Introduction

About Verovio

Verovio is open-source music notation engraving software. It supports files encoded using the Music Encoding Initiative, as well as MusicXML, Humdrum **kern , and Plaine and Easie. There are several versions of Verovio available: A command-line utility, a JavaScript library suitable for embedding notation on web-pages, and as a library for several other languages and platforms, including Python, iOS, Android, and Java. Verovio is cross-platorm and can be used in digital environments of various kind. The output of Verovio is beautifully engraved music notation rendered using the Scalable Vector Graphics (SVG) format.

While Verovio can be used as a core component for building powerful music notation editors, it is not an editor itself. There are a number of engraving options available to help control and adjust, to a fine level of detail, the specifics of the engraving.

Verovio uses the Standard Music Font Layout (SMuFL) as the basis for its notation shapes. This means that any SMuFL-compliant font may be used to customize and personalize the look of the engraved output.

About this book

This book is intended to serve as a reference guide for how to work with Verovio, and is meant for users of all skill levels. The book is a collaborative work that brings together inputs from the many contributors to the Verovio projects under the editorial leadership of the RISM Digital Center team.

[How to cite it?]

The first three sections will provide a number of tutorials, starting at the very basic and ending at advanced topics in notation. By the end of these sections you should have a very good understanding of how to use Verovio in its different forms, and how you can start to integrate it into your own work.

The following sections will cover the specifics of Verovio, serving as a reference for the operations and options available. It will also cover how to build Verovio from the source code, and how to contribute to the active development of Verovio.

Getting help

As you work through this book, from the most basic to the most advanced topics, you may find that you are struggling to understand something. The quickest and easiest way to get help is to reach out on the #verovio channel in the MEI Community's Slack chat. If you are not already a member, you can join.

Licensing

Verovio is licensed under the OSI-approved GNU Lesser General Public License (LGPLv3). This means that Verovio can be used in any contexts that are compliant with the requirements of that license. In this section, we explain more concretely what you can do with it in your project, but also what is required or not allowed for you to do, and what we additionally recommend.

What is allowed

The LGPLv3 license allows you to use the Verovio library as-is in open-source projects that are compliant with this license. It can also be used in commercial products that are open-source or not. It can be a web application, a desktop application or a mobile one. The Verovio library can be embedded in the product and shipped with it without having your product itself to be open-source as long as the Verovio library **is not modified** and is dynamically linked to your product.

What is required

Whichever use you make of the library, you have to give **visible credit** to the Verovio library. For a web application, it has to be through a prominent notice on your web-site. For a mobile application, it has to be given in the metadata of the application (e.g., iOS App Store or the Google Play store).

Here are some minimal examples to follow:

- NomadPlay web application and in the App Store
- Trala in the App Store

Using Verovio in a product without giving credit is a clear **license violation**. However, it is also important to understand that, by giving the appropriate credits, you are not only fulfilling the very basic and free-of-charge requirements of the license but also supporting the community by recognizing its work. This will help us make Verovio better and more sustainable and will be beneficial to all users - including you - in the long-run.

What is not allowed

You are not allowed to make any modifications to the Verovio library without making all of your **changes publicly available** and under the original LGPLv3 license. For example, if you improve the layout algorithm, or add support for additional music notation elements, these improvements must be made open-source under LGPLv3. Not doing it is also a **license violation** and is un-supportive of the community.

What is recommended

Providing credit if you use Verovio, and making the source code of your modifications to the Verovio library available to the community, are the only minimal legal requirements. However, we strongly encourage you to go one step further and to ask for your changes to be integrated into the original code-base of Verovio with a **pull-request** to the rism-digital/verovio repository. Before your changes can be integrated into the repository, we will need you to accept the Verovio Contributor License Agreement (CLA). This is a standard procedure for open-source projects and will allow for the community to benefit directly from your work.

We would also be happy to hear about your use of Verovio in your applications. Please get in touch if you are using Verovio, and let us know where we can learn more about your project!

Overview

There?

Tutorial 1: Web-based notation

Introduction

The first tutorial will look at how you can use Verovio to render music notation on a web page, using the pre-built JavaScript library. In this tutorial you will be building a small HTML page, with a minimal amount of JavaScript, to create an SVG rendering of an MEI file. In-depth technical expertise is not necessary, but you should be familiar with the basic principles of HTML to get the most out of this tutorial, and have access to a plain-text editor, preferably with facilities for automatically highlighting HTML and JavaScript code. (The Atom editor is a good choice if you need a recommendation.)

By the end of this tutorial, you should have a good understanding of the following:

1. How to load the Verovio JavaScript library using the <script> tag;
2. How to initialize Verovio, and how to set some basic rendering options;
3. How to load an MEI file from a URL and pass it to Verovio to render;
4. How to navigate between pages a multi-page score.

Later tutorials will cover more in-depth topics, such as how to have more control over rendering options, how to interact with the rendered notation, and how to play the notation back using MIDI.

Basic browser skills

A good skill to have in working through these tutorials is how to access and use the JavaScript error console in your browser. Every modern browser comes with this facility. This feature is useful to see what might be causing problems since these problems may not be otherwise noticeable; your page just may not work, or it may not do what you expect.

Accessing the JavaScript console is slightly different in each browser.

Chrome

Keyboard shortcut:

- Ctrl + Shift + J (Windows/Linux)
- Command + Option + J (Mac)

Menu location:

- Menu > More Tools > Developer Tools > Console tab

Chrome documentation

Firefox

Keyboard shortcut:

- Ctrl + Shift + K (Windows/Linux)
- Command + Option + K (Mac)

Menu location:

- Menu > Developer > Web Console

Firefox documentation

Internet Explorer / Edge

Keyboard shortcut: F12

Menu location: Menu “three dots” icon > F12 Developer Tools > Console tab

Edge documentation

Safari

Keyboard shortcut:

- Command + Option + C

Menu location:

The Safari developer tools must be enabled before use.

1. Safari > Preferences > Advanced > enable “Show Develop menu in menu bar”
2. Develop > Show Error Console

Safari documentation

Source

Getting started

To get started with Verovio, you need to load the JavaScript library in a web page. If you were building your own website, you may choose to host this on your own servers, but in this tutorial we will use a version that is hosted on the Verovio website.

You can start with the following HTML page:

HTML / JAVASCRIPT

```
<html>
  <head>
    <script src="http://www.verovio.org/javascript/latest/verovio-toolkit-wasm.js" defer></script>
  </head>
  <body>
    <h1>Hello Verovio!</h1>
    <div id="notation"></div>
  </body>
</html>
```

Save this in a plain text file somewhere on your hard-drive, and then open it with your browser. (The name does not matter, but it should end in `.html`; `verovio.html` is a good choice.) You should text in a large font that says “Hello Verovio!” but not much else. If you have your browser console open (discussed in the introduction), you should see no errors.

To start Verovio, you should add the following to your page in the head, after the `<script>` tag that loads the Verovio toolkit:

HTML / JAVASCRIPT

```
<script>
  document.addEventListener("DOMContentLoaded", (event) => {
      Module.onRuntimeInitialized = async _ => {
        let tk = new verovio.toolkit();
      }
  });
</script>
```

(If you are unsure, scroll to the bottom of this page; the full example is given below.)

When you refresh your page, you should still see nothing, and there should be no errors in the browser console. To help you understand what this is doing, let’s start from the inside out.

The line `tk = new verovio.toolkit();` creates a new instance of the Verovio toolkit. This is what we will eventually use to render the notation. However, we first need to wait until the Verovio library is fully downloaded and ready to use by your browser. The `Module.onRuntimeInitialized` line, and the `document.addEventListener` lines do just that – they tell your browser to wait until other things have happened before trying to work with Verovio. This is a good, safe way to ensure all the requirements are met before we try to start working with Verovio.

Logging to the Console

While you are developing, it can be useful to write little notes to yourself to let you know what types of data you have, or to see what is happening at any given point in your code. As you proceed to more advanced uses you may wish to explore the browser's built-in debugger, but until then a quick and easy way to do this is to use your browser's error console.

In your page, just after the line where you instantiate a new Verovio toolkit, insert the following:

```
console.log("Verovio has loaded!");
```

When you refresh your page, you can see this note to yourself appear in the browser console. If no other errors appear, this gives you a critical pieces of information: Your browser has reached that point in execution, which means it has successfully loaded and initialized Verovio. If you do not see this, go back through the examples to see where you may have gone wrong. If you still cannot find this, you can find the full example for this stage of the tutorial below.

End of Section 1

At the end of this first section you should have a working web page, with a message printed to your browser console, and no other errors showing up. In the next section we will look at how to load and render some basic music notation in this page.

