Reference Book

For Verovio version 6.0

Hosted in this repostiory and generated on 28 January 2026 from a85874b

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.

License

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 the Music 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

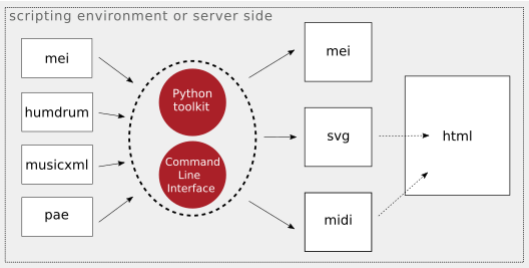
Architecture possibilities

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.

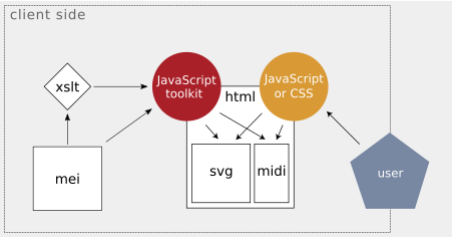
First of all, it can be built and used as a standalone command-line tool. This option is well suited to scripting environments and applications. The command-line tool can be used to render music notation files into SVG or into MIDI files. These files can be embedded in HTML files with everything happening 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).

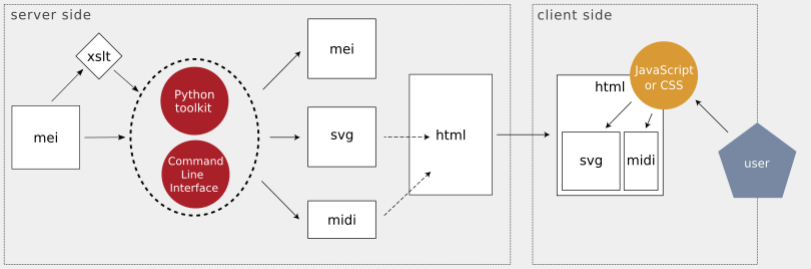
Resulting SVG or MEI documents can then be embedded in a HTML page or used as such.



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.

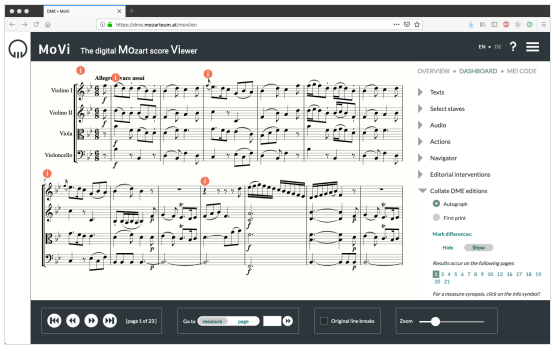


Application examples

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.

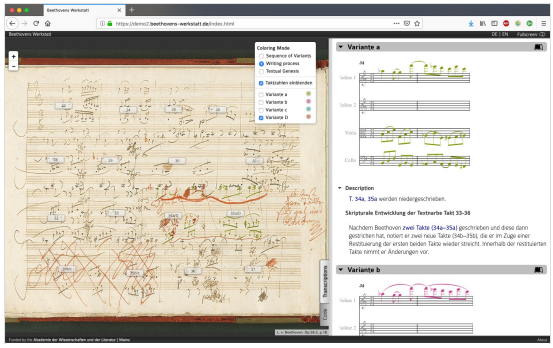
Critical editions

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).



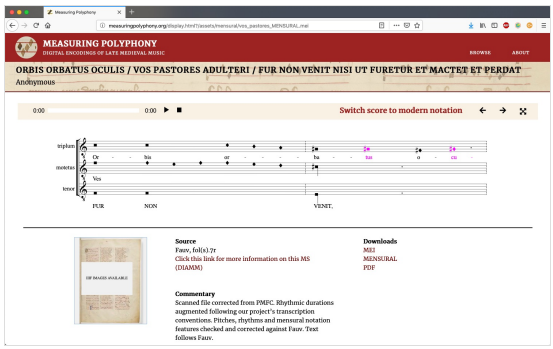
Genetic editing

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.



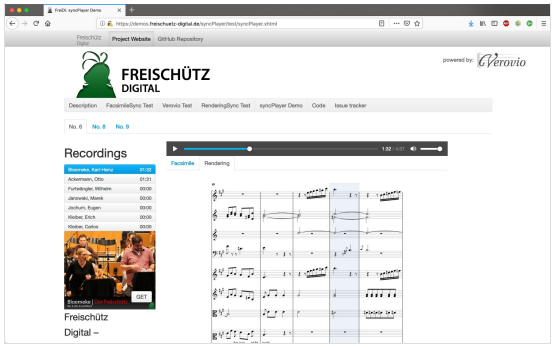
Early music

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.



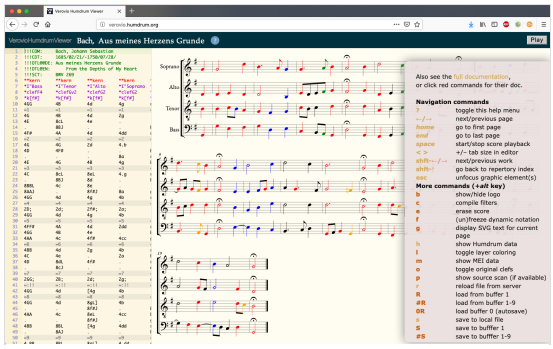
Audio alignment

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.



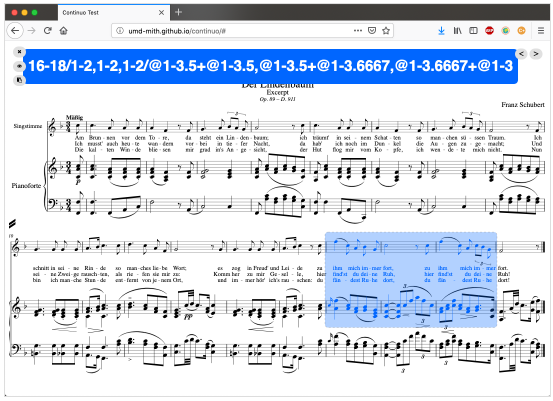
Music notation editing

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.



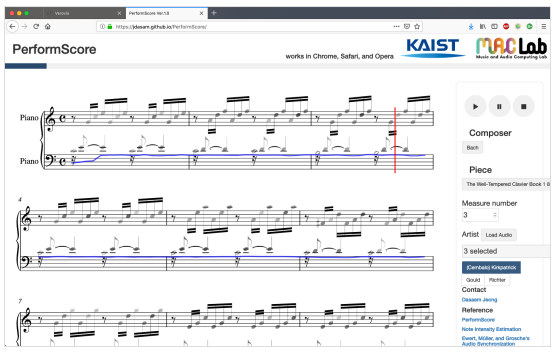
Music addressability

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.



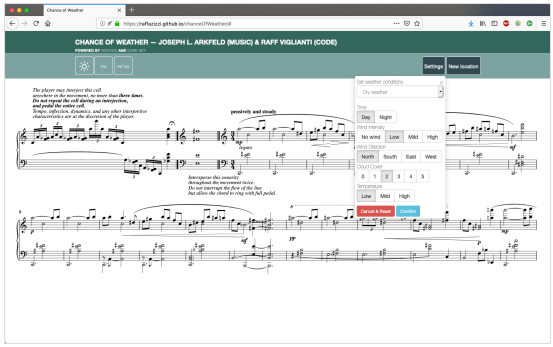
Visualisation

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.



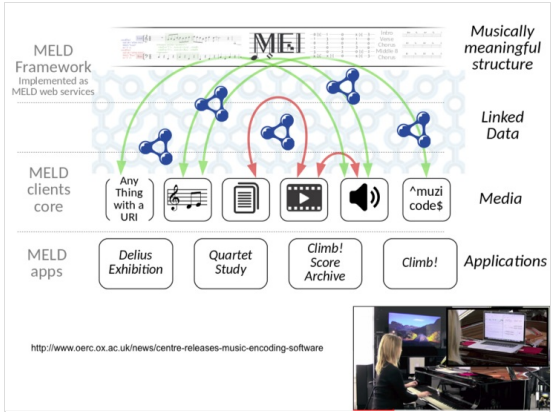
Composition

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.



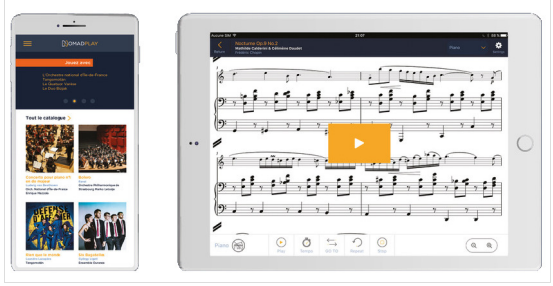
Performance

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:

- Enote in the App Store
- Trala in Google Play
- NomadPlay web application

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!

Example uses

Resources using Verovio

Name	Type	Description
Verovio Humdrum Viewer	editor	An online semi-graphical Humdrum data editor (can also be used to textually edit other digital scores compliant with verovio).
MoVI	repertory	The digital Mozart digital score VIewer at the Mozarteum
Tasso in Music Project	repertory	Musical settings of the poetry of Torquato Tasso
Measuring Polyphony	repertory	Late medieval music in black mensural and modern notations
Probstücke Digital	repertory	open and critical digital edition of Mattheson’s test pieces
370 Bach Chorales	repertory	Online edition of Bach chorales, including an interactive typesetter page that allows for creating musical examples for online display or use in papers.
Humdrum Notation Plugin	tool	Javascript interface to verovio for displaying multiple musical examples on a webpage
Music Sheet Viewer	tool	WordPress plugin for displaying graphical music from MEI data

Digital score repositories on Github

Here is a list of digital score repositories on Github that can be displayed with verovio:

Link	Encoding	Description
MEI complete examples	MEI	86 various works encoded in MEI
Mozart Piano Sonatas	Humdrum	17 Piano sonatas by W.A. Mozart from the Alte Mozart-Ausgabe (in VHV)
Beethoven Piano Sonatas	Humdrum	32 Piano sonatas by L. van Beethoven, edited by Paul Dukas (in VHV)
Josquin Research Project	Humdrum	Over 1000 scores of early Renaissance music in modern editions (website)
Tasso in Music Project	Humdrum	Critical edition of 650 Late Renaissance madrigals using the poetry of Torquato Tasso for lyrics. (website)
Music of Scott Joplin	Humdrum	Digital scores of most of Scott Joplins music
Chopin mazurkas	Humdrum	Digital scores of Chopin’s mazurkas
Chopin preludes	Humdrum	Digital scores of Chopin’s op. 24 preludes
J.N. Hummel preludes, op. 67	Humdrum	24 improvisatory prelude examples in every key
370 Bach chorales	Humdrum	Chorales collected by C.P.E. Bach after his father’s death (website)
Deutscher Liederschatz	Humdrum	200 harmonized songs from vol. 1, edited by Ludwig Erk
Beethoven string quartets	Humdrum	18 string quartets by Ludwig van Beethoven

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="https://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) => {
      verovio.module.onRuntimeInitialized = () => {
        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 `verovio.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="https://www.verovio.org/javascript/latest/verovio-toolkit-wasm.js" defer></script>
    <script>
      document.addEventListener("DOMContentLoaded", (event) => {
          verovio.module.onRuntimeInitialized = () => {
            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="https://www.verovio.org/javascript/latest/verovio-toolkit-wasm.js" defer></script>
    <script>
      document.addEventListener("DOMContentLoaded", (event) => {
        verovio.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 given as percentage of the normal (100) output. 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());
// for the default values
console.log("Verovio options:", tk.getDefaultOptions());
```

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="https://www.verovio.org/javascript/latest/verovio-toolkit-wasm.js" defer></script>
    <script>
      document.addEventListener("DOMContentLoaded", (event) => {
          verovio.module.onRuntimeInitialized = async _ => {
            let tk = new verovio.toolkit();
            console.log("Verovio has loaded!");
            tk.setOptions({
              scale: 30,
              landscape: true,
              adjustPageWidth: true
            });
            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

[in preparation]

CSS and SVG

Understanding the structure of the SVG

The SVG produced by Verovio can be manipulated further. In this tutorial, you are going to use CSS to highlight some content of the output.

One key feature of Verovio is that it preserves the structure of the MEI in the SVG output. For example, a chord with two notes encoded in MEI:

XML

```
<chord xml:id="c1">
  <note/>
  <note/>
</chord>
```

will have a the following structure in the SVG:

XML

```
<g class="chord" id="c1">
  <g class="note"/>
  <g class="note"/>
</g>
```

You will notice that both the tree structure is preserved and that the MEI element names are passed as `@class` attribute values in the SVG elements, as well as the `@xml:id` of the MEI element as `@id` in the SVG.

Since SVG can be styled with CSS, it is straightforward to modify the appearance of elements and their contents.

Modifying the appearance can be done with a CSS file, or programmatically.

Applying CSS to the SVG

In the CSS file you need to create rules to be applied to the SVG `<g>` elements - simply `g` in CSS - together with the class selector corresponding to the MEI element name. For example, `g.tempo` modifies MEI `<tempo>` elements.

