Reference Book

For Verovio version 3.4

Generated on 5 May 2021 from 0750505

DOI: 10.5448/7em6-my23

Introduction

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.

This initial chapter gives an introduction to Verovio and the history of the project as well as an overview on how to use it.

The following two chapters provides a number of tutorials, starting at the very basic and ending at advanced topics in notation. By the end of these 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 chapter on advanced-topics provides some more in-depths explanation of specifics of Verovio.

The last chapters provides a reference for the operations and options available. They also cover how to build and install Verovio, including from the source code, and how to contribute to the active development of Verovio.

Reference

This book is identified with the DOI 10.5448/7em6-my23 which refers to the currently applicable version of the book documenting the latest release 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.

History of the project

Engraving music notation by computer is a notoriously complex task, and the most powerful music notation rendering engines are, for the most part, the result of long-term developments of commercial music notation editors in which considerable resources had to be invested. They each have their own internal structure and file formats. Furthermore, the music notation rendering engines of music notation editors are not very modular and cannot easily be used or integrated into other applications.

Besides music notation editors, some music notation rendering engines are also available as command-line tools. These are easier to integrate than desktop applications, however with occasionally quite significant dependencies and requirements that limit the contexts in which their use is possible. This is the case for LilyPond, a very popular and powerful typesetting engine

Such tools have been used for many years within theMusic Encoding Initiative (MEI) community for engraving scores encoded in MEI. Using them, however, meant converting the MEI to another encoding scheme that could be used as input format for the engraving tool. Whichever solution was used to do this, it remained clearly suboptimal. MEI users willing to benefit from all the strengths of MEI were facing the problem of not being able to render their data properly. Converting a music encoding format to another one is known to become quickly problematic. It is particularly true when converting from MEI markup that is rich and detailed, a feature that distinguishes MEI from other encoding schemes. With a conversion step, it is likely that not all the information will be preserved in the rendering, or at least only in cumbersome ways and with sometimes quite limited results.

At that time, by about thirty years after the initial development of music notation software applications, the digital domain had significantly changed with the advent of the online world. For music notation, this translated into new possibilities but also new challenges to be faced. While most music notation engraving tools target PDF, this format is clearly not the ideal one in web-based environments. It can be published online, but some web browsers still require a dedicated viewer plugin to be installed for this to be possible.

They yield inconsistent viewing and document browsing experiences, which is far from ideal. To embed PDF files directly in web pages code, they need to be converted to images, which creates an overhead and additional complications in the publication process, with often poor results in the display quality.

Early stages

In 2013, the RISM Digital Center launched the development of Verovio for rendering the music incipits or the RISM project. The main idea behind the development of Verovio was to implement a tool that could render the RISM music incipits directly but also to support MEI natively. That is, without having MEI converted to another format, either explicitly or internally in the software application used for rendering. With Verovio, the MEI markup is parsed and rendered as notation with a single tool and in one step.

One of the reasons for choosing to implement a library from scratch rather than modifying an existing library was that it will allow to operate on a memory representation of MEI, which will make it significantly easier to render complex MEI features in the long run. Previous experience has indeed shown that modifying an existing solution can be very quick to develop at the beginning, but that the development curve eventually reaches a plateau.

Another idea behind the development of Verovio was to have a tool that would be easy to use in web environments. Instead of targeting PDF output, Verovio uses the Scalable Vector Graphics (SVG) format developed and maintained by the W3C. The advantage of SVG over other output formats, and Postscript and PDF in particular, is that it can easily be used in a web-based environment because it is rendered natively in most modern web browsers with no plug-in required. In addition, since SVG is a vector format, the output can also be used for high-quality printing, which means that it offers the best of both digital and paper-based worlds.

With the same goal in mind, Verovio was designed to be light and fast and has no external dependencies, making it very flexible and easy to use or integrate into digital environments.

Interacting with music encoding

Today, partly in response to the development of MIR applications, rendering of music notation can be necessary in very different contexts, for example within standalone desktop applications, in server-side web application scenarios, or directly in a web browser. Music notation might need to be rendered for displaying search results or for visualising analysis outputs. Another example is score-following applications, where the passage currently played needs to be displayed and possibly highlighted. These are different use-cases of interactive applications where music notation plays a key role, including many cases where the notation itself has to be an interactive component.

Several design features of the Verovio library make it highly suitable for interactive music notation applications. It is a software library that can run in a wide range of environments (and not a full software application) and it is light and fast. The JavaScript version of Verovio is particularly promising because it provides a fast in-browser music MEI typesetting engine that can easily be integrated into web-based applications. This setup makes it possible to design ground-breaking web applications where the MEI encoding is rendered on the fly. In such designs we can rethink the interface and avoid mimicking page output. We can instead adjust the layout dynamically to the screen of the device employed by the user. The layout can be calculated to fill the size of the screen, or interactively changed according to a zoom level adjusted by the user. This opens up new responsive web-interfaces to be designed and developed based on dynamic music notation reflow. This works particularly well with SVG, especially since it is now supported by all modern web browsers. However innovative the dynamic layout of music notation may be, it remains a very basic interaction. Verovio aims to go further and to produce a graphic output that can then be the foundation for more complex interactions.

Because SVG is XML, it has an advantage over raster image formats in that every graphical element is addressable. This feature makes it intrinsically well suited for interaction, and this is also true for music notation. In a web environment, the addressability can be used for highlighting graphical elements such as notes or any other music symbols. One additional characteristic of SVG is that its XML tree can be

structured as desired, and an innovative design feature of Verovio was to go further in the structuring of the output by leveraging this characteristic. Since Verovio implements the MEI structure internally, this key feature of SVG made it possible to preserve the MEI structure in the design of the SVG output in Verovio. Preserving the MEI structure in the SVG output is a considerable overhead in the rendering process but makes it a unique feature of Verovio.

As a result, Verovio output in SVG is not the end of a unidirectional rendering process. Quite on the contrary, it should instead be seen as an intermediate layer standing between the MEI encoding and its rendering that can act as the cornerstone for a bi-directional interaction: from the encoding to the notation, but also from the notation to the encoding through the user interface.

Design principles

The basis for interactivity offered by MEI coupled with Verovio follows some important design principles. First for all, the principle of *availability* and *discoverability*. That is, all the content (e.g., all the MEI editorial variants) is available. Alternative text can be made discoverable, for example with CSS highlighting. It also follows the design principle of *scalability*. Verovio is light and fast. It can run on small devices, but it also supports large files in higher resource environments.

They are also some technical principles that are followed as far as possible. They include*reusability* and *durability*. By providing only the interaction foundation and not making any assumption in interface design, especially with a software library that has no dependency, reusability is undeniably maximised. So is the durability, although durability is hard to predict in software development, particularly for digital humanities projects which have slow development cycles in comparison with the development of the technology itself. Reducing dependencies as much as possible is one way to increase durability. In the case of MEI rendering, keeping the rendering engine separate from larger applications that will use it is another way.

In terms of editions and interface design, there is much still to invent. This will need to be done hand in hand with the development of MEI. It is obvious that merely imitating printed output in a digital environment will not be satisfactory. Most effort should be spent on developing the added value that digital environments can offer. Parallel with the development of the online world is the appearance of new devices, such as tablets with wireless network access. They offer new possibilities in terms of digital access and change the manner and location in which digital content can be read. Developing these possibilities will not preclude the co-existence of printed editions, which have and will continue to retain their own added value. The challenge now is neither to replicate nor to supplant existing media or applications, but to expand horizons by exploring new ways of conceiving the information to which we have access, and MEI and Verovio are a decisive and exciting step in this direction.

Use-case scenarios

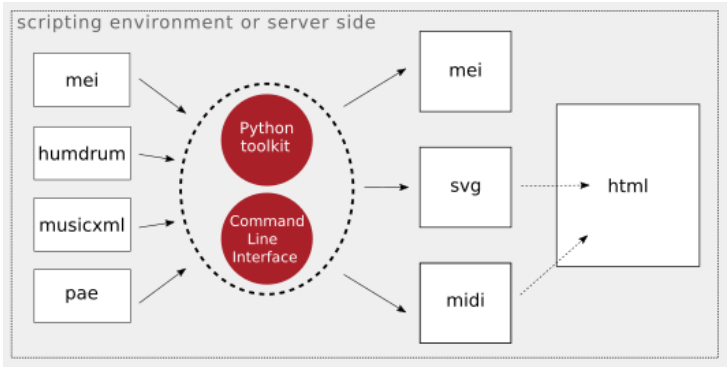
Architecture possibilities

Verovio is a C++ codebase that can be compiled and wrapped into different programming languages and integrated into various environments and several use-cases can be imagined for the Verovio toolkit.