Full example

HTML / JAVASCRIPT

```
<html>
  <head>
    <script src="http://www.verovio.org/javascript/latest/verovio-toolkit-wasm.js" defer></script>
    <script>
      document.addEventListener("DOMContentLoaded", (event) => {
          Module.onRuntimeInitialized = async _ => {
            let tk = new verovio.toolkit();
            console.log("Verovio has loaded!");
          }
      });
    </script>
  </head>
  <body>
    <h1>Hello Verovio!</h1>
    <div id="notation"></div>
  </body>
</html>
```

Basic rendering

At the end of part 1, we finished with a page that was successfully loading the Verovio library, but with nothing to display. In this part of the tutorial We will write some JavaScript that will fetch an MEI file from a URL, and then pass that MEI file to Verovio. This will turn the MEI file into an Scalable Vector Graphics (SVG) file that we can then embed in our page.

> *Scalable Vector Graphics* (SVG) is an image format that can be directly embedded into web pages. Vector graphics can be made larger or smaller with no pixellation, unlike other image formats you may be familiar with such as JPEG or PNG.

Fetching MEI with JavaScript

The first step is to fetch an MEI file from a URL. To do this, you can write the following in your HTML file, immediately after the `console.log` statement:

JAVASCRIPT

```
fetch("https://www.verovio.org/examples/downloads/Schubert_Lindenbaum.mei")
  .then( (response) => response.text() )
  .then( (meiXML) => {
    let svg = tk.renderData(meiXML, {});
    document.getElementById("notation").innerHTML = svg;
  });
```

To break this down a bit, we start with a `fetch` statement with a URL; this tells your browser to try and load the file available at this address from a remote server. If it's successful, then it should extract the XML data from the server: `then( (response) => response.text() )`.

Finally, we take this MEI response and pass it off to our Verovio instance. Remember that we 'started' Verovio by creating a new Toolkit and assigning it to the variable `tk`? Well, now we are using this toolkit to render the MEI file. The result, as you might guess by the variable name (`let svg = ...`), will be some SVG.

Once we have this SVG, we look through the page for HTML element with the `id` of "notation". You should see a `<div id="notation"></div>` line already in your HTML file. We set the content of this element (the `innerHTML`) to the SVG output of Verovio.

If you refresh your HTML page now, you should see a rendered version of a Schubert lied, "Der Lindenbaum". Congratulations! If you do not see this, go back and double-check that you do not have any errors in your browser console.

End of Section 2

At the end of this section, you should have a page with some rendered music notation on it. It's probably a bit too big, though, to read comfortably on your screen. You may also be wondering how Verovio handles larger scores, with lots of pages. We will answer these two questions in the next sections by looking at how we can control the layout options, and how we can use JavaScript to navigate the score dynamically.

Full example

HTML / JAVASCRIPT

```
<html>
  <head>
    <script src="http://www.verovio.org/javascript/latest/verovio-toolkit-wasm.js" defer></script>
    <script>
      document.addEventListener("DOMContentLoaded", (event) => {
          Module.onRuntimeInitialized = async _ => {
            let tk = new verovio.toolkit();
            console.log("Verovio has loaded!");

            fetch("https://www.verovio.org/examples/downloads/Schubert_Lindenbaum.mei")
              .then( (response) => response.text() )
              .then( (meiXML) => {
                let svg = tk.renderData(meiXML, {});
                document.getElementById("notation").innerHTML = svg;
              });
          }
      });
    </script>
  </head>
  <body>
    <h1>Hello Verovio!</h1>
    <div id="notation"></div>
  </body>
</html>
```

Layout options

Now that we have successfully rendered an MEI file to a web page, we can start to explore how to customize the SVG output. There are many possible options, most of which you will never need.

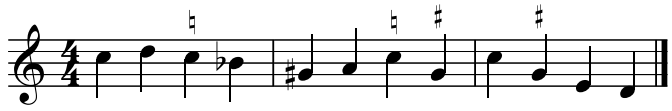
To start, we will first try and reduce the size of the image output, to demonstrate how we can scale the music notation to fit the screen.

Score navigation

Tutorial 2: Interactive notation

Introduction

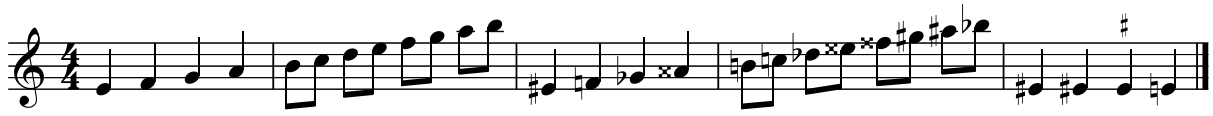
Here is one example - we show measure 1 and 3 with a separator



XML

```
<measure n="1">
  <staff n="1">
    <layer n="1">
      <note dur="4" oct="5" pname="c" stem.dir="down">
        <accid accid.ges="n"/>
      </note>
      <note dur="4" oct="5" pname="d" stem.dir="down">
        <accid accid.ges="n"/>
      </note>
      <note dur="4" oct="5" pname="c" stem.dir="down">
        <accid accid="n" func="edit"/>
      </note>
      <note dur="4" oct="4" pname="b" stem.dir="down">
        <accid accid="f"/>
      </note>
    </layer>
  </staff>
</measure>
<!-- ... -->
<measure right="end" n="3">
  <staff n="1">
    <layer n="1">
      <note dur="4" oct="5" pname="c" stem.dir="down">
        <accid accid.ges="n"/>
      </note>
      <note dur="4" oct="4" pname="g" stem.dir="up">
        <accid accid="s" func="edit"/>
      </note>
      <note dur="4" oct="4" pname="e" stem.dir="up">
        <accid accid.ges="n"/>
      </note>
      <note dur="4" oct="4" pname="d" stem.dir="up">
        <accid accid.ges="n"/>
      </note>
    </layer>
  </staff>
</measure>
```

Here is another one - we show measure 1 and 2 without separator



XML

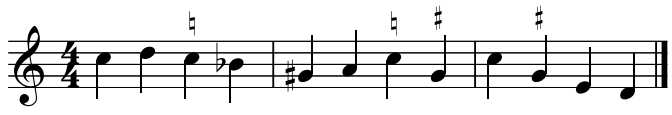
```
<measure n="1">
  <staff n="1">
    <layer n="1">
      <note dur="4" oct="4" pname="c" accid.ges="n"/>
      <note dur="4" oct="4" pname="d" accid.ges="n"/>
      <note dur="4" oct="4" pname="e" accid.ges="n"/>
      <note dur="4" oct="4" pname="f" accid.ges="n"/>
    </layer>
  </staff>
</measure>
<measure n="2">
  <staff n="1">
    <layer n="1">
      <beam>
        <note dur="8" oct="4" pname="g" accid.ges="n"/>
        <note dur="8" oct="4" pname="a" accid.ges="n"/>
      </beam>
      <beam>
        <note dur="8" oct="4" pname="b" accid.ges="n"/>
        <note dur="8" oct="5" pname="c" accid.ges="n"/>
      </beam>
      <beam>
        <note dur="8" oct="5" pname="d" accid.ges="n"/>
        <note dur="8" oct="5" pname="e" accid.ges="n"/>
      </beam>
      <beam>
        <note dur="8" oct="5" pname="f" accid.ges="n"/>
        <note dur="8" oct="5" pname="g" accid.ges="n"/>
      </beam>
    </layer>
  </staff>
</measure>
```

Inspecting the SVG

[Describe how to inspect the SVG output in the browser; useful to know for later chapters on how to style them!]

Working with CSS and SVG

Here is one example - show first note



XML

```
<measure n="1">
  <staff n="1">
    <layer n="1">
      <note dur="4" oct="5" pname="c" stem.dir="down">
        <accid accid.ges="n"/>
      </note>
      <note dur="4" oct="5" pname="d" stem.dir="down">
        <accid accid.ges="n"/>
      </note>
      <note dur="4" oct="5" pname="c" stem.dir="down">
        <accid accid="n" func="edit"/>
      </note>
      <note dur="4" oct="4" pname="b" stem.dir="down">
        <accid accid="f"/>
      </note>
    </layer>
  </staff>
</measure>
<!-- ... -->
<measure right="end" n="3">
  <staff n="1">
    <layer n="1">
      <note dur="4" oct="5" pname="c" stem.dir="down">
        <accid accid.ges="n"/>
      </note>
      <note dur="4" oct="4" pname="g" stem.dir="up">
        <accid accid="s" func="edit"/>
      </note>
      <note dur="4" oct="4" pname="e" stem.dir="up">
        <accid accid.ges="n"/>
      </note>
      <note dur="4" oct="4" pname="d" stem.dir="up">
        <accid accid.ges="n"/>
      </note>
    </layer>
  </staff>
</measure>
```

Here is another one - show first and last note

[SVG file is missing and need to be generated]

XML

[MEI file is missing and need to be generated]

XPath queries

Beyond tutorials: Advanced topics

Introduction

Transposition

SMuFL fonts

Embedded fonts

Selecting a font

Controlling the SVG output

HTML5

Towards SVG 2.0

Converting to PDF

Mensural notation

Duration alignment

Layout

Ligatures

Toolkit Reference

Input formats

MEI

humdrum

MusicXML

Plain and Easy

ABC

Output formats

SVG

MEI

MIDI

Timemap

Toolkit methods

Edit

Parse the editor actions passed as JSON string.

