to False.

ExcludeStaff (staff number) If a passage selection already exists in the current active score, an individual stave may be removed from the selection using this method.

HideSelectedEmptyStaves () If the current selection contains staves that are empty, they will be hidden (equivalent to selecting a passage and choosing Layout > Hiding Staves > Hide Empty Staves).

IncludeStaff (staff number) If a passage selection already exists in the current active score, a non-consecutive stave may be added to the selection using this method.

Paste ([clipboard Id[,reset positions]]) Pastes the music from a given clipboard to the start of the selection in the current active score. If no clipboard Id is specified, or if it is set to 0 , the data will be pasted from Sibelius's internal clipboard. Any other numeric value you pass in will take the data from a temporary clipboard you must have previously created with a call to Copy (see above). Returns True if successful.

If reset positions is False, the positions of any objects that have been moved by the user in the source selection will be retained in the copy. This is the default behavior. If you wish Sibelius to reset objects to their default positions, set this flag to True. This can be useful when copying one or more single objects (which is a non-passage selection).

Note that pasting into a score using this method will overwrite any existing music. Only one copy of the music will ever be made, so if your selection happens to span more bars or staves than is necessary, the data will not be duplicated to fill the entire selection area.

PasteToPosition (stave num, bar num, position[, clipboard Id[,reset positions]]) Pastes the music from a given clipboard to a specific location in the current active score. The optional parameters and pasting behavior works in the same way as calls to Paste.

RestoreSelection () Restores the selection previously recorded with a call to StoreCurrentSelection. Usefully called at the end of a plug-in to restore the initial selection.

SelectPassage (start barNum [, end barNum [, top staveNum [, bottom staveNum[,start pos[, end pos]]]]]) Programmatically makes a passage selection around a given area of the current active score. When no end barNum is given, only the start barNum will be selected. If neither a top-nor bottom staveNum has been specified, every stave in the score will be selected, whereas if only a top staveNum has been supplied, only that one staff will be selected. Sibelius will begin the selection from the start of the first bar if no start pos has been given, similarly completing the selection at the end of the final bar if no end pos has been supplied.
? The start pos and end pos you supply may be altered by ManuScript: Sibelius requires a passage selection to begin and end at a NoteRest if it doesn't encompass the entire bar.

SelectSystemPassage (start barNum [, end barNum [, start pos[, end pos]]]) Programmatically makes a system selection around a given area of the current active score. When no end barNum is given, only the start barNum will be selected. Sibelius will begin the selection from the start of the first bar if no start pos has been given, similarly completing the selection at the end of the final bar if no end pos has been supplied.

[^2]StoreCurrentSelection () Stores the current selection in the active score internally. Can be retrieved with a call to RestoreSelection (see below). Usefully called at the start of a plug-in to store the initial selection.

Transpose (degree, interval type[,keep double accs[,transpose keys]]) Transposes the currently selected music up or down by a specified degree and interval type. To transpose up, use positive values for degree; to transpose down, use negative values. Note that degrees are 0 -based, so 0 is equal to a unison, 1 to a second and so on. For descriptions of the various available interval types, see Global Constants. By default, Sibelius will transpose using double sharps and flats where necessary, but this behavior may be suppressed by setting the keep double accs flag to False. Sibelius will also transpose any key signatures within the selection by default, but can be overridden by setting the fourth parameter to False.

For help in calculating the interval and degree required for a particular transposition, see the documentation for the Sibelius.CalculateInterval and Sibelius.CalculateDegree methods.

## Variables

BottomStaff The number of the bottom staff of a passage (read only).
FirstBarNumber The internal bar number of the first bar of a passage (read only).
FirstBarNumberString The external bar number (including any bar number format changes) of the first bar of a passage (read only).

FirstBarSr The position of the start of the passage selection in the first bar (read only).
IsPassage True if the selection represents a passage, as opposed to a multiple selection (read only).
IsSystemPassage True if the selection includes the system staff (read only).
LastBarNumber The internal bar number of the last bar of a passage (read only).
LastBarNumberString The external bar number (including any bar number format changes) of the last bar of a passage (read only).
LastBarSr The position of the end of the passage selection in the last bar (read only).
TopStaff The number of the top staff of a passage (read only).

## Copying Entire Bars

Copying passages from one location in a score to another-or even from one score to another-is very simple. Here is an example function demonstrating how one might go about achieving this:

```
CopyBar(scoreSrc, barFirstSrc, barLastSrc, scoreDest, barFirstDest,
    barLastDest) // This is the function signature
{
    sel = scoreSrc.Selection;
    sel.SelectPassage (barFirstSrc.BarNumber, barLastSrc.BarNumber,
        barFirstSrc.ParentStaff.StaffNum,
        barLastSrc.ParentStaff.StaffNum) ;
    sel.Copy(0) ;
    selDest = scoreDest.Selection;
    selDest.SelectPassage (barFirstDest.BarNumber, barLastDest.BarNumber,
                                    barFirstDest.ParentStaff.StaffNum,
                                    barLastDest.ParentStaff.StaffNum);
    selDest.Paste(0);
}
```

Note that you may use any temporary clipboard or Sibelius's own internal clipboard if the source and destination locations are in the same score, however you can only use Sibelius's internal clipboard if the data is being transfered between two individual scores. This is because the temporary clipboards belong to the Selection object itself.

## Copying Multiple Selections from One Bar to Another

Using a combination of the BarObject's Select method and the Selection object's Copy and PasteToPosition methods, it is possible to copy an individual or multiple selection from one location in a score to another. Bear in mind that Paste will always paste the material to the very start of the selection, so if you're copying a selection that doesn't start at the very beginning of a bar, you'll have to store the position of the first item and pass it to PasteToPosition when you later come to paste the music to another bar.

This example code below copies all items from position 256 or later from one bar to another. It is assumed that sourceBar is a valid BarObject, and destStaffNum and destBarNum contain the destination staff number and bar number respectively:

```
sel = Sibelius.ActiveScore.Selection; // Get a Selection object for this score
sel.Clear(); // Clear the current selection
clipboardToUse = 1; // This clipboard ID we're going to use
copyFromPos = 256; // Copy all objects from this point in the source bar
posToCopyTo = 0; // Variable used to store the position of the first object copied
for each obj in sourceBar { // Iterate over all objects in the bar
    if (obj.Position >= copyFromPos) { // Ignore objects before the start threshold
            obj.Select(); // Select each relevant object in turn
            if (posToCopyTo = 0) {
                    posToCopyTo = obj.Position; // Remember the position of the first item
            {
    }
}
sel.Copy(clipboardToUse); // Copy the objects we've selected to the clipboard
sel.PasteToPosition(destStaffNum, destBarNum, posToCopyTo, clipboardToUse); // And
paste them to the destination bar at the relevant offset
```


## Sibelius

There is a predefined variable that represents the Sibelius program. You can use the Sibelius object to open scores, close scores, display dialogs or (most commonly) to get currently open Score objects.
for each variable in returns each open score.

## Methods

AppendLineToFile (filename, text[,use_unicode]) Appends a line of text to the file specified (adds line feed). See comment for AppendTextFile above for explanation of the use_unicode parameter. Returns True if successful.

AppendLineTortffile (filename, text) Appends a line of text to the file specified. Times New Roman 12pt is used, unless you specify a change of formatting. To change formatting, use the following backslash expressions:
> $\backslash B \backslash$ bold on
> $\backslash I \backslash$ italic on
> $\backslash \cup \backslash$ underline on
> $\backslash n \backslash$ new line
> $\backslash b \backslash$ bold off
> $\backslash i \backslash i t a l i c$ off
> $\backslash \mathbf{u} \backslash$ underline off
> $\backslash \mathbf{f}$ fontname $\backslash$ change to given font name (for example $\backslash f A r i a l \backslash$ to switch to Arial)
> $\backslash s p o i n t s \backslash$ set the font size to a specific point size (for example $\backslash \boldsymbol{s} 16 \backslash$ to set the font to 16 pts ).

Note the difference in meaning of $\backslash s$ in the context of adding data to an RTF file, versus its use in the context of styling text directly within Sibelius (see Syntax following).

AppendTextFile (filename, text[,use_unicode]) Appends text to the file specified. If the optional Boolean parameter use_unicode is True, then the string specified will be exported in Unicode format; if this parameter is False then it will be converted to 8-bit Latin-1 before being added to the text file. This parameter is True by default. Returns True if successful.

CalculateDegree (source pitch, dest pitch, upward interval) Takes two note names in the form of a string (for example C, G\#, Bb, Fx or Ebb) and a boolean that should be True if the interval you're wishing to calculate is upward. Returns a 0 -based number describing the degree between the two notes.

For example, CalculateDegree("C\#", "G", False) would return 3.

CalculateInterval (source pitch, dest pitch, upward interval) Takes two note names in the form of a string (for example C, G\#, $\mathrm{Bb}, \mathrm{Fx}$ or Ebb ) and a boolean that should be True if the interval you're wishing to calculate is upward. Returns a number representing an Interval Type (see Global Constants). You can use the value returned in calls to NoteRest. Transpose and Selection.Transpose.

For example, CalculateInterval ("Bb", "G\#", True) would return IntervalAugmented.
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 Fal se 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).

CommandExists (CommandID) Returns a boolean to reflect if a given CommandID is available for execution.
For example:

## if (Sibelius.CommandExists(toggle_review_mode)) \{\}

CreateFolder (foldername) Creates the folder of specified foldername; returns the Folder object created if successful, or null if it fails.

CreateProgressDialog (caption, min value, max value) Creates the progress dialog, which shows a slider during a long operation.

CreateRTFFile (filename) Creates the Rich Text Format (RTF) file specified. Any existing file with the same name is destroyed. Returns True if successful.

CreateTextFile (filename) Creates the plain text file specified. Any existing file with the same name is destroyed. Returns True if successful.

DestroyProgressDialog() Destroys the progress dialog.
EnableControlById (plugin, dialog, controlID, enable) Dynamically enables or disables a given control on a plug-in dialog: plug-in is a Plugin object, for example Self; dialog is a Dialog object, and therefore should not be passed in quotation marks; controlID is the string corresponding to the control to be enabled or disabled; and enable is a Boolean parameter, which enables the control when set to True and disables the control when set to False.

EnableNthControl ( $n$th control, enable) Dynamically enables or disables a given control on a plug-in dialog. Can be called either before a dialog has been displayed (in which case the operation will apply to the next dialog you show), or while a dialog is already visible (in which case the operation will affect the top-most currently visible dialog).

Note that, using this method, controls can only be identified according to their order upon creation; for this reason, you are strongly recommended to use EnableControlById () instead. To find out the creation order, open the appropriate dialog in the plug-in editor, right click on the dialog's client area and choose Set Creation Order from the contextual menu that appears. Note that nth control expects a 0 -based number, unlike the display given by Set Creation Order. By default, all controls will be enabled; to disable any given control, set enable to false.

Execute (CommandID) Executes the specified command. This lets you create scriptable plug-ins that can execute any number of commands one after the other. Using Sibelius. Execute () you can create plug-ins to speed up workflows that include multi-step repetitive tasks and then execute them all in one step. Any command listed in the Commands list on the Home tab can be used with Sibelius.Execute ().

Use Cmd () in conjunction with Sibelius.Execute () to execute commands by name rather than by CommandID. For example: Sibelius.Execute (Cmd("Select All"));

Of course, you can use other ManuScript methods for features not in the Commands list in combination with Sibelius.Execute () to create plug-ins for almost any workflow.

You can further speed up your workflow by assigning a keyboard shortcut to your plug-in.
FileExists (filename) Returns True if a file exists or False if it doesn't.
FolderExists (foldername) Returns True if a folder exists or False if it doesn't.
FindCommandId (Command Name) Translates the name of a command (as shown in Sibelius) into a Commandld, and returns that Commandld as a string. The Command Name argument is case-insensitive. It is also specific to the current language set in Sibelius. If no match is found, an empty string will be returned. For example:

Sibelius.FindCommandId("ToGGle ReVieW mODe");
FindCommandName (Id) Translates a CommandId or a StyleId to the corresponding Command Name or Style Name in the current language set in Sibelius and returns that Name as a string. If no match is found, an empty string is returned. For example:

```
Sibelius.FindCommandName("toggle_review_mode"); //CommandId
Sibelius.FindCommandName("line.staff.bend"); //StyleId
```

FindStyleId (Style Name) Translates the name of a Style (as shown in Sibelius) into a Styleld, and returns that Styleld as a string. The Style Name argument is case-insensitive. It is also specific to the current language set inside Sibelius. If no match is found, an empty string is returned. For example:

Sibelius.FindStyleId("Crescendo");
GetDocumentsFolder () Returns the user's My Documents (Windows) or Documents (Mac) folder.
GetElapsedCentiSeconds (timer number) Returns the time since ResetStopWatch was called for the given stop watch, in 100ths of a second.

GetElapsedMilliSeconds (timer number) Returns the time since ResetStopWatch was called for the given stop watch, in 1000ths of a second.

GetElapsedSeconds (timer number) Returns the time since ResetStopWatch was called for the given stop watch in seconds.
GetFile(file path) Returns a new File object representing a file path for example file=Sibelius.GetFile
("c:<br>onion<br>foo.txt");
GetFolder (file path) Returns a new Folder object representing a file path for example folder=Sibelius.GetFolder("c:\");

GetListOfCommandNamesInCategory (Localized Command Category Name) Returns a sparse array of all External Localized Command Names (in alphabetical order). For example:

```
fileTabCommands = Sibelius.GetListOfCommandNamesInCategory("File Tab")
```

GetNotesForChord (chord name) Returns a ManuScript array giving the MIDI pitches corresponding to the named chord symbol.
GetNotesForGuitarChord (chord name) Returns a ManuScript array giving the MIDI pitches and string numbers corresponding to the named guitar chord, using the most suitable fingering according to the user's preferences. Strings are numbered starting at 0 for the bottom string and increasing upwards. The array returned has twice as many entries as the number of notes in the chord, because the pitches and string numbers are interleaved thus:

```
array [0] = MIDI pitch for note 0
array [1] = string number for note 0
array [2] = MIDI pitch for note 1
array [3] = string number for note 1
```

GetScoresFolder () Returns a new Folder object representing the default Scores folder (as defined on the Files page of File 0 Preferences).

GetSyllabifier() Returns a new Syllabifier object, providing access to Sibelius's internal syllabification engine.
GetUserApplicationDataFolder () Returns the user's Application Data (Windows) or Application Support (Mac) folder.

GetUserSibeliusFolder () Returns the path to the Sibelius folder inside the user Documents folder by default (this value may be overridden in File > Preferences > Saving and Exporting > Saving scores). For example:

On Windows: C:IUsersl<your user name>\Documents\Scores
On Mac: /Users/<your user name>/Documents/Scores
GoToEnd () Moves the playback line to the end of the score.
GoToStart() Moves the playback line to the start of the score.
IsDynamicPartOpen (dynamic part) Returns True if the specified part and its corresponding Score is valid and is visible in a Score window within Sibelius.

IsFontFamilyInstalled (font name) Returns True if a font with the name font name exists on the system, otherwise False.
LaunchApplication (path[, parameters[, hide]]) Launches an external application specified via its path, which must be a complete path to the application to be launched. You can optionally pass in a sparse array of parameters (or a string if you want to pass in only a single parameter); omit this or set it to null to pass no parameters to the launched application. To prevent the launched application from gaining the focus once it is launched, set the optional hide parameter to True; if unspecified, this defaults to False, so the launched application will gain the focus.

LiveTempoTap () Equivalent to tapping a beat during Live Tempo recording.
MakeSafeFileName (filename) Returns a "safe" version of filename. The function removes characters that are illegal on Windows or Unix, and truncates the name to 31 characters so it will be viewable on Mac OS 9 .

MessageBox (string) Shows a message box with the string and an OK button.
MoveActiveViewToBar (bar number[, position]) Brings a given internal bar number into view. Has the same effect as Go to Bar in Sibelius. An optional position within the bar may also be specified, but if omitted, the very start of the bar will be brought into view.

MoveActiveViewToSelection ([start of selection]) Brings the object(s) currently selected into view. If start of selection is False, the end of the selection will be brought into view. If the optional argument is True or omitted, the start of the selection will be visible. Has the same effect as Shift + Home/End in Sibelius.

New ([manuscript paper]) Creates and shows a new score. If the optional parameter manuscript paper is not supplied, Sibelius will create a blank score; manuscript paper should be the filename of the manuscript paper you want to create, minus its .sib file extension, optionally including the name of the category (subfolder) in which it is located, for example both "String orchestra" and "Orchestral/String orchestra" will work. Returns the score object corresponding to the new score.

NthScore (score index from 0) Returns the $n$th open score (zero-based), or null if the specified index is not valid.
Open (filename [, quiet]) Opens and displays the given file. Filename must include its extension, for example Song.sib. If the optional boolean parameter quiet is set to True, then no error messages or dialogs will be displayed, even if the file could not be opened for some reason. Returns True if the file is opened successfully, False otherwise.

Play () Plays the current score, from the current position of the playback line.
PlayFromSelection() Plays from the current selection.
PlayFromStart() Plays from the start of the score.
PrependScreenreaderText (string) Prepends string to the default screen reader description.
Print (number of copies [, dynamic part[, showdialog]]) Prints the specified number of copies of the current score or dynamic part using default settings. If number of copies is missing or a negative number, then the default number of copies for the score or part is printed, and if set to 0 no printing occurs. The optional dynamic part parameter must be a valid object of the active Score (this does not affect or use Score. CurrentDynamicPart for the Score printed); if it is not supplied, the active Score is printed instead. Returns True for success, False for failure. The second optional parameter, showdialog, is a Boolean: if set to True, Sibelius will show the Print dialog, and if not specified or set to False, Sibelius will not show the dialog.

PrintAllDynamicParts ([score]) Prints the default number of copies of all dynamic parts, but does not print the full score. Prints the currently-active Score if the optional score parameter is not passed in. Returns True for success, False for failure.

RandomNumber () Returns a random number between 0 and 32,767.
RandomSeed (start number) Restarts the random number sequence from the given number.

RandomSeedTime () Restarts the random number sequence based on the current time.
RefreshDialog() Refreshes the data being displayed by any controls on the currently active plug-in dialog. For example, if a text object gets its string from a global variable and the value stored in this global variable has changed whilst the dialog is visible, calling RefreshDialog will update the text object on the dialog accordingly. Returns True if successful.

ResetStopWatch (timer number) Resets the given stop watch. timer number must be an integer greater than 0 .
ReadTextFile (filename) Reads the given filename into an array of strings, one per line. The file will be treated as ANSI (that is 8 -bit) text by default, unless it starts with a valid UTF-16 byte-order marker (BOM), in which case it will be treated as Unicode.

The resulting array can be used in two ways:

```
lines = Sibelius.ReadTextFile("file.txt");
for each l in lines {
    trace(l) ;
}
Or:
lines = Sibelius.ReadTextFile("file.txt");
for i=0 to lines.NumChildren {
    trace(lines[i]);
}
```

ScreenreaderText (string) Replaces Sibelius's default screen reader description with string.
SelectFileToOpen (caption, file, initial_dir, default extension, default type, default type description) Shows a dialog prompting the user to select a file to open. All parameters are optional. The method returns a file object describing the selection. For example:

```
file=Sibelius.SelectFileToOpen("Save Score","*.sib","c:\","sib","SIBE","Sibelius
File");
```

Note that the initial_dir parameter has no effect on Mac, because it is unsupported by macOSSelectFileToSave (caption, file, initial_dir, default extension, default type, default type description)

Shows a dialog prompting the user to select a file to save to. All parameters are optional. The method returns a File object describing the selection. File types and extensions:

| Description | Type | Extension |
| :--- | :--- | :--- |
| EMF graphics | "EMF" | emf |
| Windows bitmap | "BMP" | bmp |
| Macintosh PICT bitmap"PICT" | pict |  |
| Sibelius score | "SIBE" | sib |
| MIDI file | "Midi" | mid |
| House style file | "SIBS" | lib |
| PhotoScore file | "SCMS" | opt |
| Web page | "TEXT" | html |
| TIFF graphics | "TIFF" | tif |
| PNG graphics | "PNG" | png |

Note that the initial_dir parameter has no effect on Mac, because it is unsupported by macOS.
SelectFolder ([caption]) Allows the user to select a folder and returns a Folder object. The optional string parameter caption sets the caption of the dialog that appears.

SetCurrentScoreViewType (view type) Allows plug-ins to switch between Panorama and normal view; values are ViewTypePage (0) and ViewTypePanorama (1).

SetFocusToControl (pluginName, dialogName, controlID) Sets the focus on a specific control in a plug-in dialog. pluginName will normally be set to self, dialogName is the name of the dialog in which the control is found, and controlID is the ID of the control to receive the focus, which must be specified in quotation marks.

ShowDialog (dialogName, pluginName) Shows a dialog dialogName from a dialog description and sends messages and values to the given Plugin object pluginName (normally set to Self). Returns the value True (1) or False (0) depending on which button you clicked to close the dialog (typically OK or Cancel).