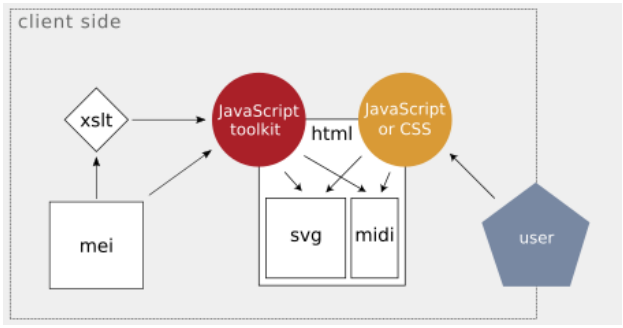
First of all, it can be built and used as a standalone command-line tool. This option is well suited to scripting environments and applications. The command-line tool can be used to render music notation files into SVG or into MIDI files. These files can be embedded in HTML files with everything happening on the server side. Verovio can also be used to convert data (e.g., MusicXML or Humdrum) to MEI. Typical use cases would be :

- generate SVG and MIDI from MEI documents or other supported formats,
- generate MEI documents from other supported formats (e.g., convert files).

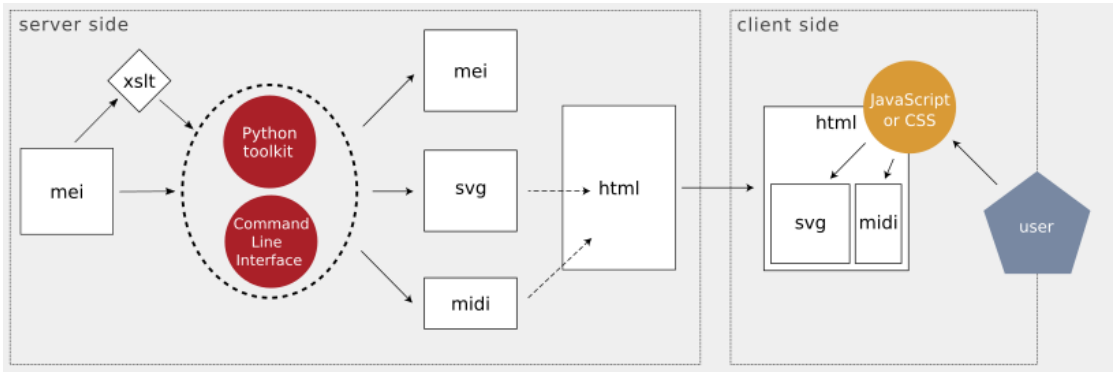
Resulting SVG or MEI documents can then be embedded in a HTML page or used as such.



The JavaScript toolkit makes it possible to generate SVG and MIDI directly in the browser. It is easy to set up and platform independent. Interaction with the user can then be handled with basic JavaScript or CSS. An example of how to handle events is given in the tutorial. It is also possible to process the MEI via XSLT in the browser before loading it into Verovio.



Both approaches can be combined: one may choose to process the MEI and to generate the SVG server side for better performance, and then handle interactions client side with JavaScript and CSS.

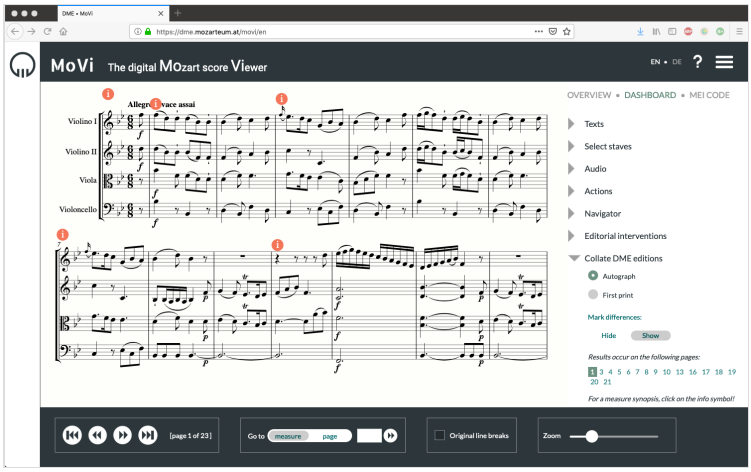


Application examples

Interactive applications in which the MEI and Verovio pair is being used are very diverse. In this section, we list some example application uses-cases based on this pair and where interaction is an important component. Most of the projects selected are research projects or research tools, but not only.

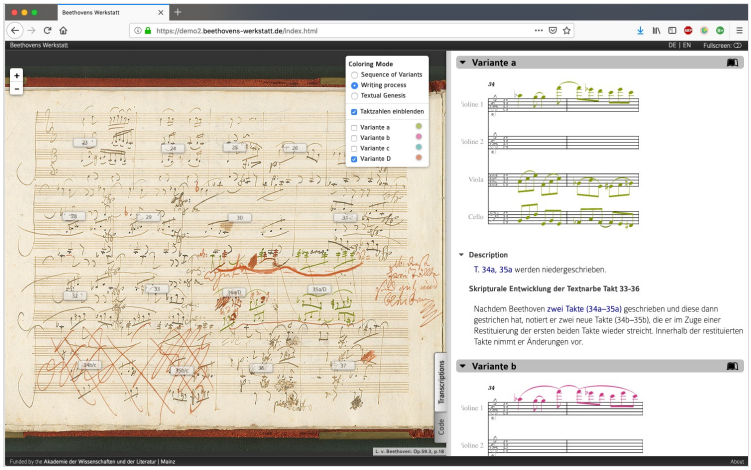
Critical editions

The Digital Interactive Mozart Edition (DIME) is a joint project of the Salzburg Mozarteum Foundation and the Packard Humanities Institute in California. It is one example project in the field of digital critical editions that takes advantage of very rich and powerful markup possibilities offered by the MEI schema. In this context, interaction capabilities open completely new and welcome perspectives in interface design. Critical editions traditionally encompass extremely dense information networks that have to be laid out on paper with all the associated bi-dimensional constraints. Variant display is notoriously cumbersome and the information often has to be scattered between various part of the books (e.g., the critical notes referring to the music scores listed at the end of a volume).



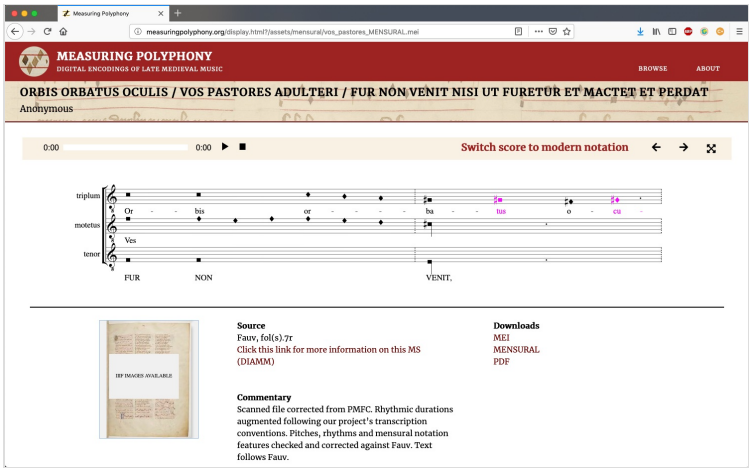
Genetic editing

Genetic editing is still an exploratory field in music. In this context, MEI is in active development under the lead of the Beethovens Werkstatt project. In genetic editing, time is a key dimension to be taken into account in the representation of differences. The differences in genetic editing represent different stages of writing for which it is not always possible to determine clearly their scope, their order in time or even their content because it is not always readable. This yields potentially very complex and large datasets for which the music notation content cannot be visualised as a whole. Only subsets of the data can here be reasonably visualised at a time, and interaction is the perfect approach for allowing highlighting, selection and navigation in the data.