Only available for Emscripten-based compiles

Returns

bool

Parameters

Name	Type	Default	Description
json_editorAction	const std::string &	∅	

Original header

C++

```
bool vrv::Toolkit::Edit(const std::string &json_editorAction)
```

Example call

PYTHON

```
result = toolkit.edit(json_editorAction)
```

EditInfo

Returns

std::string

Original header

C++

std::string vrv::Toolkit::EditInfo()

Example call

PYTHON

result = toolkit.editInfo()

GetAvailableOptions

Returns

std::string

Original header

C++

std::string vrv::Toolkit::GetAvailableOptions() const

Example call

PYTHON

result = toolkit.getAvailableOptions()

More info here

Example how to extended the documentation for a method

GetElementAttr

Return element attributes as a JSON string.

Returns

std::string

Parameters

Name	Type	Default	Description
xmlId	const std::string &	∅	

Original header

C++

std::string vrv::Toolkit::GetElementAttr(const std::string &xmlId)

Example call

PYTHON

result = toolkit.getElementAttr(xmlId)

GetElementsAtTime

Returns array of IDs of elements being currently played.

Returns

std::string

Parameters

Name	Type	Default	Description
millisec	int	∅	

Original header

C++

std::string vrv::Toolkit::GetElementsAtTime(int millisec)

Example call

PYTHON

result = toolkit.getElementsAtTime(millisec)

GetExpansionIdsForElement

Returns a vector of ID strings of all elements (the notated and the expanded) for a given element.

Returns

std::string

Parameters

Name	Type	Default	Description
xmlId	const std::string &	∅	

Original header

C++

std::string vrv::Toolkit::GetExpansionIdsForElement(const std::string &xmlId)

Example call

PYTHON

result = toolkit.getExpansionIdsForElement(xmlId)

GetHumdrum

Returns

void

Parameters

Name	Type	Default	Description
output	std::ostream &	∅	

Original header

C++

void vrv::Toolkit::GetHumdrum(std::ostream &output)

Example call

PYTHON

toolkit.getHumdrum(output)

GetHumdrum

Returns

std::string

Original header

C++

std::string vrv::Toolkit::GetHumdrum()

Example call

PYTHON

result = toolkit.getHumdrum()

GetHumdrumBuffer

Returns

const char *

Original header

C++

const char* vrv::Toolkit::GetHumdrumBuffer()

Example call

PYTHON

result = toolkit.getHumdrumBuffer()

GetHumdrumFile

Returns

bool

Parameters

Name	Type	Default	Description
filename	const std::string &	∅	

Original header

C++

bool vrv::Toolkit::GetHumdrumFile(const std::string &filename)

Example call

PYTHON

result = toolkit.getHumdrumFile(filename)

GetInputFrom

Returns

int

Original header

C++

int vrv::Toolkit::GetInputFrom()

Example call

PYTHON

result = toolkit.getInputFrom()

GetLog

Concatenates the vrv::logBuffer into a string an returns it.

This is used only for Emscripten-based compilation. The vrv::logBuffer is filled by the vrv::LogXXX functions.

Returns

std::string

Original header

C++

std::string vrv::Toolkit::GetLog()

Example call

PYTHON

result = toolkit.getLog()

GetMEI

Get the MEI as a string.

Options (JSON) can be: pageNo: integer; (1-based), all pages if none (or 0) specified scoreBased: true false; true by default (noXmlIds: true false; false by default - remove all @xml:id not used in the data - not implemented)

Returns

std::string

Parameters

Name	Type	Default	Description
jsonOptions	const std::string &	∅	

Original header

C++

std::string vrv::Toolkit::GetMEI(const std::string &jsonOptions)

Example call

PYTHON

result = toolkit.getMEI(jsonOptions)

GetMIDIValuesForElement

Return MIDI values of the element with the ID (xml:id).

RenderToMidi() must be called prior to using this method.

Returns

std::string

Parameters

Name	Type	Default	Description
xmlId	const std::string &	∅	

Original header

C++

std::string vrv::Toolkit::GetMIDIValuesForElement(const std::string &xmlId)

Example call

PYTHON

```
result = toolkit.getMIDIValuesForElement(xmlId)
```

GetNotatedIdForElement

Returns the ID string of the notated (the original) element.

Returns

std::string

Parameters

Name	Type	Default	Description
xmlId	const std::string &	∅	

Original header

C++

```
std::string vrv::Toolkit::GetNotatedIdForElement(const std::string &xmlId)
```

Example call

PYTHON

```
result = toolkit.getNotatedIdForElement(xmlId)
```

GetOption

Returns

std::string

Parameters

Name	Type	Default	Description
option	const std::string &	∅	
defaultValue	bool	false	

Original header

C++

```
std::string vrv::Toolkit::GetOption(const std::string &option, bool defaultValue=false) const
```

Example call

PYTHON

```
result = toolkit.getOption(option, defaultValue)
```

GetOptions

Returns

std::string

Parameters

Name	Type	Default	Description
defaultValues	bool	∅	

Original header

C++

```
std::string vrv::Toolkit::GetOptions(bool defaultValues) const
```

Example call

PYTHON

```
result = toolkit.getOptions(defaultValues)
```

GetOptions

Return the Options object of the Toolkit instance.

Original header

C++

```
Options* vrv::Toolkit::GetOptions()
```

Example call

PYTHON

```
result = toolkit.getOptions()
```

GetOutputTo

Returns

int

Original header

C++

```
int vrv::Toolkit::GetOutputTo()
```

Example call

PYTHON

```
result = toolkit.getOutputTo()
```

GetPageCount

Returns

int

Original header

C++

```
int vrv::Toolkit::GetPageCount()
```

Example call

PYTHON

```
result = toolkit.getPageCount()
```

GetPageWithElement

Return the page on which the element is the ID (xml:id) is rendered.

This takes into account the current layout options. Returns 0 if no element is found.

Returns

int

Parameters

Name	Type	Default	Description
xmlId	const std::string &	∅	

Original header

C++

int vrv::Toolkit::GetPageWithElement(const std::string &xmlId)

Example call

PYTHON

result = toolkit.getPageWithElement(xmlId)

GetScale

Returns

int

Original header

C++

int vrv::Toolkit::GetScale()

Example call

PYTHON

result = toolkit.getScale()

GetTimeForElement

Return the time at which the element is the ID (xml:id) is played.

RenderToMidi() must be called prior to using this method. Returns 0 if no element is found.

Returns

int

Parameters

Name	Type	Default	Description
xmlId	const std::string &	∅	

Original header

C++

int vrv::Toolkit::GetTimeForElement(const std::string &xmlId)

Example call

PYTHON

result = toolkit.getTimeForElement(xmlId)

GetTimesForElement

Return a JSON object string with the following key values for a given note: scoreTimeOnset, scoreTimeOffset, scoreTimeTiedDuration, realTimeOnsetMilliseconds, realTimeOffsetMilliseconds, realTimeTiedDurationMilliseconds.

Returns 0 if no element is found.

Returns

std::string

Parameters

Name	Type	Default	Description
xmlId	const std::string &	∅	

Original header

C++

std::string vrv::Toolkit::GetTimesForElement(const std::string &xmlId)

Example call

PYTHON

result = toolkit.getTimesForElement(xmlId)

GetUuid

Return the ID of the Toolkit instance.

Returns

std::string

Original header

C++

std::string vrv::Toolkit::GetUuid()

Example call

PYTHON

result = toolkit.getUuid()

GetVersion

Returns the version number as a string.

This is used only for Emscripten-based compilation.

Returns

std::string

Original header

C++

std::string vrv::Toolkit::GetVersion()

Example call

PYTHON

result = toolkit.getVersion()

IdentifyInputFrom

Returns

FileFormat

Parameters

Name	Type	Default	Description
data	const std::string &	∅	

Original header

C++

FileFormat vrv::Toolkit::IdentifyInputFrom(const std::string &data)

Example call

PYTHON

result = toolkit.identifyInputFrom(data)

LoadData

Load a string data with the type previously specified in the options.

By default, the methods try to auto-detect the type.

Returns

bool

Parameters

Name	Type	Default	Description
data	const std::string &	∅	A string with the data (e.g., MEI data) to be loaded

Original header

C++

bool vrv::Toolkit::LoadData(const std::string &data)

Example call

PYTHON

result = toolkit.loadData(data)

LoadFile

Load a file from the file system.

Previously convert UTF16 files to UTF8 or extract files from MusicXML compressed files.

Returns

bool

Parameters

Name	Type	Default	Description
filename	const std::string &	∅	The filename to be loaded

Original header

C++

bool vrv::Toolkit::LoadFile(const std::string &filename)

Example call

PYTHON

result = toolkit.loadFile(filename)

LoadZipDataBase64

Load a MusicXML compressed file passed as base64 encoded string.