ShowDynamicPart (dynamic part[, newWindow]) Shows the specified dynamic part. The second optional Boolean parameter newWindow allows you to specify whether the part should open in a new tab (specify False, the default) or a new window (specify True). Returns True if the specified part can be shown, False otherwise. Can be used to bring a Score to the front by way of Sibelius.ShowDynamicPart (Score.CurrentDynamicPart).

ShowTraceWindow () Shows the Plug-in Trace Window, or forces it to the front if it is already shown but currently behind another window.

StartLiveTempoRecording () Starts recording Live Tempo; equivalent to choosing Play > Record Live Tempo
StopLiveTempoRecording() Stops recording Live Tempo.
Stop () Stops the current score from playing.
UpdateProgressDialog (progress pos,status message) Returns 0 if the user clicked Cancel.
YesNoMessageBox (string) Shows a message box with Yes and No buttons. Returns True if Yes is chosen, else False.

## Variables

ActiveScore Is the active Score object (read/write). Setting Sibelius. ActiveScore makes active the current dynamic part (which may be the full score rather than a part) of the score. If that window is not currently shown, a new window may be created according to the user's preferences. Returns null if it fails to make the specified score or part active.

ApplicationLanguage Returns the language of the version of S.ibelius currently running, always in English—such as English, German, French and so on. (read only)

ApplicationLanguageIsoString Returns the two-letter ISO 3166 identifier of the language in which Sibelius is currently running, such as en, de, $\mathbf{f r}$, and so on (read only).

AvailableCommands Returns a sparse array of all available CommandIDs in alphabetical order.
CommandCategories Returns a sparse array of all Command Categories in the local language (in system order).
CurrentTime Returns a string containing the current time in the format hh:mm:ss, based on your own computer's locale (read only).
CurrentDateShort Returns a string containing the current date in the format $\mathrm{dd} / \mathrm{mm} / \mathrm{yyyy}$, based on your own computer's locale (read only).

CurrentDateLong Returns a string containing the current date in the format dd MM yyyy, based on your own computer's locale (read only).

CurrentDate Returns the current date and time as a DateTime object in local time (read only).
FontFamilies Returns a sparse array of strings containing the names of all the available font families on the system (read only).
HouseStyles The list of house styles available, as a ComponentList.
LocalizedApplicationLanguage Returns the language in which Sibelius is currently running, in the localized language, for example it returns Deutsch when running in German (read only).

ManuscriptPapers The list of manuscript papers available, as a ComponentList.

OSVersionString The current operating system in which the plug-in is running, as one of the following strings:

| Windows 95 | Mac OS X |
| :--- | :--- |
| Windows 98 | Mac OS X Jaguar |
| Windows ME | Mac OS X Panther |
| Windows NT 3.x | Mac OS X Tiger |
| Windows NT 4 | Mac OS X Leopard |
| Windows 2000 | Mac OS X Snow Leopard |
| Windows XP | Mac OS X Lion |
| Windows Vista | Mac OS X Mountain Lion |
| Windows 7 | macOS Yosemite |
| Windows 8 | macOS El Capitan |
| Windows 10 | macOS Sierra |
|  | macOS Mojave |
|  | macOS Catalina |

If the operating system is unrecognized, the variable returns Unknown system version.
PathSeparator Returns the current path separator character (which is "\" on Windows, "/" on Mac).
Plugins The list of plug-ins installed. See the documentation for the Plugin object
Playing Is True if a score is currently being played (read only).
ProgramVersion The current version of Sibelius in which the plug-in is running, as an integer in the following format:
$($ major version) $* 1000+($ minor version $) * 100+($ revision $) * 10$
So Sibelius 3.1.3 would be returned as 3130 .
ScoreCount Is the number of scores being edited (read only).
SuppressDefaultScreenreaderText Set to True to suppress the default score description for screen readers for blind and visually impaired users (read/write).

ViewAnnotations Is True if View > Invisibles > Annotations is enabled (read/write).
ViewAttachmentLines Is True if View > Invisibles > Attachment Lines is enabled (read/write).
ViewBarNumbers Is True if View > Invisibles > Bar Numbers is enabled (read/write).
ViewComments Is True if View > Invisibles > Comments is enabled (read/write).
ViewHandles Is True if View > Invisibles > Handles is enabled (read/write).
ViewHiddenObjects Is True if View > Invisibles > Hidden Objects is enabled (read/write).
ViewHighlights Is True if View > Invisibles > Highlights is enabled (read/write).
ViewLayoutMarks Is True if View > Invisibles > Layout Marks is enabled (read/write).
ViewNoteVelocities Is True if View > Live Playback Velocities is enabled (read/write).
ViewNoteColors The current View > Note Colors setting used (read/write).

| Description | Value |
| :--- | :--- |
| None | 0 |
| NotesoutofRange | 1 |
| Pitch Spectrum | 2 |
| VoiceCobrs | 3 |

ViewPageMargins Is True if View > Invisibles > Page Margins is enabled (read/write).
ViewPlaybackLine Is True if View > Invisibles > Playback Line is enabled (read/write).
ViewReplayMarker Is True if View > Invisibles > Replay Line is enabled (read/write).

## SoundInfo

The SoundInfo object contains information about the playback of a given staff.
To get the SoundInfo object for a staff, use for example:
staff = Sibelius.ActivateScore.NthStaff(1);
soundinfo $=$ staff.SoundInfoAtPosition(1, 0,0 );
The SoundInfo object can be moved around the staff once you have created it, and it will return information about the sound IDs in use throughout the staff.

## Methods

Clone () Returns a new SoundInfo object using the same credentials as the object on which the method is called.
CreateAt ([barNumber, [position, [nthRepeat]]]) Returns a new SoundInfo object at the specified bar number, at the specified rhythmic position in the bar (for example 256 for the second quarter note position), as if played through at the $n$th repeat (for example 2 for the second repeat). If no bar number is specified, the information returned will refer to bar 1 . If no position is specified, the information will refer to the start of the bar. If nth repeat is not specified, the information returned will refer to the first pass through the score.

MoveTo ([barNumber,[position,[nthRepeat]]]) Uses the same parameters as CreateAt (). Allows the caller to move the existing SoundInfo object to an entirely new location on the current staff.

MoveToNext () Moves to the next sound change event. If there are no more changes, it returns false and doesn't move.
MoveToPrevious () Moves to the previous sound change event. If there are no changes before, it returns false and doesn't move.

## Variables

ActualSoundId The actual sound ID at the current location (read only).
BarNum Returns the current location's bar number (read only).
IsDrumStave Returns true if the current location is on a drum staff (read only).
NthRepeat Returns or sets the $n$th repeat (or pass) of the current location's bar (read/write).
NumTimesBarPlayed The number of times the bar at the current location is played (read only).
PatchName The name of the patch in use at the current location (read only).
Position Returns or sets the current location's position within the bar (read/write).
RequestedSoundId The requested sound ID at the current location (read only).
SoundChangeIndex The current index in the bar play sequence. This allows differentiation between different identical sound changes (read only).

SoundSetName Returns the name of the sound set in use at the current location (read only).
StaveNum Returns the 1-based stave number (read only)

## SparseArray

To create a sparse array, use the built-in method CreateSparseArray ( $a 1, a 2, a 3, a 4 \ldots a n$ ).
for each allows you to iterate over the contents of a sparse array.

## Methods

Concat (array1, array2 $\ldots \operatorname{arrayN}$ ) Concatenate zero or more sparse arrays to this one, and return it as a one-level deep copy (so if a sparse array contains other arrays, for example, then the new sparse array will contain references to those arrays, not copies of them). This method does not modify the original sparse array.

Join ([separator]) Returns the array as a string, with each populated element separated by the optional separator. If you don't specify separator, the default separator is a comma.

Push (value1, value2, value3 ... valueN) Pushes one or more values to the end of the array.
Pop () Returns the last element of the array, and removes it from the array.
Reverse () Reverses the sparse array in place, modifying the sparse array being operated on. The reversed array only populates the elements needed to create the reversed array.

Slice (start[, end]) Returns a new sparse array of the elements starting from start and up to, but not including, the optional end. start and end can be negative indices referring to offsets from the end of the array.

## Variables

Length Returns or sets the length of the array (read/write).
ValidIndices Returns a sparse array containing only the populated indices of the original sparse array, that is those that are not null.

## Converting Old-style Arrays to New Sparse Arrays

The SparseArray object is a replacement for the old Array object, which was a more limited kind of array that could only hold strings and integers, but no other kind of objects. You are recommended to use the new SparseArray object for all arrays in your plug-ins, but if you have an existing plug-in in which old-style Arrays are used, you can convert them to SparseArrays as follows:

Array.ConvertToSparseArray () Returns a new SparseArray object, populated with strings converted from the old-style Array.

## SpecialBarline

Derived from a Bar object, these can only be found in system staves.

## Methods

None.

## Variables

BarlineType The name of the type of special barline, expressed as a string.
BarlineInternalType The type of the barline, expressed as a numeric ID which maps to one of the SpecialBarline global constants (see Global Constants).

## Staff

These can be normal staves or the system staff. The system staff contains objects that apply to all staves, such as SpecialBarlines and text using a system text style.

A Staff contains Bar objects.
for each variable in returns each object in the staff.
for each type variable in returns each item of type in the staff in chronological order (that is in order of rhythmic position in each bar).

## Methods

AddClef (pos, concert pitch clef[, transposed pitch cleff) Adds a clef to the staff at the specified position. concert pitch clef determines the clef style when Notes > Transposing Score is switched off; the optional transposed pitch clef parameter determines the clef style when this is enabled. Clef styles should be an identifier like "clef.treble"; for a complete list of available clef styles, see Clef Styles. Alternatively you can give the name of a clef style, such as "Treble," but bear in mind that this may not work in non-English versions of Sibelius.

AddLine (pos, duration, line style ,[dx,[dy,[voicenumber, [hidden]]]]) Adds a line to staff (please see the documentation in Bar object below).

AddNote (pos, sounding pitch,duration,[tied [, voice [, diatonic pitch[,string number]]]]) Adds a note to staff, adding to an existing NoteRest if already at this position (in which case the duration is ignored); otherwise creates a new NoteRest. Will add a new bar if necessary at the end of the staff. The position is in $1 / 256$ th quarters from the start of the score. The optional tied parameter should be True if you want the note to be tied. Voice 1 is assumed unless the optional voice parameter (with a value of $1,2,3$ or 4 ) is specified. You can also set the diatonic pitch, that is the number of the "note name" to which this note corresponds, 7 per octave ( $35=$ middle C, $36=\mathrm{D}, 37=\mathrm{E}$ and so on). If a diatonic pitch of zero is given then a suitable diatonic pitch will be calculated from the MIDI pitch. The optional string number parameter gives a string number for this note, which is only meaningful if the note is on a tablature stave. If this parameter is not supplied then a default string number is calculated based on the current tablature stave type and the guitar tab fingering options (specified on the Note Input page of File $>$ Preferences). Returns the Note object created (to get the NoteRest containing the note, use Note. ParentNoteRest).

When adding very short notes to tuplets, Sibelius may be unable to find a legal place for the note in the bar. Should this happen, Sibelius will return null. You should therefore check for a valid object if there is any likelihood that this situation may arise in your code.

> Q" If you add a note to a score that intersects an existing tuplet, Sibelius will try to snap the note to the closest sensible place within that tuplet. However, you are advised to use Tuplet.AddNote() for this purpose as it is void of any ambiguity.

AddStaffAbove (ossia, [start bar number[, end bar number[, start pos[, end pos]]]]) Adds a new staff above the staff. Set ossia to True to create an ossia (small) staff. The other, optional parameters determine where the staff should be visible: if you do not specify a start bar number, the staff will be visible from the start of the score; if you do not specify an end bar number, the staff will be visible to the end of the score. If you specify a start and/or end bar number, the staff will be hidden outside that range by way of an instrument change to the No instrument (hidden) instrument type. start pos and end pos represent the rhythmic position within the start bar number and end bar number respectively, and if not specified, start pos will default to the start of the bar, and end pos will default to the end of the bar. Returns the staff created, or null if the call fails.

AddStaffBelow (ossia, [start bar number[, end bar number[, start pos[, end pos]]]]) Adds a new staff below the staff. See AddStaffAbove () above for details.

AddSymbol (pos, symbol index or name) Adds a symbol to staff (please see the documentation in Bar object below).
CurrentKeySignature (bar number) Returns a KeySignature valid at the bar number passed.
NthBar ( $n$ ) Returns the $n$th bar in the staff, counting from 1.
ResetSpaceAroundStaff (above, below[, from bar[, to bar]]) Does the equivalent of Layout > Reset Space Above Staff and/or Reset Space Below Staff for the given range of bars in a staff. Set above to True to reset the space above the staff, and below to True to reset the space below the staff. If from bar is not specified, Sibelius sets it to $\mathbf{1}$; if to bar is not specified, Sibelius sets it to the last bar of the score.

ScaleFactorChange (StaveScaleFactor, startBarNum, endBarNum) Equivalent to setting the Staff Size property within the Inspector. Valid StaveScaleFactor types include: NormalSize (0), MediumSize (1), SmallSize (2), and ExtraSmallSize (3). It requires a valid range specified by startBarNum and endBarNum.

SetSound (styleID[, set SoundStage]) Changes the initial playback sound of this staff to be the default sound for the given default instrument styleID. For a complete list of default instrument style IDs in Sibelius, see Instrument Types. If the optional Boolean parameter is set to False, then the SoundStage information (volume, pan and distance) for this staff will be unchanged. If it is omitted or set to True, then the SoundStage information will be set to the default for the new sound.

SetSoundID (soundID) Changes the initial playback sound of this staff to the given soundID.
SoundIDAtPosition ([bar number, [position,[nth repeat]]]) Returns a new SoundInfo object at the specified bar number, at the specified rhythmic position in the bar (for example 256 for the second quarter note position), as if played through at the nth repeat (for example 2 for the second repeat). If no bar number is specified, the information returned will refer to bar 1. If no position is specified, the information will refer to the start of the bar. If $n$th repeat is not specified, the information returned will refer to the first pass through the score.

Staff [array element] Returns the $n$th bar (counting from 1) for example Staff [1].

## Variables

BankHigh Controls MIDI controller 0, used to select the "coarse" bank number for this stave, and corresponding to the Mixer control of the same name. The range is $0-127$, or -1 if you don't want to send this controller message at the start of playback. Note that not all MIDI devices support multiple banks (read/write).

BankLow Controls MIDI controller 32, used to select the "fine" bank number for this stave, and corresponding to the Mixer control of the same name. The range is $0-127$, or -1 if you don't want to send this controller message at the start of playback. Note that not all MIDI devices support multiple banks (read/write).

BarCount Number of bars in the staff (read only).
BarNumber The bar number of this bar. This is the internal bar number, which always runs consecutively from 1 (read only).
Channel The MIDI channel number of this staff, numbered $1-16$ (read/write).
Distance The reverb "distance" of this staff, corresponding to the control of the same name in the Mixer. This is a percentage, used to scale the overall reverb settings from the Performance dialog (read/write).

FullinstrumentName Gives the full instrument name of the staff, empty for an unnamed staff (read/write).
FullinstrumentNameWithFormatting Gives the full instrument name of the staff including any changes of font or style, if any (read/write).

FullStaffName Returns the initial full staff name (read/write).
FullStaffNameWithFormatting Returns the initial full staff name including any changes of font or style, if any (read/write).
NumStavesInSameInstrument The number of staves belonging to the default instrument from which this staff was created (read only).

InitialClefStyle The name of the initial clef on a staff, depending on the state of Notes > Transposing Score (read only).
InitialClefStyleId The style identifier of the initial clef on a staff, depending on the state of Notes > Transposing Score (read only).

InitialInstrumentType Returns an InstrumentType object for the instrument type at the start of the staff.
InitialKeySignature Returns the KeySignature object at the start of this staff (read only).
InitialStyleId Returns the style identifier of the staff (read only). To create an instrument from such an ID, pass the style as the first argument to Score. CreateInstrument. For a complete list of all the default instrument names in Sibelius, see Instrument Types.

InstrumentName Gives the full instrument name of the staff in the form that is displayed on the Instruments and Staves dialog in Sibelius (read only). For an unnamed stave, this will be "[Piano]" for example, where Piano is the default instrument name of the stave (see below). To get the internal name (which will be empty for unnamed staves), use the read/write variables
FullInstrumentName or ShortInstrumentName instead.
IsSystemStaff True or False depending on whether this staff is a system staff or not (read only).
IsVocalStaff Returns True if the instrument type used by the staff has the Vocal staff option switched on, meaning that the default positions of dynamics should be above the staff rather than below (read only).

MuteMode Specifies whether or not this stave will play back. Corresponds to the mute button in the Mixer. The supported values are defined as global constants (see Global Constants) and are Muted, HalfMuted and NotMuted (read/write).

Pan The MIDI stereo panning position of this staff (corresponding to the pan control in the Mixer). Permissible values are -100 to 100 , with positive values being to the right and negative to the left (read/write).

ParentScore Returns the staff's parent Score object (read only).
ShortInstrumentName Gives the short instrument name of the staff, empty for an unnamed staff (read/write).
ShortInstrumentNameWithFormatting Gives the short instrument name of the staff including any changes of font or style, if any (read/write).

ShortStaffname Returns the initial short staff name (read/write).

ShortStaffNameWithFormatting Returns the initial short staff name including any changes of font or style, if any (read/write).
ShowInFocusOnStaves If True then this staff will be shown when Layout > Focus on Staves is enabled (see also Score. FocusOnStaves). This variable cannot be set to False unless it is also True for at least one other staff in the score (read/write).

Solo True or False depending on whether this staff plays back in "solo" mode, corresponding to the Mixer button of the same name (read/write).

SoundIdOverrideIfAny Returns a string containing the sound ID override set in the mixer for the staff. If no override has been set, an empty string is returned (read only).

Small True if the staff is small (such as an ossia staff), False if it is normal sized (read/write).
Staffnum Returns the number of this stave, counting from 1 at the top of the currently-viewed part. Returns 0 for SystemStaff objects (read only).

Volume The overall MIDI volume of this staff, corresponding to its fader in the Mixer. Permissible values are $0-127$ (read/write).

## Syllabifier

Acts as a wrapper around Sibelius's internal Syllabification engine, exposing its functionality to ManuScript.

## Methods

AbbreviateUsingApostrophe (useApostrophe) When the abbreviate flag is set to True when calling Syllabify, Sibelius will replace vowels that have been combined with the previous syllable with an apostrophe if this option is enabled-for example Vege-ta-bles vs Veg'-ta-bles. Calling this method will cause the syllabification engine to recalculate its result if necessary.

GetNthSyllable ( $n$ ) Once a string has been syllabified by calling the Syllabify method, you can use this method to return each individual syllable as a string

NthSyllableEndsWord ( $n$ ) Once a string has been syllabified by calling the Syllabify method, you can use this method to find out whether each syllable occurs at the end of a word

Syllabify (textToSyllabify[, language[, abbreviate $=$ False]]) Breaks a string down into its syllabic components, returning the number of syllables in the resultant syllabification, or 0 if an error has occurred. The rules of the specified language will be used, and you may legally supply either a language ID, or the localized language name. To get the individual syllables, you should call the GetNthSyllable and NthSyllableEndsWord methods documented below.

If the language argument is omitted, Sibelius will attempt to automatically identify the language of the text. If this is not possible, or if an unrecognised language ID or name has been supplied, 0 will be returned.

When abbreviate is True, each ambiguous word in the string will be syllabified using the minimal number of syllables. For example, syllabifying "Everybody likes vegetables" would return "Eve-ry-bod-y likes vege-ta-bles" with this flag set to True, otherwise "E-ve-ry-bod-y likes veg-e-ta-bles".

## Variables

AbbreviateUsingApostrophe Returns True/False depending on whether the syllabification engine is set to abbreviate combined syllables with an apostrophe (read only - call method with same name for write access).

AvailableLanguageIds Returns an array containing a list of the available syllabification languages as three-letter non-translatable IDs - such as ENG (English), GER (German), LAT (Latin). These IDs are identical in all localized versions of Sibelius (read only).

AvailableLanguages Returns an array containing a list of the available syllabification languages as localized strings (read only).
NumberOfSyllables Returns the number of syllables in the hyphenated string generated by calling the Syllabify method (read only).

SyllabifiedString Returns the resultant hyphenated string generated by calling the Syllabify method (read only).

## Symbolltem and SystemSymbolltem

Derived from a BarObject. For system symbols (such as symbols belonging to the system staff, retrieved with for each on the SystemStaff object), the type of symbol objects is SystemSymbolItem, not SymbolItem.

## Methods

None.

## Variables

Index The index of this symbol in the list of symbols. This corresponds to its position in the Create $>$ Symbol dialog, counting from zero left-to-right and top-to-bottom (read only).

Name The name of this symbol. May be translated in non-English language versions of Sibelius (read only).
Size The draw size of the symbol, corresponding to the four available options in the Symbols dialog in Sibelius. The four available values are NormalSize, CueSize, GraceNoteSize and CueGraceNoteSize, all defined as global constants (read/write).

## SystemObjectPositions

Accessed from a Score object. Corresponds to the settings in House Style > System Object Positions.