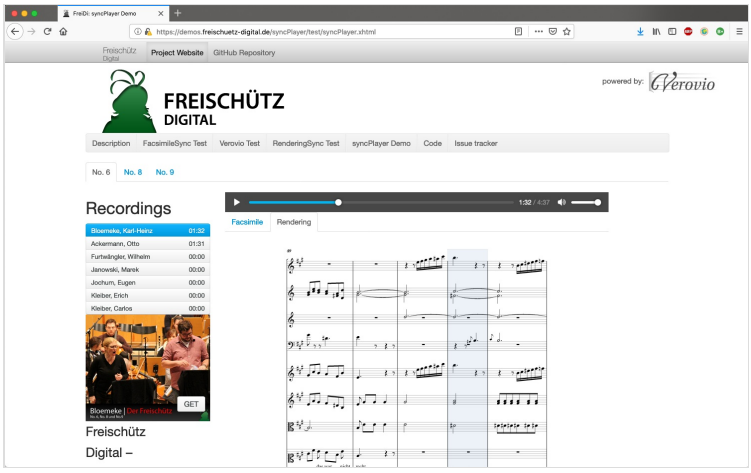
Early music

Thanks to the overall simple structure of its notation (e.g., monophony for chant), early music has often been at the forefront of development of digital projects. Nonetheless, most of the time they remained isolated because of the need to develop dedicated encoding schemes and tools. The Measuring Polyphony project, a repository of digital encoding of late medieval polyphony at Brandeis University, is a good example of a change. The same ecosystem as for CWMN is used here. The MEI modularity allows for precise representation of the mensural notation, and the development of MEI and Verovio allow, for the first time, early music notation to be properly encoded accurately regarding the ternary and binary durations in the music. Interaction perspectives can be seen for linking original notation and modern transcriptions, which remains desirable for non-expert audiences.



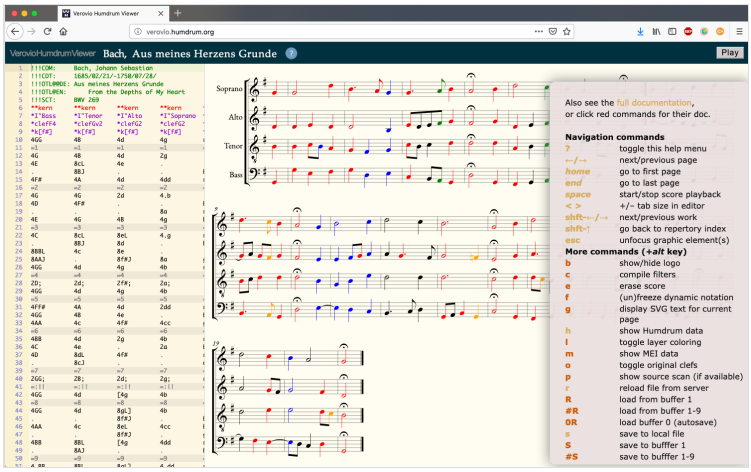
Audio alignment

Alignment of scores with audio recordings, also known as score following, is a typical music information retrieval task. The main challenge is to generate the alignment data taking into account the fact that performances vary in tempo and that sections of the score can be repeated in some performances. The Freischütz Digital is an example project where the alignment data is stored in MEI with synchronisation information at the measure level generated for multiple recordings. The playback is synchronised with Verovio using measure xml:id for following the score or jumping anywhere in it. Clicking anywhere on the score can conversely be used to jump to the corresponding place in the recording. In the case of this project, because the MEI data also contains mapping of the measures with their corresponding zone in the facsimile image of the handwritten manuscript, the same synchronisation can be realized with it.



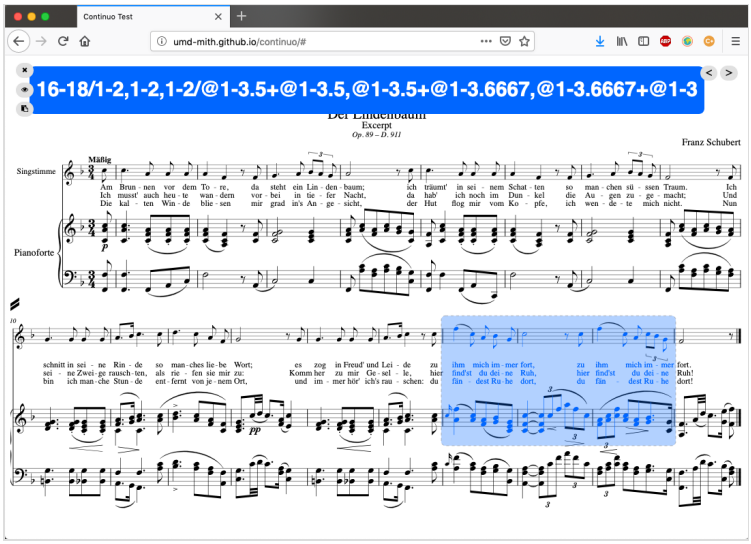
Music notation editing

Interaction with music notation can take the form of data editing, either in a WYSIWYG manner or by allowing music encoding text editing. The Neon.js project for neume notation is an example of the former approach. It is currently going in-depth refactoring for switching from a previous ad-hoc rendering solution to Verovio rendering. The later editing approach is implemented in the Verovio Humdrum Viewer (VHV) project where editing of the encoding (Humdrum or MEI) is updated on the fly. The same setup has recently been integrated into Atom as a plugin package, MEI-tools-atom. In both the VHV and the Atom package, the rendered notation can be clicked to navigate in the encoding.



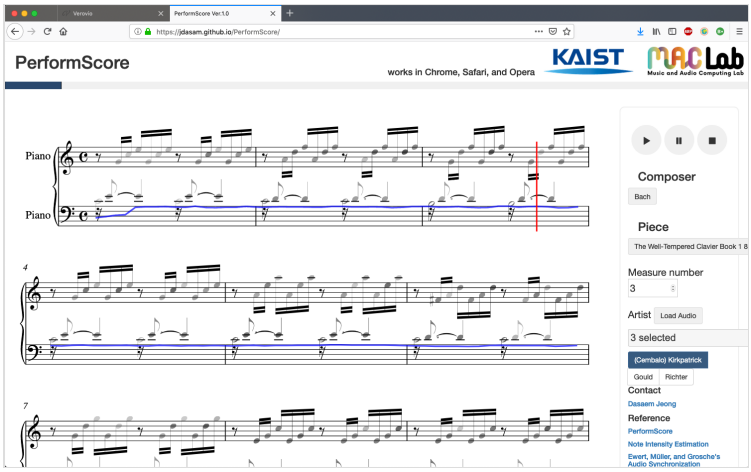
Music addressability

In music literature or in music practice, addressing music notation generally relies on movement names and measure numbers, and additionally voice or instrument names and beat numbers when necessary. However, there is no formalised concept behind this practical approach. Addressing music notation in the digital world has been recently the focus of the Enhancing Music Notation Addressability (EMA) project at the University of Maryland. The goal of the project is to develop a generic system for expressing addresses in music notation documents. In order to evaluate it, the project developed a web service with an API for addressing MEI documents, the Open MEI Addressability Service (OMAS). The Verovio rendering is used to display a selection. Conversely, the rendered music notation can serve as the basis for selecting interactively a zone to be transformed into an address in the music notation data.