```
g.tempo {
  // ... some CSS properties
}
```

To modify the color, you need to change the `fill` property, and - in some cases - also the `color` property. The CSS rule will look like:

```
{
  fill: crimson;
  color: crimson;
}
```

You can also select elements based on their hierarchy. For example, you can select `<artic>` within `<chord>` with:

```
g.chord g.artic {
}
```

CSS can be animated, for example by making the colors pulsing. You need to specify an animation name with a duration and an iteration count together with corresponding key frames:

```
{
  animation-name: pulse;
  animation-duration: 1.0s;
  animation-iteration-count: infinite;
}

@keyframes pulse {
  0%   { fill: orange; }
  50%  { fill: brown; }
  100% { fill: orange; }
}
```

CSS can also be used to change the opacity of an element. The opacity value can range from 0.0 (transparent) to 1.0 (normal default value).

Adjusting the style programmatically

In applications, it is often useful to modify the CSS programmatically, for example in response to some user interactions.

To do so, elements can be accessed by element and class name in the same way as with CSS. For example, for retrieving all rests, you can do:

JAVASCRIPT

```
let rests = document.querySelectorAll('g.rest');
```

You can then loop through the list of rests with:

JAVASCRIPT

```
for (let rest of rests) {
    // you have now access to the rest one by one and can modify their style
}
```

To modify the style of an element, you can assign the desired value to the corresponding key. For example, in order to change the color (fill) or a rest (element), you need to do:

JAVASCRIPT

```
rest.style.fill = "dodgerblue";
```

Using custom data-* attributes

The attributes in the SVG `<g>` elements corresponding to the MEI elements is not limited to the @class carrying the MEI element name and the MEI `@xml:id` passed as `@id`. Verovio also passes MEI `@type` values as additional `@class` in the SVG.

However, in many cases, applications need to have access at other attribute values. To do so, one can use the `svgAdditionalAttribute` option to specify which attributes can be made available in the SVG output. For example, for making the note pitch name and the note octave accessible, you can add the following option values to Verovio's `setOptions()` method:

JSON

```
{
  svgAdditionalAttribute: ["note@pname", "note@oct"]
}
```

With this option, each `g.note` element in the SVG will also have a `data-pname` and a `data-oct` attribute carrying the original MEI attribute value. For example, a note in MEI and the corresponding SVG element will be:

XML

```
<note pname="c" oct="5"/>
```

XML

```
<g class="note" data-pname="c" data-oct="5"/>
```

This can be used in the query selector to restrict the matches to elements having specific attribute values. For example, for selecting the C5 notes, you would do:

JAVASCRIPT

```
let c5s = document.querySelectorAll('g[data-pname="c"][data-oct="5"]');
```

Accessing MEI attribute values programmatically

Custom `data-*` attributes are straightforward and easy to use with CSS selectors. However, selectors can have some limits, and it is not always possible to know in advance all the attributes that needed in the SVG. Furthermore, if the list of attributes becomes too long, the SVG might become overloaded.

In this case, it is possible and preferable to access them programmatically with JavaScript. This can be done through the `getElementAttr()` toolkit method that gives access to all the MEI attributes of a given element, including attributes not currently supported or not used by Verovio. It takes an `xml:id` value as the input parameter and returns a JSON object with all the attributes for that element from the MEI encoding. For example, given this MEI:

XML

```
<rest xml:id="r123" dur="4" dots="1">

```js
let attr = tk.getElementAttr("r123");
```

You can then look at any attributes specifically in the JSON object returned, for example `attr.dur` for the MEI `@dur` of the rest:

JAVASCRIPT

```
if (attr.dur && attr.dur == "1") {
  // This is a whole note rest
}
```
```

Full example

Open this example in a new window.

HTML /
JAVASCRIPT

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">
    <title>Understanding the structure of the SVG</title>
    <!-- A stylesheet for modifying the appearance of the output -->
    <link href="solution.css" rel="stylesheet" type="text/css" />
    <!-- Verovio -->
    <script src="https://www.verovio.org/javascript/develop/verovio-toolkit-wasm.js" defer></script>
  </head>
  <body>
    <script>
      document.addEventListener("DOMContentLoaded", (event) => {
        verovio.module.onRuntimeInitialized = function () {
          // This line initializes the Verovio toolkit
          const tk = new verovio.toolkit();

          let zoom = 80;
          let pageHeight = document.body.clientHeight * 100 / zoom;
          let pageWidth = document.body.clientWidth * 100 / zoom;

          options = {
            pageHeight: pageHeight,
            pageWidth: pageWidth,
            scale: zoom,
            // Add an option to pass note@pname and note@oct as svg @data-*
            svgAdditionalAttribute: ["note@pname", "note@oct"]
          };
          tk.setOptions(options);

          // This line fetches the MEI file we want to render...
          fetch('https://www.verovio.org/examples/downloads/Schubert_Lindenbaum.mei')
            // ... then receives the response and "unpacks" the MEI from it
            .then((response) => response.text())
            .then((mei) => {
              // ... then we can load the data into Verovio ...
              tk.loadData(mei);
              // ... and generate the SVG for the first page
              let svg = tk.renderToSVG(1);
              // ... and finally gets the <div> element with the ID we specified,
              // and sets the content (innerHTML) to the SVG that we just generated.
              document.getElementById("notation").innerHTML = svg;

              // Get all the rests by selecting <g> with attribute class 'rest' ...
              let rests = document.querySelectorAll('g.rest');
              // ... and change their color by setting their style.fill value
              for (let rest of rests) {
                rest.style.fill = "dodgerblue";
              }

              // Get all the notes with @pname="c" and @oct="5" and change their color
              let c5s = document.querySelectorAll('g[data-pname="c"][data-oct="5"]');
              for (let c5 of c5s) {
                c5.style.fill = "aqua";
              }

              // Get all the verses ...
              let verses = document.querySelectorAll('g.verse');
              // ... and use the 'getElementAttr()' to retrieve all attributes ...
              for (let verse of verses) {
                let attr = tk.getElementAttr(verse.id);
                // ... and change to color when @n exists and is greater than 1
                if (attr.n && attr.n > 1) verse.style.fill = "darkcyan";
              }
            });
        }
      });
    </script>
    <!-- The div where we are going to insert the SVG -->
    <div id="notation" />
  </body>
</html>
```

Encoding formats

The primary notation encoding format used with Verovio is MEI; however, Verovio supports conversion from a number of other formats, including MusicXML. In this tutorial we will look at how we can get Verovio to convert a compressed MusicXML file to MEI.

Saving as MEI

When loading a MusicXML file into Verovio, it converts this internally into MEI, which we will be able to export as MEI.

To do this, and to make our lives easier, we will use a JavaScript library that helps us save a file. In the `<head>` section of your file, add the following `<script>` tag:

HTML /
JAVASCRIPT

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/FileSaver.js/2.0.0/FileSaver.min.js"></script>
```

To download the MEI file, we will add a button to our page that will trigger a save of the MEI content from Verovio. Just like in previous tutorials, add a “click” handler for a button:

JAVASCRIPT

```
document.getElementById("saveMEI").addEventListener("click", (event) => {
    let meiContent = tk.getMEI();
    var myBlob = new Blob([meiContent], {type: "application/xml"});
    saveAs(myBlob, "meifile.mei");
});
```

That is, we get the button element (`id="saveMEI"`), and then tell the button what to do when it is clicked. To get the MEI output we can use the `getMEI()` method on the toolkit. This will return a formatted string containing the MEI XML output.

Then we do a few JavaScript things to get the download to work. First we create a new “Blob”, which is just a wrapper around some arbitrary data. Then we call the `saveAs` function from the FileSaver.js library we loaded earlier.

Compressed MusicXML

You may not be aware of it, but there are actually two forms of MusicXML files! Typically, those that end with `.xml` or `.musicxml` are “plain” XML files, and we can load them directly. Here we are going to load a “compressed” MusicXML file, normally ending with a `.mxl` extension. These files are just ZIP files, but have a fixed file-and-folder structure within them. Verovio supports loading these types of files as well, but with a bit of special handling needed.

To display this we follow the same methods as loading previous files, except for two main differences:

- We use `response.arrayBuffer()` instead of `response.text()` to read the initial response;
- We use Verovio’s `loadZipDataBuffer` toolkit method, instead of the regular `loadData` method.

Wrapping up

With Verovio you can easily convert MusicXML files, in both compressed and uncompressed formats, to MEI. There are a number of other formats that Verovio supports as well, but some need to be specially enabled if you wish to use them.

Check out the chapter on Input formats in the Verovio book for more details.

Full example

Open this example in a new window.

HTML /
JAVASCRIPT

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">
    <title>Convert compressed MusicXML</title>
    <script src="https://www.verovio.org/javascript/develop/verovio-toolkit-wasm.js" defer></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/FileSaver.js/2.0.0/FileSaver.min.js"></script>
  </head>
  <body>
    <button id="saveMEI">Save my MEI!</button>
    <div id="notation"></div>
    <script>
      /**
        We need to wait for the whole page to load before we try to
        work with Verovio.
      **/
      document.addEventListener("DOMContentLoaded", (event) => {
        verovio.module.onRuntimeInitialized = function () {
          // This line initializes the Verovio toolkit
          const tk = new verovio.toolkit();

          document.getElementById("saveMEI").addEventListener("click", (event) => {
            let meiContent = tk.getMEI();
            var myBlob = new Blob([meiContent], {type: "application/xml"});
            saveAs(myBlob, "meifile.mei");
          });

          fetch("https://www.verovio.org/examples/musicxml/Schubert_Staendchen_D.923.mxl")
            .then((response) => response.arrayBuffer())
            .then((mxlData) => {
              tk.loadZipDataBuffer(mxlData);
              let svg = tk.renderToSVG(1);
              document.getElementById("notation").innerHTML = svg;
            }).catch(e => {
              console.log("An error occurred when loading the file!");
              console.log(e);
            });
        }
      });
    </script>
  </body>
</html>
```

Playing the MIDI output

Verovio can produce basic MIDI files and this feature is also available in the JavaScript toolkit. It can be used to play an MEI file directly in the browser as demonstrated in this tutorial.

Add a MIDI player

MIDI playback is not built-in to the web browser, nor available directly in Verovio. This means that we need to add a MIDI player to our page. For this tutorial we are going to use MIDIjs. You need to add a <script> tag with the following src attribute:

```
https://www.midijs.net/lib/midi.js
```

We also need buttons to handle the start and stop playing events. Add them a the top of the body above the notation div:

HTML / JAVASCRIPT

```
<button id="playMIDI">Play</button>
<button id="stopMIDI">Stop</button>
```

You also need to make sure they do something when clicking on them:

JAVASCRIPT

```
document.getElementById("playMIDI").addEventListener("click", playMIDIHandler);
document.getElementById("stopMIDI").addEventListener("click", stopMIDIHandler);
```

We now need to define what actually happens when the user clicks. That is, defining the playMIDIHandler and stopMIDIHandler functions we just bound to the button event listeners. We can scaffold them with:

JAVASCRIPT

```
const playMIDIHandler = function () {
  // do something to start playing
}

const stopMIDIHandler = function () {
  // do something to stop playing
}
```

At this stage they do nothing. To start playing, we need to get the MIDI data produced by Verovio and pass it to the player. We will use the Verovio `renderToMIDI` method that returns a MIDI file encoded as a base64 string, which we can pass to MIDIjs:

JAVASCRIPT

```
// Get the MIDI file from the Verovio toolkit
let base64midi = tk.renderToMIDI();
// Add the data URL prefixes describing the content
let midiString = 'data:audio/midi;base64,' + base64midi;
// Pass it to play to MIDIjs
MIDIjs.play(midiString);
```

Stop playing is even simpler. You only need to tell MIDIjs to do so:

JAVASCRIPT

```
MIDIjs.stop();
```

The examples above will be followed to write the body of the `playMIDIHandler` and `stopMIDIHandler` functions.

Highlighting the notes while playing

The MIDIjs player provides us with a callback function that gives us the current playback time. We can use this for highlighting the notes as the MIDI file plays! Each time the callback function is called, we can highlight the notes that are currently played, and automatically move to the next page if necessary.

You need to start by defining a callback function, similar to the button event we wrote earlier:

JAVASCRIPT

```
const midiHightlightingHandler = function (event) {
  // Do something everytime the callback function is called
}
```

You will notice that the function has an `event` parameter that will give us information about the current event. What we need to use is `event.time` that indicates the current playing time in seconds. We are going to use this and the Verovio `getElementsAtTime` method that retrieves all elements being played at a given time in order to obtain the list of notes being played:

JAVASCRIPT

```
// Get elements at a time in milliseconds (time from the player is in seconds)
let currentElements = tk.getElementsAtTime(event.time * 1000);
```

Now we should check that a page number was set. This is just to ensure we do not end up in an undefined state; for example, if the file is not loaded, or if we asked for elements that do not exist. If the page is 0, something went wrong and we should return:

JAVASCRIPT

```
if (currentElements.page == 0) return;
```

We should also check that we are currently rendering the correct page. If not, we should load it first:

JAVASCRIPT

```
if (currentElements.page != currentPage) {
  currentPage = currentElements.page;
  document.getElementById("notation").innerHTML = tk.renderToSVG(currentPage);
}
```

To do the highlighting of the notes we are going to use a CSS rule to be defined in the `style.css` file. A simple way to do it is to add a class `playing` to be applied to `g.notes` and that changes the color:

```
g.note.playing {
  fill: crimson;
}
```

Now we can actually highlight the notes. To do so, we are going to loop over the list of notes listed in `currentElements` and simply add the `playing` class to them:

JAVASCRIPT

```
// Get all notes playing and set the class
for (note of currentElements.notes) {
  let noteElement = document.getElementById(note);
  if (noteElement) noteElement.classList.add("playing");
}
```

Finally, we need to bind the MIDIjs player with the callback function we have defined. This will be done with:

JAVASCRIPT

```
MIDIjs.player_callback = midiHightlightingHandler;
```

This should not work! However, if it does, you will notice that we also need to de-highlight the notes that are not played anymore, otherwise they will stay highlighted. This should be done at the beginning of the `midiHightlightingHandler` callback function by removing the `playing` class to the notes that currently have it:

JAVASCRIPT

```
// Remove the attribute 'playing' of all notes previously playing
let playingNotes = document.querySelectorAll('g.note.playing');
for (let playingNote of playingNotes) playingNote.classList.remove("playing");
```

The code above needs to be placed within the body of the `midiHighlightingHandler` function.

Full example

Open this example in a new window.