## Methods

GetNthStaffShowsSystemObjects (staffNum) Returns True if the given staff number staffNum (relative to the current part) is showing system objects above it, otherwise False.

SetNthStaffshowsSystemObjects (staffNum, show) Tells the staff with the given staff number staffNum (relative to the current part) either to show or not show system objects above it. This will have no effect if you pass in the top staff in the part, or if the maximum number of staves allowed to show system objects has already been met.

Clear ([removeBelowBottomStaff]) Allows you to clear all the system object positions (apart from the compulsory one above the top staff) in a single operation; set the optional Boolean parameter removeBelowBottomStaff to True to also clear the Below bottom staff system object position.

## Variables

NumStavesShowingSystemObjects Returns the current number of staves showing system object positions (read only).
ShowSystemObjectsBelowBottomStaff Returns True if system objects should show below the bottom staff, otherwise False (read/write).

## SystemStaff, Staff, Selection, Bar and, all BarObject-derived Objects

## Variables

IsALine Returns true if the object is a line object. (Note that this is a variable, not a method, unlike the IsObject () method for all objects.)

Type A string giving the name of the type of an object. The strings for the first 4 types above are "SystemStave", "Stave", "MusicSelectionList", and "Bar". Note that this variable is also a member of all objects that occur in bars.

## SystemStaff

There is one Systemstaff object per score. The SystemStaff contains objects which apply to all staves, such as Special Barlines and text using a system text style. Unlike normal staves, the SystemStaff does not appear in the score itself. As such, most of the variables and methods supported for Staff objects are not available on a SystemStaff. Those that are supported by SystemStaff are as follows.

## Methods

CurrentKeySignature (bar number) Returns a KeySignature valid at the bar number passed.
CurrentTimeSignature (bar number) Returns a TimeSignature valid at the bar number passed.
NthBar ( $n$ ) Returns the $n$th bar in the staff, counting from 1.
SystemStaff [array element] Returns the $n$th bar (counting from 1) for example SystemStaff [1].

## Variables

BarCount Number of bars in the staff (read only).
InitialKeySignature Returns the KeySignature object at the start of this staff (read only).
IsSystemStaff Returns True for a SystemStaff (read only).

## Text and SystemTextItem

Derived from a BarObject. For system text (such as text belonging to the system staff, retrieved with for each on the SystemStaff object), the type of text objects is SystemTextItem, not Text.

## Methods

None.

## Variables

InitialFontName Returns the font name used at the start of a styled string (read only).
InitialFontSize Returns the font size in SUs (1/32nd space) if the text object is of a single size, or -1 if the text object uses multiple font sizes (read only).

JumpAtEndOfBar Returns True if the system text object has Jump at bar end (in the Playback panel of the Inspector) set, otherwise False. Always returns False for staff text objects (read/write).

StyleAsText The text style name (read/write).
StyleId The identifier of the text style of this piece of text (read/write).
Text The text as a string (read/write).
TextWithFormatting Returns an array containing the various changes of font or style (if any) within the string in a new element (read only). For example, "This text is $\backslash B \backslash b o l d \backslash b \backslash$, and this is $\backslash \ l i t a l i c l i l "$ would return an array with eight elements containing the following data:

```
arr[0] = "This text is "
arr[1] = "\B\"
arr[2] = "bold"
arr[3] = "\b\"
arr[4] = ", and this is "
arr[5] = "\I\"
arr[6] = "italic"
arr[7] = "\i\"
```

TextWithFormattingAsString The text including any changes of font or style (read only).

## TimeSignature

## Derived from a BarObject.

## Methods

None.

## Variables

AllowCautionary Returns True if the time signature is set to show a cautionary at the end of the previous system, if it occurs at the start of a system (read/write).

Denominator The time signature's bottom number (read only).
Numerator The time signature's top number (read only).
Text The time signature as text. You can use this to detect common time and alla breve time signatures by comparing it to the global constants CommonTimeString and AllaBreveTimeString, which define the Unicode characters used by these symbols. Other time signatures will be of the form " $4 \backslash n 4$ " (read only).

## TreeNode

These are used internally by ManuScript to implement arrays and hashes (returned with the CreateArray and CreateHash methods), and to represent global data (defined in the plug-in editor). Each TreeNode can contain a label, a piece of data and a list of "children," which are also TreeNodes. Normally, any access to a TreeNode object will access the data that is held, so that you don't need to know anything about them, but there are also some extra variables and methods that may be useful in some circumstances. These can be called on any array, hash or global variable, and on any member of such a structure.

## Methods

WriteToString Returns a string that represents the structure of this TreeNode object. In this representation, the data of a TreeNode is surrounded by double quotes and the label is not. Note that a label need not be defined. Any children of the TreeNode (also TreeNode objects themselves) are contained within curly braces $\{$ and $\}$. To obtain child TreeNodes, use the normal array operator, as described in the documentation for arrays and hashes.

## Variables

Label The label of this TreeNode.
NumChildren The number of child TreeNodes belonging to this TreeNode object.

## Tuplet

Derived from a BarObject.

## Methods

AddNestedTuplet (posInTuplet, left, right, unit[, style[, bracket[,fullDuration]]]]) Nests a new tuplet bracket within an existing tuplet at a position relative to the duration and scale-factor of the existing tuplet. The left and right parameters specify the ratio of the new tuplet, for example 3 (left) in the time of 2 (right). The unit parameter specifies the note value (in $1 / 256$ th quarters) on which the tuplet should be based. For example, if you wish to create an eighth note (quaver) triplet group, you would use the value 128. The optional style and bracket parameters take one of the pre-defined constants that affect the visual appearance of the created tuplet; see Global Constants. If fullDuration is true, the bracket of the tuplet will span the entire duration of the tuplet. Returns the Tuplet object created.

[^3]AddNote (posInTuplet, pitch, duration[, tied[, diatonic pitch[, string number]]]]) Adds a note to an existing tuplet, adopting the same voice number as used by the tuplet itself. Please note that posInTuplet is relative to the duration and scale-factor of the tuplet bracket itself. Therefore, if you wanted to add a quarter note/crotchet to the second beat of a quarter note/crotchet triplet, you would simply use the value 256 , not 341 !
utils.SplitTuplet (tuplet, splitpoint) Split the Tuplet object tuplet at the specified splitpoint, which is a number in relation to the tuplet's parent bar. It then splits a nest of tuplets at that point in the bar. This method is provided by the utils.plg-see Utils.

## Variables

Bracket The bracket type of the tuplet (such as. none, auto; see Global Constants).
FullDuration True if the bracket of the tuplet spans its entire duration.
Left The left side of the tuplet, for example 3 in 3:2 (read only).
ParentTupletIfAny If the tuplet intersects a tuplet, the innermost Tuplet object at that point in the score is returned. Otherwise, null is returned (read only).

PlayedDuration The true rhythmic duration of the tuplet, for example for quarter-note (crotchet) triplet this would be the duration of a minim (read only).

PositionInTuplet Returns the position of the tuplet relative to the duration and scale-factor of its parent tuplet. If the tuplet does not intersect a tuplet, its position within the parent Bar is returned as usual (read only).

Right The right side of the tuplet, for example 2 in 3:2 (read only).
Style The style of the tuplet (for example, number, ratio, ratio + note; see Global Constants).
Text The text shown above the tuplet (read only).
Unit The unit used for the tuplet, for example 256 for a triplet of quarter notes (read only).

## Utils

Sibelius installs a plug-in called utils.plg that contains a set of useful and common methods that can be called directly by other plug-ins. It is not intended to be run as a plug-in in its own right, so does not appear in the Plug-ins menu.

## Methods

The methods available via utils.plg are as follows:
utils.AbsoluteValue (value) Returns the absolute value of a number, that is its numerical value without regard to its sign.
utils. AddFractions $(x, y)$ Adds two fractions $x$ and $y$, passed in as ManuScript arrays. Returns an array with the result of the addition.
utils.BinaryString ( $x$ ) Returns a binary string (such as "101010") equivalent to the number $x$.
utils.bwAND ( $\mathbf{x}, \mathrm{y}$ ) Equivalent to the C++ bitwise AND $(\&)$ operator. For example, utils.bwand $(129,1)$ is equal to 1.
utils. $\operatorname{bwOR}(\mathbf{x}, \mathrm{y})$ Equivalent to the C++ bitwise inclusive $\operatorname{OR}(\mathrm{I})$ operator. For example, utils.bwOR (64,4) is equal to 68.
utils.bwXOR ( $\mathbf{x}, \mathbf{y}$ ) Equivalent to the C++ bitwise exclusive $\operatorname{XOR}(\wedge)$ operator. For example, utils.bwXOR (4,6) is equal to 2.
utils.CapableOfDeletion() Returns True if the object can be deleted using Delete (), which is determined by checking Sibelius's version number.
utils.CaseInsensitiveComparison(s1, s2) Returns True if the two strings $s 1$ and s 2 match, ignoring case.
utils.CastToBool (x) Returns the variable x explicitly cast as a Boolean.
utils.CastToInt(x) Returns the variable $x$ explicitly cast as an integer.
utils.CastToStr (x) Returns the variable x explicitly cast as a string.
utils. CombineArraysOfBool (arr1, arr2) Concatenates two arrays containing Boolean values and returns the result.
utils.CombineArraysOfInt (arr1, arr2) Concatenates two arrays containing integral values and returns the result.
utils. CombineArraysOfString (arr1, arr2) Concatenates two arrays containing string values and returns the result.
utils.CopyTextFile (source, dest) Copies an existing text file from one location to another, returning True if successful.
utils.CreateArrayBlanket (value, size) Returns an array with size elements, each containing a blanket value specified by the first parameter.
utils.DeleteStaff (score, nth staff, retain selection) Deletes an entire staff and its content from a given score, returning True if successful. If retain selection is True, Sibelius will ensure any item(s) that were selected prior to the staff's deletion are still selected.
utils. DenaryValue ( $x$ ) Returns a number in base 10 equivalent to binary number $x$, which must be provided as a string.
utils. DivideFractions $(x, y)$ Divides fraction $x$ by fraction $y$, passed in as ManuScript arrays. Returns an array with the result of the division.
utils.ExtractFileName (filename) Returns just the filename portion of a string filename containing both a path and a filename.
utils. Format (str, [val1, val2, val3 ...]) Provides a simple means of replacing human-readable data types in a string. Each successive instance of \%s in str is replaced with the value of the next remaining unused argument. for example s = utils.Format ("The \%s brown \%s jumps \%s the lazy \%s", "quick", "fox", "over", "dog");
utils.FormatTime (ms) Formats a time, given in milliseconds, to a human-readable string using the format mm'ss.z (where z is centiseconds).
utils.FractionAsDecimal ( $x$ ) Returns the decimal equivalent of the fraction $x$, which is passed in as an array.
utils.FractionDenominator ( $x$ ) Returns the denominator of fraction $x$, which is passed in as an array.
utils.FractionNumerator ( $x$ ) Returns the numerator of fraction $x$, which is passed in as an array.
utils.GetAppDir () Returns the path of the Sibelius executable as a string.
utils.GetArrayIndex (arr, value) Returns the index of value in the array arr, or -1 if it doesn't exist in the array.
utils. GetBits ( $\mathbf{x}$ ) Returns an array containing the list of powers of two whose cumulative sum equates to the value of x .
utils.GetGlobalApplicationDataDir () Returns the path of the system's global application data area as a string.
utils.GetLocationTime (score, barNum, position) Returns the precise time (in milliseconds) of a given location in a score. The position should be local to the start of the bar number you have supplied. Use the utils library to achieve this if your plug-in needs to be backwards compatible with Sibelius 4; otherwise call the Score object's function with the same name.
utils. GetMillisecondsFromTime (time) If you pass in a time expressed in milliseconds (one minute being 60,000), this function returns the milliseconds portion of the number (in this case 60,000 modulus $1000=0$ ).
utils. GetMinutesFromTime (time) If you pass in a time expressed in milliseconds, this function returns the minutes portion of the number (for example if time $=120,262$ milliseconds, this function returns 2 ).
utils.GetObjectTime (score, obj) Returns the precise time (in milliseconds) that the object obj occurs from the start of a given score, taking into account tempo changes, performance markings and any other events in the score that have an effect on playback. Use this method to achieve this if your plug-in needs to be backwards compatible with Sibelius 4; otherwise use the Time property of the BarObject object whose time you wish to determine.
utils.GetPluginId(plug-in) This enables you to identify a plug-in by entering the line of code PluginUniqueID $=$
"someUniqueId"; in a plug-in's Initialize method. When you pass a Plugin object to this function, it scans the plug-in's code and returns its unique ID if it has one, otherwise an empty string.
utils.GetSibeliusPluginsFolder () This is a wrapper around the deprecated GetPluginsFolder () function, and returns the path of the Plugins folder.
utils.GetSibMajorVersion() Returns the major version number of Sibelius.
utils. GreatestCommonDivisor ( $m, n$ ) Returns the greatest common divisor of two non-zero integers, that is the largest positive integer that divides both numbers without remainder.
utils.IsInArray (arr, value) Returns True if value exists in the array arr.
utils.IsNumeric (str[, integer only]) Returns True if the string str is numeric. Set the optional Boolean parameter integer only to True if you want the method to only return True if $s t r$ is an integer (so that you can disallow floating point numbers).
utils.LowerCase (str) Returns the ANSI string str in lowercase.
utils. MakeFraction $(x, y)$ Creates a fraction with $x$ as the numerator and $y$ as the denominator. The fraction is returned as a normal ManuScript array. (Manipulating fractions means you never have to worry about rounding errors.)
utils.max $(x, y)$ Returns the greater of two numbers.
utils.min $(x, y)$ Returns the lesser of two numbers.
utils.MultiplyFractions $(x, y)$ Multiplies fraction $y$ by fraction $x$, passed in as ManuScript arrays. Returns an array with the result of the multiplication.
utils. PatternCount (pattern, str) Returns the number of times the substring pattern exists in str.
utils. Pos (subStr, str) Returns the zero-based position of the first instance of the sub-string subStr in str, or -1 if it isn't found.
utils. PosReverse (subStr, str) Returns the zero-based position of the last instance of the sub-string subStr in str, or -1 if it isn't found.
utils.RaisePower $(x, y)$ Raises $x$ to the $y$ th power, where $y$ is a positive integer.
utils. Replace (inStr, toFind, replaceWith, replaceAll) Replaces a sub-string in a string with a new value. It looks for toFind in the string inStr, and if it finds it, replaces it with replaceWith. If the Boolean replaceAll is False, it only changes the first instance found; if it's True, it replaces all instances.
utils.ReverseArrayOfBool (arr) Reverses the order of the elements in an array of Booleans.
utils.ReverseArrayOfInt (arr) Reverses the order of the elements in an array of integers.
utils.ReverseArrayOfString (arr) Reverses the order of the elements in an array of strings.
utils. RoundToNDecimalPlaces (number, precision) Returns a string containing the number rounded to precision decimal places. The method handles the input as a string, in order to avoid rounding errors which would otherwise spoil results beyond the tenth decimal place or so.
utils.SetDefaultIfNotInArray (value, arr, Defaultindex) Scans the array arr for the value specified by the first parameter. Value is returned if it exists in the array, otherwise, arr [DefaultIndex].
utils.shl $(x, y)$ Bitwise left-shift. Shifts the value x left by y bits. Equivalent to $\mathrm{C}++$ << operator.
utils. $\operatorname{shr}(x, y)$ Bitwise right-shift. Shifts the value x right by y bits. Equivalent to $\mathrm{C}++\gg$ operator.
utils. SortArray (arr, show progress) Sorts the array arr using a case-insensitive alphabetic sort. Set show progress to True to see a progress bar while the sort is carried out, or set it to False if you don't want to see a progress bar.
utils. SortArrayCustom (arr, show progress, plug-in name, method) Sorts the array arr using a custom sort order routine method, which must be passed into this method. plug-in name is the name of the plug-in that contains the sort order routine method. You can write your own sort order routine: it must be a method that takes two strings ( $\operatorname{str} A$ and $\operatorname{str} B$ ) and returns 1 or 0 based on the results of the comparison.
utils.SortArrayNumeric (arr, show progress) Sorts the array arr in ascending numeric order. Set show progress to True to see a progress bar while the sort is carried out, or set it to False if you don't want to see a progress bar.
utils.SplitTuplet (tuplet, splitpoint) Split the Tuplet object tuplet at the specified splitpoint, which is a number in relation to the tuplet's parent bar. It then splits a nest of tuplets at that point in the bar.
utils.StartComponentManager (componentName, callbackFunc) Returns an array of filenames (strings) found on the system inside a folder with a given name, following the same rules of precedence as Sibelius's internal component manager. Files in the user's application data area take priority over those in the global application data area, followed lastly by those in the Sibelius's application directory itself.
callbackFunc should point to a function in the calling script that scans a supplied directory for files with a specific extension.

Such a function might look something like this:

```
GetFooFiles(dir) { // This is the function signature
    components = CreateArray();
    for each FOO file in dir {
    components[components.NumChildren] = file.NameWithExt;
    }
    return(components);
}
```

In the scenario above, the call to start the component manager would look like this (where "Foo Files" is the name of the directory containing your files):

```
files = utils.StartComponentManager("Foo Files",
"myPlugin.GetFooFiles");
```

utils. SubtractFractions $(x, y)$ Subtracts fraction $y$ from fraction $x$, passed in as ManuScript arrays. Returns an array with the result of the subtraction.
utils. UpperCase (str) Returns the ANSI string str in uppercase.

## VersionHistory

Each Score object has a VersionHistory object (obtained by way of the score. GetVersions () method), which in turn provides a list of Version objects. Each Version object represents a specific version, and also provides a list of VersionComment objects, which represent the per-version comments (as opposed to bar-attached comments, which are represented to ManuScript as Comment objects, derived from BarObject objects).

## Methods

AddVersion ([name[, comment]]) Adds a new Version object and returns it if successful (or null if not), with an optional name and comment for the version.

DeleteNthVersion ( $n$ ) Deletes the $n$th Version object, returning True if successful.
GetNthVersion ( $n$ ) Returns the $n$th Version object.

## Variables

NumChildren Returns the number of versions in the score's VersionHistory object.

## Version

Accessed via a Score object's VersionHistory object.

## Methods

AddComment (text) Adds a new comment with the specified text, and returns the VersionComment object created.
Close () Closes all views of the version that are currently open in Sibelius, returning True if it has actually closed anything.
GetNthComment ( $n$ ) Gets the $n$th comment as a VersionComment object, or returns null if the index is out of range.
DeleteNthComment ( $n$ ) Deletes the $n$th comment, returning True if successful, or null if the index is out of range.
OpenAndReturnScore () Opens the specified version in Sibelius (if it's not already open) and returns its Score object.

## Variables

EndDate Returns a DateTime object representing the version's end date (read only). IsOpen returns True if the version is currently open in Sibelius (read only).

Name Returns the name of the version (read/write).
NumComments Returns the number of comments in the version (read only).
StartDate Returns a DateTime object representing the version's start date (read only).

## VersionComment

Accessed via Version objects.

## Methods

None.

## Variables

Text Returns or changes the text of the comment, and this cannot be undone (read/write).
TimeStamp Returns a DateTime object representing the time at which the comment was created.
UserName Returns the name of the user who created the comment (read only).

## Command IDs

The following is a complete list of Command IDs that can be passed to the Sibelius.Execute() method.