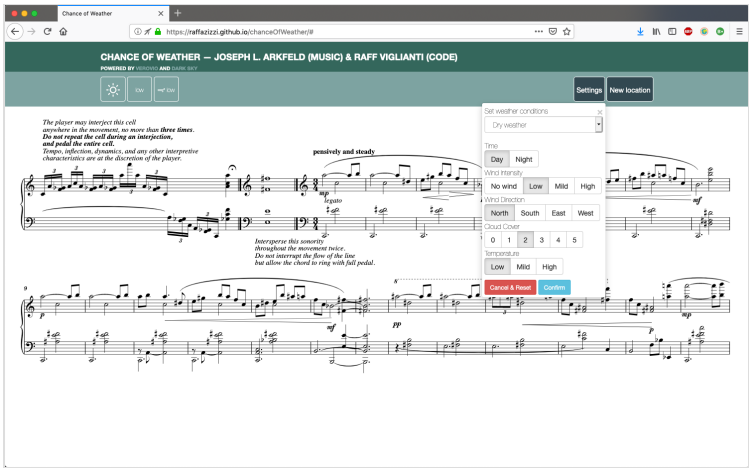
Visualisation

Visualisation is an important field of research and experimentation in digital humanities. With digital publications and digital devices, interactivity significantly increases the visualisation possibilities. For example, the visualisation scope or perspective can change dynamically following the choice of the user or the content of the data. With dynamic music notation rendering, it is possible to augment it with additional visualisation layers as demonstrated by the performance analysis and re-synthesis of piano music PerformScore project at the Music and Audio Computing Lab. A player featuring score following for multiple performances to be selected by the user as seen with the Freichütz Digital project is enhanced here with the visualisation of additional characteristics of the performance being played. They include tempo and dynamic changes but also the intensity of individual notes through colour and opacity adjustment. Louder notes become darker with high opacity and softer notes thinner with low opacity.



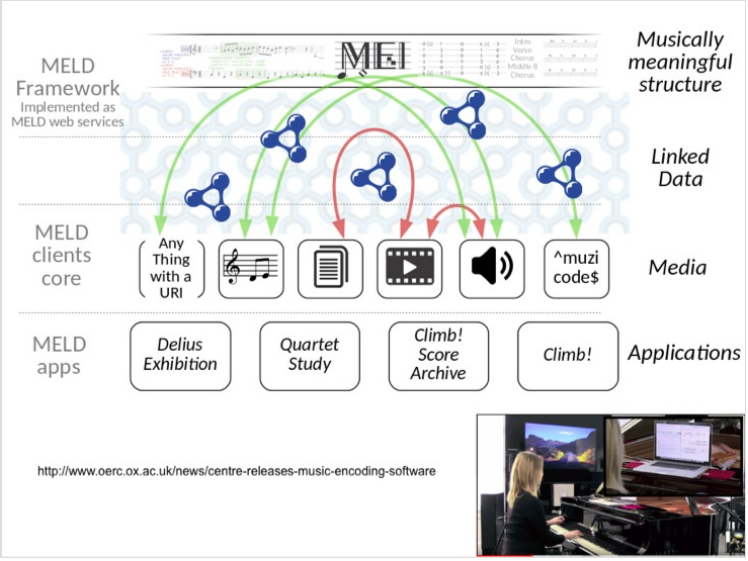
Composition

Contemporary music compositions can rely directly on the distinct features of digital score technology. An example is the Chance Of Weather composition by Joseph Arkfeld based on Emily Dickinson's poetic fragments "Fortitude - flanked with Melody". The idea behind this project is to apply in the composition process the paradigm of fragment and variation as found in critical editions. The composition is made up of a set of fragments inspired by the poem and the encoding of the score is itself based on markup traditionally used for critical editions. Ultimately, the choice of the fragments for a particular instance of the composition is determined by an external data source, namely weather conditions (wind, cloud cover, temperature, etc.) at a geographical place to be chosen by the user. The weather conditions are transformed into a query that selects the corresponding fragments.



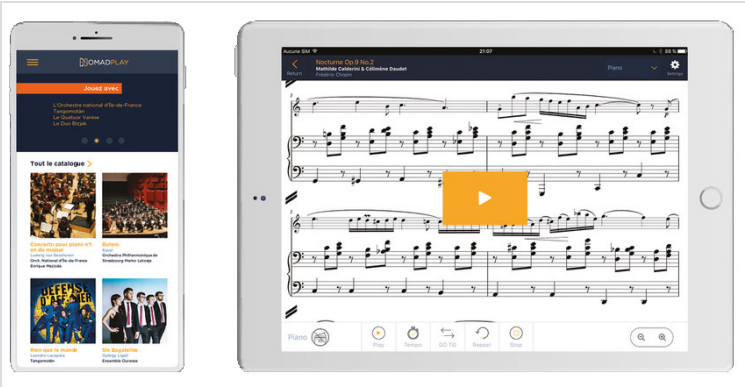
Performance

Interaction with music notation is quite common in the domain of performance. However, a significant breakthrough came on stage with the Music Encoding and Linked Data (MELD) framework and Climb!, a music composition that mixes the idea of classical virtuoso piece and computer game. The major innovation of the project is that the dataset is stored as Linked Data using MELD. Climb! is a non-linear composition also made from a set of fragments moving from the bottom to the top of a graph representing a mountain. The path of a performance is not pre-determined and changes at each performance. At some stages, the performer has to play some excerpts, whose accuracy is dynamically verified in order to decide if the performer can proceed to the next stage. Feedback to the performer can be provided by the highlighting of score fragments.



Education

In the field of music education, interactive applications are more and more common and increasingly sophisticated. They typically link music notation with recordings, but also with user feedback (measure tempo, tuning, etc.). They are often built as mobile device applications, such as the NomadPlay application. NomadPlay features a catalogue of recordings of a wide range of pieces from which the user can select his instrument. He can then rehearse the piece with the score of his instrument being displayed and synchronized with the recording but with the sound of his instrument removed. It is also possible to loop a difficult passage, or to change the tempo of the recording interactively.



Verovio 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!

Tutorial 1: First Steps

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 understand 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.

> You may notice that Verovio prints some warnings to your browser console. We can ignore these options for this tutorial, but if you are working with your own encoded scores and see these warnings it may help you track down problems or unexpected behaviours when rendering your scores.

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.

Passing options to Verovio

Passing options to Verovio is as easy as creating a set of key and value pairs, and using the setOptions method on the toolkit. To scale the output we will use the `scale` option. Add the following to your page, after we have instantiated the toolkit but before we render the data:

JAVASCRIPT

```
tk.setOptions({
  scale: 30
});
```

When you refresh your page, you should see your score scaled to 30% of its original size. Try experimenting with other values to see their effects! (Hint: you can use sizes above 100%.)

Defaults

All of the options have default values. You can use the `getOptions` method to view the list of all the options and their default values. We will use the browser console to explore these defaults. Add the following line:

JAVASCRIPT

```
console.log("Verovio options:", tk.getOptions());
```

When you refresh your page and open your browser's console you should see the text "Verovio options:" followed by a small disclosure triangle. Clicking this triangle will produce a long list of options that you can pass to `setOptions`. Let's try a few more.

Change the page orientation

You may have noticed that, by default, Verovio renders the score in "portrait" orientation; that is, the width of the score is shorter than the length. To change this, we can use the `landscape` and `adjustPageWidth` options:

JAVASCRIPT

```
tk.setOptions({
  scale: 30,
  landscape: true,
  adjustPageWidth: true
});
```

When you refresh the page you should notice that your SVG has changed orientation! But wait… the score is now cut off! Where did the rest of it go?

It turns out that Verovio has the ability to split scores into "pages" automatically. When it calculates the notation cannot fit on the current page, Verovio will automatically push it to the next page. Adjusting the different options will have an effect on this calculation, so it is worth looking through the options that we printed out, and trying some on your own. You may wish to change the `pageWidth` option, for example, to a bigger or smaller value and see what the result is.

End of Section 3

In this section we have explored Verovio's default options, and looked at how to adjust them to change the rendering output. In the next section we will look at how we can adjust these options dynamically, using on-screen controls to provide a user interface for building interactive music notation displays.