HTML /
JAVASCRIPT

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width">
        <title>MIDI playback</title>
        <!-- A stylesheet for modifying the appearance of the notes being played -->
        <link href="midi.css" rel="stylesheet" type="text/css" />
        <!-- Verovio -->
        <script src="https://www.verovio.org/javascript/develop/verovio-toolkit-wasm.js" defer></script>
        <!-- A JavaScript MIDI player -->
        <script src='https://www.midijs.net/lib/midi.js'></script>
    </head>
    <body>
        <button id="playMIDI">Play</button>
        <button id="stopMIDI">Stop</button>
        <div id="notation"></div>
        <script>
            /**
                We need to wait for the whole page to load before we try to
                work with Verovio.
            **/
            document.addEventListener("DOMContentLoaded", (event) => {
                verovio.module.onRuntimeInitialized = function () {
                    // This line initializes the Verovio toolkit
                    const tk = new verovio.toolkit();

                    tk.setOptions({
                        pageWidth: document.body.clientWidth,
                        pageHeight: document.body.clientHeight,
                        scaleToPageSize: true,
                    });

                    // The current page, which will change when playing through the piece
                    let currentPage = 1;

                    /**
                     The handler to start playing the file
                    **/
                    const playMIDIHandler = function () {
                        // Get the MIDI file from the Verovio toolkit
                        let base64midi = tk.renderToMIDI();
                        // Add the data URL prefixes describing the content
                        let midiString = 'data:audio/midi;base64,' + base64midi;
                        // Pass it to play to MIDIjs
                        MIDIjs.play(midiString);
                    }

                    /**
```

```
            The handler to stop playing the file
            **/
            const stopMIDIHandler = function () {
                MIDIjs.stop();
            }

            const midiHightlightingHandler = function (event) {
                // Remove the attribute 'playing' of all notes previously playing
                let playingNotes = document.querySelectorAll('g.note.playing');
                for (let playingNote of playingNotes) playingNote.classList.remove("playing");

                // Get elements at a time in milliseconds (time from the player is in seconds)
                let currentElements = tk.getElementsAtTime(event.time * 1000);

                if (currentElements.page == 0) return;

                if (currentElements.page != currentPage) {
                    currentPage = currentElements.page;
                    document.getElementById("notation").innerHTML = tk.renderToSVG(currentPage);
                }

                // Get all notes playing and set the class
                for (note of currentElements.notes) {
                    let noteElement = document.getElementById(note);
                    if (noteElement) noteElement.classList.add("playing");
                }
            }

            /**
                Wire up the buttons to actually work.
            */
            document.getElementById("playMIDI").addEventListener("click", playMIDIHandler);
            document.getElementById("stopMIDI").addEventListener("click", stopMIDIHandler);
            /**
             Set the function as message callback
            */
            MIDIjs.player_callback = midiHightlightingHandler;

            // This line fetches the MEI file we want to render...
            fetch("https://www.verovio.org/examples/downloads/Schubert_Lindenbaum.mei")
            // ... then receives the response and "unpacks" the MEI from it
            .then((response) => response.text())
            .then((meiXML) => {
                // ... then we can load the data into Verovio ...
                tk.loadData(meiXML);
                // ... and generate the SVG for the first page ...
                let svg = tk.renderToSVG(1);
                // ... and finally gets the <div> element with the ID we specified,
                // and sets the content (innerHTML) to the SVG that we just generated.
                document.getElementById("notation").innerHTML = svg;
            });
        }
    });
  </script>
 </body>
</html>
```

Score content selection

Verovio can extract segments of a score and display only these segments. This can be useful if you have a larger score and want to display a segment in a webpage to highlight a particular segment or portion. This can also be combined with the techniques from the previous tutorials, so you can also highlight or even play back these segments using MIDI.

Selecting parts of a score

Verovio has a `select` method available on the toolkit. This method takes a JSON object where you can specify a range of measures in the format “[first]-[last]”. The selection syntax is based on a subset of the `measureRange` syntax from the Enhancing Music Notation Addressability API. The difference is that Verovio only supports a single measure range. For example:

JAVASCRIPT

```
tk.select({measureRange: "1-10"});
```

Once the measures have been selected, calls to render the score to SVG will render only that selected portion. Importantly, it will also reduce the number of “pages” that are available to only the number that are needed to represent the selection.

You can clear a selection by passing in an empty JSON Object:

JAVASCRIPT

```
tk.select({});
```

You can also select the entire score by using start and end :

JAVASCRIPT

```
tk.select({measureRange: "start-end"});
```

Full example

Open this example in a new window.

HTML /
JAVASCRIPT

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">
    <title>Select an excerpt of a score interactively</title>
    <script src="https://www.verovio.org/javascript/develop/verovio-toolkit-wasm.js" defer></script>
  </head>
  <body>
    <label>Select: </label>
    <input id="start" type="text" placeholder="from..." />
    <input id="end" type="text" placeholder="...to" />
    <button id="applySelection">Apply</button>
    <button id="prevPage">Previous page</button>
    <button id="nextPage">Next page</button>
    <div id="notation"></div>
    <script>
      /**
       Load Verovio
      **/
      document.addEventListener("DOMContentLoaded", (event) => {
        verovio.module.onRuntimeInitialized = function () {
          // This line initializes the Verovio toolkit
          const tk = new verovio.toolkit();

          tk.setOptions({
            pageWidth: document.body.clientWidth,
            pageHeight: document.body.clientHeight,
            scale: 75,
            scaleToPageSize: true,
          });

          // Keep a variable to the notation div id
          let notationElement = document.getElementById("notation");

          // The current page, which will change when playing through the piece
          let currentPage = 1;

          /**
           The handler to apply the selection
          **/
          const applySelectionHandler = function () {
            let start = document.getElementById("start").value;
            if (start === '') start = "start";
            let end = document.getElementById("end").value;
            if (end === '') end = "end";
            let range = `${start}-${end}`;
            tk.select({measureRange: range});
            tk.redoLayout();
            notationElement.innerHTML = tk.renderToSVG(currentPage);
          }

          /**
            Wire up the button to actually work.
          */
          const nextPageHandler = function () {
            currentPage = Math.min(currentPage + 1, tk.getPageCount());
```

```
                notationElement.innerHTML = tk.renderToSVG(currentPage);
            }

            const prevPageHandler = function () {
                currentPage = Math.max(currentPage - 1, 1);
                notationElement.innerHTML = tk.renderToSVG(currentPage);
            }

            /**
                Wire up the buttons to actually work.
            */
            document.getElementById("nextPage").addEventListener("click", nextPageHandler);
            document.getElementById("prevPage").addEventListener("click", prevPageHandler);
            document.getElementById("applySelection").addEventListener("click", applySelectionHandler
);

            // This line fetches the MEI file we want to render...
            fetch("https://www.verovio.org/examples/downloads/Schubert_Lindenbaum.mei")
                // ... then receives the response and "unpacks" the MEI from it
                .then((response) => response.text())
                .then((meiXML) => {
                    // ... then we can load the data into Verovio ...
                    tk.loadData(meiXML);
                    // ... and generate the SVG for the first page ...
                    let svg = tk.renderToSVG(1);
                    // ... and finally gets the <div> element with the ID we specified,
                    // and sets the content (innerHTML) to the SVG that we just generated.
                    notationElement.innerHTML = svg;
                });
        }
    });
  </script>
 </body>
</html>
```

Interacting with editorial markup

MEI has a feature that lets us encode variant “readings” of a musical text. These readings may come from different sources of the same piece. A common type of alternate reading is a “contrafactum”, or alternate text. Typically this might occur between a Latin sacred text and a secular text in a vernacular, such as English, both set to the same music.

Verovio supports the selection of variant readings encoded with MEI <app> , containing <lem> and <rdg> elements. Only one variant can be displayed at a time, and this is selected when the file is loaded. By default, Verovio selects the <lem> (or the first <rdg> if no <lem> is provided).

In this example we are going to create a basic interface to switch between variants by applying XPath queries for selecting specific readings, and then highlight the editorial markup in different colours. The MEI file we will use for this purpose comes from the Marenzio edition:

```
https://raw.githubusercontent.com/marenzio/marenzio.github.io/master/mei/M-04-6/M_04_6_02_Di_nettare_amoroso_ebro_la_mente.mei
```

Selection with an xpath query

The first thing we need is a variable to store the XPath queries. This must be an array, but we can start with an empty one, which applies the default behaviour:

JAVASCRIPT

```
let appXPath = [];
```

Since Verovio selects the elements to be displayed when loading the file, we need to define a loadFile() function that applies the options, loads the file into Verovio, and renders it:

JAVASCRIPT

```
// A function that loads a file
function loadFile() {
  fetch("https://raw.githubusercontent.com/marenzio/marenzio.github.io/master/mei/M-04-6/M_04_6_02_Di_nettare_amoroso_ebro_la_mente.mei")
    .then((response) => response.text())
    .then((meiXML) => {
      tk.setOptions({
        pageWidth: document.body.clientWidth,
        pageHeight: document.body.clientHeight,
        scale: 50,
        scaleToPageSize: true,
        appXPathQuery: appXPath
      });
      tk.loadData(meiXML);
      notationElement.innerHTML = tk.renderToSVG(currentPage);
    });
}
```

To load, or reload, the file we can now call the function:

JAVASCRIPT

```
loadFile();
```

In the CSS file we also have defined two rules:

```
g.lem {
  fill: darkcyan;
}
g.rdg {
  fill: crimson;
}
```

At this stage, because we have a editorial-markup.css with some rule highlighting for lem and rdg classes, the default <lem> should appear highlighted.

Switching between readings

To switch between the original and a reading (an English contrafactum text, in this case), we add two buttons, with two handlers that bind to the button event listeners:

HTML / JAVASCRIPT

```
<button id="original">Original</button>
<button id="contrafactum">Contrafactum</button>
```

JAVASCRIPT

```
const originalHandler = function () {
  // Do something to render the original
}

const contrafactumHandler = function () {
 // Do something to render the contrafactum
}

document.getElementById("original").addEventListener("click", originalHandler);
document.getElementById("contrafactum").addEventListener("click", contrafactumHandler);
```

The file example we are using has some readings encoded as follows:

XML

```
<app>
  <lem source="Italian">
    <verse>
      <syl>Di</syl>
    </verse>
  </lem>
  <rdg source="English">
    <verse>
      <syl>When</syl>
    </verse>
  </rdg>
</app>
```

To select the “English” reading (i.e., in the contrafactumHandler), we can write an xPath query selecting <rdg> elements with the corresponding @source value, in this case “English”, and then reload the file:

JAVASCRIPT

```
appXPath = ["./rdg[@source='English']"];
loadFile();
```

For the original, we can reset Verovio by passing in an empty appXPath array:

JAVASCRIPT

```
appXPath = [];
loadFile();
```

When switching between the views you will notice that the colour of the text differs between the original <lem> and the English <rdg> .

Full example

Open this example in a new window.

HTML /
JAVASCRIPT

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="utf-8">
      <meta name="viewport" content="width=device-width">
      <title>Interacting with editorial markup</title>
      <!-- A stylesheet for modifying the appearance of the editorial markup -->
      <link href="editorial-markup.css" rel="stylesheet" type="text/css" />
      <!-- Verovio -->
      <script src="https://www.verovio.org/javascript/develop/verovio-toolkit-wasm.js" defer></script>
   </head>
   <body>
      <button id="original">Original</button>
      <button id="contrafactum">Contrafactum</button>
      <button id="prevPage">Previous page</button>
      <button id="nextPage">Next page</button>
      <div id="notation"></div>
      <script>
         /**
          Load Verovio
         **/
         document.addEventListener("DOMContentLoaded", (event) => {
         verovio.module.onRuntimeInitialized = function () {
            // This line initializes the Verovio toolkit
            const tk = new verovio.toolkit();

            // An array to keep xpath queries for selecting editorial markup
            let appXPath = [];

            // Keep a variable to the notation div id
            let notationElement = document.getElementById("notation");

            // A function that loads the file
            function loadFile() {

               fetch("https://raw.githubusercontent.com/marenzio/marenzio.github.io/master/mei/M-04-6/M_
04_6_02_Di_nettare_amoroso_ebro_la_mente.mei")
                  // ... then receives the response and "unpacks" the MEI from it
                  .then((response) => response.text())
                  .then((meiXML) => {
                  // First we set the options, including the appXPathQuery one
                  tk.setOptions({
                     pageWidth: document.body.clientWidth,
                     pageHeight: document.body.clientHeight,
                     scale: 50,
                     scaleToPageSize: true,
                     appXPathQuery: appXPath
                  });
                  // ... then we can load the data into Verovio ...
                  tk.loadData(meiXML);
                  // ... and generate and set the SVG for the current page ...
                  notationElement.innerHTML = tk.renderToSVG(currentPage);
                  });
               }

               // The current page, which will change when playing through the piece
               let currentPage = 1;

               /**
```

```
            Wire up the button to actually work.
          */
          const nextPageHandler = function () {
            currentPage = Math.min(currentPage + 1, tk.getPageCount());
            notationElement.innerHTML = tk.renderToSVG(currentPage);
          }

          const prevPageHandler = function () {
            currentPage = Math.max(currentPage - 1, 1);
            notationElement.innerHTML = tk.renderToSVG(currentPage);
          }

          const originalHandler = function () {
            appXPath = [];
            loadFile();
          }

          const contrafactumHandler = function () {
            appXPath = ["./rdg[@source='English']"];
            loadFile();
          }

          /**
            Wire up the buttons to actually work.
          */
          document.getElementById("nextPage").addEventListener("click", nextPageHandler);
          document.getElementById("prevPage").addEventListener("click", prevPageHandler);
          document.getElementById("original").addEventListener("click", originalHandler);
          document.getElementById("contrafactum").addEventListener("click", contrafactumHandler);

          // This line fetches the MEI file we want to render...
          loadFile();
        }
      });
    </script>
  </body>
</html>
```

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.

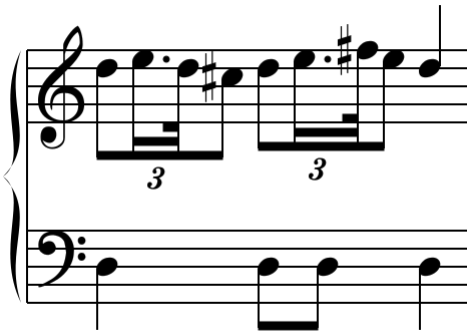
SVG structure

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>
```