CommandID Command Name
128th_note 128th note
16_tremolos 16 tremolos
16th_note 16th note (semiquaver)
256th_note 256th note
2_tremolos 2 tremolos
32_tremolos 32 tremolos
32nd_note 32nd note (demisemiquaver)
4_tremolos 4 tremolos
512th_note 512th note
64th_note 64th note (hemidemisemiquaver)
8_tremolos 8 tremolos
8th_note Eighth note (quaver)
accent Accent
accessibility_preferences Accessibility Settings
acciacatura Acciaccatura
add_2nd_above Add interval 2nd above
add_3rd_above Add interval 3rd above
add_4th_above Add interval 4th above
add_5th_above Add interval 5th above
add_6th_above Add interval 6th above
add_7th_above Add interval 7th above
add_9th_above Add interval 9th above
add_dynamic_guitar_staff Add dynamic guitar staves
add_bar_at_end Bar at End
add_multiple_bars Other Bar
add_octave_above Add interval an octave above
add_ossia_above Add Ossia Above
add_ossia_below Add Ossia Below
add_single_bar Single Bar
add_stave_above Add Staff Above
add_stave_below Add Staff Below
add_unison Add unison
add_video Add Video
advance_caret Advance Caret
advanced_filter Advanced Filter
align_horizontally Align in a Column
align_staves Align Staves
align_vertically Align in a Row
annotate Annotate
append_score Append Score
appoggiatura Appoggiatura
arpeggio Arpeggio
arpeggio_down Arpeggio Down
arpeggio_up Arpeggio Up
arrange Arrange
auto_breaks Auto Breaks
bar_number_change Bar Number Change
bar_rest Bar Rest
barline_between_staves Between Staves Barline
barline_dashed Dashed Barline
barline_double Double Barline
barline_end_repeat End Repeat Barline
barline_final Final Barline
barline_invisible Invisible Barline
barline_normal Normal Barline
barline_short Short Barline
barline_start_repeat Start Repeat Barline
barline_ticks Tick Barline
brace Brace
bracket Bracket
bracket_accidental Bracket accidental
bracket_notehead Bracket notehead
buzz_roll Buzz roll (Z on stem)
calibrate_live_tempo Calibrate Live Tempo
cancel_stop_selectnone Cancel/Stop/Select None

| cascade Cascade <br> change_appearance_current_part Change Current Part Appearance | create_pitch_d Create D <br> create_pitch_e Create E <br> create_pitch_f Create F |
| :---: | :---: |
| change_appearance_of_all_parts Change Appearance of All Parts | create_pitch_g Create G |
| change_appearance_of_open_parts Change Appearance of | create_title_page Title Page |
| Open Parts | cross_stave_move_down Move Down a Staff |
| change_tie_style Next tie style | cross_stave_move_up Move Up a Staff |
| chord_diagram_toggle Add/Remove Chord Diagram | cross_stave_reset Move to Original Staff |
| chord_symbols Chord Symbol | custom_articulation1 Custom Articulation 1 |
| chord_text_root_toggle Add/Remove Chord Text Root | custom_articulation2 Custom Articulation 2 |
| chord_text_toggle Add/Remove Chord Text | custom_articulation3 Custom Articulation 3 |
| clear_live_tempo Clear Live Tempo | cut Cut |
| clef_dialog Show Clef dialog | decrease_instrument_stave_size Decrease Instrument Staff |
| clefs Clef | Size |
| close_all Close All | decrease_spacing Decrease Spacing |
| close_tab Close Tab | decrease_spacing_a_lot Decrease Spacing a Lot |
| close_window Close Window | default_positions Default Positions |
| color Color | delete Delete |
| color_note_none Note Colors None | delete_bar Delete Bar |
| color_note_voices Voice Colors | delete_current_part_or_subset Delete current part or subset |
| color_out_of_range_notes Notes out of Range | delete_part Delete Part |
| command_search Activate Command Search | document_setup Document Setup |
| comment Comment | doit Doit |
| compare_window Compare | double_dot Double dot |
| configure_live_tempo Configure Live Tempo | double_flat Double flat |
| consecutive_rehearsal_mark Rehearsal Mark (consecutive) | double_sharp Double sharp |
| copy Copy | double_whole_note Double whole note (breve) |
| copy_and_change_instrument_in_part Copy and Change In- | downbow Downbow |
| strument | duplet Duplet |
| copy_as_idea Capture Idea | edit_noteheads Notehead type gallery |
| copy_part_layout Copy Part Layout | edit_all_fonts Edit All Fonts |
| create_highlight Highlight | edit_arrange_styles Edit Arrange Styles |
| create_hitpoint Create Hit Point | edit_chord_diagram Edit Chord Diagram |
| create_live_tempo_tap_point Create Live Tempo Tap Point | edit_chord_symbols Edit Chord Symbols |
| create_new_part Create New Part | edit_hitpoints Hit points |
| create_pitch_a Create A | edit_instruments Edit Instruments |
| create_pitch_b Create B | edit_lines Edit Lines |
| create_pitch_c Create C |  |
| Command IDs | 97 |

edit_plugins Edit Plug-ins
edit_staves_in_part Edit Staves in Part
edit_symbols Edit Symbols
edit_text Edit Text
edit_text_styles Edit Text Styles
edit_worksheets Edit Worksheets
end_beam End beam
end_line End line
end_slur End slur
engraving_rules Engraving Rules
equivalent_chord_text Equivalent Chord Text
exit_program Exit
export_as_audio Export Audio
export_as_graphic Export Graphics
export_as_manuscript_paper Export Manuscript Paper
export_as_midi Export MIDI File
export_as_musicxml Export MusicXML
export_as_older_version Export Previous Sibelius Version
export_as_pdf Export PDF
export_as_sib_student Export Sibelius Student file
export_as_video Export Video File
export_house_style Export House Style
extend_selection_down Extend Selection Down
extend_selection_left Extend Selection Left
extend_selection_right Extend Selection Right
extend_selection_to_bottom_staff Extend Selection to Bottom Staff
extend_selection_to_end_of_bar Extend Selection to End of Bar
extend_selection_to_start_of_bar Extend Selection to Start of Bar
extend_selection_up Extend Selection Up
extend_selection_to_top_of_staff Extend Selection to Top of Staff
extract_parts Extract Parts
fall Fall
fast_forward Fast-forward
feathered_beam_accel Feathered beam accel.
feathered_beam_rit Feathered beam rit.
filter_2nd_note 2nd from Top
filter_2nd_note_for_deletion Filter 2nd from Top (For Deletion)
filter_2nd_or_single_note_for_deletion Filter 2nd Note or Single Notes (For Deletion)
filter_2nd_pr_single_note Filter 2nd Note or Single Notes (For Copying)
filter_3rd_note 3rd from Top
filter_3rd_note_for_deletion Filter 3rd from Top (For Deletion)
filter_3rd_or_single_note Filter 3rd Note or Single Notes (For Copying)
filter_3rd_or_single_note_for_deletion Filter 3rd Note or Single Notes (For Deletion)
filter_all_barlines Filter All Barlines
filter_bar_numbers Filter Bar Numbers
filter_bottom_note Bottom Note
filter_bottom_note_for_deletion Filter Bottom Note (For Deletion)
filter_bottom_or_single_note Filter Bottom Note or Single Notes (For Copying)
filter_bottom_or_single_note_for_deletion Filter Bottom Note or Single Notes (For Deletion)
filter_chord_symbols Filter Chord Symbols
filter_comments Filter Comment items
filter_dynamics Filter Dynamics
filter_expression_text Filter Expression Text
filter_grace_notes Filter Grace Notes
filter_hairpins Filter Hairpins
filter_hidden_objects Filter Hidden Objects
filter_instrument_changes Filter Instrument Changes
filter_lyrics Filter Lyrics
filter_nonspecial_barlines Filter Non-Special Barlines
filter_non_tied_notes Filter notes that are not tied or that start a tie
filter_notes_and_chords Filter Notes and Chords
filter_page_numbers Filter Page Numbers
filter_pedal_lines Filter Pedal Lines
filter_player1 Filter Player 1 (For Deletion)
filter_player2 Filter Player 2 (For Deletion)
filter_rehearsal_marks Filter Rehearsal Marks
filter_repeat_bars Filter Repeat Bars
filter_rests Filter Rests
filter_slurs Filter Slurs
filter_special_barlines Filter Special Barlines
filter_staff_text Filter Staff Text
filter_symbols Filter Symbols
filter_system_text Filter System Text
filter_technique_text Filter Technique Text
filter_top_note Top Note
filter_top_note_for_deletion Filter Top Note (For Deletion)
filter_top_or_single_note Filter Top Note or Single Notes (For Copying)
filter_top_or_single_note_for_deletion Filter Top Note or Single Notes (For Deletion)
filter_tuplets Filter Tuplets
filter_voice1 Filter Voice 1
filter_voice1_only Filter Voice 1 Only
filter_voice2 Filter Voice 2
filter_voice2_only Filter Voice 2 Only
filter_voice3 Filter Voice 3
filter_voice3_only Filter Voice 3 Only
filter_voice4 Filter Voice 4
filter_voice4_only Filter Voice 4 Only
find Find
find_next Find Next
find_next_collision Find Next Collision
find_next_comment Find Next Comment
find_previous_collision Find Previous Collision
find_previous_comment Find Previous Comment
flat Flat
flexitime_input Flexi-time Input
flexitime_options Flexi-time Options
flip Flip
focus_on_staves Focus on Staves
freeze_magnetic_layout_positions Freeze Magnetic Layout Positions
fretboard_window Fretboard
full_screen Full Screen
goto_bar Go to Bar
goto_page Go to Page
goto_playback_line Go to Playback Line
goto_selection_end Go to selection end
goto_selection_start Go to selection start graphic_adjust_color Adjust Colours
graphic_change_link_source Change Link Source
graphic_flip_horizontal Flip Horizontal
graphic_flip_vertical Flip Vertical
graphic_rotate_180 Rotate 180 degrees
graphic_rotate_270 Rotate 90 degrees CCW
graphic_rotate_90 Rotate 90 degrees CW
graphics Graphic
guitar_scale_diagram Guitar Scale Diagram
half_note Half note (minim)
harmonic_circle Harmonic/Open
hide_all_invisibles Hide All Invisibles
hide_all_panels Hide Tool Windows
hide_empty_staves Hide Empty Staves
hide_show_accidentals_toggle Hide or show accidentals
hide_show_toggle Hide or Show
ideas_window Ideas
import_file Import
import_house_style Import House Style
import_lyrics Create Lyrics From Text File
increase_instrument_stave_size Increase Instrument Staff Size
increase_spacing Increase Spacing
increase_spacing_a_lot Increase Spacing a Lot
input_notes Input Notes
install_plugins Install Plug-ins
instrument_and_stave_dialog Instruments and Staves
instrument_change Instrument Change
keep_bars_together Keep Bars Together
key_signature_dialog Show Key Signature dialog
key_signatures Key Signature
keyboard_window Keyboard
keypad Keypad
keypad0 0 on keypad
keypad1 1 on keypad
keypad2 2 on keypad
keypad3 3 on keypad
keypad4 4 on keypad
keypad5 5 on keypad
keypad6 6 on keypad
keypad7 7 on keypad
keypad8 8 on keypad
keypad9 9 on keypad
keypad_*(Windows)_(Mac)
keypad_+ '+' on keypad
keypad_-(Mac)
keypad_-(Windows)_*(Mac)
keypad_. .' on keypad
keypad_/(Windows)_=(Mac)
keypad_ente Enter on keypad
keypad_first_layout First keypad layout
keypad_next_layout Next keypad layout
keypad_palette1 Common notes
keypad_palette2 More notes
keypad_palette3 Beams/tremolos
keypad_palette4 Articulations
keypad_palette5 Jazz Articulations
keypad_palette6 Accidentals
lines Line
lines_dialog Show Lines dialog
live_playback Live Playback
live_playback_velocities Live Playback Velocities
live_tempo_tap_points Live Tempo Tap Points
lock_format Lock Format
long Long
magnetic_layout_off Turn off Magnetic Layout for item
magnetic_layout_on Turn on Magnetic Layout for item
magnetic_layout_options Magnetic Layout Options
make_into_bar Make Into Bar
make_into_page Make Into Page
make_into_score_subset Make into score subset
make_into_system Make Into System
manuscript_language_reference ManuScript Language Reference
marcato Marcato
mid_beam Middle of beam
minimize_expand_ribbon Minimize/Expand Ribbon
minimize_window Minimize Window
mixer Mixer
move_down_chromatically Move Down Chromatically
move_instruments_down Move Instruments Down
move_instruments_up Move Instruments Up
move_object_down Move Object Down
move_object_down_a_lot Move Object Down a Lot
move_object_left Move Object Left
move_object_left_a_lot Move Object Left a Lot
move_object_right Move Object Right
move_object_right_a_lot Move Object Right a Lot
move_object_up Move Object Up
move_object_up_a_lot Move Object Up a Lot
move_playback_to_selection Move Playback Line to Selection
move_screen_left_a_bit Move Screen Left a Bit
move_screen_right_a_bit Move Screen Right a Bit
move_selection_to_playback Move Selection to Playback Line move_staff_down_a_lot Move Staff Down a Lot (absolute)
move_staff_down_select_note_next_voice Move Staff Down (absolute) and Select Note in Next Voice
move_staff_down_relative Move Staff Down (relative)
move_staff_down_relative_a_lot Move Staff Down a Lot (relative) and Select Note in Staff Below
move_staff_up_a_lot Move Staff Up a Lot (absolute)
move_staff_up_select_note_previous_voice Move Staff Up (absolute) and Select Note in Previous Voice
move_staff_up_relative Move Staff Up (relative)
move_staff_up_relative_a_lot Move Staff Up a Lot (relative) and Select Note in Staff Above
move_up_chromatically Move Up Chromatically
natural Natural
natural_flat Natural flat
natural_sharp Natural sharp
nav_down Move stave down (relative) and Select part of object below
nav_up Move stave up (relative) / Select part of object above navigator Navigator
new_file New
new_window New Window
next_part Next Part
next_window_tab Select Next Tab
no_beam No beam
nonuplet Nonuplet
note_spacing_rules Note Spacing Rule
notehead_0 Notehead 0
notehead_1 Notehead 1
notehead_2 Notehead 2
notehead_3 Notehead 3
notehead_4 Notehead 4
notehead_5 Notehead 5
notehead_6 Notehead 6
notehead_7 Notehead 7
notehead_8 Notehead 8
notehead_9 Notehead 9
noteheads Edit Noteheads
object_rulers Object Rulers
octuplet Octopus
open_file Open
optimize_staff_spacing Optimize Staff Spacing
ordering_move_back Send Back
ordering_move_forward Bring Forward
ordering_move_to_back Send to Back
ordering_move_to_front Bring to Front
ordering_reset_to_default Reset to Default Order
override_pitch_before_duration Override pitch before duration
page_bottom Move bottom of page
page_break Page Break
page_down Move down full screen
page_down_a_bit Move page down a bit
page_first Move to first page
page_last Move to last page
Command IDs
page_left Move left full screen
page_number_change Page Number Change
page_right Move right full screen
page_top Move to top of page
page_up Move up full screen
page_up_a_bit Move page up a bit
panorama_view Panorama
paste Paste
paste_as_cue Paste as Cue
pause Fermata (pause)
performance_interpretation Performance
play/stop Play/Stop
play/stop_or_extend_line Play/Stop or snap line/lyric to next note
play_from_selection Play From Selection
play_half_speed_extend_lines Play at half-speed or snap line/lyric to previous note
play_live_tempo Live Tempo
playback_all_notes_off All Notes Off
playback_backward_one_frame Move backward a single frame
playback_devices Playback Devices
playback_dictionary Playback Dictionary
playback_forward_one_frame Move forward a single frame
playback_line_to_end Move Playback Line to End
playback_line_to_start Move Playback Line to Start
playback_stop Stop
plop Plop
plus Plus/Closed
prebend_note Pre-bend note
preferences Preferences
previous_part Previous Part
previous_window_tab Select Previous Tab
print Print
print_all_parts Print All Parts
print_copies_of_parts Print Copies of Parts
print_dialog OS Print Dialog
publish_to_score_exchange Publish to Score Exchange quadruplet Quadruplet
quarter_flat Quarter flat
quarter_note Quarter note (crotchet)
quarter_sharp Quarter sharp
quick_start Quick Start
quintuplet Quintuplet
reapply_color Reapply Color
rebeam Re-beam
record_live_tempo Record Live Tempo
record_with_audioscore Record with AudioScore
redo Redo
redo_dialog Redo History
remove_accidental Remove accidentals
remove_articulation Remove articulations
remove_staves Remove Staves
repeat Repeat
repeat_2_bars 2 bar Repeat Bar
repeat_4_bars 4 bar Repeat Bar
repeat_bar Repeat Bar
repeat_interpretation Repeats
repitch Re-input Pitches
replay Replay
reset_beam_groups Reset Beam Groups
reset_design Reset Design
reset_magnetic_layout Use default Magnetic Layout settings
reset_note_spacing Reset Note Spacing
reset_position Reset Position
reset_space_above_staff Reset Space Above Staff
reset_space_below_staff Reset Space Below Staff
reset_stems_and_beam_positions Reset Stems and Beam Positions
reset_tab_fingering Reset Guitar Tab Fingering
reset_to_score_design Reset to Score Design
reset_to_score_position Reset to Score Position
respell Respell Accidental/Edit Text
respell_chord_text Respell Chord Text
rest Rest
restart_rehearsal_marks Restart Rehearsal Marks
revoice_chord_diagram Revoice Chord Diagram

```
show_empty_staves Show Empty Staves
show_handles Handles
show_hidden_objects Hidden Objects
show_highlights Highlights
show_in_all Show in All
show_in_parts Show in Parts
show_in_score Show in Score
show_inspector Inspector
show_layout_marks Layout Marks
show_magnetic_layout_collisions Magnetic Layout Collisions
show_magnetic_layout_groups View Magnetic Layout Groups
show_magnetic_layout_original_positions View Magnetic
Layout Original Positions
show_page_margins Page Margins
show_playback_line Playback Line
show_plugin_trace Show Plugin Trace Window
show_replay_marker Replay Line
sibelius_help Sibelius Help
sibelius_tutorials Sibelius Tutorials
single_pages_horizontally Single Pages Horizontally
single_pages_vertically Single Pages Vertically
slide Slide
slide_notes_or_rests_left Slide notes or rests left
slide_notes_or_rests_right Slide notes or rests right
snap_line_next_note Snap line/lyric to next note
special_page_break Special Page Break
split_multirest Split Multirest
split_system Split System
spreads_horizontally Spreads Horizontally
spreads_vertically Spreads Vertically
squarepause Long fermata
staccatissimo Staccatissimo
staccato Staccato
staff_rulers Staff Rulers
start_beam Start beam
start_sub_beam Start secondary beam
stemlet Stemlet
```

subbracket Sub-bracket

| transform_live_playback Transform Live Playback | zoom_12.5\% Zoom 12.5\% |
| :---: | :---: |
| transport Transport | zoom_125\% Zoom 125\% |
| transpose Transpose | zoom_150\% Zoom 150\% |
| tremolo_with_next_note Tremolo with next note | zoom_200\% Zoom 200\% |
| tripause Short fermata | zoom_25\% Zoom 25\% |
| triple_dot Triple dot | zoom_300\% Zoom 300\% |
| triplet Triplet | zoom_37.5\% Zoom 37.5\% |
| tuplet Tuplet | zoom_400\% Zoom 400\% |
| turn_note_into_grace_note Turn note into gracenote | zoom_50\% Zoom 50\% |
| undo Undo | zoom_62.5\% Zoom 62.5\% |
| undo_dialog Undo History | zoom_75\% Zoom 75\% |
| unlock_format Unlock Format | zoom_800\% Zoom 800\% |
| upbow Upbow | zoom_87.5\% Zoom 87.5\% |
| use_multirests Use Multirests | zoom_actual_size Zoom to actual size |
| versions_edit Edit Versions | zoom_edit Zoom |
| versions_export_current Export Current Version | zoom_in Zoom In |
| versions_export_log Export Version Log | zoom_out Zoom Out |
| versions_make_current Make Current Version | zoom_to_fit Zoom to fit page |
| versions_save Save Version | zoom_to_fit_page Zoom Window |
| versions_show_next Show Next Version | zoom_to_fit_two_pages Zoom to fit 2 pages |
| versions_show_previous Show Previous Version | zoom_to_page_height Zoom to page height |
| video_double_size Double Size | zoom_to_page_width Zoom to fit page width |
| video_fullscreen Full Screen | zoom_to_system_width Zoom to system width |
| video_half_size Half Size |  |
| video_normal_size Full Size |  |
| video_window Video |  |
| view_live_tempo View Live Tempo |  |
| voice1 Voice 1 |  |
| voice2 Voice 2 |  |
| voice3 Voice 3 |  |
| voice4 Voice 4 |  |
| voice_all Voice All |  |
| wedge Wedge |  |
| whats_new What's New |  |
| whole_note Whole note (semibreve) |  |
| worksheet_creator Worksheet Creator |  |
| zoom_100\% Zoom 100\% |  |

## Global Constants

## Global Constants

These are useful variables held internally within ManuScript and are accessible from any plug-in. They are called "constants" because you are encouraged not to change them.

Many of the constants are the names of note values, which you can use to specify a position in a bar. For example, instead of writing 320 you can write Quarter+Sixteenth or equally Crotchet+Semiquaver.