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!");
            tk.setOptions({
              scale: 30,
              landscape: true,
              adjustPageWidth: true
            });
            tk.renderToSVG
            console.log("Verovio options:", tk.getOptions());

            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>
```

Score navigation

In this final part of the introductory tutorial, we will take what we have learned about Verovio and produce an interactive score, where your users can adjust the behaviour of Verovio and see the display updated.

Creating the controls

Before we start we will need to create some HTML form controls. These controls will do the following:

- A slider to adjust the scaling factor;
- “Next page” and “Previous page” buttons for navigating the score;
- A checkbox for adjusting the orientation (portrait or landscape)

> If you are not familiar with how HTML form controls are created, you may wish to consult the Basic form controls and the HTML5 input types documentation.

Tutorial 2: Interactive notation

Introduction

[in preparation]

Working with CSS and SVG

[in preparation]

XPath queries

[in preparation]

Working with MIDI

[in preparation]

Beyond tutorials: Advanced topics

Introduction

This chapter covers several advanced topics that require more in-depth documentation.

Internal structure

Verovio provides a self-contained typesetting engine that is directly capable of rendering MEI to a graphical representation in high quality. Its main goal is to develop a library with an internal structure identical to MEI as far as possible.

For practical reasons, however, the Verovio library uses a page-based customization of MEI internally. Since the modifications introduced by the customization are very limited, the Verovio library can also be used to render un-customized MEI files. With the page-based customization, the content of the music is encoded in `<page>` elements that are themselves contained in a `<pages>` element within `<mdiv>`.

A `<page>` element contains `<system>` elements. From there, the encoding is identical to standard MEI. That is, a `<system>` element will contain `<measure>` elements or `<staff>` elements that are both un-customized, depending on whether the music is measured or un-measured.

Layout and positioning

The idea of a page-based customization is also to make it possible to encode the positioning of elements directly in the content tree. This can be useful where the encoding represents one single source with one image per page. This is typically the case with optical music recognition applications. Verovio supports both positioned elements and automatic layout, which is the default when un-customized MEI files are rendered.

The page-based organization is modeled by a MEI customization that defines the structure described above. The ODD file of the customization and the corresponding RNG schema are available from the MEI Incubator. This is still work-in-progress.

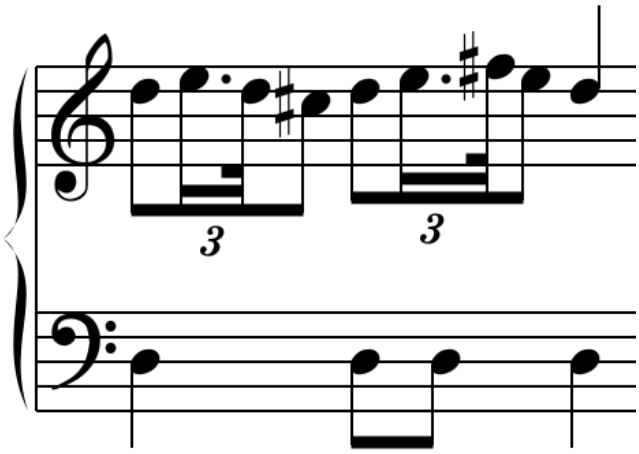
SVG structure

One advantage of SVG rendering over other formats (e.g., images or PDF) is that SVG is rendered natively in all modern web-browsers. Because it is in XML, it also has the advantage that it is well suited to interaction in the browser, since every graphic is an XML element that is easy addressable in the DOM. With Verovio, we also have the advantage that the SVG is organized in such a way that the MEI structure is preserved as much as possible.

To give an example, a `<note>` element with an `xml:id` attribute in the MEI file will have a corresponding `<g>` element in the SVG with and `class` attribute with a value of `"note"` and an `id` attribute corresponding to the `xml:id`. This makes interaction with the SVG using JavaScript very easy. The hierarchy of the element is also preserved as shown below.

XML

```
<tuplet xml:id="t1" num="3" numbase="2">
  <beam xml:id="b1">
    <note xml:id="n1" pname="d" oct="5" dur="8" />
    <note xml:id="n2" pname="e" oct="5" dur="16" dots="1"/>
    <note xml:id="n3" pname="d" oct="5" dur="32" />
    <note xml:id="n4" pname="c" oct="5" dur="8" accid="s"/>
  </beam>
</tuplet>
<beam xml:id="b2">
  <tuplet xml:id="t2" num="3" numbase="2">
    <note xml:id="n5" pname="d" oct="5" dur="8" />
    <note xml:id="n6" pname="e" oct="5" dur="16" dots="1"/>
    <note xml:id="n7" pname="f" oct="5" dur="32" accid="s"/>
    <note xml:id="n8" pname="e" oct="5" dur="8"/>
  </tuplet>
</beam>
```



XML

```
<g class="tuplet" id="t1">
  <g class="beam" id="b1">
    <g class="note" id="n1"></g>
    <g class="note" id="n2"></g>
    <g class="note" id="n3"></g>
    <g class="note" id="n4"></g>
  </g>
</g>
<g class="beam" id="b2">
  <g class="tuplet" id="t2">
    <g class="note" id="n5"></g>
    <g class="note" id="n6"></g>
    <g class="note" id="n7"></g>
    <g class="note" id="n8"></g>
  </g>
</g>
```

Transposition

Transposition in Verovio uses the base-40 system that allows for an arbitrary maximum sharp/flat count (where base-40 can handle up to double sharps/flats). The option `--transpose` can be given two types of data: (1) a chromatic interval, or (2) a tonic pitch in the new key with optional direction and octave of transposition added.

Transposition by chromatic interval

For transposition by chromatic intervals, the format is an optional sign, followed by a chromatic quality followed by a diatonic number of steps. Examples: `+M2` = up major second, `-d5` = down diminished fifth

The direction of the interval, with `-` indicating down and no sign or a `+` means up. A special cases is `P1` which is a perfect unison, so `+P1` and `-P1` are equivalent since there is no movement up or down.

For the chromatic quality of the interval, `P` means perfect, `M` means major, `m` means minor, `d` means diminished, `A` means augmented, `dd` means doubly diminished (and so on), `AA` means doubly augmented (and so on). For `[PdA]` the case of the letter does not matter so `[pDa]` should be interpreted as equivalent. `M` and `m` are case-sensitive (major and minor).

The diatonic interval is any (reasonable) positive integer. A unison is `1`, a second is `2`, and so on. Compound intervals an octave and above can also be represented, such as `8` for an octave, a `9` for a ninth (octave plus a second), `10` for a tenth (octave plus a third), `15` is two octaves, and `16` is two octaves plus a second.

Verovio will print an error message if the string option is not formatted correctly, and it will return an error interval which is a very large interval going down.

Example interval names:

name	**meaning**
P1	perfect unison
M2	major second up
+M2	major second up
-M2	major second down
m2	minor second up
d2	diminished second up
dd2	doubly diminished second up
A2	augmented second up
AA2	doubly augmented second up
M3	major third up
P4	perfect fourth up
d4	diminished fourth up
A4	augmented fourth up
P8	perfect octave up
P15	two perfect octaves up
m10	perfect octave plus minor third up

Transposition by tonic pitch

For transposition by tonic pitch names, the format is made up of an optional direction, a pname and an accid.

If no direction is given, then the smallest interval will be chosen. For example if starting from C major and transposing to G major, the calculated interval will be down a perfect fourth, since the G below C is closer than the G above C.

When the direction is +, the next higher pitch that matches the new tonic will define the interval. For C major to G major, this is a perfect fifth up. When the direction is -, the next lowest pitch that matches the new tonic will define the interval. For C major to G major, this is a perfect fourth down.

The + or - direction can be doubled/tripled/etc. to indicate additional octave transpositions. For example --g from C major means to transpose down an octave and a fourth: The fourth to the G below, and then the octave to the next lower G. Likewise, +++g from C major means to transpose up two octaves and a fifth: A fifth to the G above, then ++ means two octaves above that G.

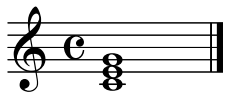
When using a case-insensitive @pname for the tonic of the new key, use ([A-Ga-g]) followed by an optional accid for the new key tonic. This is also case-insensitive: ([Ss]*|[Ff]*).

Examples:

tonic parameter	meaning
g	transpose current tonic to closest G tonic note (up or down a fourth from current tonic)
+g	transpose to the next higher G tonic
-g	transpose down to next lower G tonic
++g	transpose to second next higher G tonic
--g	transpose to second next lower G tonic
ff	transpose to nearest F-flat
-cs	transpose to next lower C-sharp
++BF	transpose up to second next higher B-flat

Illustrated examples

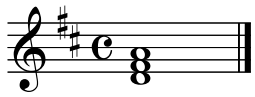
Here is a test example music to transpose - note the @key.sig is expected for transposition to work properly:



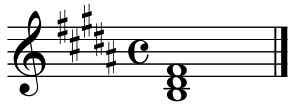
XML