Controlling the SVG output

Units and page dimensions

Verovio abstract unit

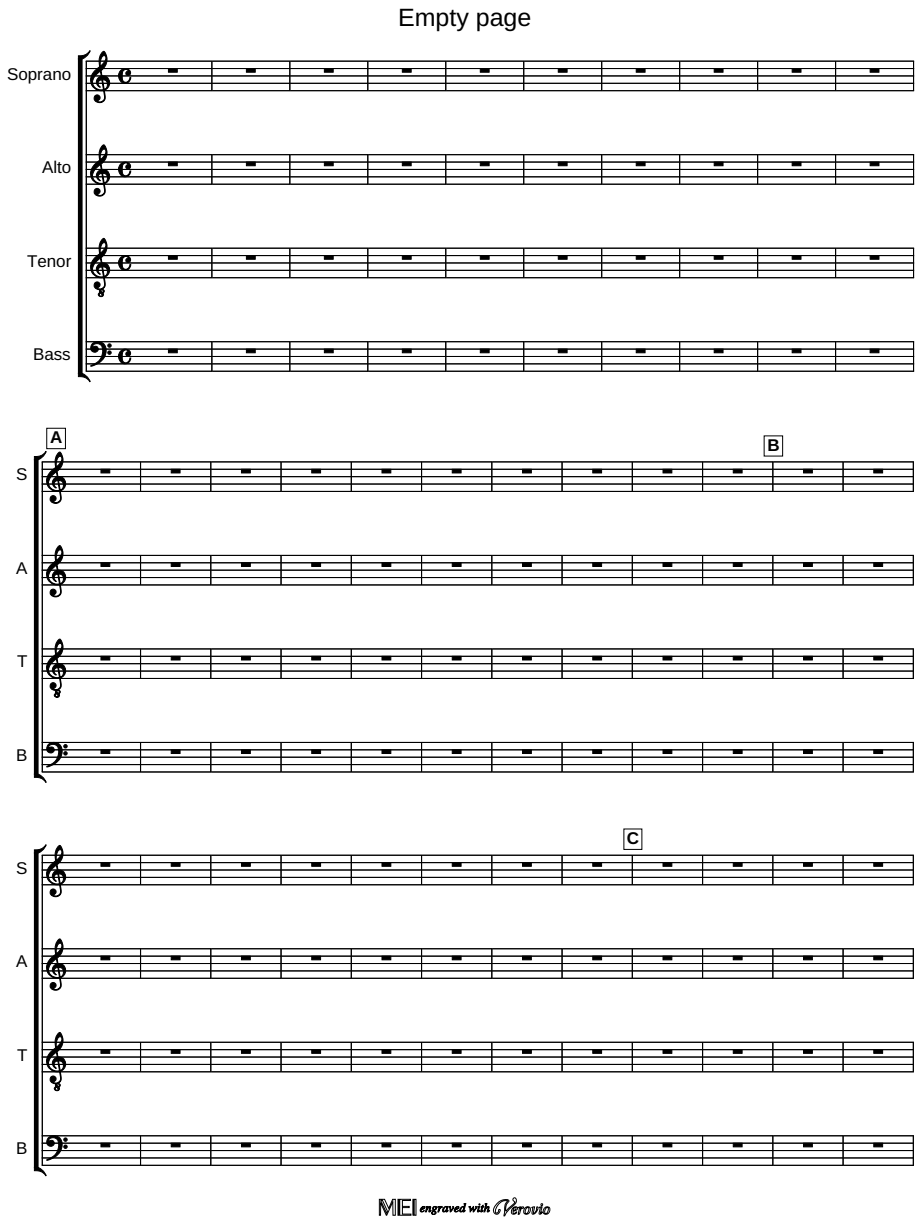
Verovio layout calculation is based on an internal abstract unit. This abstract unit is also used for specifying a few options, such as the page dimensions. By default, the page height is 2970 and the page width is 2100. These are equivalent to the dimension of an A4 page in portrait orientation in tenths of a millimeter. When generating SVG, these units are interpreted as pixels, which means that the default SVG image size is **2970px** height by **2100px** width.

The example below shows an empty page with the default dimensions – and the option --justifyVertically enabled.

[SVG file is missing and need to be generated]

Page margins (--page-margin-bottom, --page-margin-left, --page-margin-right and --page-margin-top) are also specified in abstract units, with a default value of 50. That is **50px** with the SVG image output.

Changing the page dimension will increase the amount of music that fits on the page. The example below if the same file rendered with a page height of 3050 and a page width of 2290, a more typical paper size for sheet music than A4.



MEI unit

Most of the options in Verovio are given in MEI units. An MEI unit (or MEI virtual unit) corresponds to half the distance between adjacent staff lines where the interline space is measured from the middle of a staff line. The value of the MEI unit in Verovio is given in abstract units and determines the size of the staff on a page. By default, the MEI unit is 9.0 , which means that a staff space is 9 abstract units, or **9px** in the SVG image output with the default options.

In traditional music engraving, the staff size corresponds to the raster which would be chosen depending of the type and size of score to be engraved. However, in digital environments the size of the notation can be changed on the screen depending on the size and orientation of the screen (i.e., “responsive” environments), and the size of the raster can remain fixed. Adjusting the size of the notation in Verovio is usually changed by adjusting the page size and scaling factors, which are described in the next section.

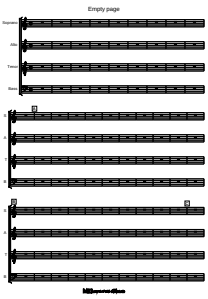
Scaling

Using the SVG ViewBox

For simple cases where the output SVG image is embedded in a web environment, enabling the --svg-view-box is the simplest way to have the image scaled down to the fit its container. It includes responsive environments when the container size can change interactively. The example below is the default output page with the option --svg-view-box enabled and embedded in a with a with of 210px . As a result, the SVG image is scaled down to fit in it.

HTML / JAVASCRIPT

```
<div style="width: 210px;">
  <!-- SVG image included here -->
</div>
```



Using the Verovio option

The SVG output in Verovio can be scaled by using the --scale option. The option value is an integer representing a scaling percentage. It is 100 percent by default.

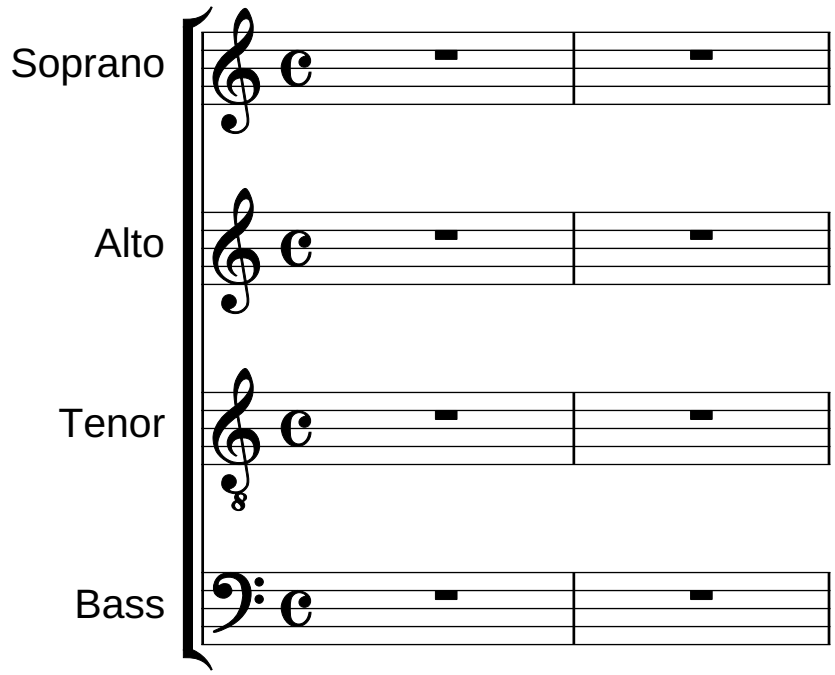
When changing the scale option, Verovio will by default change the size of the output SVG image. For example, with the default page size and a scale option set to 50 percent, the resulting SVG image will have a size of **1485px** by **1050px**. The same amount of music will be engraved on the page as with the default scale value.

In responsive environments, Verovio can be used to create user interfaces where the user can change the magnification ("zoom"). This can be achieved by changing the scale and the page dimensions. Zooming out means increasing the page dimensions and reducing the scale by the same factor, and zooming in the opposite. For example, if the window in which the output of Verovio will be displayed is **1800px** by **800px**, these can be set as a page height and page width and Verovio will produce an SVG image that fits the window with the default scale value of 100 percent. To implement a zooming out function, for example by a factor 2, the page dimensions have to be changed to 3600 by 1600 and the scale to 50 . The output SVG image will then still have a size that fits the window.

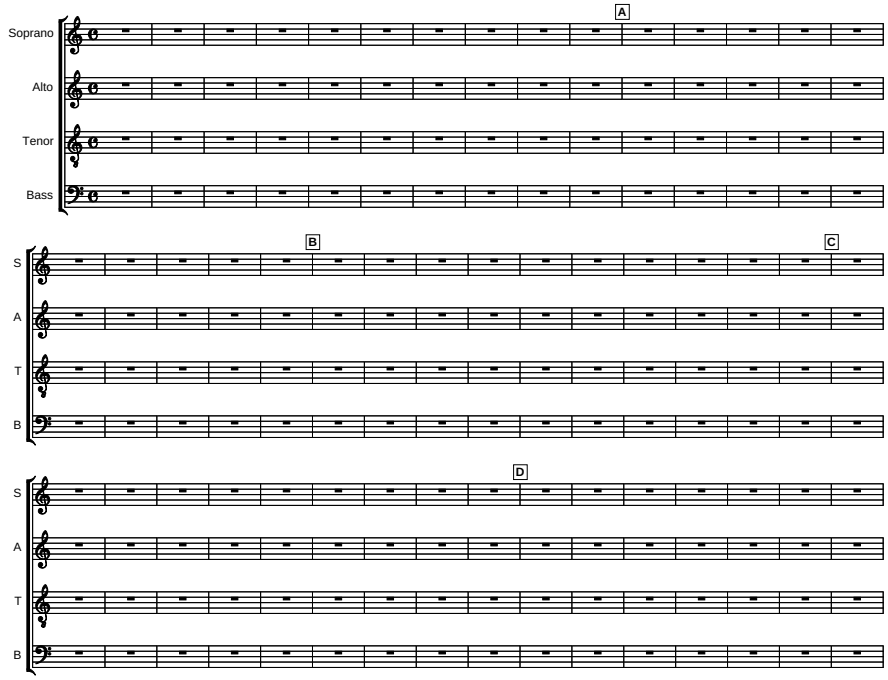
Scaling to the page size

Verovio has a --scale-to-page-size option that simplifies the scaling process described above. Using this option is recommended in responsive environments. The advantage is that does not require the page dimensions to be calculated and changed by the user. With this option, the SVG output image will always have the same size independently from the scale percentage. The scale percentage determines how the rendering is scaled within this image. For example, reducing the scale percentage will increase the amount of music on the page. When this option is enabled, the layout needs to be recalculated when the scale value is changed – see below.

The example below shows a file rendered with a page height of 800 , a width of 900 and the default scale of 100 percent.



The example below is the same file rendered with the same page dimensions as above, but with the option --scale-to-page-size enabled and a scale of 30 percent. The SVG image size remains the same but the amount of music rendered has increased accordingly.



Changing the values of some options often requires the layout to be recalculated. For example, when the ratio of the page dimension are changed, or the margins are changed, then a call to RedoLayout must be made before rendering a page again. It is also important to keep in mind that redoing the layout might yield a different number of pages and that it is important to check the a page still exists with GetPageCount before rendering it.

When the --scale-to-page-size option is **not** enabled on (default), then changing only the scale option does not require the layout to be recalculated before rendering a page again because the amount of music per page and the number of pages will **not** change. However, when the option --scale-to-page-size is enabled, then the layout recalculation and the page existence check need to happen before rendering a page.

SVG optimised for PDF generation

The SVG image output in pixel units is well suited to digital environments and rendering on screens. However, in some cases, the SVG will be subsequently converted to a PDF for printing. In such uses, it is recommended to enable the option --mm-output to change the page dimensions to millimeters. In this case, the SVG image produced with the default page height and page width will have a size of **297mm** by **210mm**. The page margins, with their default value of 50 , will have a size of **5mm**.

If you want to increase or decrease the amount of music on a page, there are two solutions. The first one is to enable the --scale-to-page-size option described above and to adjust the scale value. Because the image produced with the option remains fixed, the page will have a larger or a smaller amount of music on the page depending if the scale option was decreased or increased respectively. The other solution for changing the amount of music on a page is to adjust the MEI unit, which is described below.