## Truth Values

| True | 1 |
| :--- | :--- |
| False | 0 |

## Measurements

| Space | 32 |
| :--- | :--- |
| StaffHeight | 128 |

## Positions and Durations

| Long | 4096 |
| :--- | :--- |
| Breve | 2048 |
| DottedBreve | 3072 |
| Whole or Semibreve | 1024 |
| DottedWhole | 1536 |
| Half or Minim | 512 |
| DottedHalf or DottedMinim | 768 |
| Quarter or Crotchet | 256 |
| DottedQuarter or DottedCrotchet | 384 |
| Eighth or Quaver | 128 |
| DottedEighth or DottedQuaver | 192 |
| Sixteenth or Semiquaver | 64 |
| DottedSixteenth or DottedSemiquaver | 32 |
| ThirtySecond or Demisemiquaver | 48 |
| DottedThirtySecond or DottedDemisemiquaver | 24 |
| SixtyFourth or Hemidemisemiquaver |  |
| DottedSixtyFourth or DottedHemidemisemiquaver |  |


| OneHundredTwentyEighth or Semihemidemisemiquaver | 8 |
| :--- | :--- |
| DottedOneHundredTwentyEighth or DottedSemihemidemisemiquaver | 12 |

## Style Names

For the ApplyStyle () method of Score objects. Instead of the capitalized strings in quotes, you can use the equivalent variables in mixed upper and lower case. Note again that the constant HOUSE refers to the options in House Style > Engraving Rules and Layout > Document Setup only; to apply the entire House Style, use the ALLSTYLES constant.

| Engraving Rules | "ENGRAVINGRULES" | Document Setup | "DOCSETUP" |
| :--- | :--- | :--- | :--- |
| House | "HOUSE" | Dictionary | "DICTIONARY" |
| Text | "TEXT" | SpacingRule | "SPACINGRULE" |
| Symbols | "SYMBOLS" | CustomChordNames | "CUSTOMCHORDNAMES" |
| Lines | "LINES" | DefaultPartAppearance | "DEFAULTPARTAPPEARANCE" |
| Noteheads | "NOTEHEADS" | InstrumentsAndEnsembles | "INSTRUMENTSANDENSEMBLES" |
| Clefs | "CLEFS" | AlIStyles | "ALLSTYLES" |

## Bar Number Formats

These constants can be used for the format argument of the AddBarNumber method.

| BarNumberFormatNormal | 0 |
| :--- | :--- |
| BarNumberFormatNumberLetterLower | 1 |
| BarNumberFormatNumberLetterUpper | 2 |

## Text Styles

Here is a list of all the text style identifiers which are guaranteed to be present in any score in Sibelius. In previous versions of ManuScript text styles were identified by a numeric index; this usage has been deprecated but will continue to work for old plug-ins. New plug-ins should use the identifiers given below. For each style we first give the English name of the style and then the identifier.

| Instrument <br> names | "text.instrumentname" | Time signatures <br> (one staff only) | "text.staff.timesig.onestaffonly" |
| :--- | :--- | :--- | :--- |
| 1st and 2nd <br> endings | "text.staff.1st_n_2nd_endings" | Tuplets | "text.staff.tuplets" |
| Auto page <br> break <br> warnings | "text.staff.autopagebreak.warnings" | Bar numbers | "text.system.barnumber" |
| Boxed text | "text.staff.boxed" | Metronome mark | "text.system.metronome" |
| Expression | "text.staff.expression" | Multirests <br> (numbers) | "text.system.multirestnumbers" |
| Chord <br> diagram <br> fingering | "text.staff.fingering.chord_diagrams" | Composer | "text.system.page_aligned.composer" |
| Footnote | "text.staff.footnote" | Composer <br> (on title page) | "text.system.page_aligned.composer.ontitlepage" |
| Block lyrics | "text.staff.lyrics.block" | Copyright | "text.system.page_aligned.copyright" |


| Multirests (tacet) | "text.staff.multirests.tacet" | Dedication | "text.system.page_aligned.dedication" |
| :---: | :---: | :---: | :---: |
| Plain text | "text.staff.plain" | Footer (inside edge) | "text.system.page_aligned.footer.inside" |
| Small text | "text.staff.small" | Footer (outside edge) | "text.system.page_aligned.footer.outside" |
| Chord symbol | "text.staff.space.chordsymbol" | Worksheet footer (first page, I) | "text.system.page_aligned.footer.worksheet.left" |
| Figured bass | "text.staff.space.figuredbass" | Header | "text.system.page_aligned.header" |
| Fingering | "text.staff.space.fingering" | Worksheet header (first page, I) | "text.system.page_aligned.header.worksheet.left" |
| Chord diagram fret | "text.staff.space.fretnumbers" | Worksheet header (first page, r) | "text.system.page_aligned.header.worksheet.right" |
| Lyrics above staff | "text.staff.space.hypen.lyrics.above" | Header (after first page) | "text.system.page_aligned.header_notp1" |
| Lyrics (chorus) | "text.staff.space.hypen.lyrics.chorus" | Header (after first page, inside edge) | "text.system.page_aligned.header_notp1.inside" |
| Lyrics line 1 | "text.staff.space.hypen.lyrics.verse1" | Instrument name at top left | "text.system.page_aligned.instrnametopleft" |
| Lyrics line 2 | "text.staff.space.hypen.lyrics.verse2" | Lyricist | "text.system.page_aligned.lyricist" |
| Lyrics line 3 | "text.staff.space.hypen.lyrics.verse3" | Page numbers | "text.system.page_aligned.pagenumber" |
| Lyrics line 4 | "text.staff.space.hypen.lyrics.verse4" | Subtitle | "text.system.page_aligned.subtitle" |
| Lyrics line 5 | "text.staff.space.hypen.lyrics.verse5" | Title | "text.system.page_aligned.title" |
| Nashville chord numbers | "text.staff.space.nashvillechords" | Title (on title page) | "text.system.page_aligned.title.ontitlepage" |
| Common symbols | "text.staff.symbol.common" | Rehearsal mark | "text.system.rehearsalmarks" |
| Figured bass (extras) | "text.staff.symbol.figured.bass.extras" | Repeat (D.C./ <br> D.S./To Coda) | "text.system.repeat" |
| Note tails | "text.staff.symbol.noteflags" | Tempo | "text.system.tempo" |
| Special noteheads | "text.staff.symbol.noteheads.special" | Timecode | "text.system.timecode" |
| Percussion instruments | "text.staff.symbol.percussion" | Duration at end of score | "text.system.timecode.duration" |
| Special symbols | "text.staff.symbol.special" | Hit points | "text.system.timecode.hitpoints" |
| Tablature letters | "text.staff.tab.letters" | Time signatures (huge) | "text.system.timesig.huge" |
| Tablature numbers | "text.staff.tab.numbers" | Time signatures (large) | "text.system.timesig.large" |


| Technique | "text.staff.technique" | Time signatures | "text.system.timesig.normal" |
| :--- | :--- | :--- | :--- |

## Line Styles

| Arpeggio | "line.staff.arpeggio" | Bracketed slur below | "line.staff.slur.down.bracketed" |
| :---: | :---: | :---: | :---: |
| Arpeggio down | "line.staff.arpeggio.down" | Dashed slur below | "line.staff.slur.down.dashed" |
| Arpeggio up | "line.staff.arpeggio.up" | Dotted slur below | "line.staff.slur.down.dotted" |
| Unused 2 | "line.staff.arrow" | Slur above | "line.staff.slur.up" |
| Arrow | "line.staff.arrow.black.right" | Bracketed slur above | "line.staff.slur.up.bracketed" |
| Dashed arrow | "line.staff.arrow.black.right.dashed" | Dashed slur above | "line.staff.slur.up.dashed" |
| Double arrow | "line.staff.arrow.black.right.left" | Dotted slur above | "line.staff.slur.up.dotted" |
| Vertical arrow (2) | "line.staff.arrow.black.vertical" | String indicator above (1) | "line.staff.string.above.1" |
| White arrow | "line.staff.arrow.white.right" | String indicator above (2) | "line.staff.string.above.2" |
| Dashed white arrow | "line.staff.arrow.white.right.dashed" | String indicator above (3) | "line.staff.string.above.3" |
| Double white arrow | "line.staff.arrow.white.right.left" | String indicator above (4) | "line.staff.string.above.4" |
| Vertical arrow | "line.staff.arrow.white.vertical" | String indicator above (5) | "line.staff.string.above.5" |
| Beam | "line.staff.beam" | String indicator above (6) | "line.staff.string.above.6" |
| Guitar Bend | "line.staff.bend" | String indicator above (7) | "line.staff.string.above.7" |
| Guitar hold bend | "line.staff.bend.hold" | String indicator above (8) | "line.staff.string.above.8" |
| Box | "line.staff.box" | String indicator below (1) | "line.staff.string.below.1" |
| Bracket above | "line.staff.bracket.above" | String indicator below (2) | "line.staff.string.below.2" |
| Bracket above (end) | "line.staff.bracket.above.end" | String indicator below (3) | "line.staff.string.below.3" |
| Bracket above (start) | "line.staff.bracket.above.start" | String indicator below (4) | "line.staff.string.below.4" |
| Bracket below | "line.staff.bracket.below" | String indicator below (5) | "line.staff.string.below.5" |
| Bracket below (end) | "line.staff.bracket.below.end" | String indicator below (6) | "line.staff.string.below.6" |
| Bracket below (start) | "line.staff.bracket.below.start" | String indicator below (7) | "line.staff.string.below.7" |
| Vertical bracket | "line.staff.bracket.vertical" | String indicator below (8) | "line.staff.string.below.8" |


| Vertical bracket 2 | "line.staff.bracket.vertical.2" | Tie | "line.staff.tie" |
| :---: | :---: | :---: | :---: |
| Dashed line | "line.staff.dashed" | Trill | "line.staff.trill" |
| Vertical dashed line | "line.staff.dashed.vertical" | Tuplet | "line.staff.tuplet" |
| Dotted line | "line.staff.dotted" | Vertical line | "line.staff.vertical" |
| Glissando (straight) | "line.staff.gliss.straight" | Vibrato | "line.staff.vibrato" |
| Glissando (wavy) | "line.staff.gliss.wavy" | Guitar vibrato bar | "line.staff.vibrato.bar" |
| Guitar effect | "line.staff.guitareffect" | Wide vibrato | "line.staff.vibrato.wide" |
| Crescendo | "line.staff.hairpin.crescendo" | Dashed system line | "line.system.dashed" |
| Bracketed crescendo | "line.staff.hairpin.crescendo.bracketed" | Wide dashed system line | "line.system.dashed.wide" |
| Dashed crescendo | "line.staff.hairpin.crescendo.dashed" | 1st ending | "line.system.repeat.1st" |
| Dotted crescendo | "line.staff.hairpin.crescendo.dotted" | 1st and 2nd ending | "line.system.repeat.1st_n_2nd" |
| Crescendo from silence | "line.staff.hairpin.crescendo.fromsilence" | 2nd ending | "line.system.repeat.2nd" |
| Diminuendo | "line.staff.hairpin.diminuendo" | 2nd ending (closed) | "line.system.repeat.2nd.closed" |
| Bracketed diminuendo | "line.staff.hairpin.diminuendo.bracketed" | 3rd ending | "line.system.repeat.3rd" |
| Dashed diminuendo | "line.staff.hairpin.diminuendo.dashed" | Repeat ending (closed) | "line.system.repeat.closed" |
| Dotted diminuendo | "line.staff.hairpin.diminuendo.dotted" | Repeat ending (open) | "line.system.repeat.open" |
| Diminuendo to silence | "line.staff.hairpin.diminuendo.tosilence" | Accel. | "line.system.tempo.accel" |
| Guitar artificial harmonic | "line.staff.harmonic.artificial" | Accel. (italic) | "line.system.tempo.accel.italic" |
| Guitar harp harmonic | "line.staff.harmonic.harp" | Accel. <br> (italic, text only) | "line.system.tempo.accel.italic.textonly" |
| Guitar pinch harmonic | "line.staff.harmonic.pinch" | Molto accel. | "line.system.tempo.accel.molto" |
| Guitar touch harmonic | "line.staff.harmonic.touch" | Molto accel. (text only) | "line.system.tempo.accel.molto.textonly" |
| Guitar harmonics | "line.staff.harmonics" | Poco accel. | "line.system.tempo.accel.poco" |
| Hauptstimme | "line.staff.hauptstimme" | Poco accel. (text only) | "line.system.tempo.accel.poco.textonly" |
| Guitar let ring | "line.staff.letring" | Accel. (text only) | "line.system.tempo.accel.textonly" |
| Lyric line | "line.staff.lyric" | Tempo change (arrow right) | "line.system.tempo.arrowright" |
| Guitar palm mute | "line.staff.mute.palm" | Rall. | "line.system.tempo.rall" |
| Nebenstimme | "line.staff.nebenstimme" | Rall. (italic) | "line.system.tempo.rall.italic" |
| 2 octaves down | "line.staff.octava.minus15" | Rall. <br> (italic, text only) | "line.system.tempo.rall.italic.textonly" |


| Octave down | "line.staff.octava.minus8" | Molto rall. | "line.system.tempo.rall.molto" |
| :--- | :--- | :--- | :--- |
| 2 octaves up | "line.staff.octava.plus15" | Molto rall. <br> (text only) | "line.system.tempo.rall.molto.textonly" |
| Octave up | "line.staff.octava.plus8" | Poco rall. | "line.system.tempo.rall.poco" |
| Pedal | "line.staff.pedal" | Poco rall. (text only) | "line.system.tempo.rall.poco.textonly" |
| Pedal lift | "line.staff.pedal.lift" | Rall. (text only) | "line.system.tempo.rall.textonly" |
| Pedal lift again | "line.staff.pedal.lift.again" | Rit. | "line.system.tempo.rit" |
| Pedal lift finally | "line.staff.pedal.lift.finally" | Rit. (italic) | "line.system.tempo.rit.italic" |
| Pedal (no line) | "line.staff.pedal.noline" | Rit. (italic, text only) | "line.system.tempo.rit.italic.textonly" |
| Guitar pick scrape | "line.staff.pick.scrape" | Molto rit. | "line.system.tempo.rit.molto" |
| Line | "line.staff.plain" | Molto rit. (text only) | "line.system.tempo.rit.molto.textonly" |
| Portamento | "line.staff.port.straight" | Poco rit. | "line.system.tempo.rit.poco" |
| Guitar rake | "line.staff.rake" | Poco rit. (text only) | "line.system.tempo.rit.poco.textonly" |
| Guitar slide | "line.staff.slide" | Rit. (text only) | "line.system.tempo.rit.textonly" |
| Slur below | "line.staff.slur.down" |  |  |

## Clef Styles

Here is a list of all the clef style identifiers that are guaranteed to be present in any score in Sibelius, for use with the Stave. AddClef method. For each style we first give the English name of the style, and then the identifier.

| Alto | "clef.alto" | Small tab | "clef.tab.small" |
| :--- | :--- | :--- | :--- |
| Baritone C | "clef.baritone.c" | Small tab (taller) | "clef.tab.small.taller" |
| Baritone F | "clef.baritone.f" | Tab (taller) | "clef.tab.taller" |
| Bass | "clef.bass" | Tenor | "clef.tenor" |
| Bass down 8 | "clef.bass.down.8" | Tenor down 8 | "clef.tenor.down.8" |
| Bass up 15 | "clef.bass.up.15" | Treble | "clef.treble" |
| Bass up 8 | "clef.bass.up.8" | Treble down 8 | "clef.treble.down.8" |
| Null | "clef.null" | Treble (down 8) | "clef.treble.down.8.bracketed" |
| Percussion | "clef.percussion" | Treble down 8 (old) | "clef.treble.down.8.old" |
| Percussion 2 | "clef.percussion_2" | Treble up 15 | "clef.treble.up.15" |
| Soprano | "clef.soprano" | Treble up 8 | "clef.treble.up.8" |
| Mezzo-soprano | "clef.soprano.mezzo" | French violin | "clef.violin.french" |
| Tab | "clef.tab" | Sub-bass F | "clef.sub-bass.f" |

## Instrument Types

Here is a list of all the instrument type identifiers that are guaranteed to be present in any score in Sibelius. For each style we first give the English name of the style and then the identifier. Note that only the tablature stave types can be used with guitar frames; the rest are included for completeness.

| Alp-Horn in F | instrument.brass.alp-horn.f |
| :---: | :---: |
| Alp-Horn in G | instrument.brass.alp-horn.g |
| Baritone Bugle in G | instrument.brass.bugle.baritone.g |
| Contrabass Bugle in G | instrument.brass.bugle.contrabass.g |
| Euphonium Bugle in G | instrument.brass.bugle.euphonium.g |
| Mellophone Bugle in G | instrument.brass.bugle.mellophone.g |
| Soprano Bugle in G | instrument.brass.bugle.soprano.g |
| Cimbasso in $\mathbf{B b}$ | instrument.brass.cimbasso.bflat |
| Cimbasso in Eb | instrument.brass.cimbasso.eflat |
| Cimbasso in F | instrument.brass.cimbasso.f |
| Cornet in A | instrument.brass.cornet.a |
| Cornet in Bb | instrument.brass.cornet.bflat |
| Soprano Cornet in Eb | instrument.brass.cornet.soprano.eflat |
| Euphonium in Bb [treble clef] | instrument.brass.euphonium |
| Euphonium in Bb [bass clef, treble transp.] | instrument.brass.euphonium.bassclef |
| Euphonium in C [bass clef] | instrument.brass.euphonium.bassclef.bassclef |
| Euphonium in Bb [bass clef] | instrument.brass.euphonium.bflat.bassclef.bassclef |
| Flugelhorn | instrument.brass.flugelhorn |
| Horn in A [no key] | instrument.brass.horn.a.nokeysig |
| Horn in Ab alto [no key] | instrument.brass.horn.alto.aflat.nokeysig |
| Alto Horn in Eb | instrument.brass.horn.alto.eflat |
| Alto Horn in F | instrument.brass.horn.alto.f |
| Horn in B [no key] | instrument.brass.horn.b.nokeysig |
| Baritone in Bb [treble clef] | instrument.brass.horn.baritone |
| Baritone in C [treble clef] | instrument.brass.horn.baritone. 2 |
| Baritone in Bb [bass clef, treble transp.] | instrument.brass.horn.baritone.bassclef |
| Baritone in C [bass clef] | instrument.brass.horn.baritone.bassclef.bassclef |
| Bass in $\mathbf{B b}$ | instrument.brass.horn.bass.bflat |
| Bass in Bb [bass clef, treble transp.] | instrument.brass.horn.bass.bflat.bassclef |
| Bass in C | instrument.brass.horn.bass.c |
| Bass in Eb | instrument.brass.horn.bass.eflat |
| Bass in Eb [bass clef, treble transp.] | instrument.brass.horn.bass.eflat.bassclef |
| A Basso Horn [no key] | instrument.brass.horn.basso.a.nokeysig |
| Bb Basso Horn [no key] | instrument.brass.horn.basso.bflat.nokeysig |
| C Basso Horn [no key] | instrument.brass.horn.basso.c.nokeysig |
| Horn in Bb [no key] | instrument.brass.horn.bflat.nokeysig |


| Horn in C [no key] | instrument.brass.horn.c.nokeysig |
| :---: | :---: |
| Horn in D [no key] | instrument.brass.horn.d.nokeysig |
| Horn in Db [no key] | instrument.brass.horn.dflat.nokeysig |
| Horn in E [no key] | instrument.brass.horn.e.nokeysig |
| Horn in Eb | instrument.brass.horn.eflat |
| Horn in Eb [no key] | instrument.brass.horn.eflat.nokeysig |
| Horn in F | instrument.brass.horn.f |
| Horn in F [bass clef] | instrument.brass.horn.f.bassclef |
| Horn in F [no key] | instrument.brass.horn.f.nokeysig |
| Horn in F\# [no key] | instrument.brass.horn.fsharp.nokeysig |
| Horn in G [no key] | instrument.brass.horn.g.nokeysig |
| Tenor Horn | instrument.brass.horn.tenor |
| Mellophone in Eb | instrument.brass.mellophone.eflat |
| Mellophone in F | instrument.brass.mellophone.f |
| Mellophonium in Eb | instrument.brass.mellophonium.eflat |
| Mellophonium in F | instrument.brass.mellophonium.f |
| Ophicleide | instrument.brass.ophicleide |
| Brass | instrument.brass.section |
| Serpent | instrument.brass.serpent |
| Sousaphone in Bb | instrument.brass.sousaphone.bflat |
| Sousaphone in Eb | instrument.brass.sousaphone.eflat |
| Trombone | instrument.brass.trombone |
| Alto Trombone | instrument.brass.trombone.alto |
| Bass Trombone | instrument.brass.trombone.bass |
| Trombone in Bb [bass clef, treble transp.] | instrument.brass.trombone.bassclef.trebleclef |
| Contrabass Trombone | instrument.brass.trombone.contrabass |
| Tenor Trombone | instrument.brass.trombone.tenor |
| Trombone in Bb [treble clef] | instrument.brass.trombone.trebleclef |
| Trumpet in A | instrument.brass.trumpet.a |
| Trumpet in B [no key] | instrument.brass.trumpet.b.nokeysig |
| Bass Trumpet in Bb | instrument.brass.trumpet.bass.bflat |
| Bass Trumpet in Eb | instrument.brass.trumpet.bass.eflat |
| Trumpet in Bb | instrument.brass.trumpet.bflat |
| Trumpet in Bb [no key] | instrument.brass.trumpet.bflat.nokeysig |
| Trumpet in C | instrument.brass.trumpet.c |
| Trumpet in D | instrument.brass.trumpet.d |
| Trumpet in Db | instrument.brass.trumpet.dflat |