```
<score>
  <scoreDef>
    <staffGrp>
      <staffDef n="1" lines="5" clef.shape="G" clef.line="2" meter.sym="common" key.sig="0"/>
    </staffGrp>
  </scoreDef>
  <section>
    <measure right="end" n="1">
      <staff n="1">
        <layer n="1">
          <chord dur="1">
            <note oct="4" pname="c"/>
            <note oct="4" pname="e"/>
            <note oct="4" pname="g"/>
          </chord>
        </layer>
      </staff>
    </measure>
  </section>
</score>
```

Setting transpose: "M2" will transpose the music up a major second from C to D:



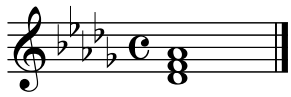
Setting transpose: "-m2" To go down a minor second from C to B:



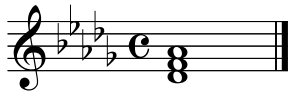
Common intervals: m3 = minor third, M3 = major 3rd, P4 = perfect fourth, P5 = perfect fifth, d5 = diminished fifth, A4 = augmented fourth.

It is also possible to give semitone steps, with 1 being one semitone, 2 being two semitones, etc. This method is less precise, and the computer will make an automatic calculating to minimize the number of accidentals in the target key signature.

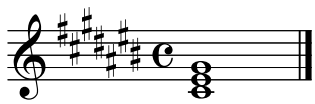
For example transpose: "1" will display in D-flat major:



This is equivalent to going up a minor second with transpose: "-m2" :

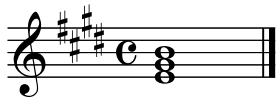


If you need to transpose to C-sharp major, then you cannot use integers, but must use the full musical interval, which in this case is transpose: "A1" for an augmented unison:

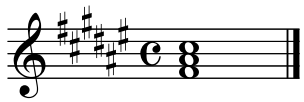


(a1 and A1 are the same, but m2 and M2 are not equivalent).

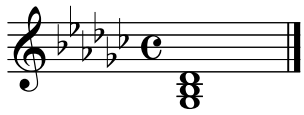
It is also possible to give the tonic note of the new key. For example transpose: "E" means to transpose to E major (or minor, since the mode will not be changed). This feature requires that the music contain key information which is not always present in MusicXML data. It can also be incorrect, which may cause problems, so use this option with care in an automatic situation.



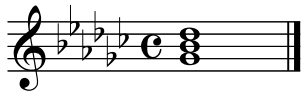
F-sharp major with transpose: "F#" , which is equivalent to a transposition of A4 :



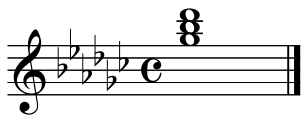
G-flat major with transpose: "Gb" , which is equivalent to d5 :



Notice that this method moves to the closest tonic. To force G-flat major above, add a + with transpose: "+Gb" :



To go another octave above, add two ++ with transpose: "++Gb" :



Algorithm for transposition by tonic

The algorithm for transposition by tonic proceeds as follow:

- Find the key information at the start of the music in each part. If all parts have the same transposition (or no parts have transposition), then use the @pname and @accid as the reference pitch for which an interval will be calculated for the input transposition target tonic.
- If all parts do not have the same transposition, then choose a part that does not have a transposition from which to extract the key information. If all parts have transpositions, but the transpositions are different, then apply transposition to the key information to get it to sounding pitch for one of those instruments and use this transposed pitch as the basis for the key transposition.
- The key information may be stored in one of two main locations:
  staffDef@key.pname / staffDef@key.accid (the most common currently) or

keySig@pname / keySig@accid . The staffDef@key.mode / keySig@mode is not needed. This key information must come before the first notes on the staff. keySig may be found as a child of staffDef , or may be found outside of the staffDef (at the start layer) or in scoreDef if it applies to all staves in the score (or the majority of staves in the score?).

- If there is no key information found before the first notes of the music, print an error warning and do not transpose.
- Once the original key is known, then the interval necessary for transposition can be calculated. The next step is to identify the closest new tonic's octave. For extra + or - in the tonic string, add an octave to the interval to calculate the final interval for transposition.

At this point the key transposition process becomes equivalent to the interval transposition process.

SMuFL fonts

Most music notation software applications use music fonts for rendering music symbols or parts of music symbols. These may include clefs, note heads, time signatures or articulation signs. However, these fonts often have incompatible code points – the internal location within the font that points to a symbol. They are most of the time developed with no common agreement on which code point represents which character. The code point for the G clef symbol in one font may be the code point used for a quarter rest in another, or may be simply undefined. Furthermore, they usually have their own metric and positioning system for specifying what the size of the glyph is and where its baseline is. Because of this, music fonts are difficult to use interchangeably.

To address this, the Standard Music Font Layout (SMuFL) specification has been developed to attempt to harmonize code points across music fonts by specifying code points and symbol sizes for music fonts. SMuFL gives users the ability to reference specific Unicode code points with the understanding that it would represent the same, or similar, symbol across fonts. This presents new opportunities for exploring visual representations of music within a music encoding system without necessarily tying them to a particular font. While previous music encoding systems could not reference font code points without becoming tied to that font for representation, the introduction of SMuFL to music encoding can provide a reference to a particular graphical symbol that should be used to render a given encoding.

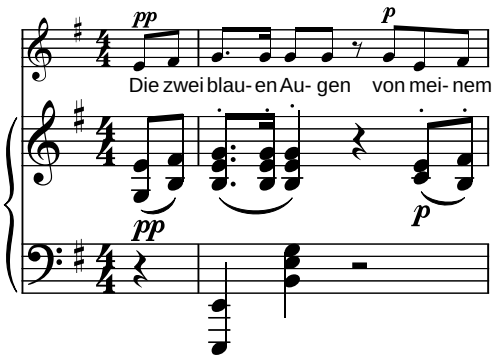
Verovio follows the SMuFL specification. It means that it is possible to easily change the music font used in Verovio for personalised output. Verovio includes the Leipzig font, its own SMuFL-compliant music font. Leipzig was initially developed by Etienne Darbellay and Jean-François Marti as part of the Wolfgang music notation software. It is SMuFL compliant since version 5.0 and distributed under the SIL Open Font License.

Verovio also supports and includes the Bravura font designed by Daniel Spreadbury, and the Gootville and Leland fonts designed by the MuseScore community.

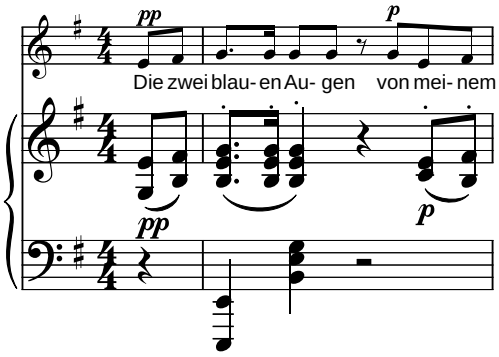
Fonts included can be selected by setting the --font option. For example, the Bravura font can be selected with the --font Bravura option on the command-line tool or by adding { font: "Bravura" } in the JavaScript toolkit options.

Examples

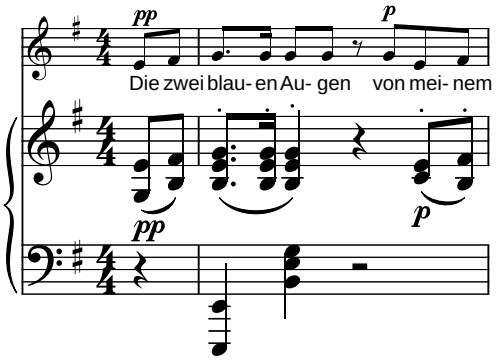
Leipzig



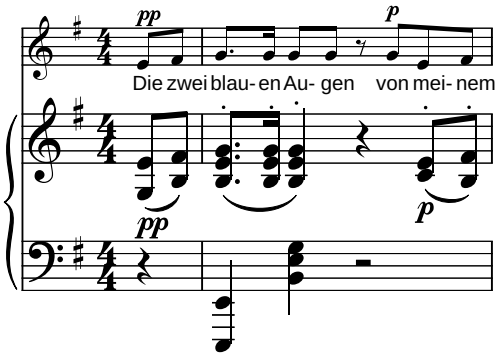
Bravura



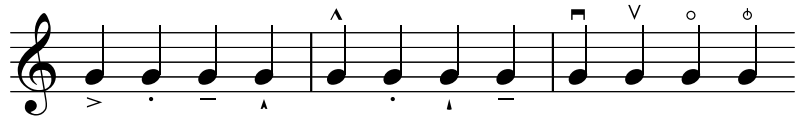
Gootville



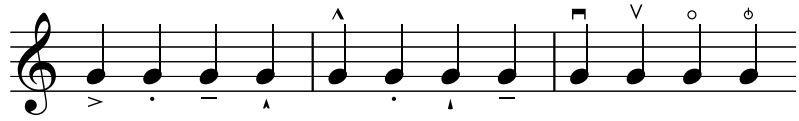
Leland



Leipzig



Bravura

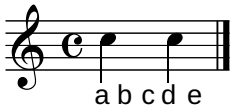


Music symbols in text

For cases when music symbols are displayed within text, Verovio uses theVerovioText font. This font is a based on Leipzig and includes a limited set of symbols. They include:

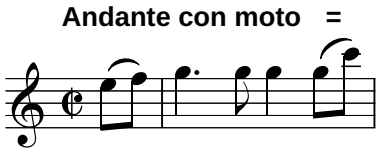
- Note symbols for tempo indications
- Lyric elision symbols
- Figured bass symbols
- Dynamic symbols

Examples



XML