Adjusting the MEI unit

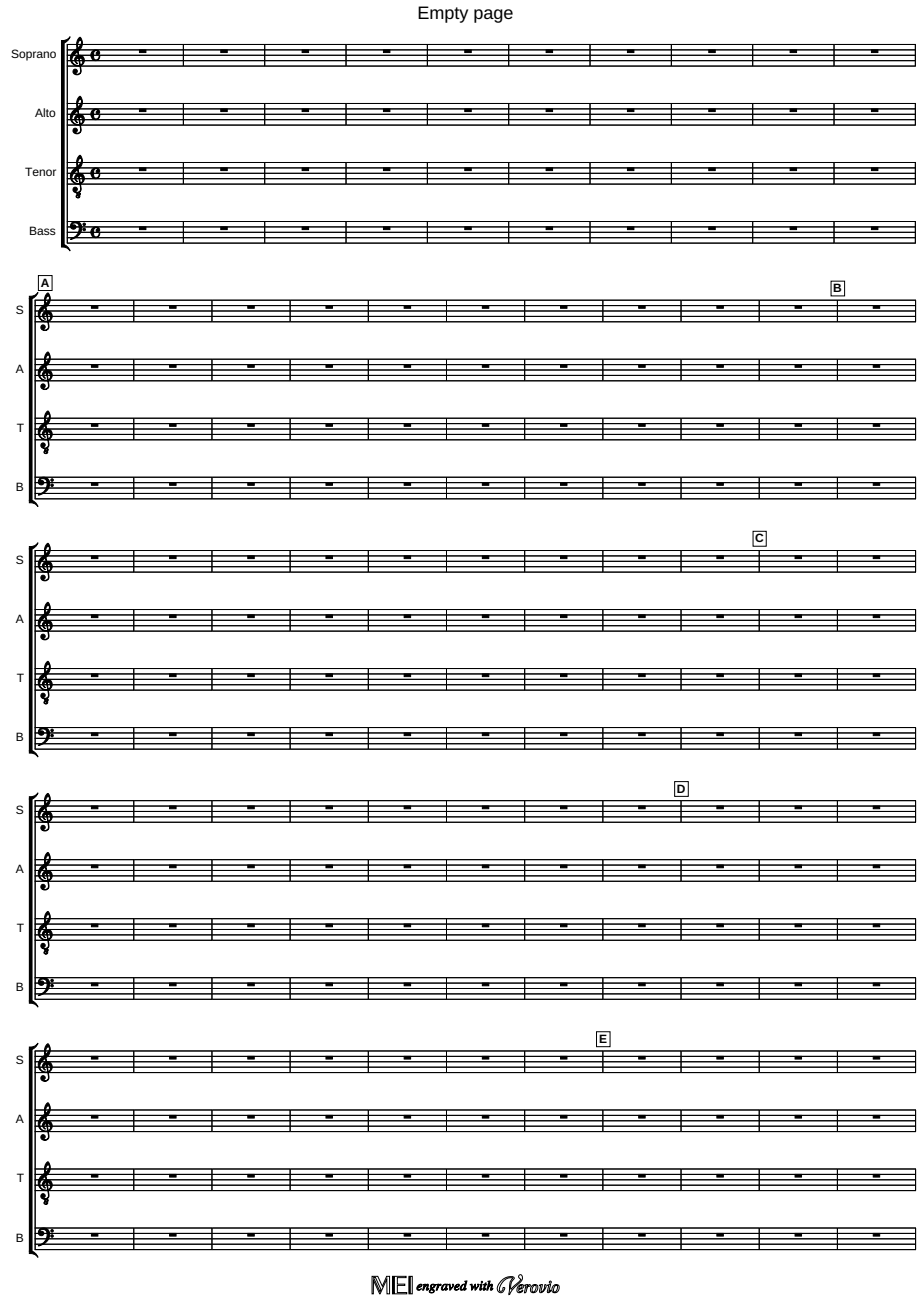
If you want to replicate a print layout with a specific traditional page and staff size, you need to control the size of the staff (or raster). With Verovio, the raster can be adjusted with the --unit option, which adjusts the definition of an MEI unit. One MEI unit (or MEI virtual unit) corresponds to half the distance between adjacent staff lines. In terms of staff size (or raster), it means a staff size of **7.2mm** with the --mm-output option enabled. Bear in mind that this size do not factor in the width of the staff line. Because the MEI unit size is measured from the middle of a staff line, the actual staff height will be two half staff line widths more.

The table below gives an indication of values for the MEI unit in Verovio corresponding to raster sizes (without staff line width factored in) as found in the literature or some music notation software applications.

MEI unit	Raster	Staff size in mm	Example use
11.5	0	9.2	Educational music
9.875	1	7.9	
9.25	2	7.4	Piano music
8.75	3	7.0	Single-staff parts
8.125	4	6.5	
7.5	5	6.0	
6.875	6	5.5	Choral music
6.0	7	4.8	

MEI unit	Raster	Staff size in mm	Example use
4.625	8	3.7	Full score

The example below shows the same file as above with the default A4 page size but with a unit value of 6.0 . More music is rendered on a page because the staff size is smaller.



Layout options

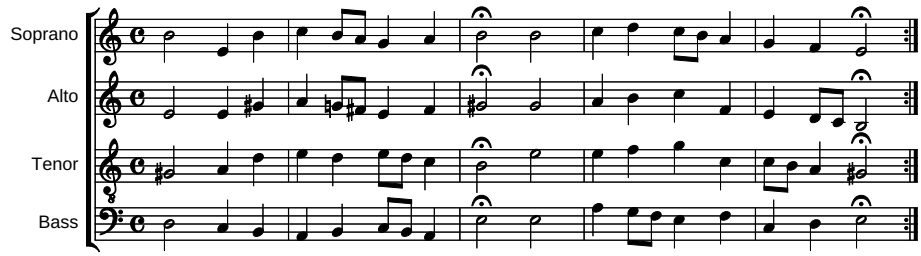
Output layout

By default, Verovio generates an output where the content is organized in pages, which size can be changed with the --page-height and --page-width options. The content of the music will laid out on one or more pages. It is possible to adjust the option --breaks to control how the layout is organized, namely where system and page breaks occur.

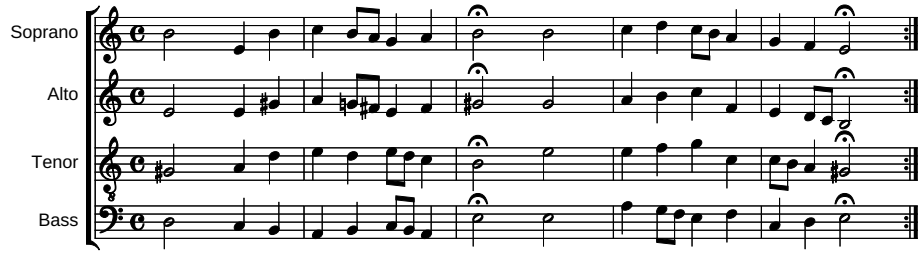
By setting the option --breaks to none , no system and page breaks will occur, and Verovio will output a single system with the entire music content. With this option, the page width will be adjusted (e.g., increased) automatically to ensure that it can contain the entire content. Be aware that this can produce very large files, regarding both the dimension of the SVG image and the actual file size.

Content spacing

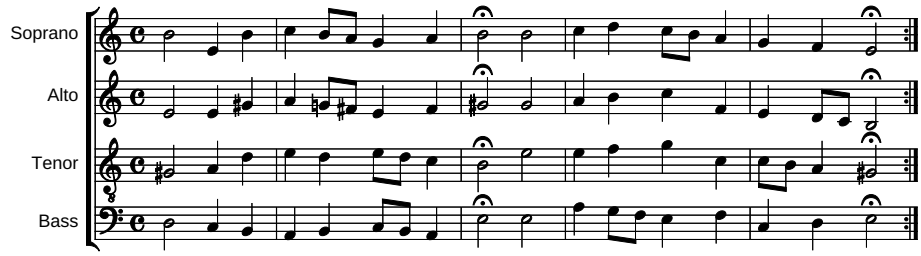
The spacing of the rhythmic values (notes and rests) is adjusted based on their durations, each value taking a bit more space than the next shorter value. This is the default output:



The spacing can be controlled with --spacing-linear and --spacing-non-linear options. In general, if one of them is increased, the other should be decrease not to have an exaggeratedly expanded - or the other way around. The default values are 0.25 for --spacing-linear and 0.6 for --spacing-non-linear. If you want all measures to have the same width, which means making no spacing difference according to the duration of the notes and rests, the --spacing-non-linear value needs to be set to 1.0. This is what Verovio will produce together with --spacing-linear set to 0.03:



Alternatively, if --spacing-non-linear is reduced to 0.35 and --spacing-linear increased to 1.0, there will be less difference in spacing between notes of different durations:

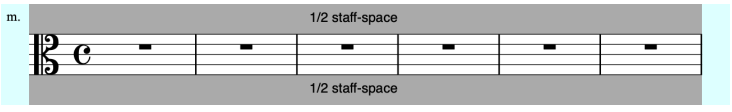


Staff and system spacing

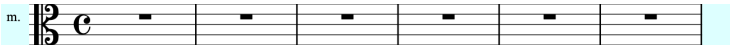
Staff and system spacing in Verovio is controlled by two options, namely --spacing-staff and --spacing-system. Their value is given in MEI units and the default value is 12 units. Since a five line staff is 8 units, it means the default spacing between two staves will be one and a half staff height, and an additional equivalent system spacing between two systems.

Verovio adds half a staff space above the first staff of a system and below the last one. This is the illustrated below with the default spacing options.

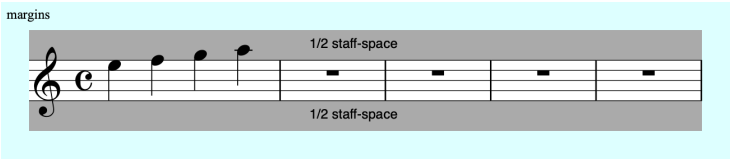
When removing the top and bottom page margins (--page-margin-top 0 and --page-margin-bottom 0), there will be only the half staff space above and below.



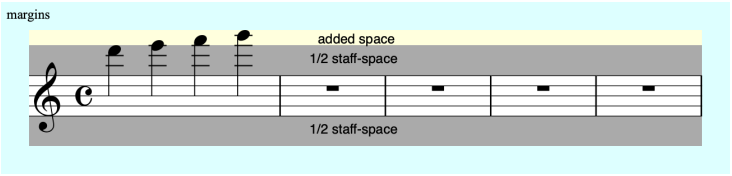
To remove the spacing above and below the staff completely, the staff space has to be removed with --spacing-staff 0.



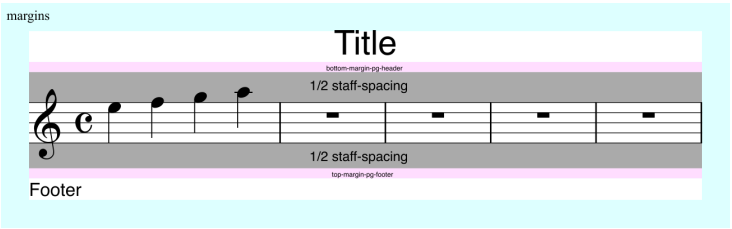
The half staff space above and below the staff means that having music content above or below the staff line does not change vertical positioning when the content fits within that space. Because the default value for --spacing-staff is 12 MEI units, it means that up to 3 staff spaces (e.g., up to a D6 with a G-2 clef) will fit in that space.



When the content takes more space than half a staff spacing, then space is added. On a page, it means that the position of the staff will be lowered accordingly.



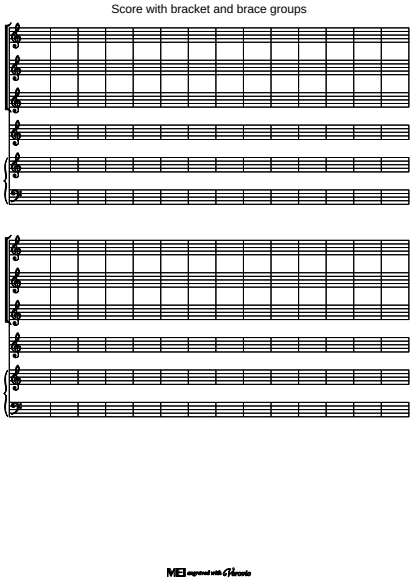
When there is a header or a footer, additional spacing is added between between them and the music content. By default, the spacing is 2.0 MEI units. The value can be adjusted with the options --bottom-margin-pg-header or --top-margin-pg-footer.



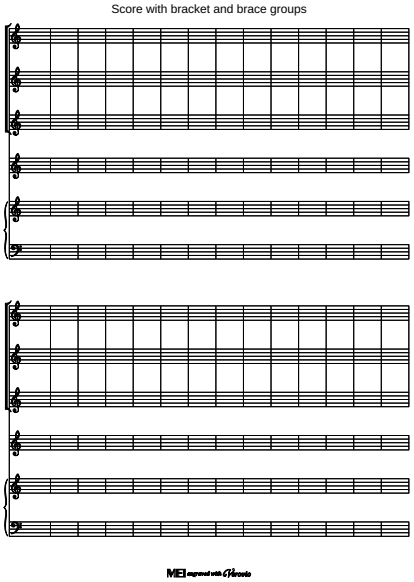
Vertical justification

When producing page-like layouts, it is often desirable to justify the content vertically in order to have the staves distributed on the page in a balance way.

By default, no vertical justification is applied, and rendering a page will place all systems at the top of the page. The spacing of the staff and the system will be simply determined by the --spacing-staff and --spacing-system values.

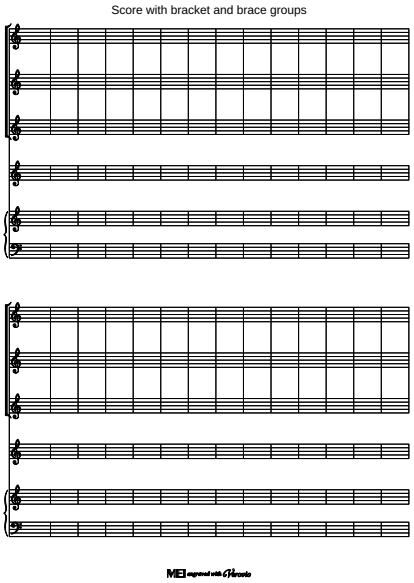


To have the content justified, the option --justify-vertically has to be enabled.

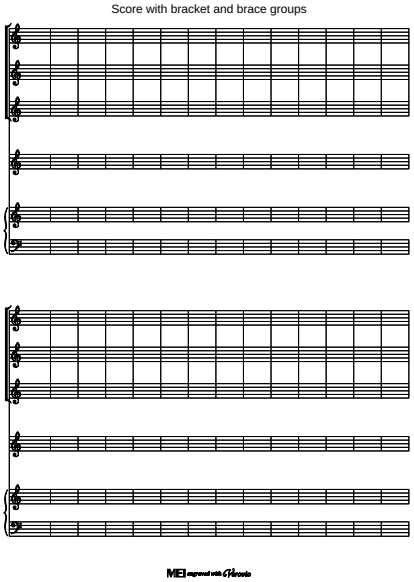


This option applies justification over all staff spaces in the score in a linear way. Verovio allows for a more fine-tuned justification with different spacing parameters for staves grouped by a brace or a bracket. The options to adjust are `--justification-brace-group`, `--justification-bracket-group`, `--justification-staff` and `--justification-system`. Each of these take a parameter from `0.0` to `10.0` acting as a factor on the justification applied. The default value is `1.0` for all of them.

Setting one of these options to `0.0` will result in no justification space added between the corresponding staves. For example, the same example as above rendered with `--justification-brace-group 0.0` will have a layout where the staff spacing between the staves in the brace groups remains unchanged.