| Trumpet in E [no key] | instrument.brass.trumpet.e.nokeysig |
| :---: | :---: |
| Trumpet in Eb | instrument.brass.trumpet.eflat |
| Trumpet in F | instrument.brass.trumpet.f |
| Trumpet in G [no key] | instrument.brass.trumpet.g.nokeysig |
| Piccolo Trumpet in A | instrument.brass.trumpet.piccolo.a |
| Piccolo Trumpet in Bb | instrument.brass.trumpet.piccolo.bflat |
| Tenor Trumpet in Eb | instrument.brass.trumpet.tenor.eflat |
| Tuba | instrument.brass.tuba |
| Tuba in F | instrument.brass.tuba.f |
| Tenor Tuba (Wagner, in Bb) | instrument.brass.tuba.tenor |
| Tenor Tuba [bass clef] | instrument.brass.tuba.tenor.bassclef |
| Wagner Tuba in Bb | instrument.brass.tuba.wagner.bflat |
| Wagner Tuba in F | instrument.brass.tuba.wagner.f |
| Applause | instrument.exotic.applause |
| Birdsong | instrument.exotic.birdsong |
| Helicopter | instrument.exotic.helicopter |
| Ondes Martenot | instrument.exotic.ondes-martenot |
| Sampler | instrument.exotic.sampler |
| Seashore | instrument.exotic.seashore |
| Tape | instrument.exotic.tape |
| Telephone | instrument.exotic.telephone |
| Theremin | instrument.exotic.theremin |
| Bajo [notation] | instrument.fretted.bajo.5lines |
| Bajo, 6-string [tab] | instrument.fretted.bajo.tab |
| Bajo, 4-string [tab] | instrument.fretted.bajo.tab.4lines |
| Bajo, 5-string [tab] | instrument.fretted.bajo.tab.5lines |
| Alto Balalaika [notation] | instrument.fretted.balalaika.alto.5lines |
| Alto Balalaika [tab] | instrument.fretted.balalaika.alto.tab |
| Bass Balalaika [notation] | instrument.fretted.balalaika.bass.5lines |
| Bass Balalaika [tab] | instrument.fretted.balalaika.bass.tab |
| Contrabass Balalaika [notation] | instrument.fretted.balalaika.contrabass.5lines |
| Contrabass Balalaika [tab] | instrument.fretted.balalaika.contrabass.tab |
| Prima Balalaika [notation] | instrument.fretted.balalaika.prima.5lines |
| Prima Balalaika [tab] | instrument.fretted.balalaika.prima.tab |
| Second Balalaika [notation] | instrument.fretted.balalaika.second.5lines |
| Second Balalaika [tab] | instrument.fretted.balalaika.second.tab |
| Bandola [notation] | instrument.fretted.bandola.5lines |


| Bandola [tab] | instrument.fretted.bandola.tab |
| :---: | :---: |
| Bandolón [notation] | instrument.fretted.bandolon.5lines |
| Bandolón [tab] | instrument.fretted.bandolon.tab |
| Bandurria [notation] | instrument.fretted.bandurria.5lines |
| Bandurria [tab] | instrument.fretted.bandurria.tab |
| Banjo [notation] | instrument.fretted.banjo.5lines |
| Banjo (aDADE tuning) [tab] | instrument.fretted.banjo.aDADE.tab |
| Banjo (aEADE tuning) [tab] | instrument.fretted.banjo.aEADE.tab |
| Banjo (gCGBD tuning) [tab] | instrument.fretted.banjo.gCGBD.tab |
| Banjo (gCGCD tuning) [tab] | instrument.fretted.banjo.gCGCD.tab |
| Banjo (gDF\#AD tuning) [tab] | instrument.fretted.banjo.gDFAD.tab |
| Banjo (gDGBD tuning) [tab] | instrument.fretted.banjo.gDGBD.tab |
| Banjo (gDGCD tuning) [tab] | instrument.fretted.banjo.gDGCD.tab |
| Tenor Banjo [notation] | instrument.fretted.banjo.tenor.5lines |
| Tenor Banjo [tab] | instrument.fretted.banjo.tenor.tab |
| Bordonúa [notation] | instrument.fretted.bordonua.5lines |
| Bordonúa [tab] | instrument.fretted.bordonua.tab |
| Cavaquinho [notation] | instrument.fretted.cavaquinho.5lines |
| Cavaquinho [tab] | instrument.fretted.cavaquinho.tab |
| Charango [notation] | instrument.fretted.charango.5lines |
| Charango [tab] | instrument.fretted.charango.tab |
| Cuatro [notation] | instrument.fretted.cuatro.5lines |
| Cuatro, Puerto Rico [tab] | instrument.fretted.cuatro.puerto-rico.tab |
| Cuatro, Venezuela [tab] | instrument.fretted.cuatro.venezuela.tab |
| Resonator guitar [notation] | instrument.fretted.guitar.resonator.5lines |
| Resonator Guitar, A6 tuning [tab] | instrument.fretted.guitar.resonator.a6.tab |
| Resonator Guitar, B11 tuning [tab] | instrument.fretted.guitar.resonator.b11.tab |
| Resonator Guitar, C\#m tuning [tab] | instrument.fretted.guitar.resonator.c\#m.tab |
| Resonator Guitar, C6+A7 tuning [tab] | instrument.fretted.guitar.resonator.c6-a7.tab |
| Resonator Guitar, C6 + high G tuning [tab] | instrument.fretted.guitar.resonator.c6-highg.tab |
| Resonator Guitar, standard tuning [tab] | instrument.fretted.guitar.resonator.c6.tab |
| Resonator Guitar, C\#m7 tuning [tab] | instrument.fretted.guitar.resonator.cm7.tab |
| Resonator Guitar, E13 Hawaiian tuning [tab] | instrument.fretted.guitar.resonator.e13-hawaiian.tab |
| Resonator Guitar, E13 Western tuning [tab] | instrument.fretted.guitar.resonator.e13-western.tab |
| Resonator Guitar, open A tuning [tab] | instrument.fretted.guitar.resonator.open.A.tab |
| Resonator Guitar, open G tuning [tab] | instrument.fretted.guitar.resonator.open.G.tab |
| Dulcimer | instrument.fretted.dulcimer |


| Dulcimer [notation] | instrument.fretted.dulcimer.5lines |
| :---: | :---: |
| Dulcimer (DAA tuning) [tab] | instrument.fretted.dulcimer.daa.tab |
| Dulcimer (DAD tuning) [tab] | instrument.fretted.dulcimer.dad.tab |
| Gamba [notation] | instrument.fretted.gamba.5lines |
| Gamba [tab] | instrument.fretted.gamba.tab |
| 12-string Acoustic Guitar [notation] | instrument.fretted.guitar.12-string.5lines |
| 12-string Acoustic Guitar, DADGAD tuning [tab] | instrument.fretted.guitar.12-string.dadgad.tab |
| 12-string Acoustic Guitar, double D tuning [tab] | instrument.fretted.guitar.12-string.double-d.tab |
| 12-string Acoustic Guitar, dropped D tuning [tab] | instrument.fretted.guitar.12-string.dropped-d.tab |
| 12-string Acoustic Guitar, open D tuning [tab] | instrument.fretted.guitar.12-string.open-d.tab |
| 12-string Acoustic Guitar, open E tuning [tab] | instrument.fretted.guitar. 12-string.open-e.tab |
| 12-string Acoustic Guitar, open G tuning [tab] | instrument.fretted.guitar. 12-string.open-g.tab |
| 12-string Acoustic Guitar, standard tuning (no rhythms) [tab] | instrument.fretted.guitar.12-string.tab |
| 12-string Acoustic Guitar, standard tuning [tab] | instrument.fretted.guitar.12-string.tab.rhythms |
| Acoustic Guitar [notation] | instrument.fretted.guitar.acoustic.5lines |
| Acoustic Guitar, DADGAD tuning [tab] | instrument.fretted.guitar.acoustic.dadgad.tab |
| Acoustic Guitar, double D tuning [tab] | instrument.fretted.guitar.acoustic.double-d.tab |
| Acoustic Guitar, dropped D tuning [tab] | instrument.fretted.guitar.acoustic.dropped-d.tab |
| Acoustic Guitar, modal D tuning [tab] | instrument.fretted.guitar.acoustic.modal-d.tab |
| Acoustic Guitar, Nashville tuning [tab] | instrument.fretted.guitar.acoustic.nashville.tab |
| Acoustic Guitar, open A tuning [tab] | instrument.fretted.guitar.acoustic.open-a.tab |
| Acoustic Guitar, open C tuning [tab] | instrument.fretted.guitar.acoustic.open-c.tab |
| Acoustic Guitar, open D tuning [tab] | instrument.fretted.guitar.acoustic.open-d.tab |
| Acoustic Guitar, open Dm cross-note tuning [tab] | instrument.fretted.guitar.acoustic.open-dm.tab |
| Acoustic Guitar, open E tuning [tab] | instrument.fretted.guitar.acoustic.open-e.tab |
| Acoustic Guitar, open G tuning [tab] | instrument.fretted.guitar.acoustic.open-g.tab |
| Acoustic Guitar, standard tuning (no rhythms) [tab] | instrument.fretted.guitar.acoustic.tab |
| Acoustic Guitar, standard tuning [tab] | instrument.fretted.guitar.acoustic.tab.rhythms |
| 4-string Bass Guitar [notation] | instrument.fretted.guitar.bass.4-string.5lines |
| 4-string Bass Guitar [tab] | instrument.fretted.guitar.bass.4-string.tab |
| 5-string Bass Guitar [notation] | instrument.fretted.guitar.bass.5-string.5lines |
| 5-string Bass Guitar [tab] | instrument.fretted.guitar.bass.5-string.tab |
| Bass Guitar [notation] | instrument.fretted.guitar.bass.5lines |
| 6-string Bass Guitar [notation] | instrument.fretted.guitar.bass.6-string.5lines |
| 6-string Bass Guitar [tab] | instrument.fretted.guitar.bass.6-string.tab |
| Acoustic Bass [notation] | instrument.fretted.guitar.bass.acoustic.5lines |


| Acoustic Bass [tab] | instrument.fretted.guitar.bass.acoustic.tab |
| :---: | :---: |
| 5-string Electric Bass [notation] | instrument.fretted.guitar.bass.electric.5-string.5lines |
| 5-string Electric Bass [tab] | instrument.fretted.guitar.bass.electric.5-string.tab |
| Electric Bass [notation] | instrument.fretted.guitar.bass.electric.5lines |
| 6-string Electric Bass [notation] | instrument.fretted.guitar.bass.electric.6-string.5lines |
| 6-string Electric Bass [tab] | instrument.fretted.guitar.bass.electric.6-string.tab |
| 5-string Fretless Electric Bass | instrument.fretted.guitar.bass.electric.fretless.5-string.5lines |
| 5-string Fretless Electric Bass [tab] | instrument.fretted.guitar.bass.electric.fretless.5-string.tab |
| Fretless Electric Bass [notation] | instrument.fretted.guitar.bass.electric.fretless.5lines |
| 6-string Fretless Electric Bass | instrument.fretted.guitar.bass.electric.fretless.6-string.5lines |
| 6-string Fretless Electric Bass [tab] | instrument.fretted.guitar.bass.electric.fretless.6-string.tab |
| Fretless Electric Bass [tab] | instrument.fretted.guitar.bass.electric.fretless.tab |
| Electric Bass [tab] | instrument.fretted.guitar.bass.electric.tab |
| 5-string Fretless Bass Guitar [notation] | instrument.fretted.guitar.bass.fretless.5-string.5lines |
| 5-string Fretless Bass Guitar [tab] | instrument.fretted.guitar.bass.fretless.5-string.tab |
| Fretless Bass Guitar [notation] | instrument.fretted.guitar.bass.fretless.5lines |
| 6-string Fretless Bass Guitar [notation] | instrument.fretted.guitar.bass.fretless.6-string.5lines |
| 6-string Fretless Bass Guitar [tab] | instrument.fretted.guitar.bass.fretless.6-string.tab |
| Fretless Bass Guitar [tab] | instrument.fretted.guitar.bass.fretless.tab |
| Semi-Acoustic Bass [notation] | instrument.fretted.guitar.bass.semi-acoustic.5lines |
| Semi-Acoustic Bass [tab] | instrument.fretted.guitar.bass.semi-acoustic.tab |
| Bass Guitar [tab] | instrument.fretted.guitar.bass.tab |
| Bass Guitar [tab, with rhythms] | instrument.fretted.guitar.bass.tab.rhythms |
| Classical Guitar [notation] | instrument.fretted.guitar.classical.5lines |
| Classical Guitar, DADGAD tuning [tab] | instrument.fretted.guitar.classical.dadgad.tab |
| Classical Guitar, double D tuning [tab] | instrument.fretted.guitar.classical.double-d.tab |
| Classical Guitar, dropped D tuning [tab] | instrument.fretted.guitar.classical.dropped-d.tab |
| Classical Guitar, open D tuning [tab] | instrument.fretted.guitar.classical.open-d.tab |
| Classical Guitar, open E tuning [tab] | instrument.fretted.guitar.classical.open-e.tab |
| Classical Guitar, open G tuning [tab] | instrument.fretted.guitar.classical.open-g.tab |
| Classical Guitar, standard tuning (no rhythms) [tab] | instrument.fretted.guitar.classical.tab |
| Classical Guitar, standard tuning [tab] | instrument.fretted.guitar.classical.tab.rhythms |
| Electric Guitar [notation] | instrument.fretted.guitar.electric.5lines |
| 7-string Electric Guitar, low A tuning [tab] | instrument.fretted.guitar.electric.7-string.low-a.tab |
| 7-string Electric Guitar, low B tuning [tab] | instrument.fretted.guitar.electric.7-string.tab |
| Electric Guitar, DADGAD tuning [tab] | instrument.fretted.guitar.electric.dadgad.tab |
| Electric Guitar, double D tuning [tab] | instrument.fretted.guitar.electric.double-d.tab |


| Electric Guitar, dropped D tuning [tab] | instrument.fretted.guitar.electric.dropped-d.tab |
| :---: | :---: |
| Electric Guitar, open D tuning [tab] | instrument.fretted.guitar.electric.open-d.tab |
| Electric Guitar, open E tuning [tab] | instrument.fretted.guitar.electric.open-e.tab |
| Electric Guitar, open G tuning [tab] | instrument.fretted.guitar.electric.open-g.tab |
| Electric Guitar, standard tuning (no rhythms) [tab] | instrument.fretted.guitar.electric.tab |
| Electric Guitar, standard tuning [tab] | instrument.fretted.guitar.electric.tab.rhythms |
| Kora | instrument.fretted.guitar.kora |
| Semi-acoustic Guitar [notation] | instrument.fretted.guitar.semi-acoustic.5lines |
| Semi-acoustic Guitar, DADGAD tuning [tab] | instrument.fretted.guitar.semi-acoustic.dadgad.tab |
| Semi-acoustic Guitar, double D tuning [tab] | instrument.fretted.guitar.semi-acoustic.double-d.tab |
| Semi-acoustic Guitar, dropped D tuning [tab] | instrument.fretted.guitar.semi-acoustic.dropped-d.tab |
| Semi-acoustic Guitar, open D tuning [tab] | instrument.fretted.guitar.semi-acoustic.open-d.tab |
| Semi-acoustic Guitar, open E tuning [tab] | instrument.fretted.guitar.semi-acoustic.open-e.tab |
| Semi-acoustic Guitar, open G tuning [tab] | instrument.fretted.guitar.semi-acoustic.open-g.tab |
| Semi-acoustic Guitar, standard tuning (no rhythms) [tab] | instrument.fretted.guitar.semi-acoustic.tab |
| Semi-acoustic Guitar, standard tuning [tab] | instrument.fretted.guitar.semi-acoustic.tab.rhythms |
| 10-string Hawaiian Steel Guitar [tab] | instrument.fretted.guitar.steel.hawaiian.10-string.tab |
| Hawaiian Steel Guitar [notation] | instrument.fretted.guitar.steel.hawaiian.5lines |
| 6-string Hawaiian Steel Guitar, standard tuning [tab] | instrument.fretted.guitar.steel.hawaiian.6-string.tab |
| 6-string Hawaiian Steel Guitar, alternate tuning [tab] | instrument.fretted.guitar.steel.hawaiian.6-string.tab.alternative |
| 6-string Hawaiian Steel Guitar, slack key Bb Mauna Loa tuning [tab] | instrument.fretted.guitar.steel.hawaiian.6-string.tab.bflat.mauna.loa |
| 6-string Hawaiian Steel Guitar, slack key C Mauna Loa tuning [tab] | instrument.fretted.guitar.steel.hawaiian.6-string.tab.c.mauna.loa |
| 6-string Hawaiian Steel Guitar, slack key Wahine CGDGBD tuning [tab] | instrument.fretted.guitar.steel.hawaiian.6-string.tab.cgdgbd.wahine |
| 6-string Hawaiian Steel Guitar, slack key Wahine CGDGBE tuning [tab] | instrument.fretted.guitar.steel.hawaiian.6-string.tab.cgdgbe.wahine |
| 6-string Hawaiian Steel Guitar, slack key Wahine DGDF\#BD tuning [tab] | instrument.fretted.guitar.steel.hawaiian.6-string.tab.dgdfbd.wahine |
| 6-string Hawaiian Steel Guitar, slack key G Mauna Loa tuning [tab] | instrument.fretted.guitar.steel.hawaiian.6-string.tab.g.mauna.loa |
| 6-string Hawaiian Steel Guitar, slack key G Taro Patch tuning [tab] | instrument.fretted.guitar.steel.hawaiian.6-string.tab.g.taro.patch |
| 6-string Hawaiian Steel Guitar, slack key Wahine GCDGBE tuning [tab] | instrument.fretted.guitar.steel.hawaiian.6-string.tab.gcdgbe.wahine |
| 8-string Hawaiian Steel Guitar [tab] | instrument.fretted.guitar.steel.hawaiian.8-string.tab |
| 8-string Hawaiian Steel Guitar, alternate tuning [tab] | instrument.fretted.guitar.steel.hawaiian.8-string.tab.alternative |
| Hawaiian Steel Guitar [tab] | instrument.fretted.guitar.steel.hawaiian.tab |
| Pedal Steel Guitar [notation] | instrument.fretted.guitar.steel.pedal.5lines |


| Pedal Steel Guitar [tab] | instrument.fretted.guitar.steel.pedal.tab |
| :---: | :---: |
| Guitarra [notation] | instrument.fretted.guitarra.5lines |
| Guitarra, Coimbra [tab] | instrument.fretted.guitarra.coimbra.tab |
| Guitarra, Lisboa [tab] | instrument.fretted.guitarra.lisboa.tab |
| Guitarra, Portuguesa [tab] | instrument.fretted.guitarra.portuguesa.tab |
| Guitarrón [notation] | instrument.fretted.guitarron.5lines |
| Guitarrón [tab] | instrument.fretted.guitarron.tab |
| Laúd [notation] | instrument.fretted.laud.5lines |
| Laúd [tab] | instrument.fretted.laud.tab |
| Tenor Lute [notation] | instrument.fretted.lute.5lines |
| Bass Lute [notation] | instrument.fretted.lute.bass-d.french.english.5lines |
| Bass Lute, D tuning, French/English [tab] | instrument.fretted.lute.bass-d.french.english.tab |
| Bass Lute, D tuning, Italian [tab] | instrument.fretted.lute.bass-d.italian.tab |
| Bass Lute, D tuning, Spanish [tab] | instrument.fretted.lute.bass-d.spanish.tab |
| Tenor Lute, G tuning, Italian [tab] | instrument.fretted.lute.italian.tab |
| Tenor Lute, G tuning, Spanish [tab] | instrument.fretted.lute.spanish.tab |
| Tenor Lute, G tuning, French/English [tab] | instrument.fretted.lute.tab |
| Tenor Lute, A tuning, French/English [tab] | instrument.fretted.lute.tenor-a.french.english.tab |
| Tenor Lute, A tuning, Italian [tab] | instrument.fretted.lute.tenor-a.italian.tab |
| Tenor Lute, A tuning, Spanish [tab] | instrument.fretted.lute.tenor-a.spanish.tab |
| Treble Lute [notation] | instrument.fretted.lute.treble-d.french.english.5lines |
| Treble Lute, D tuning, French/English [tab] | instrument.fretted.lute.treble-d.french.english.tab |
| Treble Lute, D tuning, Italian [tab] | instrument.fretted.lute.treble-d.talian.tab |
| Treble Lute, D tuning, Spanish [tab] | instrument.fretted.lute.treble-d.spanish.tab |
| Mandolin [notation] | instrument.fretted.mandolin.5lines |
| Mandolin [tab] | instrument.fretted.mandolin.tab |
| Oud [notation] | instrument.fretted.oud.5lines |
| Oud [tab] | instrument.fretted.oud.tab |
| Qanoon | instrument.fretted.qanoon.5lines |
| Requinto [notation] | instrument.fretted.requinto.5lines |
| Requinto [tab] | instrument.fretted.requinto.tab |
| Santoor | instrument.fretted.santoor.5lines |
| Sitar [notation] | instrument.fretted.sitar.5lines |
| Sitar (Ravi Shankar) [tab] | instrument.fretted.sitar.ravi-shankkar.tab |
| Sitar (Vilayat Khan) [tab] | instrument.fretted.sitar.vilayat-khan.tab |
| Tambura (Female) [notation] | instrument.fretted.tambura.female |
| Tambura (Male) [notation] | instrument.fretted.tambura.male |