Returns

bool

Parameters

Name	Type	Default	Description
data	const std::string &	∅	A ZIP file in base64 encoded string

Original header

C++

bool vrv::Toolkit::LoadZipDataBase64(const std::string &data)

Example call

PYTHON

result = toolkit.loadZipDataBase64(data)

LoadZipDataBuffer

Load a MusicXML compressed file passed as a buffer of bytes.

True if loading the buffer succeed, false otherwise

Returns

bool – True if loading the buffer succeed, false otherwise

Parameters

Name	Type	Default	Description
data	const unsigned char *	∅	A ZIP file as a buffer of bytes
length	int	∅	The size of the data buffer

Original header

C++

bool vrv::Toolkit::LoadZipDataBuffer(const unsigned char *data, int length)

Example call

PYTHON

result = toolkit.loadZipDataBuffer(data, length)

RedoLayout

Redo the layout of the loaded data.

This can be called once the rendering option were changed, For example with a new page (sceen) height or a new zoom level.

Returns

void

Original header

C++

void vrv::Toolkit::RedoLayout()

Example call

PYTHON

toolkit.redoLayout()

RedoPagePitchPosLayout

Redo the layout of the pitch postitions of the current drawing page.

Only the note vertical positions are recalculated with this method. RedoLayout() needs to be called for a full recalculation.

Returns

void

Original header

C++

void vrv::Toolkit::RedoPagePitchPosLayout()

Example call

PYTHON

toolkit.redoPagePitchPosLayout()

RenderToDeviceContext

Render the page to the deviceContext.

Page number is 1-based.

Returns

bool

Parameters

Name	Type	Default	Description
pageNo	int	∅	
deviceContext	``	∅	

Original header

C++

bool vrv::Toolkit::RenderToDeviceContext(int pageNo, DeviceContext *deviceContext)

Example call

PYTHON

result = toolkit.renderToDeviceContext(pageNo, deviceContext)

RenderToMIDI

Creates a midi file, opens it, and returns it (base64 encoded).

Returns

std::string

Original header

C++

std::string vrv::Toolkit::RenderToMIDI()

Example call

PYTHON

result = toolkit.renderToMIDI()

RenderToMIDIFile

Creates a midi file, opens it, and writes to it.

currently generates a dummy midi file.

Returns

bool

Parameters

Name	Type	Default	Description
filename	const std::string &	∅	

Original header

C++

bool vrv::Toolkit::RenderToMIDIFile(const std::string &filename)

Example call

PYTHON

result = toolkit.renderToMIDIFile(filename)

RenderToPAE

Render the content to Plaine and Easie.

Only the top staff / layer is exported.

Returns

std::string

Original header

C++

std::string vrv::Toolkit::RenderToPAE()

Example call

PYTHON

result = toolkit.renderToPAE()

RenderToPAEFile

Export the content to a Plaine and Easie file.

Returns

bool

Parameters

Name	Type	Default	Description
filename	const std::string &	∅	

Original header

C++

bool vrv::Toolkit::RenderToPAEFile(const std::string &filename)

Example call

PYTHON

result = toolkit.renderToPAEFile(filename)

RenderToSVG

Render the page in SVG and returns it as a string.

Page number is 1-based

Returns

std::string

Parameters

Name	Type	Default	Description
pageNo	int	1	
xml_declaration	bool	false	

Original header

C++

std::string vrv::Toolkit::RenderToSVG(int pageNo=1, bool xml_declaration=false)

Example call

PYTHON

result = toolkit.renderToSVG(pageNo, xml_declaration)

RenderToSVGFile

Render the page in SVG and save it to the file.

Page number is 1-based.

Returns

bool

Parameters

Name	Type	Default	Description
filename	const std::string &	∅	
pageNo	int	1	

Original header

C++

bool vrv::Toolkit::RenderToSVGFile(const std::string &filename, int pageNo=1)

Example call

PYTHON

result = toolkit.renderToSVGFile(filename, pageNo)

RenderToTimemap

Creates a timemap file, and return it as a JSON string.

Returns

std::string

Original header

C++

std::string vrv::Toolkit::RenderToTimemap()

Example call

PYTHON

result = toolkit.renderToTimemap()

RenderToTimemapFile

Returns

bool

Parameters

Name	Type	Default	Description
filename	const std::string &	∅	

Original header

C++

bool vrv::Toolkit::RenderToTimemapFile(const std::string &filename)

Example call

PYTHON

result = toolkit.renderToTimemapFile(filename)

SaveFile

Save an MEI file.

This is a lond description for Save.

Returns

bool

Parameters

Name	Type	Default	Description
filename	const std::string &	∅	This parameter is the filename
jsonOptions	const std::string &	∅	There are the options. It cannot be null

Original header

C++

bool vrv::Toolkit::SaveFile(const std::string &filename, const std::string &jsonOptions)

Example call

PYTHON

result = toolkit.saveFile(filename, jsonOptions)

SetHumdrumBuffer

Returns

void

Parameters

Name	Type	Default	Description
contents	const char *	∅	

Original header

C++

void vrv::Toolkit::SetHumdrumBuffer(const char *contents)

Example call

PYTHON

toolkit.setHumdrumBuffer(contents)

SetInputFrom

Returns

bool

Parameters

Name	Type	Default	Description
inputFrom	std::string const &	∅	

Original header

C++

bool vrv::Toolkit::SetInputFrom(std::string const &inputFrom)

Example call

PYTHON

result = toolkit.setInputFrom(inputFrom)

SetInputFrom

Returns

void

Parameters

Name	Type	Default	Description
format	FileFormat	∅	

Original header

C++

void vrv::Toolkit::SetInputFrom(FileFormat format)

Example call

PYTHON

toolkit.setInputFrom(format)

SetOption

Returns

bool

Parameters

Name	Type	Default	Description
option	const std::string &	∅	
value	const std::string &	∅	

Original header

C++

bool vrv::Toolkit::SetOption(const std::string &option, const std::string &value)

Example call

PYTHON

result = toolkit.setOption(option, value)

SetOptions

Returns

bool

Parameters

Name	Type	Default	Description
jsonOptions	const std::string &	∅	

Original header

C++

bool vrv::Toolkit::SetOptions(const std::string &jsonOptions)

Example call

PYTHON

result = toolkit.setOptions(jsonOptions)

SetOutputTo

Returns

bool

Parameters

Name	Type	Default	Description
outputTo	std::string const &	∅	

Original header

C++

bool vrv::Toolkit::SetOutputTo(std::string const &outputTo)

Example call

PYTHON

result = toolkit.setOutputTo(outputTo)

SetResourcePath

Set the resource path for the Toolkit instance.

This method needs to be called if the constructor had initFont=false or if the resource path needs to be changed.

Returns

bool

Parameters

Name	Type	Default	Description
path	const std::string &	∅	The path to the resource directory

Original header

C++

bool vrv::Toolkit::SetResourcePath(const std::string &path)

Example call

PYTHON

result = toolkit.setResourcePath(path)

SetScale

Returns

bool

Parameters

Name	Type	Default	Description
scale	int	∅	

Original header

C++

bool vrv::Toolkit::SetScale(int scale)

Example call

PYTHON

result = toolkit.setScale(scale)

Toolkit

If initFont is set to false, Resources::InitFonts will have to be called explicitely.

Parameters

Name	Type	Default	Description
initFont	bool	true	

Original header

C++

vrv::Toolkit::Toolkit(bool initFont=true)

Example call

PYTHON

result = toolkit.toolkit(initFont)

Toolkit options

Base short options

All of the base options are short options in the command-line version of the toolkit. Most of them are command-line options that have no direct corresponding JSON key.