```
<verse n="1">
  <syl con="b">a</syl>
  <syl con="b">b</syl>
  <syl>c</syl>
</verse>
```



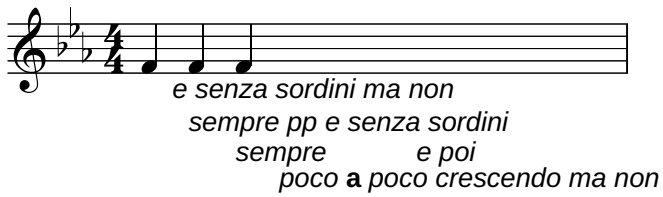
XML

```
<tempo staff="1" tstamp="1.000000">Andante con moto <rend fontname="VerovioText">&#xE1D5;</rend> = 70</tempo>
```

Characters in tempo indications can be encoded as Unicode characters or as entities (e.g., ). See the section on MEI in Output formats for more information on how to control them.

Dynamics

For dynamics, the font is used only where text and dynamic symbols are mixed together. Verovio automatically detects dynamic symbols within text and displays them appropriately. In such cases, however, the music font will always be VerovioText and the font specified with the --font option will not be used.



XML

```
<dynam staff="1" tstamp="1.000000">ff e senza sordini ma non sfz</dynam>
<dynam staff="1" tstamp="2.000000">
  <rend fontfam="Times">sempre pp e senza sordini</rend>
</dynam>
<dynam staff="1" tstamp="3.000000">sempre fff rfz e poi<lb/>poco <rend fontstyle="normal" fontweight="
bold">a</rend> poco crescendo ma non ffff<lb/>troppo</dynam>
```

In some cases, it might be desirable to disable the automatic detection of dynamic symbols and the use of the music font. This can be achieved by setting a text font explicitly, as illustrated with the <rend fontfam="Times"> in the second dynamic in the example above.

Controlling the SVG output

[in preparation]

HTML5

Towards SVG 2.0

Converting to PDF

Mensural notation

[in preparation]

Duration alignment

Layout

Ligatures

Toolkit Reference

Input formats

MEI

> This is still experimental

Humdrum

MusicXML

Plain and Easy

ABC

Output formats

SVG

MEI

MIDI

Timemap

Toolkit methods

Edit

Edit the MEI data.

Returns

bool – True if the edit action was successfully applied

Parameters

Name	Type	Default	Description
editorAction	const std::string &	∅	The editor actions as a stringified JSON object

Original header

C++

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

Example call

PYTHON

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

EditInfo

Return the editor status.

Returns

std::string – The editor status as a string

Original header

C++

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

Example call

PYTHON

```
result = toolkit.editInfo()
```

GetAvailableOptions

Return all available options grouped by category.

For each option, returns the type, the default value, and the minimum and maximum value (when available)

Returns

std::string – A stringified JSON object

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.

The attributes returned include the ones not supported by Verovio

Returns

std::string – A stringified JSON object with all attributes

Parameters

Name	Type	Default	Description
xmlId	const std::string &	∅	the ID (xml:id) of the element being looked for

Original header

C++

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

Example call

PYTHON

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

The method performs a lookup in the loaded MEI tree and will return all attributes for the retrieved element. This includes attributes currently not supported by Verovio. Looking in the MEI tree means that looking for

elements added dynamically for the rendering by Verovio will no be found. This is the case for system elements when loading score-based MEI, or meterSig or clef elements displayed at the beginning of a system.

GetElementsAtTime

Returns array of IDs of elements being currently played.

Returns

std::string – A stringified JSON object with the page and notes being played

Parameters

Name	Type	Default	Description
millisec	int	∅	The time in milliseconds

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 – A stringified JSON object with all IDs

Parameters

Name	Type	Default	Description
xmlId	const std::string &	∅	the ID (xml:id) of the element being looked for

Original header

C++

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

Example call

PYTHON

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

GetHumdrum

Get the humdrum buffer.

Returns

std::string – The humdrum buffer as a string

Original header

C++

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

Example call

```
PYTHON
result = toolkit.getHumdrum()
```

GetHumdrumFile

Write the humdrum buffer to the file.

This methods is not available in the JavaScript version of the toolkit.

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)
```

GetLog

Get the log content for the latest operation.

Returns

std::string – The log content as a string

Original header

```
C++
std::string vrv::Toolkit::GetLog()
```

Example call

```
PYTHON
result = toolkit.getLog()
```

GetMEI

Get the MEI as a string.

Returns

std::string

Parameters

Name	Type	Default	Description
jsonOptions	const std::string &	""	A stringified JSON object with the output options pageNo: integer; (1-based), all pages if none (or 0) specified; scoreBased: true or false; true by default; removeIds: true or false; false by default - remove all @xml:id not used in the data;

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 – A stringified JSON object with the MIDI values

Parameters

Name	Type	Default	Description
xmlId	const std::string &	∅	the ID (xml:id) of the element being looked for

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 – A stringified JSON object with all IDs

Parameters

Name	Type	Default	Description
xmlId	const std::string &	∅	the ID (xml:id) of the element being looked for

Original header

C++

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

Example call

PYTHON

result = toolkit.getNotatedIdForElement(xmlId)

GetOption

Get the value for an option.

Returns

std::string – The option value as a string

Parameters

Name	Type	Default	Description
option	const std::string &	∅	The name of the option
defaultValue	bool	false	True to get the default value of the option

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

Return a dictionary of all the options.

Returns

std::string – A stringified JSON object

Parameters

Name	Type	Default	Description
defaultValues	bool	∅	True for getting the default values and false for the current values

Original header

C++

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

Example call

PYTHON

result = toolkit.getOptions(defaultValues)

GetPageCount

Return the number of pages in the loaded document.

The number of pages depends one the page size and if encoded layout was taken into account or not.

Returns

int – The number of pages

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

int – the page number (1-based) where the element is (0 if not found)

Parameters

Name	Type	Default	Description
xmlId	const std::string &	∅	the ID (xml:id) of the element being looked for

Original header

C++

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

Example call

PYTHON

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

GetScale

Get the scale option.

Returns

int – the scale option as integer

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

int – The time in milliseconds

Parameters

Name	Type	Default	Description
xmlId	const std::string &	∅	the ID (xml:id) of the element being looked for

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.

Return scoreTimeOnset, scoreTimeOffset, scoreTimeTiedDuration, realTimeOnsetMilliseconds, realTimeOffsetMilliseconds, realTimeTiedDurationMilliseconds.

Returns

std::string – A stringified JSON object with the values

Parameters

Name	Type	Default	Description
xmlId	const std::string &	∅	the ID (xml:id) of the element being looked for

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 – The ID as as string

Original header

C++

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

Example call

PYTHON

```
result = toolkit.getUuid()
```

GetVersion

Return the version number.