| Tiple [notation] | instrument.fretted.tiple.5lines |
| :---: | :---: |
| Tiple, Argentina [tab] | instrument.fretted.tiple.argentina.tab |
| Tiple, Colombia ADF\#B tuning [tab] | instrument.fretted.tiple.colombia.tab.adfb |
| Tiple, Colombia DGBE tuning [tab] | instrument.fretted.tiple.colombia.tab.dgbe |
| Tiple, Cuba [tab] | instrument.fretted.tiple.cuba.tab |
| Tiple, Peru [tab] | instrument.fretted.tiple.peru.tab |
| Tiple, Santo Domingo [tab] | instrument.fretted.tiple.santo.domingo.tab |
| Tiple, Uruguay [tab] | instrument.fretted.tiple.uruguay.tab |
| Tres [notation] | instrument.fretted.tres.5lines |
| Tres, GCE tuning [tab] | instrument.fretted.tres.tab |
| Tres, ADF\# tuning [tab] | instrument.fretted.tres.tab.adf |
| Tres, GBE tuning [tab] | instrument.fretted.tres.tab.gbe |
| Ukulele [notation] | instrument.fretted.ukulele.5lines |
| Ukulele [tab] | instrument.fretted.ukulele.tab |
| Vihuela [notation] | instrument.fretted.vihuela.5lines |
| Vihuela [tab] | instrument.fretted.vihuela.tab |
| Zither | instrument.fretted.zither |
| Keyboard | instrument.keyboard |
| Accordion | instrument.keyboard.accordion |
| Bandoneon | instrument.keyboard.bandoneon |
| Celesta | instrument.keyboard.celesta |
| Clavichord | instrument.keyboard.clavichord |
| Harmonium | instrument.keyboard.harmonium |
| Harpsichord | instrument.keyboard.harpsichord |
| Keyboards | instrument.keyboard.keyboards |
| Tape Sampler Keyboard [Brass] | instrument.keyboard.tape sampler.brass |
| Tape Sampler Keyboard [Choir] | instrument.keyboard.tape sampler.choir |
| Tape Sampler Keyboard [Flute] | instrument.keyboard.tape sampler.flute |
| Tape Sampler Keyboard [Strings] | instrument.keyboard.tape sampler |
| Melodeon | instrument.keyboard.melodeon |
| Electric Organ | instrument.keyboard.organ.electric |
| Organ [manuals] | instrument.keyboard.organ.manuals |
| Manual [solo organ manuals] | instrument.keyboard.organ.manuals.solo |
| Ped. [Organ pedals] | instrument.keyboard.organ.pedals |
| Pedal [solo organ pedals] | instrument.keyboard.organ.pedals.solo |
| Piano | instrument.keyboard.piano |
| Electric Piano | instrument.keyboard.piano.electric |


| Electric Clavichord | instrument.keyboard.piano.electric.clavichord |
| :---: | :---: |
| Electric Stage Piano | instrument.keyboard.piano.electric.stage |
| Overdriven Electric Piano | instrument.keyboard.piano.electric.overdriven |
| Honky-tonk Piano | instrument.keyboard.piano.honky-tonk |
| Synthesizer | instrument.keyboard.synthesizer |
| Unnamed (2 lines) | instrument.other.2lines |
| Unnamed (3 lines) | instrument.other.3lines |
| Unnamed (4 lines) | instrument.other.4lines |
| Unnamed (bass staff) | instrument.other.bassclef |
| No instrument (barlines shown) | instrument.other.none.barlines |
| No instrument (bar rests shown) | instrument.other.none.barrests |
| No instrument (hidden) | instrument.other.none.hidden |
| Solo | instrument.other.solo.real |
| Unnamed (treble staff) | instrument.other.trebleclef |
| Almglocken | instrument.pitchedpercussion.almglocken |
| Antique Cymbals | instrument.pitchedpercussion.antiquecymbals |
| Chimes | instrument.pitchedpercussion.bells.chimes |
| Chimes [no key] | instrument.pitchedpercussion.bells.chimes.nokeysig |
| Bell lyre [marching band] | instrument.pitchedpercussion.bells.marching |
| Orchestral Bells | instrument.pitchedpercussion.bells.orchestral |
| Tubular Bells | instrument.pitchedpercussion.bells.tubular |
| Cimbalom | instrument.pitchedpercussion.cimbalom |
| Crotales | instrument.pitchedpercussion.crotales |
| Steel Drums | instrument.pitchedpercussion.drums.steel |
| Steel Drums [bass clef, treble transp.] | instrument.pitchedpercussion.drums.steel.bassclef |
| Gamelan Kengong | instrument.pitchedpercussion.gamelan.kengong |
| Gamelan Slentam | instrument.pitchedpercussion.gamelan.slentam |
| Glockenspiel | instrument.pitchedpercussion.glockenspiel |
| Alto Glockenspiel | instrument.pitchedpercussion.glockenspiel.alto |
| Soprano Glockenspiel | instrument.pitchedpercussion.glockenspiel.soprano |
| Handbells | instrument.pitchedpercussion.handbells |
| Harp | instrument.pitchedpercussion.harp |
| Lever Harp | instrument.pitchedpercussion.harp.lever |
| Kalimba | instrument.pitchedpercussion.kalimba |
| Marimba [grand staff] | instrument.pitchedpercussion.marimba |
| Marimba [treble staff] | instrument.pitchedpercussion.marimba.trebleclef |
| Alto Metallophone | instrument.pitchedpercussion.metallophone.alto |


| Bass Metallophone | instrument.pitchedpercussion.metallophone.bass |
| :---: | :---: |
| Soprano Metallophone | instrument.pitchedpercussion.metallophone.soprano |
| Roto-toms | instrument.pitchedpercussion.roto-toms |
| Temple Blocks | instrument.pitchedpercussion.templeblocks |
| Timpani [with key] | instrument.pitchedpercussion.timpani |
| Timpani [no key] | instrument.pitchedpercussion.timpani.nokeysig |
| Vibraphone | instrument.pitchedpercussion.vibraphone |
| Wood Blocks [5 lines] | instrument.pitchedpercussion.woodblocks |
| Xylophone | instrument.pitchedpercussion.xylophone |
| Alto Xylophone | instrument.pitchedpercussion.xylophone.alto |
| Bass Xylophone | instrument.pitchedpercussion.xylophone.bass |
| Contra Bass Bar | instrument.pitchedpercussion.xylophone.contrabass.bar |
| Gyil | instrument.pitchedpercussion.xylophone.gyil |
| Soprano Xylophone | instrument.pitchedpercussion.xylophone.soprano |
| Xylorimba | instrument.pitchedpercussion.xylorimba |
| Alto | instrument.singers.alto |
| Solo Alto | instrument.singers.alto.solo |
| Altus | instrument.singers.altus |
| Baritone | instrument.singers.baritone |
| Solo Baritone | instrument.singers.baritone.solo |
| Bass | instrument.singers.bass |
| Solo Bass | instrument.singers.bass.solo |
| Bassus | instrument.singers.bassus |
| Cantus | instrument.singers.cantus |
| Choir | instrument.singers.choir |
| Contralto | instrument.singers.contralto |
| Countertenor | instrument.singers.counter-tenor |
| Mean | instrument.singers.mean |
| Mezzo-soprano | instrument.singers.mezzo-soprano |
| Quintus | instrument.singers.quintus |
| Secundus | instrument.singers.secundus |
| Soprano | instrument.singers.soprano |
| Solo Soprano | instrument.singers.soprano.solo |
| Tenor | instrument.singers.tenor |
| Solo Tenor | instrument.singers.tenor.solo |
| Treble | instrument.singers.treble |
| Solo Treble | instrument.singers.treble.solo |


| Voice | instrument.singers.voice |
| :---: | :---: |
| Voice [male] | instrument.singers.voice.male |
| Contrabass | instrument.strings.contrabass |
| Bass [Double] | instrument.strings.contrabass.bass |
| Double Bass | instrument.strings.contrabass.double-bass |
| Solo Contrabass | instrument.strings.contrabass.solo |
| String Bass | instrument.strings.contrabass.string |
| Upright Bass | instrument.strings.contrabass.upright |
| Hurdy-gurdy | instrument.strings.hurdy-gurdy |
| Sarangi | instrument.strings.sarangi |
| Strings | instrument.strings.section |
| Strings [reduction] | instrument.strings.section.reduction |
| Bass Viol | instrument.strings.viol.bass |
| Tenor Viol | instrument.strings.viol.tenor |
| Treble Viol | instrument.strings.viol.treble |
| Viola | instrument.strings.viola |
| Solo Viola | instrument.strings.viola.solo |
| Violin 1 | instrument.strings.violin. 1 |
| Violin 2 | instrument.strings.violin. 2 |
| Violin I | instrument.strings.violin.I |
| Violin II | instrument.strings.violin.ii |
| Solo Violin | instrument.strings.violin.solo |
| Violoncello | instrument.strings.violoncello |
| Solo Violoncello | instrument.strings.violoncello.solo |
| Anvil | instrument.unpitched.anvil |
| Cha-cha bell [1 line] | instrument.unpitched.bells.cha-cha |
| Mambo bell [1 line] | instrument.unpitched.bells.mambo |
| Sleigh Bells | instrument.unpitched.bells.sleigh |
| Brake Drum [1 line] | instrument.unpitched.brake-drum. 1 line |
| Cabasa [1 line] | instrument.unpitched.cabasa |
| Cabasa [2 lines] | instrument.unpitched.cabasa.2lines |
| Castanets | instrument.unpitched.castanets |
| Shaker, Caxixi [1 line] | instrument.unpitched.caxixi. 1 line |
| Claves [1 line] | instrument.unpitched.claves |
| Shaker, Cocoa Bean Rattle [1 line] | instrument.unpitched.cocoa bean. 1 line |
| Finger Cymbals [1 line] | instrument.unpitched.cymbals.finger. 1 line |
| Percussion [1 line] | instrument.unpitched.drums. 1 line |


| Percussion [2 lines] | instrument.unpitched.drums.2lines |
| :---: | :---: |
| Berimbau | instrument.unpitched.drums.2lines.berimbau |
| Percussion [3 lines] | instrument.unpitched.drums.3lines |
| Percussion [4 lines] | instrument.unpitched.drums.4lines |
| Percussion [5 lines] | instrument.unpitched.drums.5lines |
| Agogos [2 lines] | instrument.unpitched.drums.agogos |
| Bass Drum | instrument.unpitched.drums.bass |
| Bass Drum [5 lines] | instrument.unpitched.drums.bass.5lines |
| Marching Bass Drum [3 lines] | instrument.unpitched.drums.bass.marching.3lines |
| Marching Bass Drum [5 lines] | instrument.unpitched.drums.bass.marching.5lines |
| Itótele [Batá Drum] | instrument.unpitched.drums.bata.itotele |
| Iyá [Batá Drum] | instrument.unpitched.drums.bata.iya |
| Okónkolo [Batá Drum] | instrument.unpitched.drums.bata.okonkolo |
| Bongos [2 lines] | instrument.unpitched.drums.bongos |
| Bongo Bell [High] | instrument.unpitched.drums.bongos.bell.high |
| Bongo Bell [Low] | instrument.unpitched.drums.bongos.bell.low |
| Box | instrument.unpitched.drums.box.3lines |
| Cajon [2 lines] | instrument.unpitched.drums.cajon |
| Congas [2 lines] | instrument.unpitched.drums.congas |
| Congas [1 line] | instrument.unpitched.drums.congas. 1 line |
| Congas [3 lines] | instrument.unpitched.drums.congas.3lines |
| Congas [4 lines] | instrument.unpitched.drums.congas.4lines |
| Cuíca [3 lines] | instrument.unpitched.drums.cuica.3lines |
| Cymbals | instrument.unpitched.drums.cymbal |
| Marching Cymbals [5 lines] | instrument.unpitched.drums.cymbals.marching.5lines |
| Djembe [3 lines] | instrument.unpitched.drums.djembe.3lines |
| Drum Set (Rock) | instrument.unpitched.drums.drumset |
| Drum Set (Alternative) | instrument.unpitched.drums.drumset.alternative |
| Drum Set (Brushes) | instrument.unpitched.drums.drumset.brushes |
| Drum Set (Dance) | instrument.unpitched.drums.drumset.dance |
| Drum Set (Disco) | instrument.unpitched.drums.drumset.disco |
| Drum Set (Electronica) | instrument.unpitched.drums.drumset.electronic |
| Drum Set (Fusion) | instrument.unpitched.drums.drumset.fusion |
| Drum Set (Garage) | instrument.unpitched.drums.drumset.garage |
| Drum Set (Hip-hop) | instrument.unpitched.drums.drumset.hip-hop |
| Drum Set (Industrial) | instrument.unpitched.drums.drumset.industrial |
| Drum Set (Jazz) | instrument.unpitched.drums.drumset.jazz |


| Drum Set (Lo-Fi) | instrument.unpitched.drums.drumset.lo-fi |
| :---: | :---: |
| Drum Set (Metal) | instrument.unpitched.drums.drumset.metal |
| Drum Set (Motown) | instrument.unpitched.drums.drumset.motown |
| Drum Set (New Age) | instrument.unpitched.drums.drumset.new age |
| Drum Set (Pop) | instrument.unpitched.drums.drumset.pop |
| Drum Set (Reggae) | instrument.unpitched.drums.drumset.reggae |
| Drum Set (Stadium Rock) | instrument.unpitched.drums.drumset.rock.stadium |
| Drum Set (Rods) | instrument.unpitched.drums.drumset.rods |
| Drum Set (Drum Machine) | instrument.unpitched.drums.drumset.tr-808 |
| Dumbek [3 lines] | instrument.unpitched.drums.dumbek.3lines |
| Kidi [Ewe Drum] | instrument.unpitched.drums.ewe.kidi |
| Sogo [Ewe Drum] | instrument.unpitched.drums.ewe.sogo |
| Gankokwe (Bell) | instrument.unpitched.drums.gankokwe |
| Jam Blocks [2 lines] | instrument.unpitched.drums.jamblocks |
| Jawbone [1 line] | instrument.unpitched.drums.jawbone. 1 line |
| Pandeiro [2 lines] | instrument.unpitched.drums.pandeiro |
| Rain Stick (High) [1 line] | instrument.unpitched.drums.rainstick.high. 1 line |
| Rain Stick (Low) [1 line] | instrument.unpitched.drums.rainstick.low. 1 line |
| Egg Shaker (High) [1 line] | instrument.unpitched.drums.shaker.high. 1 line |
| Egg Shaker (Low) [1 line] | instrument.unpitched.drums.shaker.low. 1 line |
| Egg Shaker (Medium) [1 line] | instrument.unpitched.drums.shaker.medium. 1 line |
| Side Drum | instrument.unpitched.drums.side |
| Snare Drum | instrument.unpitched.drums.snare |
| Marching Snare Drums [5 lines] | instrument.unpitched.drums.snare.5lines |
| Surdo [2 lines] | instrument.unpitched.drums.surdo |
| Tabla | instrument.unpitched.drums.table |
| Taiko Drum | instrument.unpitched.drums.taiko |
| Tenor Drum | instrument.unpitched.drums.tenor |
| Marching Tenor Drums [5 lines] | instrument.unpitched.drums.tenor.marching |
| Quads [5 lines] | instrument.unpitched.drums.tenor.marching.quads |
| Tom-toms [5 lines] | instrument.unpitched.drums.tom-toms |
| Tom-toms [4 lines] | instrument.unpitched.drums.tom-toms.4lines |
| Udu | instrument.unpitched.drums.udu |
| Shaker, Egg Shaker [1 line] | instrument.unpitched.egg shaker. 1 line |
| Finger Click [1 line] | instrument.unpitched.fingerclick |
| Gamelan Gong Ageng (High) [1 line] | instrument.unpitched.gamelan.gong-ageng.high |
| Gamelan Gong Ageng (Low) [1 line] | instrument.unpitched.gamelan.gong-ageng.low |


| Gamelan Kempyang and Ketuk [2 lines] | instrument.unpitched.gamelan.kempyang-ketuk |
| :---: | :---: |
| Gamelan Khendang Ageng [1 line] | instrument.unpitched.gamelan.khendang-ageng |
| Gamelan Khendang Ciblon [1 line] | instrument.unpitched.gamelan.khendang-ciblon |
| Large Gong [1 line] | instrument.unpitched.gong.large. 1 line |
| Medium Gong [1 line] | instrument.unpitched.gong.medium. 1 line |
| Gourd [1 line] | instrument.unpitched.gourd |
| Guira [1 line] | instrument.unpitched.guira |
| Guiro (High) [1 line] | instrument.unpitched.guiro.high |
| Guiro (Medium) [1 line] | instrument.unpitched.guiro.medium |
| Handclap [1 line] | instrument.unpitched.handclap |
| Shaker, Kayamba [1 line] | instrument.unpitched.kayamba. 1 line |
| Maracas | instrument.unpitched.maracas |
| Shaker, Gourd Maracas [1 line] | instrument.unpitched.maracas.gourd. 1 line |
| Maracas [High] | instrument.unpitched.maracas.high |
| Maracas [Medium] | instrument.unpitched.maracas.medium |
| Mark tree [1 line] | instrument.unpitched.marktree |
| Shaker, Nsak Rattle [1 line] | instrument.unpitched.nsak. 1 line |
| Finger Snaps | instrument.unpitched.orff.fingersnaps |
| Hand Claps | instrument.unpitched.orff.handclaps |
| Patsch | instrument.unpitched.orff.patsch |
| Stamp | instrument.unpitched.orff.stamp |
| Salsa bell [1 line] | instrument.unpitched.salsa.bell |
| Shaker [1 line] | instrument.unpitched.shaker |
| Shaker, Shekere [1 line] | instrument.unpitched.shekere. 1 line |
| Tam-tam | instrument.unpitched.tam-tam |
| Tambourine | instrument.unpitched.tambourine |
| Timbales [2 lines] | instrument.unpitched.timbales.2lines |
| Timbales [5 lines] | instrument.unpitched.timbales.5lines |
| Triangle | instrument.unpitched.triangle |
| Shaker, Wasembe Rattle (High) [1 line] | instrument.unpitched.wasembe.high. 1 line |
| Shaker, Wasembe Rattle (Low) [1 line] | instrument.unpitched.wasembe.low. 1 line |
| Shaker, Wasembe Rattle (Medium) [1 line] | instrument.unpitched.wasembe.medium. 1 line |
| Whip | instrument.unpitched.whip |
| Whistle | instrument.unpitched.whistle |
| Wind Chimes [1 line] | instrument.unpitched.wind-chimes. 1 line |
| Wood Block [1 line] | instrument.unpitched.woodblock. 1 line |
| Bagpipes | instrument.wind.bagpipe |


| Basset Horn | instrument.wind.basset-horn |
| :---: | :---: |
| Bassoon | instrument.wind.bassoon |
| Contrabassoon | instrument.wind.bassoon.contrabassoon |
| Quart Bassoon | instrument.wind.bassoon.quart |
| Quint Bassoon | instrument.wind.bassoon.quint |
| Clarinet in A | instrument.wind.clarinet.a |
| Clarinet in $\mathbf{A b}$ | instrument.wind.clarinet.aflat |
| Alto Clarinet in Eb | instrument.wind.clarinet.alto.eflat |
| Alto Clarinet in Eb [bass clef, treble transp.] | instrument.wind.clarinet.alto.eflat.bassclef |
| Bass Clarinet in Bb | instrument.wind.clarinet.bass.bflat |
| Bass Clarinet in Bb [score sounds 8vb] | instrument.wind.clarinet.bass.bflat.8vb-score |
| Bass Clarinet in Bb [bass clef, treble transp.] | instrument.wind.clarinet.bass.bflat.bassclef |
| Clarinet in $\mathbf{B b}$ | instrument.wind.clarinet.bflat |
| Clarinet in C | instrument.wind.clarinet.c |
| Contra Alto Clarinet in Eb | instrument.wind.clarinet.contra.alto.eflat |
| Contra Alto Clarinet in Eb [score sounds 8vb] | instrument.wind.clarinet.contra.alto.eflat.8vb-score |
| Contra Alto Clarinet in Eb [bass clef, treble transp.] | instrument.wind.clarinet.contra.alto.eflat.bassclef |
| Contrabass Clarinet in Bb | instrument.wind.clarinet.contrabass.bflat |
| Contrabass Clarinet in Bb [score sounds 15mb] | instrument.wind.clarinet.contrabass.bflat. 15 mb -score |
| Contrabass Clarinet in Bb [bass clef, treble transp.] | instrument.wind.clarinet.contrabass.bflat.bassclef |
| Clarinet in D | instrument.wind.clarinet.d |
| Clarinet in Eb | instrument.wind.clarinet.eflat |
| Clarinet in G | instrument.wind.clarinet.g |
| Cor Anglais | instrument.wind.coranglais |
| Didgeridoo | instrument.wind.didgeridoo |
| Duduk | instrument.wind.duduk |
| English Horn | instrument.wind.englishhorn |
| Flageolet | instrument.wind.flageolet |
| Flute | instrument.wind.flute |
| Alto Flute | instrument.wind.flute.alto |
| Bansuri | instrument.wind.flute.bansuri |
| Bass Flute | instrument.wind.flute.bass |
| Eb Flute | instrument.wind.flute.eflat |
| G Flute | instrument.wind.flute.g |
| Harmonica | instrument.wind.harmonica |
| Heckelphone | instrument.wind.heckelphone |
| Mey | instrument.wind.mey |