Name and parameter	Description	See also
-a, --all-pages	Output all pages	
-h, --help	Display this message	
-f, --input-from <s>	Select input format from: “abc”, “darms”, “humdrum”, “mei”, “pae”, “xml” (musicxml) (default: “mei”)	Input formats
-o, --outfile <s>	Output file name (use “-“ as file name for standard output) (default: “svg”)	
-t, --output-to <s>	Select output format to: “mei”, “pb-mei”, “svg”, or “midi” (default: “svg”)	Output formats
-p, --page <i>	Select the page to engrave (default is 1)	
-r, --resource-path <s>	Path to the directory with Verovio resources (default: “/usr/local/share/verovio”)	SetResourcePath Building the toolkit
-s, --scale <i>	Scale of the output in percent (default: 100; min: 1; max: 1000)	
- , --stdin	Use “-“ as input file or set the “–stdin” option for reading from the standard input	
-v, --version	Display the version number	
-x, --xml-id-seed <i>	Seed the random number generator for XML IDs (default is random)	

Input and page layout options

Name and parameter	Description	See also
--adjust-page-height	Adjust the page height to the height of the content	
--adjust-page-width	Adjust the page width to the width of the content	
--breaks <s>	Define page and system breaks layout (default: “auto”; other values: [‘none’, ‘auto’, ‘line’, ‘smart’, ‘encoded’])	
--breaks-smart-sb <f>	In smart breaks mode, the portion of system width usage at which an encoded sb will be used (default: 0.66; min: 0.0; max: 1.0)	
--clef-change-factor <f>	Set the ratio of normal clefs to changing clefs (default: 0.66; min: 0.25; max: 1.0)	
--condense <s>	Control condensed score layout (default: “auto”; other values: [‘none’, ‘auto’, ‘encoded’])	
--condense-first-page	When condensing a score also condense the first page	
--condense-tempo-pages	When condensing a score also condense pages with a tempo change	
--even-note-spacing	Specify the linear spacing factor	
--expand <s>	Expand all referenced elements in the expansion (default: "")	

Name and parameter	Description	See also
--footer <s>	Control footer layout (default: “auto”; other values: [‘none’, ‘auto’, ‘encoded’, ‘always’])	
--header <s>	Control header layout (default: “auto”; other values: [‘none’, ‘auto’, ‘encoded’])	
--hum-type	Include type attributes when importing from Humdrum	
--justify-vertically	Justify spacing vertically to fill the page	
--landscape	The landscape paper orientation flag	
--mensural-to-measure	Convert mensural sections to measure-based MEI	Ligatures Layout
--min-last-justification <f>	The last system is only justified if the unjustified width is greater than this percent (default: 0.8; min: 0.0; max: 1.0)	
--mm-output	Specify that the output in the SVG is given in mm (default is px)	
--no-justification	Do not justify the system	
--open-control-events	Render open control events	
--output-indent <i>	Output indentation value for MEI and SVG (default: 3; min: 1; max: 10)	
--output-indent-tab	Output indentation with tabulation for MEI and SVG	
--output-smufl-xml-entities	Output SMuFL charachters as XML entities instead of byte codes	
--page-height <i>	The page height (default: 2970; min: 100; max: 60000)	
--page-margin-bottom <i>	The page bottom margin (default: 50; min: 0; max: 500)	
--page-margin-left <i>	The page left margin (default: 50; min: 0; max: 500)	
--page-margin-right <i>	The page right margin (default: 50; min: 0; max: 500)	
--page-margin-top <i>	The page top margin (default: 50; min: 0; max: 500)	
--page-width <i>	The page width (default: 2100; min: 100; max: 60000)	
--remove-ids	Remove XML IDs in the MEI output that are not referenced	
--shrink-to-fit	Scale down page content to fit the page height if needed	
--svg-bounding-boxes	Include bounding boxes in SVG output	
--svg-format-raw	Writes SVG out with no line indenting or non-content newlines.	
--svg-html5	Write data-id and data-class attributes for JS usage and id clash avoidance.	
--svg-remove-xlink	Removes the xlink: prefix on href attributes for compatibility with some newer browsers.	
--svg-view-box	Use viewBox on svg root element for easy scaling of document	
--unit <i>	The MEI unit (1⁄2 of the distance between the staff lines) (default: 9; min: 6; max: 20)	
--use-brace-glyph	Use brace glyph from current font	

Name and parameter	Description	See also
--use-facsimile	Use information in the element to control the layout	
--use-pg-footer-for-all	Use the pgFooter for all pages	
--use-pg-header-for-all	Use the pgHeader for all pages	

General layout options

Name and parameter	Description	See also
--bar-line-separation <f>	The default distance between multiple barlines when locked together (default: 0.8; min: 0.5; max: 2.0)	
--bar-line-width <f>	The barLine width (default: 0.3; min: 0.1; max: 0.8)	
--beam-max-slope <i>	The maximum beam slope (default: 10; min: 1; max: 20)	
--beam-min-slope <i>	The minimum beam slope	
--bracket-thickness <f>	The thickness of the system bracket (default: 1.0; min: 0.5; max: 2.0)	
--engraving-defaults <s>	Path to json file describing defaults for engraving SMuFL elements	
--font <s>	Set the music font (default: “Leipzig”)	
--grace-factor <f>	The grace size ratio numerator (default: 0.75; min: 0.5; max: 1.0)	
--grace-rhythm-align	Align grace notes rhythmically with all staves	
--grace-right-align	Align the right position of a grace group with all staves	
--hairpin-size <f>	The haripin size in MEI units (default: 3.0; min: 1.0; max: 8.0)	
--hairpin-thickness <f>	The thickness of the hairpin (default: 0.2; min: 0.1; max: 0.8)	
--justification-brace-group <f>	Space between staves inside a braced group ijustification (default: 1.0; min: 0.0; max: 10.0)	
--justification-bracket-group <f>	Space between staves inside a bracketed group justification (default: 1.0; min: 0.0; max: 10.0)	
--justification-staff <f>	The staff justification (default: 1.0; min: 0.0; max: 10.0)	
--justification-system <f>	The system spacing justification (default: 1.0; min: 0.0; max: 10.0)	
--ledger-line-extension <f>	The amount by which a ledger line should extend either side of a notehead (default: 0.54; min: 0.2; max: 1.0)	
--ledger-line-thickness <f>	The thickness of the ledger lines (default: 0.25; min: 0.1; max: 0.5)	
--lyric-hyphen-length <f>	The lyric hyphen and dash length (default: 1.2; min: 0.5; max: 3.0)	

Name and parameter	Description	See also
--lyric-line-thickness <f>	The lyric extender line thickness (default: 0.25; min: 0.1; max: 0.5)	
--lyric-no-start-hyphen	Do not show hyphens at the beginning of a system	
--lyric-size <f>	The lyrics size in MEI units (default: 4.5; min: 2.0; max: 8.0)	
--lyric-top-min-margin <f>	The minmal margin above the lyrics in MEI units (default: 2.0; min: 0.0; max: 8.0)	
--lyric-word-space <f>	The lyric word space length (default: 1.2; min: 0.5; max: 3.0)	
--measure-number <s>	The measure numbering rule (unused) (default: “system”; other values: [‘system’, ‘interval’])	
--midi-tempo-adjustment <f>	The MIDI tempo adjustment factor (default: 1.0; min: 0.2; max: 4.0)	
--min-measure-width <i>	The minimal measure width in MEI units (default: 15; min: 1; max: 30)	
--repeat-bar-line-dot-separation <f>	The default horizontal distance between the dots and the inner barline of a repeat barline (default: 0.3; min: 0.1; max: 1.0)	
--repeat-ending-line-thickness <f>	Repeat and ending line thickness (default: 0.15; min: 0.1; max: 2.0)	
--slur-control-points <i>	Slur control points - higher value means more curved at the end (default: 5; min: 1; max: 10)	
--slur-curve-factor <i>	Slur curve factor - high value means rounder slurs (default: 10; min: 1; max: 100)	
--slur-endpoint-thickness <f>	The Endpoint slur thickness in MEI units (default: 0.1; min: 0.05; max: 0.25)	
--slur-height-factor <i>	Slur height factor - high value means flatter slurs (default: 5; min: 1; max: 100)	
--slur-max-height <f>	The maximum slur height in MEI units (default: 3.0; min: 2.0; max: 6.0)	
--slur-max-slope <i>	The maximum slur slope in degrees (default: 20; min: 0; max: 60)	
--slur-midpoint-thickness <f>	The midpoint slur thickness in MEI units (default: 0.6; min: 0.2; max: 1.2)	
--slur-min-height <f>	The minimum slur height in MEI units (default: 1.2; min: 0.3; max: 2.0)	
--spacing-brace-group <i>	Minimum space between staves inside a braced group in MEI units (default: 12; min: 0; max: 48)	
--spacing-bracket-group <i>	Minimum space between staves inside a bracketed group in MEI units (default: 12; min: 0; max: 48)	
--spacing-dur-detection	Detect long duration for adjusting spacing	
--spacing-linear <f>	Specify the linear spacing factor (default: 0.25; min: 0.0; max: 1.0)	

Name and parameter	Description	See also
--spacing-non-linear <f>	Specify the non-linear spacing factor (default: 0.6; min: 0.0; max: 1.0)	
--spacing-staff <i>	The staff minimal spacing in MEI units (default: 12; min: 0; max: 48)	
--spacing-system <i>	The system minimal spacing in MEI units (default: 12; min: 0; max: 48)	
--staff-line-width <f>	The staff line width in unit (default: 0.15; min: 0.1; max: 0.3)	
--stem-width <f>	The stem width (default: 0.2; min: 0.1; max: 0.5)	
--sub-bracket-thickness <f>	The thickness of system sub-bracket (default: 0.2; min: 0.1; max: 2.0)	
--system-divider <s>	The display of system dividers (default: “auto”; other values: [‘none’, ‘auto’, ‘left’, ‘left-right’])	
--system-max-per-page <i>	Maximun number of systems per page	
--text-enclosure-thickness <f>	The thickness of the line text enclosing box (default: 0.2; min: 0.1; max: 0.8)	
--thick-barline-thickness <f>	The thickness of the thick barline (default: 1.0; min: 0.5; max: 2.0)	
--tie-endpoint-thickness <f>	The Endpoint tie thickness in MEI units (default: 0.1; min: 0.05; max: 0.25)	
--tie-midpoint-thickness <f>	The midpoint tie thickness in MEI units (default: 0.5; min: 0.2; max: 1.0)	
--tuplet-bracket-thickness <f>	The thickness of the tuplet bracket (default: 0.2; min: 0.1; max: 0.8)	
--tuplet-num-head	Placement of tuplet number on the side of the note head	