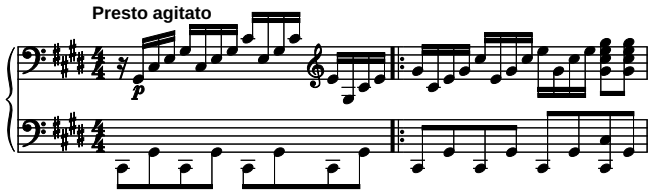


Adding `--justification-bracket-group 0.2` with reduce the space between the staves in the bracket group, even though still changing it be cause the value is not `0.0`. The spacing between the other staves and the systems is consequently increased.



Generating single-system incipits

Generating single-system incipits can be done with the options `--page-height 100` and `--adjust-page-height` enabled. This way, the first page will contain only the first system of music. Its width can be set as desired with `--page-width`, here with `1500`:



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.

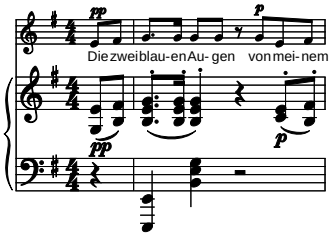
Verovio follows the SMuFL specification. It means that it is possible to easily change the music font used in Verovio for personalised output. Verovio includes the Leipzig font, its own SMuFL-compliant music font. Leipzig was initially developed by Etienne Darbellay and Jean-François Marti as part of 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.

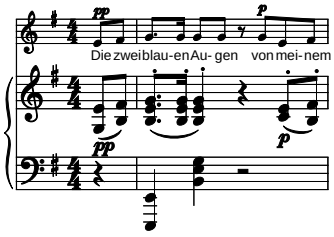
Fonts included can be selected by setting the `--font` option. For example, the Bravura font can be selected with the `--font Bravura` option on the command-line tool or by adding `{ font: "Bravura" }` in the JavaScript toolkit options.

Examples

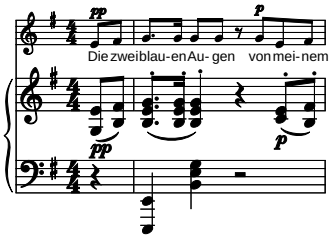
Leipzig



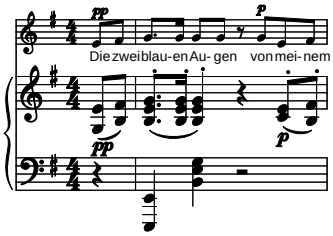
Bravura



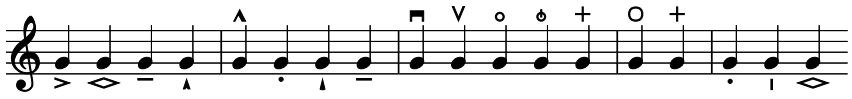
Gootville



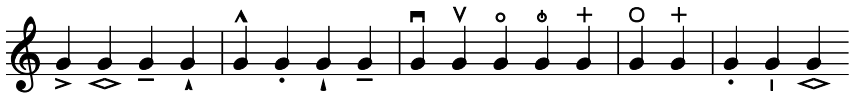
Leland



Leipzig



Bravura



Music symbols in text

For cases when music symbols are displayed within text, Verovio uses a text WOFF2 version of the selected music font. The font is included in the SVG as CSS.

The --smufl-text-font allows to change how the font is included. By default, it is simply embedded as a base64 string. This means that the SVG is fully self-contained and does not require network access for the font glyphs to be displayed. The include of the font can also be ignored with none , which can be useful when the font is included seperately in the environment.

With linked , the text font will be included in the SVG but with the following CSS import:

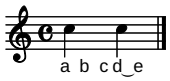
XML

```
<style type="text/css">
   @import url("https://www.verovio.org/javascript/3.13.0/data/Leipzig.css");
</style>
```

The version of the font path is based on the Verovio version release number, or is develop for the develop version of the toolkit.

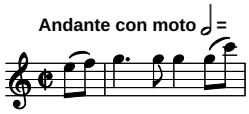
When a music glyph is displayed within text and the music font selected is not Leipzig or Bravura, Verovio will also check if the music glyph exists in the selected music font. If not, it will fallback to the Leipzig font. If other text elements include music glyphs that do exist in the selected font, then both Leipzig and the selected font will be included. In other words, the fallback to Leipzig will be enabled only for the text elements displaying a missing music glyphs but not for the others.

Examples



XML

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



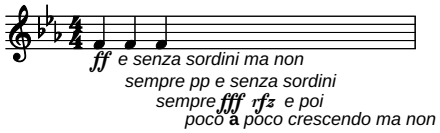
XML

```
<tempo staff="1" tstamp="1" midi.bpm="70">Andante con moto <rend glyph.auth="smufl">&#xE1D3;</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, Verovio automatically detects dynamic symbols within text and displays them appropriately. 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 this example:

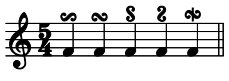


XML

```
<dynam staff="1" tstamp="1">ff e senza sordini ma non sfz</dynam>
<dynam staff="1" tstamp="2">
  <rend fontfam="Times">sempre pp e senza sordini</rend>
</dynam>
<dynam staff="1" tstamp="3">sempre fff rfz e poi<lb/>poco <rend fontstyle="normal" fontweight="bold">a
</rend> poco crescendo ma non ffff<lb/>troppo</dynam>
```

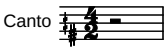
Use alternate SMuFL glyphs

For some elements, Verovio support the use of alternate SMuFL glyphs through the @glyph.auth and @glyph.name or @glyph.num attributes. The @glyph.auth is expected to the value smufl . When both @glyph.auth and @glyph.num are provided, then the priority is given to @glyph.num .



XML

```
<turn staff="1" startid="#ex_0026251991" form="lower"/>
<turn staff="1" startid="#ex_1783472943" form="upper"/>
<turn type="vertical" staff="1" startid="#ex_0290776165" glyph.auth="smufl" glyph.num="U+E56B" form="
lower"/>
<turn type="vertical" staff="1" startid="#ex_1264550794" glyph.auth="smufl" glyph.num="U+E56A" form="
upper"/>
<turn type="slashed" staff="1" startid="#ex_0939405248" glyph.auth="smufl" glyph.num="U+E569"/>
```



XML

```
<keySig>
  <keyAccid pname="b" glyph.num="U+E444"/>
  <keyAccid pname="f" glyph.name="accidentalBuyukMucennebSharp"/>
</keySig>
```

Custom fonts

Load all fonts

The `--font-load-all` boolean option makes Verovio loads all the music fonts available in the resource directory. That way, a specific font other than the default font `Leipzig` or the font set with the `--font` option can be used by specifying a `@fontname` value. At this stage, this is supporting only on `clef` and `meterSig`.

Set a specific fall back

Only Bravura and Leipzig have a complete coverage of the glyphs used in Verovio. The `--font-fallback` parameter option allows to choose between `Leipzig` (default) or `Bravura` as the fallback font to be used when the font chosen is missing a glyph.

Loading custom fonts

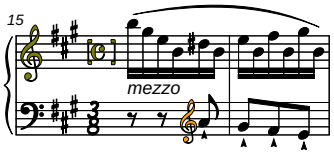
The `--font-add-custom` parameter option allows to load and use an external font not available in the resource directory. The option is repeatable, which means that more than one external font can be loaded. For bindings that use JSON options, the value(s) must be passed in an array.

The custom font must be archived in a ZIP file containing the files produced by the font script Verovio provides for extracting relevant information from an SVG font file and the corresponding SMuFL metadata. These files are:

- The XML file with bounding boxes of the included glyphs.
- The XML snippets for each glyph
- The CSS file of the font for text

The ZIP filename must correspond to the name of the font. For the JavaScript binding, the ZIP file must be encoded in Base64 and passed as a URL or as a based64 string.

Example rendered with `--font-fallback Bravura` and `--font-add-custom GoldenAge.zip` (available here) and all fonts loaded with `--font-load-all`. The elements in olive have a `@fontame="Petaluma"`. The clef in orange is a `Bravura` fallback.



XML

```
<clef shape="G" color="olive" line="2" fontname="Petaluma"/>
```

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.

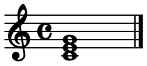
When using a case-insensitive `@pname` for the tonic of the new key, use `([A-Ga-g])` followed by an optional `accid` for the new key tonic. This is also case-insensitive: `([Ss]*|[Ff]*)`.

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

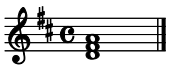
Here is a test example music to transpose - note the `@key.sig` is expected for transposition to work properly:



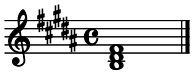
XML

```
<score>
  <scoreDef>
    <staffGrp>
      <staffDef n="1" lines="5" clef.shape="G" clef.line="2" keysig="0" meter.sym="common"/>
    </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:



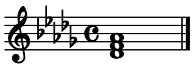
Setting transpose: "-m2" To go down a minor second from C to B:



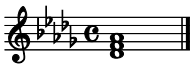
Common intervals: m3 = minor third, M3 = major 3rd, P4 = perfect fourth, P5 = perfect fifth, d5 = diminished fifth, A4 = augmented fourth.

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.

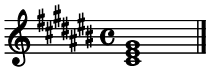
For example transpose: "1" will display in D-flat major:



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

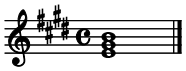


If you need to transpose to C-sharp major, then you cannot use integers, but must use the full musical interval, which in this case is transpose: "A1" for an augmented unison:

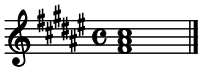


(a1 and A1 are the same, but m2 and M2 are not equivalent).

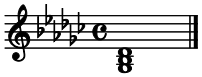
It is also possible to give the tonic note of the new key. For example transpose: "E" means to transpose to E major (or minor, since the mode will not be changed). This feature requires that the music contain key information which is not always present in MusicXML data. It can also be incorrect, which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



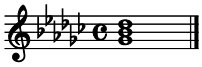
F-sharp major with transpose: "F#" , which is equivalent to a transposition of A4 :



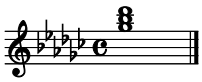
G-flat major with transpose: "Gb" , which is equivalent to d5 :



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:

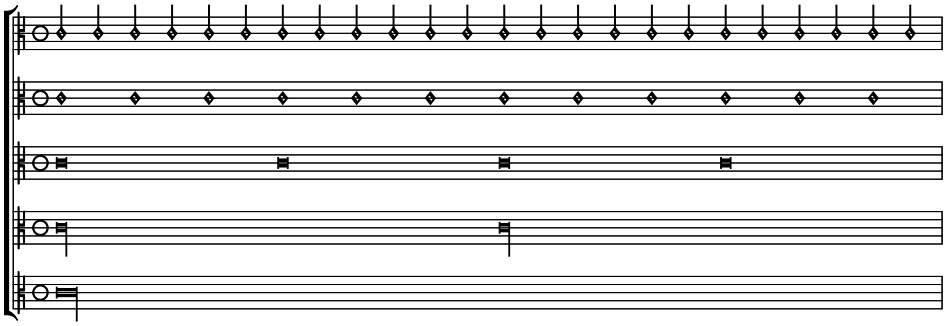
- Find the key information at the start of the music in each part. If all parts have the same transposition (or no parts have transposition), then use the @pname and @accid as the reference pitch for which an interval will be calculated for the input transposition target tonic.
- If all parts do not 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.

Mensural notation

Duration alignment

One of the unique features of Verovio is to treat mensural notation to be ternary by default. This means that when the notation type on staffDef@notation.type is set to mensural.* , duration will be aligned on a ternary basis by default. Binary alignment can be triggered with @modusmaior="2" , @modusminor="2" , @tempus="2" , or @prolatio="2" , as appropriate.



XML

```
<mensur modusmaior="2" modusminor="2" prolatio="2" tempus="3" sign="O"/>
```

Layout

[in preparation]

Ligatures

[in preparation]

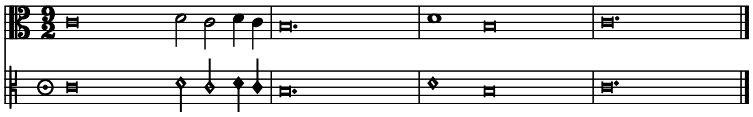
Alignment with CMN

Mensural notation in *tempus imperfectum* and *prolatio minor* aligns automatically (i.e., aligning whole notes with semibrevis and so on for durations up and down). Of course, this requires the mensural notation to be put in measures, and Verovio only takes care about the alignment of the duration themselves.

XML

```
<measure>
  <staff n="1">
    <layer n="1">
      <note dur="breve" oct="4" pname="c"/>
      <note dur="1" oct="4" pname="d"/>
      <note dur="2" oct="4" pname="d"/>
      <note dur="2" oct="4" pname="c"/>
    </layer>
  </staff>
  <staff n="2">
    <layer n="1">
      <note dur="brevis" oct="4" pname="c"/>
      <note dur="semibrevis" oct="4" pname="d"/>
      <note dur="minima" oct="4" pname="d"/>
      <note dur="minima" oct="4" pname="c"/>
    </layer>
  </staff>
</measure>
```

With *tempus perfectum* and or *prolatio maior*, then duration shorter than a breve in CMN need to be put into tuplets with the appropriate adjustments. See here num="3" and numbase="2" for the whole notes and num="9" and numbase="4" for the half notes and shorter.



XML

```
<measure>
  <staff n="1">
    <layer n="1">
      <note dur="breve" oct="4" pname="c"/>
      <tuplet num="9" numbase="4" num.visible="false" bracket.visible="false">
        <note dur="2" oct="4" pname="d"/>
      </tuplet>
      <tuplet num="9" numbase="4" num.visible="false" bracket.visible="false">
        <note dur="2" oct="4" pname="c"/>
      </tuplet>
      <tuplet num="9" numbase="4" num.visible="false" bracket.visible="false">
        <note dur="4" oct="4" pname="d"/>
      </tuplet>
      <tuplet num="9" numbase="4" num.visible="false" bracket.visible="false">
        <note dur="4" oct="4" pname="c"/>
      </tuplet>
    </layer>
  </staff>
  <staff n="2">
    <layer n="1">
      <note dur="brevis" oct="4" pname="c"/>
      <note dur="minima" oct="4" pname="d"/>
      <note dur="minima" oct="4" pname="c"/>
      <note dur="semiminima" oct="4" pname="d"/>
      <note dur="semiminima" oct="4" pname="c"/>
    </layer>
  </staff>
</measure>
```

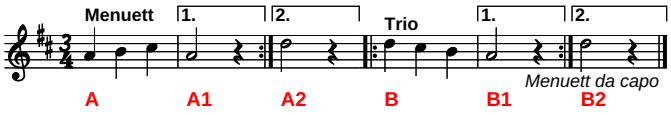
Repetition expansion

Scores may contain repetitions, endings, or directives to repeat a section from a certain location in the score, such as dal segno or similar. Such instructions allow performers to make informed decisions about the repetition structure during performance. The MEI schema provides the <expansion> element to encode specific versions of a score's repetition structure. Verovio supports this MEI element with the --expand toolkit option, which generates an unfolded version of the score in the requested output format. This realises the encoded repetitions by copying and/or deleting score elements to match the specified repetition structure.