| Nai | instrument.wind.nai |
| :---: | :---: |
| Oboe | instrument.wind.oboe |
| Baritone Oboe | instrument.wind.oboe.baritone |
| Bass Oboe | instrument.wind.oboe.bass |
| Oboe d'Amore | instrument.wind.oboe.damore |
| Ocarina | instrument.wind.ocarina |
| Panpipes | instrument.wind.panpipes |
| Piccolo | instrument.wind.piccolo |
| Military Piccolo in Db | instrument.wind.piccolo.dflat |
| Alto Recorder | instrument.wind.recorder.alto |
| Bass Recorder | instrument.wind.recorder.bass |
| Great Bass Recorder | instrument.wind.recorder.bass.great |
| Contrabass Recorder | instrument.wind.recorder.contrabass |
| Descant Recorder | instrument.wind.recorder.descant |
| Sopranino Recorder | instrument.wind.recorder.sopranino |
| Soprano Recorder | instrument.wind.recorder.soprano |
| Tenor Recorder | instrument.wind.recorder.tenor |
| Treble Recorder | instrument.wind.recorder.treble |
| Alto Saxophone | instrument.wind.saxophone.alto |
| Baritone Saxophone | instrument.wind.saxophone.baritone |
| Baritone Saxophone [score sounds 8vb] | instrument.wind.saxophone.baritone.8vb-score |
| Baritone Saxophone [bass clef, treble transp.] | instrument.wind.saxophone.baritone.bassclef |
| Bass Saxophone | instrument.wind.saxophone.bass |
| Bass Saxophone [score sounds 15mb] | instrument.wind.saxophone.bass.15mb-score |
| Bass Saxophone [bass clef, treble transp.] | instrument.wind.saxophone.bass.bassclef |
| C Melody Saxophone | instrument.wind.saxophone.c-melody |
| Contrabass (Tubax) Saxophone | instrument.wind.saxophone.contrabass |
| Contrabass (Tubax) Saxophone [score sounds 15mb] | instrument.wind.saxophone.contrabass. 15 mb -score |
| Contrabass (Tubax) Sax [bass clef, treble transp.] | instrument.wind.saxophone.contrabass.bassclef |
| F Mezzo Soprano Saxophone | instrument.wind.saxophone.mezz-soprano.f |
| Sopranino Saxophone | instrument.wind.saxophone.sopranino |
| Piccolo Saxophone in Bb [Soprillo] | instrument.wind.saxophone.sopranino.bflat |
| Soprano Saxophone | instrument.wind.saxophone.soprano |
| C Soprano Saxophone | instrument.wind.saxophone.soprano.c |
| Subcontrabass (Tubax) Saxophone | instrument.wind.saxophone.subcontrabass |
| Subcontrabass (Tubax) Saxophone [score sounds 15mb] | instrument.wind.saxophone.subcontrabass. 15 mb -score |
| Subcontrabass (Tubax) Sax [bass clef, treble transp.] | instrument.wind.saxophone.subcontrabass.bassclef |


| Tenor Saxophone | instrument.wind.saxophone.tenor |
| :--- | :--- |
| Tenor Saxophone [score sounds 8vb] | instrument.wind.saxophone.tenor.8vb-score |
| Tenor Saxophone [bass clef, treble transp.] | instrument.wind.saxophone.tenor.bassclef |
| Woodwind | instrument.wind.section |
| Shakuhachi | instrument.wind.shakuhachi |
| Tin Whistle | instrument.wind.whistle.tin |

## Beam Options

For the Beam variable of NoteRest objects.

| NoBeam | 1 |
| :--- | :--- |
| StartBeam | 2 |
| ContinueBeam | 3 |
| SingleBeam | 4 |

## Bracket Types

For the AddBracket () method of BracketList objects, and the BracketType variable of Bracket objects.

| BracketFull | 0 |
| :--- | :--- |
| BracketBrace | 1 |
| BracketSub | 2 |

## Breaks

These constants are used by the SetBreakType () method of Score objects.

| MiddleOfSystem | 1 |
| :--- | :--- |
| EndOfSystem | 2 |
| MiddleOfPage | 3 |
| EndOfPage | 4 |
| NotEndOfSystem | 5 |
| EndOfSystemOrPage | 6 |
| Default | 7 |
| SpecialPageBreak | 8 |

These constants correspond to the menu entries in the Bars panel of the Properties window in the following way:
MiddleOfSystem Middle of system. The bar can only appear in the middle of a system, not at the end.
EndOfSystem No menu entry; created by Layout > Lock Format. The bar can only appear at the end of a mid-page system, not the middle of a system or the end of a page.

MiddleOfPage Middle of page. The bar can appear anywhere except at the end of a page.
EndOfPage Page break. The bar can only appear at the end of a page.
NotEndOfSystem No menu entry. The bar can appear anywhere except the end of a mid-page system.

EndOfSystemOrPage System break. The bar can only appear at the end of a mid-page system or the end of a page.
Default No break. The bar can appear anywhere.
Note that in older versions of ManuScript the constant MiddleOfSystem was called NoBreak and the constant EndOfSystem was called SystemBreak. These older names were confusing, because they implied a correlation with the similarly-named menu items in the Properties window that was not accurate. The old names are still supported for old plug-ins, but should not be used for new plug-ins. For consistency, the old constant PageBreak has also been renamed EndOfPage, even though this did correlate correctly with the Properties window.

## Accidentals

For the Accidental variable of Note objects.

| DoubleSharp | 2 |
| :--- | :--- |
| Sharp | 1 |
| Natural | 0 |
| Flat | -1 |
| DoubleFlat | -2 |
| QuarterSharp | 0.5 |
| ThreeQuarterSharp | 1.5 |
| QuarterFlat | -0.5 |
| ThreeQuarterFlat | -1.5 |

## Note Style Names

For the NoteStyle variable of Note objects; these correspond to the noteheads available from the Notes panel of the Properties window in the manuscript papers that are supplied with Sibelius.

| NormalNoteStyle | 0 | BackSlashedNoteStyle | 12 |
| :--- | :--- | :--- | :--- |
| CrossNoteStyle | 1 | ArrowDownNoteStyle | 13 |
| DiamondNoteStyle | 2 | ArrowUpNoteStyle | 14 |
| BeatWithoutStemNoteStyle | 3 | InvertedTriangleNoteStyle | 15 |
| BeatNoteStyle | 4 | ShapedNote1NoteStyle | 16 |
| CrossOrDiamondNoteStyle | 5 | ShapedNote2NoteStyle | 17 |
| BlackAndWhiteDiamondNoteStyle | 6 | ShapedNote3NoteStyle | 18 |
| HeadlessNoteStyle | 7 | ShapedNote4StemUpNoteStyle | 19 |
| StemlessNoteStyle | 8 | ShapedNote4StemDownNoteStyle | 23 |
| SilentNoteStyle | 9 | ShapedNote5NoteStyle | 20 |
| CueNoteStyle | 10 | ShapedNote6NoteStyle | 21 |
| SlashedNoteStyle | 11 | ShapedNote7NoteStyle | 22 |

## MuteMode Constants

These are the possible values of Stave. MuteMode:

| Muted | 0 |
| :--- | :--- |
| HalfMuted | 1 |
| NotMuted | 2 |

## Articulations

Used with Note. GetArticulation and Note.SetArticulation.

| Custom3Artic | 15 |
| :--- | :--- |
| TriPauseArtic | 14 |
| PauseArtic | 13 |
| SquarePauseArtic | 12 |
| Custom2Artic | 11 |
| DownBowArtic | 10 |
| UpBowArtic | 9 |
| PlusArtic | 8 |
| HarmonicArtic | 7 |
| MarcatoArtic | 6 |
| AccentArtic | 5 |
| TenutoArtic | 3 |
| WedgeArtic | 2 |
| StaccatissimoArtic | 1 |
| StaccatoArtic | 0 |
| Custom1Artic |  |

## SyllableTypes for LyricItems

Used in LyricItem.

| MiddleOfWord | 0 |
| :--- | :--- |
| EndOfWord | 1 |

## Accidental Styles

As used by Note. AccidentalStyle.

| NormalAcc | "0" |
| :--- | :--- |
| HiddenAcc | "1" |
| CautionaryAcc | "2" |
| BracketedAcc | "3" |

## Time Signature Strings

These define the unicode characters used to draw common time and alla breve time signatures, so that you can recognize these by comparison with TimeSignature. Text.

## CommonTimeString

## AllaBreveTimeString

## Symbols

There are a lot of symbols in Sibelius. We've defined named constants for the indices of some of the most frequently used symbols, which can be passed to Bar. AddSymbol. For other symbols, you can work out the required index by "counting along" in the Create > Symbol dialog of Sibelius, or by using the method Score. Symbol Index. To help with the "counting along," we've defined a constant for the start of every group of symbols in the Create > Symbol dialog, and these are also given below. Then for example you can access the 8va symbol as OctaveSymbols +2 .

| Common Symbol Indices |  |
| :---: | :---: |
| SegnoSymbol | "1" |
| CodaSymbol | "2" |
| RepeatBeatSymbol | "5" |
| RepeatBarSymbol | "6" |
| RepeatTwoBarsSymbol | "7" |
| TrillSymbol | "32" |
| BracketedTrillSymbol | "33" |
| MordentSymbol | "36" |
| InvertedMordentSymbol | "37" |
| TurnSymbol | "38" |
| InvertedTurnSymbol | "39" |
| ReversedTurnSymbol | "40" |
| TripleMordentSymbol | "41" |
| InvertedTripleMordentSymbol | "42" |
| PedalSymbol | "48" |
| PedaIPSymbol | "49" |
| PedalUpSymbol | "50" |
| LiftPedalSymbol | "51" |
| HeelOneSymbol | "52" |
| HeelTwoSymbol | "53" |
| ToeOneSymbol | "54" |
| ToeTwoSymbol | "55" |
| CommaSymbol | "247" |
| TickSymbol | "248" |
| CaesuraSymbol | "249" |
| ThickCaesuraSymbol | "250" |


| Indices at the Start of Each Group of Symbols |  |
| :---: | :---: |
| RepeatSymbols | "0" |
| GeneralSymbols | "16" |
| OrnamentSymbols | "32" |
| KeyboardSymbols | "48" |
| ChromaticPercussionSymbols | "64" |
| DrumPercussionSymbols | "80" |
| MetallicPercussionSymbols | "96" |
| OtherPercussionSymbols | "112" |
| BeaterPercussionSymbols | "128" |
| PercussionTechniqueSymbols | "160" |
| GuitarSymbols | "176" |
| ArticulationSymbols | "208" |
| AccidentalSymbols | "256" |
| NoteSymbols | "288" |
| NoteheadSymbols | "320" |
| RestSymbols | "368" |
| ConductorSymbols | "400" |
| ClefSymbols | "416" |
| OctaveSymbols | "448" |
| BreakSymbols | "464" |
| TechniqueSymbols | "480" |
| AccordionSymbols | "496" |
| HandbellSymbols | "528" |
| MiscellaneousSymbols | "544" |
|  | Constants |
| NormalSize | "0" |
| CueSize | "1" |
| GraceNoteSize | "2" |
| CueGraceNoteSize | "3" |

## Special Page Break Types

| NoPageBreak | "0" |
| :--- | :--- |
| MusicRestartsAfterXPages | $" 1 "$ |
| MusicRestartsOnNextLeftPage | "2" |
| MusicRestartsOnNextRightPage | "3" |

## Interval Types

| IntervalDiatonic | "-1" |
| :--- | :--- |
| Interval5xDiminished | "0" |
| Interval4xDiminished | $" 1 "$ |
| Interval3xDiminished | "2" |
| Interval2xDiminished | $" 3 "$ |
| IntervalDiminished | "4" |
| IntervalMinor | $" 4 "$ |
| IntervalMajor | $" 5 "$ |
| IntervalPerfect | $" 6 "$ |
| IntervalAugmented | $" 7 "$ |
| Interval2xAugmented | $" 8 "$ |
| Interval3xAugmented | $" 9 "$ |
| Interval4xAugmented | $" 10 "$ |
| Interval5xAugmented |  |

## InMultirest Values

| NoMultirest | "0" |
| :--- | :--- |
| StartsMultirest | "1" |
| EndsMultirest | "2" |
| MidMultirest | "3" |

## Page Number Visibility Values

| PageNumberShowAII | "0" |
| :--- | :--- |
| PageNumberHideFirst | $" 1 "$ |
| PageNumberHideAII | "2" |

## Page Number Format Values

| PageNumberFormatNormal | "0" |
| :--- | :--- |
| PageNumberFormatRomanUpper | "1" |
| PageNumberFormatRomanLower | "2" |
| PageNumberFormatLetterLower | "3" |

## Special Barlines

| SpecialBarlineStartRepeat | "0" |
| :--- | :--- |
| SpecialBarlineEndRepeat | $" 1 "$ |
| SpecialBarlineDashed | $" 2 "$ |
| SpecialBarlineDouble | $" 3 "$ |
| SpecialBarlineFinal | $" 4 "$ |
| SpecialBarlineInvisible | $" 5 "$ |
| SpecialBarlineBetweenStaves | $" 6 "$ |
| SpecialBarlineNormal | $" 7 "$ |
| SpecialBarlineTick | $" 8 "$ |
| SpecialBarlineShort | $" 10 "$ |
| SpecialBarlineDotted | $" 11 "$ |
| SpecialBarlineTriple | $" 12 "$ |
| SpecialBarlineThick |  |

## Bar Rest Type Values

| WholeBarRest | "0" |
| :--- | :--- |
| BreveBarRest | "1" |
| OneBarRepeat | "2" |
| TwoBarRepeat | "3" |
| FourBarRepeat | "4" |

## GuitarScaleDiagram Type Values

| ScaleTypeMajor | "0" |
| :--- | :--- |
| ScaleTypeMinor | "1" |
| ScaleTypeHarmonicMinor | "2" |
| ScaleTypeMelodicMinor | "3" |
| ScaleTypeDorian | "4" |
| ScaleTypePhrygian | "5" |
| ScaleTypeLydian | "6" |
| ScaleTypeMixolydian | "7" |
| ScaleTypeLocrian | "8" |
| ScaleTypeWholeTone | "9" |
| ScalrTypeDiminishedHalfWhole | $" 10 "$ |
| ScaleTypeDiminishedWholeHalf | $" 11 "$ |
| ScaleTypeAlteredDominant | $" 12 "$ |
| ScaleTypeLocrianSharp2 | $" 13 "$ |


| ScaleTypeLydianFlat7 | "14" |
| :--- | :--- |
| ScaleTypeMajorBebop | "15" |
| ScaleTypeDominantBebop | $" 16 "$ |
| ScaleTypeLydianSharp5 | $" 17 "$ |
| ScaleTypePhrygianDominant | $" 18 "$ |
| ScaleTypeAugmentedArpeggio | $" 19 "$ |
| ScaleTypeMajor7thArpeggio | "20" |
| ScaleType7thArpeggio | "21" |
| ScaleTypeMin7Flat5Arpeggio | "22" |
| ScaleTypeDiminished7thArpeggio | "23" |
| ScaleTypeMajorPentatonic | $" 24 "$ |
| ScaleTypeMinorPentatonic | $" 25 "$ |
| ScaleTypeOther | $" 26 "$ |

## FeatheredBeamType Values

For the FeatheredBeamType variable of NoteRest objects.

| FeatheredBeamNone | "0" |
| :--- | :--- |
| FeatheredBeamAccel | $" 1 "$ |
| FeatheredBeamRit | "2" |

## Units Values

FortheDocumentSetup object.

| DocumentSetupUnitsmm | "0" |
| :--- | :--- |
| DocumentSetupUnitsInches | $" 1 "$ |
| DocumentSetupUnitsPoints | $" 2 "$ |

## Orientation Values

For the Orientation variable of DocumentSetup objects.

| OrientationPortrait | "0" |
| :--- | :--- |
| OrientationLandscape | $" 1 "$ |

## PageSize Values

For the PageSize variable of DocumentSetup objects.

| PageSizeLetter | "0" |
| :--- | :--- |
| PageSizeTabloid | "1" |
| PageSizeA5 | "2" |
| PageSizeB5 | "3" |


| PageSizeA4 | "4" |
| :--- | :--- |
| PageSizeB4 | "5" |
| PageSizeA3 | "6" |
| PageSizeUSBand | "7" |
| PageSizeStatement | "8" |
| PageSizeHymn | "9" |
| PageSizeOctavo | "10" |
| PageSizeExecutive | $" 11 "$ |
| PageSizeQuarto | $" 12 "$ |
| PageSizeConcert | $" 13 "$ |
| PageSizeFolio | $" 14 "$ |
| PageSizeLegal | $" 15 "$ |
| PageSize9_5x12_5 | $" 16 "$ |
| PageSize10x13 | $" 17 "$ |
| PageSizeCustom | $" 18 "$ |

## MarginType Values

For the MarginType variable of DocumentSetup objects.

| PageMarginsSame | "0" |
| :--- | :--- |
| PageMarginsMirrored | "1" |
| PageMarginsDifferent | "2" |

## StaffScaleFactor Values

For the StaffScaleFactor variable of Staff objects.

| NormaISize | "0" |
| :--- | :--- |
| MediumSize | "1" |
| SmallSize | "2" |
| ExtraSmallSize | "3" |

## Tuplets

These define the constants that can be passed as a style parameter to Bar.AddTuplet() and Tuplet.AddNestedTuplet().

| TupletNoNumber | "0" |
| :--- | :--- |
| TupletLeft | "1" |
| TupletLeftRight | "2" |
| TupletLeftRightNote | "3" |

These define the constants that can be passed as a bracket parameter: "0"

| TupletBracketOn | "1" |
| :--- | :--- |
| TupletBracketAuto | "2" |

## SingleTremolos

For the SingleTremolos variable of NoteRest objects, the constants are numbers in the range 0 to 7 , representing the number of tremolo beams on the stem of the note or chord. For a "z on stem" (for buzz rolls), use the value -1 or the constant zonStem.

## DoubleTremolo Values

For the double tremolo style variables of EngravingRules objects.

| DoubleTremolosTouchingStems | "0" |
| :--- | :--- |
| DoubleTremolosBetweenStems | "1" |
| DoubleTremolosOuterTremoloTouchingStems | "2" |

## BarNumberFrequencyCategory Values

For the BarNumberFrequencyCategory variable of EngravingRules objects.

| EverySystem | "0" |
| :--- | :--- |
| NoBarNumbers | $" 1 "$ |
| EveryNthBar | "2" |

## Instrument Name Values

For the instrument name variables of EngravingRules objects.

| InstrumentNamesFull | "0" |
| :--- | :--- |
| InstrumentNamesShort | "1" |
| InstrumentNamesNone | "2" |

## Types of Objects in a Bar

The Type field for objects in a bar can return one of the following values:
Clef, SpecialBarline, TimeSignature, KeySignature
Line, ArpeggioLine, Bend, CrescendoLine, DiminuendoLine, GlissandoLine, OctavaLine, PedalLine, RepeatTimeLine, Slur, Trill, Box, BeamLine, Tuplet, RitardLine, HighLight

LyricItem, Text, SystemTextItem, GuitarFrame, GuitarScaleDiagram, RehearsalMark, InstrumentChange
BarRest, NoteRest, Graphic, Comment, Bracket, BarNumber
SymbolItem, SystemSymbolItem

## Interpreter Options

| TreatSingleCharacterAsString | When enabled, literals such as " 1 " will be considered as strings, not characters. That has several consequences: <br> - When consisting of a single digit, these will have the value of the digit, not the value of the ASCII character. <br> - In the case of non-digit values, like "a", they will be converted to 0 , as documented elsewhere. (Not as 97 as was the case before. If the ASCII value is required, one has to use the "Asc" method explicitly.) <br> - Result of comparison like "ManuScript" = "M" will now be false. |
| :---: | :---: |
| SupportHalfSemitonePitchValues | When enabled, floating-point values are accepted as "Pitch" so that scripts can add and manipulate quarter-tone values. Pitch is still specified and returned as a semitone, but 0.5 semitone is a quartertone. The same goes for Accidentals. The following methods accept floating-point pitches: <br> - NoteRest.AddNote <br> - Bar.AddNote <br> - Stave.AddNote <br> The following methods return pitches or accidentals in float-ing-point values: <br> - Note.Pitch <br> - Note.WrittenPitch <br> - Note.Accidental <br> - Note.WrittenAccidental |

Technical Support (USA) Visit the Online Support Center at www.avid.com/support

Product Information For company and product information, visit us on the web at www.avid.com


[^0]:    圊 Cross References point to related sections in this guide and other Avid documentation.

[^1]:    Radio Button Properties dialog

[^2]:    "O' The start pos and end pos you supply may be altered by ManuScript: Sibelius requires a passage selection to begin and end at a NoteRest if it doesn't encompass the entire bar.

[^3]:    "ใิ' If AddNestedTuplet() has been given illegal parameters, it will not be able to create a valid Tuplet object. Therefore, you should test for inequality of the returned Tuplet object with null before attempting to use it.