Element selectors and processing

Name and parameter	Description	See also
--app-x-path-query * <s>	Set the xPath query for selecting child elements, for example: "./rdg[contains(@source, 'source-id')]"; by default the or the first is selected	
--choice-x-path-query * <s>	Set the xPath query for selecting child elements, for example: "./orig"; by default the first child is selected	
--mdiv-x-path-query <s>	Set the xPath query for selecting the to be rendered; only one can be rendered (default: "")	
--subst-x-path-query * <s>	Set the xPath query for selecting child elements, for example: "./del"; by default the first child is selected	
--transpose <s>	SUMMARY (default: “”)	Transposition
--transpose-selected-only	Transpose only the selected content and ignore unselected editorial content	

Element margins

Name and parameter	Description	See also
--bottom-margin-artic <f>	The margin for artic in MEI units (default: 0.75; min: 0.0; max: 10.0)	
--bottom-margin-harm <f>	The margin for harm in MEI units (default: 1.0; min: 0.0; max: 10.0)	
--bottom-margin-header <f>	The margin for header in MEI units (default: 8.0; min: 0.0; max: 24.0)	
--default-bottom-margin <f>	The default bottom margin (default: 0.5; min: 0.0; max: 5.0)	
--default-left-margin <f>	The default left margin (default: 0.0; min: 0.0; max: 2.0)	
--default-right-margin <f>	The default right margin (default: 0.0; min: 0.0; max: 2.0)	
--default-top-margin <f>	The default top margin (default: 0.5; min: 0.0; max: 6.0)	
--left-margin-accid <f>	The margin for accid in MEI units (default: 1.0; min: 0.0; max: 2.0)	
--left-margin-bar-line <f>	The margin for barLine in MEI units (default: 0.0; min: 0.0; max: 2.0)	
--left-margin-beat-rpt <f>	The margin for beatRpt in MEI units (default: 2.0; min: 0.0; max: 2.0)	
--left-margin-chord <f>	The margin for chord in MEI units (default: 1.0; min: 0.0; max: 2.0)	
--left-margin-clef <f>	The margin for clef in MEI units (default: 1.0; min: 0.0; max: 2.0)	
--left-margin-key-sig <f>	The margin for keySig in MEI units (default: 1.0; min: 0.0; max: 2.0)	
--left-margin-left-bar-line <f>	The margin for left barLine in MEI units (default: 1.0; min: 0.0; max: 2.0)	
--left-margin-m-rest <f>	The margin for mRest in MEI units (default: 0.0; min: 0.0; max: 2.0)	
--left-margin-m-rpt2 <f>	The margin for mRpt2 in MEI units (default: 0.0; min: 0.0; max: 2.0)	
--left-margin-mensur <f>	The margin for mensur in MEI units (default: 1.0; min: 0.0; max: 2.0)	
--left-margin-meter-sig <f>	The margin for meterSig in MEI units (default: 1.0; min: 0.0; max: 2.0)	
--left-margin-multi-rest <f>	The margin for multiRest in MEI units (default: 0.0; min: 0.0; max: 2.0)	
--left-margin-multi-rpt <f>	The margin for multiRpt in MEI units (default: 0.0; min: 0.0; max: 2.0)	
--left-margin-note <f>	The margin for note in MEI units (default: 1.0; min: 0.0; max: 2.0)	
--left-margin-rest <f>	The margin for rest in MEI units (default: 1.0; min: 0.0; max: 2.0)	

Name and parameter	Description	See also
--left-margin-right-bar-line <f>	The margin for right barLine in MEI units (default: 1.0; min: 0.0; max: 2.0)	
--right-margin-accid <f>	The right margin for accid in MEI units (default: 0.0; min: 0.0; max: 2.0)	
--right-margin-bar-line <f>	The right margin for barLine in MEI units (default: 0.0; min: 0.0; max: 2.0)	
--right-margin-beat-rpt <f>	The right margin for beatRpt in MEI units (default: 0.0; min: 0.0; max: 2.0)	
--right-margin-chord <f>	The right margin for chord in MEI units (default: 0.0; min: 0.0; max: 2.0)	
--right-margin-clef <f>	The right margin for clef in MEI units (default: 1.0; min: 0.0; max: 2.0)	
--right-margin-key-sig <f>	The right margin for keySig in MEI units (default: 1.0; min: 0.0; max: 2.0)	
--right-margin-left-bar-line <f>	The right margin for left barLine in MEI units (default: 1.0; min: 0.0; max: 2.0)	
--right-margin-m-rest <f>	The right margin for mRest in MEI units (default: 0.0; min: 0.0; max: 2.0)	
--right-margin-m-rpt2 <f>	The right margin for mRpt2 in MEI units (default: 0.0; min: 0.0; max: 2.0)	
--right-margin-mensur <f>	The right margin for mensur in MEI units (default: 1.0; min: 0.0; max: 2.0)	
--right-margin-meter-sig <f>	The right margin for meterSig in MEI units (default: 1.0; min: 0.0; max: 2.0)	
--right-margin-multi-rest <f>	The right margin for multiRest in MEI units (default: 0.0; min: 0.0; max: 2.0)	
--right-margin-multi-rpt <f>	The right margin for multiRpt in MEI units (default: 0.0; min: 0.0; max: 2.0)	
--right-margin-note <f>	The right margin for note in MEI units (default: 0.0; min: 0.0; max: 2.0)	
--right-margin-rest <f>	The right margin for rest in MEI units (default: 0.0; min: 0.0; max: 2.0)	
--right-margin-right-bar-line <f>	The right margin for right barLine in MEI units (default: 0.0; min: 0.0; max: 2.0)	
--top-margin-artic <f>	The margin for artic in MEI units (default: 0.75; min: 0.0; max: 10.0)	
--top-margin-harm <f>	The margin for harm in MEI units (default: 1.0; min: 0.0; max: 10.0)	

Installing or building from sources

Secret page ;-) with tests for syntax highlighting…

Compile from sources is easy.

TERMINAL

```
ls -a
echo "Hello World!
```

HTML / JAVASCRIPT

```
<html>
  <head>
    <script src="http://www.verovio.org/javascript/latest/verovio-toolkit-wasm.js" defer></script>
  </head>
  <body>
    <h1>Hello Verovio!</h1>
    <div id="notation"></div>
  </body>
</html>
```

JAVASCRIPT

```
int main() {
    var string;
    return 0;
}
```

JSON

```
{
   "adjustPageHeight":true,
   "breaks":"none",
   "pageHeight":2970,
   "pageWidth":2100,
   "header":"none",
   "footer":"none",
   "scale":50,
   "spacingStaff":4
}
```

C++

```
int main() {
    std::string string;
    return 0;
}
```

PYTHON

```
int main() {
    std::string string;
    return 0;
}
```

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="https://music-encoding.org/schema/4.0.0/mei-all.rng" type="application/xml" schematypens=
"http://relaxng.org/ns/structure/1.0" ?>
<?xml-model href="https://music-encoding.org/schema/4.0.0/mei-all.rng" type="application/xml" schematypens=
"http://purl.oclc.org/dsdl/schematron" ?>
<mei xmlns="http://www.music-encoding.org/ns/mei" meiversion="4.0.0">
  <meiHead>
    <fileDesc>
      <titleStmt>
        <title>Measure barline example</title>
      </titleStmt>
      <pubStmt>
        <date>2017-05-04</date>
      </pubStmt>
      <seriesStmt>
        <title>Verovio test suite</title>
      </seriesStmt>
      <notesStmt>
        <annot>Verovio supports various types of barline for the "measure" elements. It also supports "scale" att
ributes on "staffDef".</annot>
      </notesStmt>
    </fileDesc>
  </meiHead>
</mei>
```

Command-line version

Verovio codebase is C++17 compliant and is cross-platform. It has be tested on several operating systems and architectures. This sections describes how to build the command-line version of the toolkit from the command-line or using some of the most popular IDEs. There are currently no pre-build binaries of the command-line toolkit available since building it is very straight-forward.

MacOS or Linux

To build the the command-line tool, you need CMake to be installed on your machine as well as a compiler supporting C++17. The commands to build are the following:

TERMINAL

```
cd tools
cmake ../cmake
make
```

The generates a `verovio` binary within `./tools`. You can run Verovio from there or install it. Installing it means copying the executable and the resource files to directories which paths are globally accessible. You simply need to run:

TERMINAL