Returns

std::string – the version number as a string

Original header

C++

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

Example call

PYTHON

```
result = toolkit.getVersion()
```

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 – True if the data was successfully loaded

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 – True if the file was successfully loaded

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 – True if the data was successfully loaded

Parameters

Name	Type	Default	Description
data	const std::string &	∅	A ZIP file as a 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.

Returns

bool – True if the data was successfully loaded

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()
```

RenderToMIDI

Render the document to MIDI.

Returns

std::string – A MIDI file as a base64 encoded string

Original header

C++

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

Example call

PYTHON

```
result = toolkit.renderToMIDI()
```

RenderToMIDIFile

Render a document to MIDI and save it to the file.

This methods is not available in the JavaScript version of the toolkit.

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 a document to Plaine and Easie.

Only the top staff / layer is exported.

Returns

std::string – The PAE as a string

Original header

C++

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

Example call

PYTHON

```
result = toolkit.renderToPAE()
```

RenderToPAEFile

Render a document to Plaine and Easie and save it to the file.

Only the top staff / layer is exported. This methods is not available in the JavaScript version of the toolkit.

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 a page to SVG.

Returns

std::string – The SVG page as a string

Parameters

Name	Type	Default	Description
pageNo	int	1	The page to render (1-based)
xmlDeclaration	bool	false	True for including the xml declaration in the SVG output

Original header

C++

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

Example call

PYTHON

result = toolkit.renderToSVG(pageNo, xmlDeclaration)

RenderToSVGFile

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

This methods is not available in the JavaScript version of the toolkit.

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

Render a document to a timemap.

Returns

std::string – The timemap as a string

Original header

C++

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

Example call

PYTHON

```
result = toolkit.renderToTimemap()
```

RenderToTimemapFile

Render a document to timemap and save it to the file.

This methods is not available in the JavaScript version of the toolkit.

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

Get the MEI and save it to the file.

This methods is not available in the JavaScript version of the toolkit.

Returns

bool – True if the file was successfully written

Parameters

Name	Type	Default	Description
filename	const std::string &	∅	The output filename
jsonOptions	const std::string &	""	A stringified JSON object with the output options

Original header

C++

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

Example call

PYTHON

```
result = toolkit.saveFile(filename, jsonOptions)
```

SetInputFrom

Set the input from option.

Returns

bool – True if the option was successfully set

Parameters

Name	Type	Default	Description
inputFrom	std::string const &	∅	the input from value as string

Original header

C++

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

Example call

PYTHON

```
result = toolkit.setInputFrom(inputFrom)
```

SetOption

Set the value for an option.

Returns

bool – True if the option was successfully set

Parameters

Name	Type	Default	Description
option	const std::string &	∅	The name of the option
value	const std::string &	∅	The option value as 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

Set option values.

The name of each option to be set is to be given as JSON key.

Returns

bool – True if the options were successfully set

Parameters

Name	Type	Default	Description
jsonOptions	const std::string &	∅	A stringified JSON objects with the output options

Original header

C++

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

Example call

PYTHON

```
result = toolkit.setOptions(jsonOptions)
```

SetOutputTo

Set the output to option.

Returns

bool – True if the option was successfully set

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 – True if the resources was successfully loaded

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

Set the scale option.

Returns

bool – True if the option was successfully set

Parameters

Name	Type	Default	Description
scale	int	∅	the scale value as integer

Original header

C++

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

Example call

PYTHON

result = toolkit.setScale(scale)

Toolkit

Constructor.

Parameters

Name	Type	Default	Description
initFont	bool	true	If set to false, resource path is not initialized and SetResourcePath will have to be called explicitely

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.

-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”)

See also: 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”)

See also: 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”)

See also: 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

The version number includes major, minor and revision numbers and the last number of the git commit.

-x, --xml-id-seed <i>

Seed the random number generator for XML IDs (default is random)

Input and page layout options

--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: "")

--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

See also: 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-format-raw

Writes MEI out with no line indenting or non-content newlines.

--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)

--preserve-analytical-markup

Preserves the analytical markup in MEI

--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

--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

--alternative-octave-symbols

Use alternative symbols for displaying octaves

--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)

--dynam-dist <f>

The default distance from the staff for dynamic marks
(default: 1.0; min: 0.5; max: 16.0)

--engraving-defaults <s>

Path to json file describing defaults for engraving SMuFL elements

--font <s>

Set the music font
(default: “Leipzig”)

See also: SMuFL fonts

--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)

--harm-dist <f>

The default distance from the staff of harmonic indications
(default: 1.0; min: 0.5; max: 16.0)

--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)

--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)

--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)

--mnum-interval <i>

How frequently to place measure numbers

--multi-rest-style <s>

Rendering style of multiple measure rests
(default: “auto”; other values: [‘auto’, ‘default’, ‘block’, ‘symbols’])

--octave-line-thickness <f>

The thickness of the line used for an octave line
(default: 0.2; min: 0.1; max: 1.0)

--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)

--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

--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: “”)

See also: Transposition

--transpose-selected-only

Transpose only the selected content and ignore unselected editorial content

Element margins

--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)

--left-margin-right-bar-line <f>

The margin for right barLine in MEI units
(default: 1.0; min: 0.0; max: 2.0)

--left-margin-tab-dur-sym <f>

The margin for tabDurSym 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)

--right-margin-tab-dur-sym <f>

The right margin for tabDurSym 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

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 .`)

Basic usage

For typesetting an MEI file with the default options, you need to do:

TERMINAL

```
verovio -o output.svg Hummel_Concerto_for_trumpet.mei
```

If you use a version locally that is not installed, do not forget to add the `-r` parameter:

TERMINAL

```
./verovio -r ../data -o output.svg Hummel_Concerto_for_trumpet.mei
```

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, 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 eachrelease.

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 theGetting started section of the Tutorial 1: First steps 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-digital/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

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 .

Use of this

The convention for the pointer this is to use it for method calls and not to use if for member access because these are prefixed with m_ .

As it stands, the codebase is not consistently following this convention

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. See the section Generate code with LibMEI for detailed information on how to modify and generate this code.

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

[in preparation]

Generate code with libMEI

Verovio uses a forked version of LibMEI, a library that generates code directly from the MEI schema. It can be adapted to generate code in any language. For Verovio, it is used to generate C++ code. The code generated with LibMEI is included in the Verovio repository in the `./libmei` directory and the LibMEI repository does not need to be cloned for building Verovio.

Whenever the MEI schema is modified, this code needs to be re-generated in order to integrate these changes. However, since Verovio implements only a small subset of the MEI schema, this really needs to be done only for the changes in the schema that touch features supported by Verovio. This means that the code within the `./libmei` directory should never be edited by hand because any change will be overwritten by the LibMEI output when the code generated from the schema needs to be updated and LibMEI is run again.

Running LibMEI

In order to update to code generated with LibMEI, you need to clone the forked version of LibMEI.

LibMEI takes a compiled ODD as input. You need to run, from the LibMEI directory:

TERMINAL

```
python tools/parseschema2.py -l vrv -o /path/to/the/verovio/directory -i tools/includes/vrv mei/dev/mei-verovio_compiled.odd
```

You need to set to option `-o` to point to the Verovio directory where the `./libmei` files will be written.

Customization

Verovio currently uses an MEI customization that adds or modified a few elements. It is defined in the `./mei/dev/mei-verovio.xml` file. If you want to makes changes to it, you can make them there. You will need to re-generate the `./mei/dev/mei-verovio_compiled.odd` ODD file. This can be done using the Edirom MEI Garage. Alternatively, you can also use the MEI command-line script. To do so, you will need to a clone of the MEI repository, copy your customization file (e.g., `mei-verovio.xml`) into it and do:

TERMINAL

```
ant init
ant -lib lib/saxon/saxon9he.jar -Dcustomization.path=mei-verovio.xml
```

The ODD file will be written to ./dist/schemata/mei-verovio_compiled.odd , which you can use as new input file for LibMEI.

Adding SMuFL glyphs

All SMuFL glyphs used by Verovio have to be available in theLeipzig 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