The expansion element is expected to be the first element in a section or ending and must contain descendant expansion, ending, or rdg elements (see guidelines for section). Its @plist attribute may point to its descendant section, ending, rdg, or lem elements to indicate a particular expanded version of that excerpt of the score. See the MEI guidelines for a simple expansion example.

Minuet example

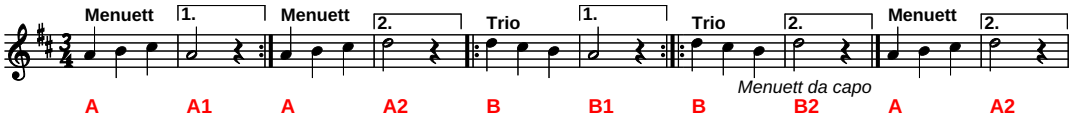
A typical MEI example (from the Verovio test suite example-expansion-001) is given below containing a straight-forward repetition structure in which the Minuet and the Trio are each repeated once with different endings. As indicated by "Menuett da capo", the performer is requested to repeat the Minuet after the Trio, but then without repetition, going directly to A2 to terminate the performance.



XML

```
<section xml:id="all">
  <expansion xml:id="expansion-default" plist="#A #A1 #A #A2 #B #B1 #B #B2 #A #A2"/>
  <expansion xml:id="expansion-minimal" plist="#A #A2 #B #B2 #A #A2"/>
  <expansion xml:id="expansion-maximal" plist="#A #A1 #A #A2 #B #B1 #B #B2 #A #A1 #A #A2"/>
  <section xml:id="A"/>
  <ending xml:id="A1" n="1."/>
  <ending xml:id="A2" n="2."/>
  <section xml:id="B"/>
  <ending xml:id="B1" n="1."/>
  <ending xml:id="B2" n="2."/>
</section>
```

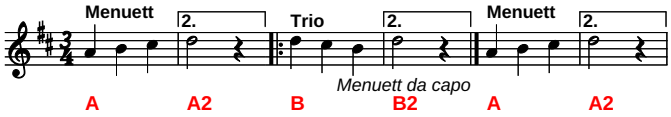
An expansion is engraved by Verovio by passing the xml:id of its expansion element as an option to the toolkit, e.g. --expand expansion-default for the command-line or expand: 'expansion-default' for Javascript/Python options, which looks like this:



This example also encodes a "minimal" expansion that omits sections A1 and B1, but still repeats the Minuet:

XML

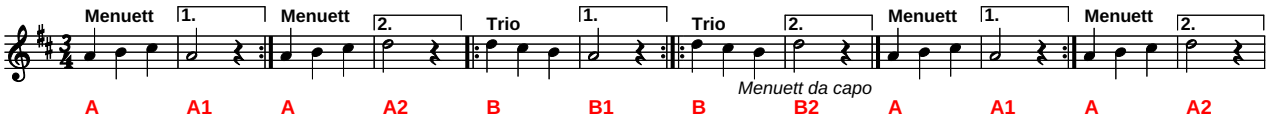
```
<expansion xml:id="expansion-minimal" plist="#A #A2 #B #B2 #A #A2"/>
```



This example also encodes a "maximal" expansion that realises all repeats, including in the repeated Minuet:

XML

```
<expansion xml:id="expansion-maximal" plist="#A #A1 #A #A2 #B #B1 #B #B2 #A #A1 #A #A2"/>
```



For an example encoding this minuet with a hierarchical expansion, see below.

Exporting an expansionmap

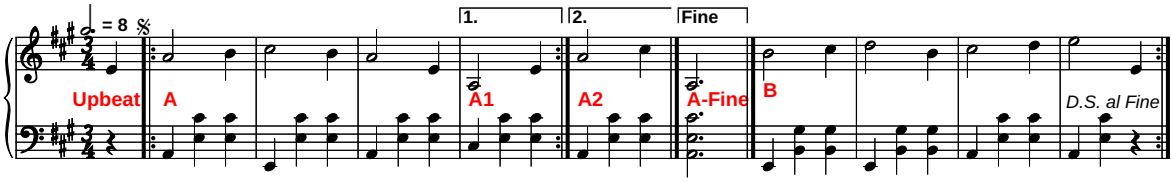
For sections that get cloned, Verovio generates predictable xml:id s for all containing elements, appending

a -rendX to the existing xml:id , where X is a number starting from 2 for the first repetition of a given element. Thus, xml:id="A-rend3" would refer to the third occurence (or the second repetition) of section "A" . To track the relationship between the original score and a unfolded repeats, Verovio provides access to the expansionmap. This JSON object contains key-value pairs with unique keys for each xml:id in the encoding (both the original and the unfolded elements) and values containing a list of related (original and unfolded) elements, e.g. ["A", "A-rend2", "A-rend3"] .

For more information on the expansionmap, see Output formats.

Example with re-ordered sections

The following example has a slightly more complex default expansion structure that requires section A to be rendered three times, each time choosing a different ending.

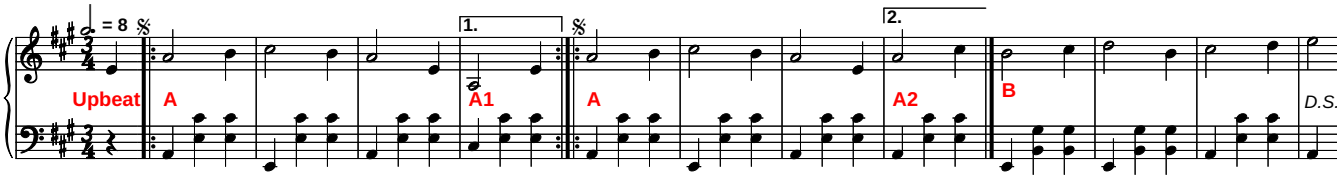


The following expansion represents a sensible default:

XML

```
<expansion xml:id="default" plist="#Upbeat #A #A1 #A #A2 #B #A #A-Fine"/>
```

Verovio accommodates this complexity automatically by re-ordering the section structure so that it looks like this:



Hierarchical expansion structure

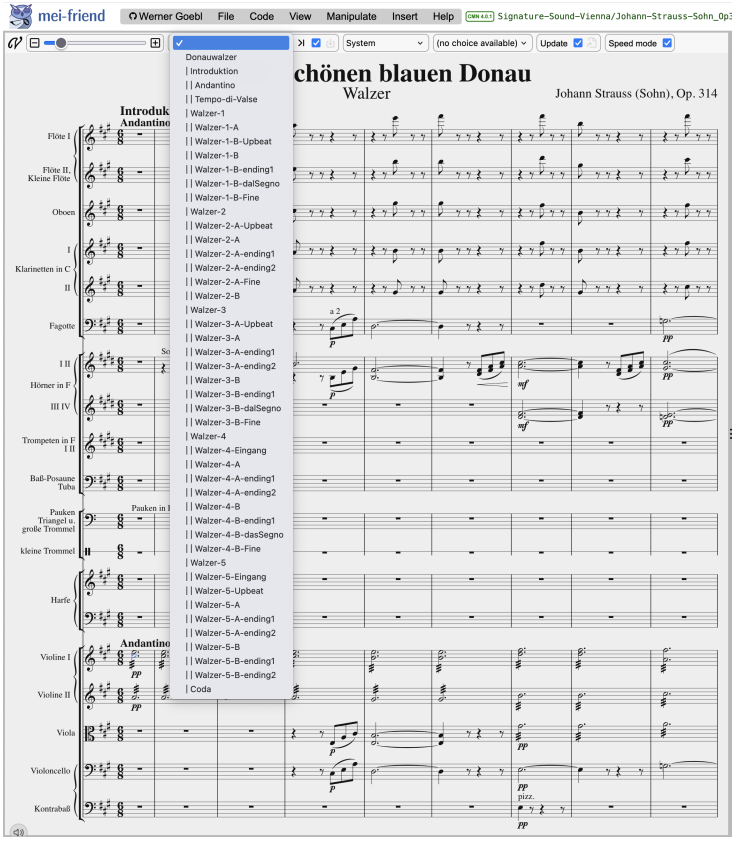
The elements referred to in the expansion@plist may themselves be expansion elements situated in descendent section elements.

The expansion structure of the above Minuet example could also be encoded in a hierarchical way, i.e., with separate sub-sections for Minuet and Trio, each hosting their own expansion element embedded, to which then the top-level expansion elements refer to (for the complete MEI encoding, see Verovio test suite example-expansion-001-hierarchical):

XML

```
<section xml:id="all">
  <expansion xml:id="exp-default" plist="#exp-menuett-default #exp-trio-default #exp-menuett-minimal"/>
  <expansion xml:id="exp-minimal" plist="#exp-menuett-minimal #exp-trio-minimal #exp-menuett-minimal"
/>
  <expansion xml:id="exp-maximal" plist="#exp-menuett-default #exp-trio-default #exp-menuett-default"/>
  <section xml:id="Menuett">
    <expansion xml:id="exp-menuett-default" plist="#A #A1 #A #A2"/>
    <expansion xml:id="exp-menuett-minimal" plist="#A #A2"/>
    <section xml:id="A"/>
    <ending xml:id="A1" n="1."/>
    <ending xml:id="A2" n="2."/>
  </section>
  <section xml:id="Trio">
    <expansion xml:id="exp-trio-default" plist="#B #B1 #B #B2"/>
    <expansion xml:id="exp-trio-minimal" plist="#B #B2"/>
    <section xml:id="B"/>
    <ending xml:id="B1" n="1."/>
    <ending xml:id="B2" n="2."/>
  </section>
</section>
```

A complex but typical example is the chained Waltz structure of An der schönen blauen Donau by Johann Strauss II. (To open a public-domain encoding in mei-friend, please click here.)



In turn, each waltz contains a set of expansion elements, each defining a repetition structure performed within the history of the Vienna New Year's Concert series. The higher-level expansion element then refers to these waltz-level expansions.

The beginning of the section structure of the first two waltzes including the respective expansion structure looks like this:

XML

```
<section xml:id="Donauwalzer">
    <expansion xml:id="Longest_Version_Krauss" plist="#Introduktion #Walzer-1-withRep-woDalSegno #Walzer-2-withRep-withDalSegno ..." />
    <expansion xml:id="Shorter_Version_Boskovsky" plist="#Introduktion #Walzer-1-woRep-woDalSegno #Walzer-2-woRep-withDalSegno ..."/>
    <section xml:id="Introduktion">
        <section xml:id="Andantino"/>
        <section xml:id="Tempo-di-Valse"/>
    </section>
    <section xml:id="Walzer-1">
        <expansion xml:id="Walzer-1-withRep-woDalSegno" plist="#Walzer-1-A #Walzer-1-B-Upbeat #Walzer-1-B #Walzer-1-B-ending1 #Walzer-1-B #Walzer-1-B-Fine"/>
        <expansion xml:id="Walzer-1-woRep-woDalSegno" plist="#Walzer-1-A #Walzer-1-B-Upbeat #Walzer-1-B #Walzer-1-B-Fine"/>
        <section xml:id="Walzer-1-A"/>
        <section xml:id="Walzer-1-B-Upbeat"/>
        <section xml:id="Walzer-1-B"/>
        <ending xml:id="Walzer-1-B-ending1" n="1."/>
        <ending xml:id="Walzer-1-B-dalSegno" n="2."/>
        <ending xml:id="Walzer-1-B-Fine" n="Fine"/>
    </section>
    <section xml:id="Walzer-2">
        <expansion xml:id="Walzer-2-withRep-withDalSegno" plist="#Walzer-2-A-Upbeat #Walzer-2-A #Walzer-2-A-ending1 #Walzer-2-A #Walzer-2-A-ending2 #Walzer-2-B #Walzer-2-A #Walzer-2-A-Fine"/>
        <expansion xml:id="Walzer-2-woRep-withDalSegno" plist="#Walzer-2-A-Upbeat #Walzer-2-A #Walzer-2-A-ending2 #Walzer-2-B #Walzer-2-A #Walzer-2-A-Fine"/>
        <section xml:id="Walzer-2-A-Upbeat"/>
        <section xml:id="Walzer-2-A"/>
        <ending xml:id="Walzer-2-A-ending1" n="1."/>
        <ending xml:id="Walzer-2-A-ending2" n="2."/>
        <ending xml:id="Walzer-2-A-Fine" n="Fine"/>
    </section>
...
```

The following expansion element refers to the longest and most typical realisation of this piece in the history of the Vienna New Year's Concert series, as first conducted by Clemens Krauss:

XML

```
<expansion xml:id="Longest_Version_Krauss" plist="#Introduktion #Walzer-1-withRep-woDalSegno #Walzer-2-withRep-withDalSegno #Walzer-3-withRep1-withRep2-woDalSegno #Walzer-4-withRep1-withRep2-woDalSegno #Walzer-5-withRep-woDalSegno #Coda"/>
```

Toolkit Reference

Input formats

When data is loaded into Verovio with no input format specifies, it tries to detect it based on the initial content of the data. MEI is assume to be the default format if auto detection fails. In such cases, the format can be given explicitly with the option `--input-from` (or `-f`).

MEI

The native input format for Verovio is MEI. Verovio supports MEI as input format from MEI 2013 onwards. From Verovio 2.x.x, the plan is to have even version numbers for Verovio releases using a stable version of MEI, and odd version numbers for releases using a development version of MEI. It means that once MEI 5.0 will be released, Verovio will move to version 4.x.x. Older versions of MEI are still supported by newer versions of Verovio.