```
sudo make install
```

If you do not install it and run it from `./tools` or from another directory, you need to use the `-r` option to set the appropriate resource directory. The parameter of the `-r` option has to be a path to the `./data` folder of the codebase.

Keep in mind that if you have installed, you should not run another version without re-installing it or using the -r options because otherwise the resources installed can be invalid. A typical problem is missing font glyphs that a newer version needs but that are not in the older version of the resources.

For seeing the command-line options, run:

TERMINAL

```
./verovio --help
```

(Until version 2.6.0, the cmake command was cmake . and not cmake ../cmake .)

Additional building options

By default the executable is not stripped. To strip it during the installation do

TERMINAL

```
sudo make install/strip
```

For building it without Plain and Easy support (without regex.h), run:

TERMINAL

```
cmake ../cmake -DNO_PAE_SUPPORT=ON
```

To allow PAE support again, you must run the command

TERMINAL

```
cmake ../cmake -DNO_PAE_SUPPORT=OFF
```

since running cmake ../cmake will not clear the state of the define variable.

The other building options are:

- NO_ABC_SUPPORT for the ABC importer to be turned on/off
- NO_HUMDRUM_SUPPORT for the Humdrum importer to be turned on/off
- MUSICXML_DEFAULT_HUMDRUM to use the MusicXML Humdrum importer by default instead of the direct MusicXML importer
- BUILD_AS_LIBRARY for Verovio to be built as dynamic shared library instead of a command-line executable

Uninstall a previous version

To uninstall a previously installed version of Verovio from the system, run:

TERMINAL

```
rm -f /usr/local/bin/verovio
rm -rf /usr/local/share/verovio
```

Occasionally there are problems with updates necessary to the Makefile when compiling a new version of Verovio with make. It may be necessary to clear out the automatically generated cmake files and regenerate them. To do that, run:

TERMINAL

```
rm -rf CMakeFiles CMakeCache.txt Makefile cmake_install.cmake
```

Windows 10

To build Verovio on Windows 10 from the command-line, you will need to have Microsoft C++ Build Tools and make installed on your computer.

Run the following commands from the *x86 Native Tools Command Prompt for VS* (with administrator privileges):

TERMINAL

```
cd <sourceCode>/tools
cmake ../cmake -G "NMake Makefiles"
nmake
nmake install
```

After the installation, add `<sourceCode>/tools` to the PATH of your system.

When running the commands, the resource path should be provided explicitly with the following option:

TERMINAL

```
-r "C:/Program Files (x86)/Verovio/share/verovio"
```

Xcode

For MacOS users, there is also an Xcode project in the Verovio root directory.

By default, humdrum support is turned off in Xcode. To turn in on, you need to use the `Verovio-Humdrum` building scheme.

Visual Studio

- Install CMake
- Go into the tools folder of Verovio
- Execute `cmake ../cmake -DNO_PAE_SUPPORT=ON` (add `-DCMAKE_GENERATOR_PLATFORM=x64` for a x64 solution)
- Open the resulting `Verovio.sln` with Visual Studio and build it from there

JavaScript and WebAssembly

Pre-build versions

The verovio.org GitHub repository provides compiled versions of the JavaScript toolkit. The toolkit is available in three options.

1. **verovio-toolkit.js** - in JavaScript (more precisely in asm.js)
2. **verovio-toolkit-wasm.js** – in WebAssembly
3. **verovio-toolkit-hum.js** – in JavaScript with the Humdrum support

A build of each of these is provided by CI for the development version as well as for each release.

The latest release is always available from:

```
https://www.verovio.org/javascript/latest/verovio-toolkit.js
```

The latest development version is available from:

```
https://www.verovio.org/javascript/develop/verovio-toolkit.js
```

Previous releases are available from their corresponding directory, e.g.:

```
https://www.verovio.org/javascript/2.7.1/verovio-toolkit.js
```

NPM

The latest stable version is available via NPM registry. The version distributed via NPM it the WebAssembly build. It can be installed with:

TERMINAL

```
npm install verovio
```

The homepage of the Verovio package includes documentation on how to use it.

Basic usage of the toolkit

For instructions on a basic usage of the JavaScript version of the toolkit, see the Getting started section of the Tutorial 1: Web-based notation chapter.

Building the toolkit

To build the JavaScript toolkit you need to have the Emscripten compiler installed on your machine. You also need CMake. You need to run:

TERMINAL

```
cd emscripten
./buildToolkit -H
```

The toolkit will be written to:

```
./emscripten/build/verovio-toolkit.js
```

Building without `-H` will include the Humdrum support, which increases the size of the toolkit by about one third. In that case, the output will be written to `verovio-toolkit-hum.js` .

If you are building with another option set than previously, or if you want to regenerate the makefiles, add the option `-M` .

Python

Pre-build versions

Pre-build versions of the Python version of the toolkit are available through PyPi for every release since version 3.1.0.

The Python versions for which a pre-build is provided are 3.6, 3.7, 3.8 and 3.9. The platforms supported are MacOS 10.9, Linux with manylinux for x86-64, Win-32 and Win-amd64.

The latest release can be installed with:

TERMINAL

```
pip install verovio
```

A previous version can be installed with:

TERMINAL

```
pip install verovio==3.2.0
```

For all platforms or architectures for which a pre-build version is not available in the PyPi repository, a source distribution is available. It can be installed with the same command as above. This will automatically trigger the compilation of the package.

Basic usage of the toolkit

Once installed, the Verovio tookit module can be imported with

PYTHON

```
import verovio
```

You can then create an instance of the toolkit and load data. For example:

PYTHON

```
tk = verovio.toolkit()
tk.loadFile("path-to-mei-file")
tk.getPageCount()
```

Once loaded, the data can be rendered to a string:

PYTHON

```
svg_string = tk.renderToSVG(1)
```

It can also be rendered to a file:

PYTHON

```
tk.renderToSVGFile( "page.svg", 1 )
```

Setting options

The options are set on the toolkit instance. For example, the following code will change the dimensions of the page and redo the layout for the previously loaded data:

PYTHON

```
tk.setOption( "pageHeight", "2100" )
tk.setOption( "pageWidth", "2900" )
tk.setScale(25)
tk.redoLayout()
tk.renderToSVGFile( "page-scaled.svg", 1 )
```

It is also possbile to collect options in a Python Dictionary and pass them as Json dump to the toolkit:

PYTHON

```
import json
options = {
    'pageHeight': 1000,
    'pageWidth': 1000
}
tk.setOptions(json.dumps(options))
tk.redoLayout()
tk.renderToSVGFile( "page-square.svg", 1 )
```

Building the toolkit

To build the Python toolkit you need to have swig and swig-python installed on your machine (see SWIG) and the Python distutils package. Version 4.0 or newer of SWIG is recommended but older versions should work too. To install SWIG in MacOS using Homebrew, type the command `brew install swig` .

The toolkit needs to be built from from the root directory of the repository content. To build it in-place, run:

TERMINAL

```
python setup.py build_ext --inplace
```

If you want to install it, run:

TERMINAL

```
python setup.py build_ext
sudo python setup.py install
```

For building it with one or more specific options (e.g., without Plain and Easy support), run:

TERMINAL

```
python setup.py build_ext --inplace --define NO_PAE_SUPPORT
```

Building a Python wheel locally

You can build a Python wheel locally with:

TERMINAL

```
python setup.py bdist
```

For a source distribution, do:

TERMINAL

```
python setup.py sdist
```

In both cases, the wheel will be written to the `./dist` directory.

Building with CMake

The Python toolkit can be built with CMake, which can be significantly faster because parallel processing can be used. This is also the approach to recommend when developing because it will not rebuild the entire codebase when a change it made to a file but only the files that actually need to rebuilt.

For this approach to work you need at least version 3.13 of CMake because it uses the option `-B` introduced in that version of CMake. The steps are:

TERMINAL

```
cd bindings
cmake ../cmake -B python -DBUILD_AS_PYTHON=ON
cd python
make -j8
```

If you want to enable or disable other specific options, you can do:

TERMINAL

```
cmake ../cmake -B python -DBUILD_AS_PYTHON=ON -DNO_PAE_SUPPORT=ON
```

Installation with CMake has not be tested yet

Resources for versions built locally

When using a version built locally, you usually have to specify the path to the Verovio resources. To do so, you can do

PYTHON

```
import verovio
tk = verovio.toolkit(False)
tk.setResourcePath("path-to-resource-dir")
```

Alternatively, you can set it before you create the instance of the toolkit

PYTHON

```
import verovio
verovio.setDefaultResourcePath("path-to-resource-dir")
tk = verovio.toolkit()
```

Other bindings

Java

To build the Java toolkit you need to have swig and swig-java installed on your machine (see SWIG) as well as Maven. You need to run:

TERMINAL

```
cd bindings/java
mvn package
mvn package
```

Note the `mvn package` command needs to be run twice. You can test it with the MEI and PAE examples. For example – replace `X.X.X` with the appropriate version number:

TERMINAL

```
cd example-mei
javac -cp .:../target/VerovioToolkit-X.X.X.jar main.java
java -cp .:../target/VerovioToolkit-X.X.X.jar main
```

This should write an `output.svg` file in the current directory. The PAE example will write the SVG to the standard output.

See this issue for SVG output problems on non US Ubuntu installations.

CocoaPods

You can use CocoaPods to install `Verovio` by adding it to your to your `Podfile` :

```
platform :ios, '12.0'
use_frameworks!
target 'MyApp' do
  pod 'Verovio', :git => 'https://github.com/rism-ch/verovio.git', :branch => 'develop'
end
```

Then, run the following command:

TERMINAL

```
pod install
```

To use Verovio in your iOS project import

C++

```
#import <Verovio/Verovio-umbrella.h>
```

See https://github.com/Noroxs/VerovioExample for an example how to use it. To build and run the example, you need to:

- Navigate in the Terminal to the cloned directory
- Execute pod update
- Open the VerovioExample.xcworkspace and NOT the VerovioExample.xcodeproj
- Build and Run on any simulator or device

Contributing

Introduction

Coding guidelines

This document describes the coding style for the Verovio project for the C++ part of the codebase.

Formatting

Verovio uses a Clang-Format (**5.0**) coding style based on the WebKit style, with a few minor modifications. The modifications include:

```
AllowShortIfStatementsOnASingleLine: true
AllowShortLoopsOnASingleLine: true
ColumnLimit: 120
ConstructorInitializerAllOnOneLineOrOnePerLine: true
PointerAlignment: Right
```

The simplest way to fullfil the Verovio coding style is to use a clang-format tool and to apply the style defined in the .clang-format file available in the project root directory.

Downloading clang-format for OS X

An easy way to install clang-format on OS X computers is to use Hombrew. Type this command in the terminal to install:

TERMINAL

```
brew install clang-format
```

Running clang-format

Please make sure you use version 5.0

To use clang-format to adjust a single file:

TERMINAL

```
clang-format -style=file -i   some-directory/some-file.cpp
```

The `-style=file` option instructs clang-format to search for the .clang-format configuration file (recursively in some parent directory). The `-i` option is used to alter the file “in-place”. If you don’t give the `-i` option, a fomatted copy of the file will be sent to standard output.

Includes and forward declarations

Includes in the header files must list first the system includes followed by the Verovio includes, if any, and then the includes for the libraries included in Verovio. All includes have to be ordered alphabetically:

C++

```
#include <string>
#include <utility>
#include <vector>

//----------------------------------------------------------------------------

#include "attclasses.h"
#include "atttypes.h"

//----------------------------------------------------------------------------

#include "pugixml.hpp"
#include "utf8.h"
```

In the header files, always use forward declarations (and not includes) whenever possible. Forward declaration have to be ordered alphabetically:

C++

```
class DeviceContext;
class Layer;
class StaffAlignment;
class Syl;
class TimeSpanningInterface;
```

In the implementation files, the first include in always the include of the corresponding header file, followed by the system includes and then the other Verovio includes with libraries at the end too, if any, also ordered alphabetically:

C++

```
#include "att.h"

//----------------------------------------------------------------------------

#include <sstream>
#include <stdlib.h>

//----------------------------------------------------------------------------

#include "object.h"
#include "vrv.h"

//----------------------------------------------------------------------------

#include "pugixml.hpp"
```

Null and boolean

The null pointer value should be written as `NULL` . Boolean values should be written as `true` and `false` .

Class, method and member names

All class names must be in upper CamelCase. The internal capitalization follows the MEI one:

C++

```
class Measure;
class ScoreDef;
class StaffDef;
```

All method names must also be in upper CamelCase:

C++

```
void Measure::AddStaff(Staff *staff) {}
```

All member names must be in lower camelCase. Instance members must be prefixed with `m_` and class (static) members with `s_` :

C++

```
class Glyph {
public:

    /** An instance member */
    int m_unitsPerEm;

    /** A static member */
    static std::string s_systemPath;
};
```

In the class declaration, the methods are declared first, and then the member variables. For both, the declaration order is `public` , `protected` , and `private` .

Comments

Comments for describing methods can be grouped using `///@{` and `///@}` delimiters together with the `@name` indication:

C++

```
/**
 * @name Add children to an editorial element.
 */
///@{
void AddFloatingElement(FloatingElement *child);
void AddLayerElement(LayerElement *child);
void AddTextElement(TextElement *child);
///@}
```

LibMEI

The code for the attribute classes of Verovio are generated from the MEI schema using a modified version of LibMEI available here. The code generated is included in the Verovio repository in ./libmei and the LibMEI repository does not need to be cloned for building Verovio.

The attribute classes generated from the MEI schema provide all the members for the element classes of Verovio. They are implemented via multiple inheritance in element classes. The element classes corresponding to the MEI elements are not generated by LibMEI but are implemented explicitly in Verovio. They all inherit from the `Object` class (of the `vrv` namespace) or from a `Object` child class. They can inherit from various interfaces used for the rendering. All the MEI member are defined through the inheritance of generated attribute classes, either grouped as interfaces or individually.

For example, the MEI <note> is implemented as a `Note` class that inherit from `Object` through `LayerElement` . It also inherit from the StemmedDrawingInterface that holds data used for the rendering.

Its MEI members are defined through the `DurationInterface` and `PitchInterface` that regroup common functionnalities for durational and pitched MEI elements respectively plus some additional individual attribute classes.

The inheritance should always list `Object` (or the `Object` child class) first, followed by the rendering interfaces, followed by the attribute class interfaces, followed by the individual attribute classes, each of them ordered alphabetically:

C++

```
class Note : public LayerElement,
             public StemmedDrawingInterface,
             public DurationInterface,
             public PitchInterface,
             public AttColoration,
             public AttGraced,
             public AttStems,
             public AttTiepresent
```

In the implementation, the same order must be followed, for the constructor calls and for the registration of the interfaces and individual attribute classes:

C++

```
Note::Note()
    : LayerElement("note-")
    , StemmedDrawingInterface()
    , DurationInterface()
    , PitchInterface()
    , AttColoration()
    , AttGraced()
    , AttStems()
    , AttTiepresent()
{
    RegisterInterface(DurationInterface::GetAttClasses(), DurationInterface::IsInterface());
    RegisterInterface(PitchInterface::GetAttClasses(), PitchInterface::IsInterface());
    RegisterAttClass(ATT_COLORATION);
    RegisterAttClass(ATT_GRACED);
    RegisterAttClass(ATT_STEMS);
    RegisterAttClass(ATT_TIEPRESENT);

    Reset();
}
```

Resetting the attributes is required and follows the same order

C++

```
void Note::Reset()
{
    LayerElement::Reset();
    StemmedDrawingInterface::Reset();
    DurationInterface::Reset();
    PitchInterface::Reset();
    ResetColoration();
    ResetGraced();
    ResetStems();
    ResetTiepresent();

    // ...
}
```

Contributing workflow

Generate code with libMEI

Adding SMuFL glyphs

All SMuFL glyphs used by Verovio have to be available in the Leipzig font. For adding support for a new SMuFL glyph, the steps are:

1. Add the glyph to the Leipzig font file
2. Generate the Leipzig font as SVG font
3. Add the glyph to the list of supported glyph in the XSL list

Make sure you always add glyphs **only** in the develop-leipzig branch because conflict solving is problematic with the process of adding a glyph, in particular for the Leipzig font file. For this reason, make sure you always pull the latest version from the develop-leipzig branch before starting your work and do not wait too long before making a PR. If changes have been made in between, you will need to add your glyphs again.

When making a PR, always add an image (e.g., screenshot of FontForge) showing the glyphs.

Adding the glyph to the Leipzig font file

The file is ./fonts/Leipzig-5.2.sfd and should be edited with FontForge. Very often it is possible to copy another existing glyph as basis for the new glyph. Leipzig is visually lighter and thinner than Bravura and new glyphs have to follow this design choice. Do not copy glyphs from Bravura. Make sure the font is valid by running “Element => “Find Problems…”.

Once the new glyph(s) has/have been added, you also need to change the version number in the font info (menu “Element” => “Font Info” and then tab “PS Names” in fields “Version” and “Copyright” and tab “Comment” where you also need to add a comment together with the version number. The file can be saved.

Generate the Leipzig font as SVG font

From FontForge, export the with menu “File” => “Generate Fonts…” and select “SVG font” (option “validate before saving” can be turned off). The file needs to be written to ./fonts/Leipzig.svg .

Add the glyph to the list of supported glyph in the XSL list

Open the file ./fonts/supported.xsl and uncomment the glyph(s) you added to Leipzig. The XSL file is then used to extract the glyphs supported by Verovio

Make a PR to the develop-leipzig branch

Once the PR will have been merged, the glyphs will be extracted from the SVG font by running the script ./fonts/generate_all.sh (from ./fonts/). This will extract all the glyphs from the SVG font file and calculate the their bounding boxes. When this is done you will see your glyphs in ./data/ and in ./include/vrv/smufl.h

Table of Contents