When loading MEI data into Verovio and outputting MEI, elements that are not supported by Verovio will be ignored. This means that they are not loaded into memory and will not be preserved in the MEI output. This includes the element themselves, but also any descendant they might have. A warning will be given in the console. For example:

```
[Warning] Unsupported '<ossia>' within <measure>
```

Support for previous version of MEI

When an MEI file in loaded into Verovio and is not of the latest version for that version of Verovio, it performs upgrade steps for the features that were supported by Verovio for that older version of MEI.

MEI 2013 files

Various attributes in `<page>` and `<measure>` for the page-based version of MEI are upgraded (experimental work).

MEI 3.0 files

The following elements / attributes are upgraded:

- beatRpt
- fTrem@slash
- instrDef@midi.volume
- mordent@form
- turn@form
- staffDef@barthru
- staffDef@label
- staffDef@label.abbr
- staffGrp@label
- staffGrp@label.abbr
- @dur.ges

Original data

XML

```
<beatRpt rend="4" />
<beatRpt rend="8" />
<beatRpt rend="16" />
<beatRpt form="4" />
```

Upgraded data

XML

```
<beatRpt slash="1" />
<beatRpt slash="1" />
<beatRpt slash="2" />
<beatRpt slash="1" />
```

Original data

XML

```
<fTrem slash="2" />
```

Upgraded data

XML

```
<fTrem beams="2" />
```

Original data

XML

```
<instrDef midi.volume="111" />
```

Upgraded data

XML

```
<instrDef midi.volume="87.40%" />
```

Original data

XML

```
<mordent form="inv" />
<mordent form="norm" />
```

Upgraded data

XML

```
<mordent form="upper" />
<mordent form="lower" />
```

Original data

XML

```
<turn form="inv" />
<turn form="norm" />
```

Upgraded data

XML

```
<turn form="lower" />
<turn form="upper" />
```

Original data

XML

```
<staff barthru="true" />
```

Upgraded data

XML

```
<staff bar.thru="true" />
```

Original data

XML

```
<staffDef label="violin I" label.abbr="vl I" />
```

Upgraded data

XML

```
<staffDef>
  <label>Violin I<label>
  <labelAbbr>Vl I<labelAbbr>
</staffDef>
```

Original data

XML

```
<note dur.ges="8p" />
<note dur.ges="32r" />
<note dur.ges="32s" />
```

Upgraded data

XML

```
<note dur.ppq="8" />
<note dur.recip="32" />
<note dur.real="32" />
```

MEI 4.0 files

The following elements / attributes are upgraded:

- mensur@tempus
- mensur@prolatio

Original data

XML

```
<mensur tempus="3" />
<mensur tempus="2" />
```

Upgraded data

XML

```
<mensur tempus="3" sign="O" />
<mensur tempus="2" sign="C" />
```

Original data

XML

```
<mensur prolatio="3" />
<mensur prolatio="2" />
```

Upgraded data

XML

```
<mensur prolatio="3" dot="true" />
<mensur prolatio="2" dot="false" />
```

Page-based MEI

> The MEI page-based model is not part of MEI. It was put in place for the development of Verovio and can still change in the future. It will be documented as input format once it is stabilized.

Humdrum

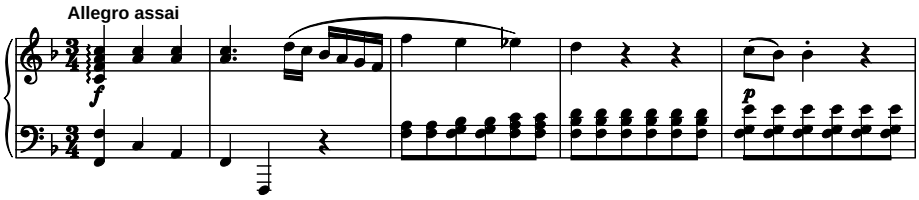
Humdrum data is an analytic music code for transcribing fully polyphonic textures. Humdrum syntax presents notes of the score in strict time sequence. Each data row represents all notes sounding or events occurring at the same time, and each column traces the melodic line of the individual parts. More information about the syntax is available on the Humdrum website.

Examples

The following example from Mozart's piano sonata in F major, K1 280 (K6 189e), mvmt. 1, is generated dynamically within this page using the JavaScript form of Verovio, inputting the Humdrum data that follows.

Piano Sonata No. 2 in F major

Wolfgang Amadeus Mozart



The data consists of three separate streams of information, called spines that usually consist of one column, but sometime more due to spine splits into subspines. The first column represents music on the bottom staff, the second column represents the top staff, and the third column contains the dynamics, which in this case apply to both staves.

```
!!!COM: Mozart, Wolfgang Amadeus
!!!OTL: Piano Sonata No. 2 in F major
!!!OMV: 1
!!!SCT1: K1 280
!!!SCT2: K6 189e
!!!OMD: Allegro assai
**kern **kern **dynam
*staff2 *staff1 *staff1/2
*clefF4 *clefG2 *
*k[b-] *k[b-] *
*F: *F: *
*M3/4 *M3/4 *
*MM152 *MM152 *
=1- =1- =1-
4FF 4F 4c: 4f: 4a: 4cc: f
4C 4a/ 4cc/ .
4AA 4a/ 4cc/ .
=2 =2 =2
4FF 4.a/ 4.cc/ .
4FFF . .
. (16ddLL .
. 16ccJJ .
4r 16b-LL .
. 16a .
. 16g .
. 16fJJ .
=3 =3 =3
8F 8AL 4ff .
8F 8A . .
8F 8G 8B- 4ee .
8F 8G 8B- . .
8F 8A 8c 4ee-) .
8F 8A 8cJ . .
=4 =4 =4
8F 8B- 8dL 4dd .
8F 8B- 8d . .
8F 8B- 8d 4r .
8F 8B- 8d . .
8F 8B- 8d 4r .
8F 8B- 8dJ . .
=5 =5 =5
8F 8G 8eL (8ccL p
8F 8G 8e 8b-J) .
8F 8G 8e 4b-' .
8F 8G 8e . .
8F 8G 8e 4r .
8F 8G 8eJ . .
=6 =6 =6
*- *- *-
```

Verovio Humdrum Viewer

The Verovio Humdrum Viewer (VHV) is a special-purpose interactive website for viewing and editing Humdrum files with the Verovio notation egraving library. You can view the full score for the above Mozart example in VHV from this link: verovio.humdrum.org/?file=mozart/sonatas/sonata02-1.krn.

When on a VHV notation page, try pressing the key "**p**" to view the scan of the original print from which the musical data was encoded. Also try pressing "**m**" to view the internal conversion to MEI data. Vi users can try pressing "**v**" to toggle between the basic and vim modes for the text editor. Use the left/right arrow keys or PageUp/PageDown to navigate to different pages. Press shift-left/right arrows to go to the next/previous work/movement in the repertory.

Sample repertories of Humdrum data displayed in the Verovio Humdrum Viewer:

- J.S. Bach chorales (When viewing a chorale, type the "o" letter key to toggle view of the original historic clefs.)
- Mozart piano sonatas
- Beethoven piano sontats
- Beethoven string quartets
- Chopin mazurkas
- Works of Scott Joplin
- Works of Josquin des Prez
- Works of Johannes Ockeghem
- Works of Pierre de la Rue
- Works of Mabrianus de Orto
- Deutscher Liederschatz, Band I (Edited by Ludwig Erk.)

Command-line interface usage

To typeset music in the Humdrum format on the command-line:

TERMINAL

```
$ verovio -f humdrum input.krn -o output.svg
```

You can usually use the auto-detection feature of verovio by omitting the option.

TERMINAL

```
$ verovio input.krn -o output.svg
```

The output filename will have the same basename as the input if the option is not given, so in this case the output will be called .

TERMINAL

```
$ verovio file.krn
```

Standard input/output can be used with the verovio command by giving a dash for standard input and to send the output to standard output.

TERMINAL

```
$ cat input.krn | verovio - -o - > output.svg
```

To convert to MEI data:

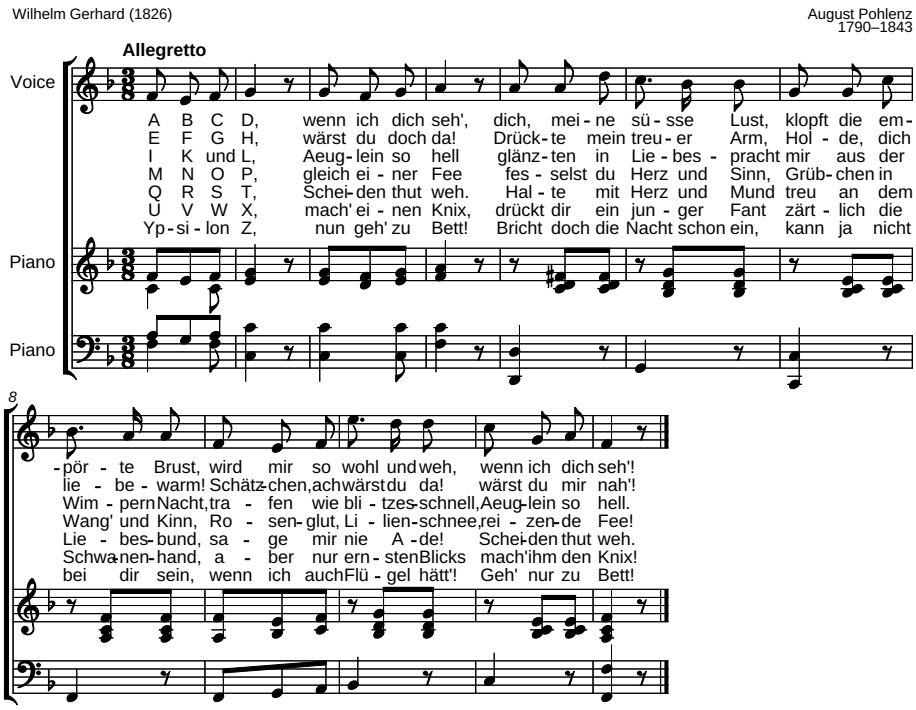
TERMINAL

```
$ verovio file.krn --no-layout --all-pages -t mei
```

A more complicated example

Below is a song for voice and piano accompaniment. Each verse is listed in a separate spine of **text in addition to the three staves of music in **kern spines and one dynamics (**dynam) spine.

Liebes-A-B-C




```
!!!OTL@@DE: Liebes-A-B-C
!!!COM: Pohlenz, August
!!!CDT: 1790/07/03/-1843/03/09/
!!!ODT: 1827
!!!OMD: Allegretto
!!!LYR: Gerhard, Wilhelm
!!!LDT: 1826
!!!OCL: Erk, Ludwig
!!!GCO: Deutscher Liederschatz, Band 1
**kern **kern **kern **text **text **text **text **text **text
*staff3 *staff2 *staff1 *staff1 *staff1 *staff1 *staff1 *staff1 *staff1
*Ipiano *Ipiano *Ivox * * * * * * *
*clefF4 *clefG2 *clefG2 * * * * * * *
*k[b-] *k[b-] *k[b-] * * * * * * *
*F: *F: *F: * * * * * * * *
*M3/8 *M3/8 *M3/8 * * * * * * *
```

```
*MM100 *MM100 *MM100 * * * * * * *
=1- =1- =1- =1- =1- =1- =1- =1- =1- =1-
*^ *^ * * * * * * * *
8AL 4F 8fL 4c 8f A E I M Q U Yp-
8G . 8e . 8e B F K N R V -si-
8AJ 8F 8fJ 8c 8f C G und O S W -lon
*v *v * * * * * * * * * *
* *v *v * * * * * * * *
=2 =2 =2 =2 =2 =2 =2 =2 =2 =2
4C 4c 4e 4g 4g D, H, L, P, T, X, Z,
8r 8r 8r . . . . . .
=3 =3 =3 =3 =3 =3 =3 =3 =3 =3
4C 4c 8e 8gL 8g wenn wärst Aeug- gleich Schei- mach' nun
. 8d 8f 8f ich du -lein ei- -den ei- geh'
8C 8c 8e 8gJ 8g dich doch so -ner thut -nen zu
=4 =4 =4 =4 =4 =4 =4 =4 =4 =4
4F 4c 4f 4a 4a seh', da! hell Fee weh. Knix, Bett!
8r 8r 8r . . . . . .
=5 =5 =5 =5 =5 =5 =5 =5 =5 =5
4DD 4D 8r 8a dich, Drück- glänz- fes- Hal- drückt Bricht
. 8c 8d 8f#L 8a mei- -te -ten -selst -te dir doch
8r 8c 8d 8f#J 8dd -ne mein in du mit ein die
=6 =6 =6 =6 =6 =6 =6 =6 =6 =6
4GG 8r 8.cc sü- treu- Lie- Herz Herz jun- Nacht
. 8B- 8d 8gL . . . . . .
. . 16b- -sse -er -bes- und und -ger schon
8r 8B- 8d 8gJ 8b- Lust, Arm, -pracht Sinn, Mund Fant ein,
=7 =7 =7 =7 =7 =7 =7 =7 =7 =7
4CC 4C 8r 8g klopft Hol- mir Grüb- treu zärt- kann
. 8B- 8c 8eL 8g die -de, aus -chen an -lich ja
8r 8B- 8c 8eJ 8cc em- dich der in dem die nicht
=8 =8 =8 =8 =8 =8 =8 =8 =8 =8
4FF 8r 8.b- -pör- lie- Wim- Wang' Lie- Schwa- bei
. 8A 8c 8fL . . . . . .
. . 16a -te -be- -pern und -bes- -nen- dir
8r 8A 8c 8fJ 8a Brust, warm! Nacht, Kinn, -bund, -hand, sein,
=9 =9 =9 =9 =9 =9 =9 =9 =9 =9
8FFL 8A 8fL 8f wird Schätz- tra- Ro- sa- a- wenn
8GG 8B- 8e 8e mir -chen, -fen -sen- -ge -ber ich
8AAJ 8c 8fJ 8f so ach wie -glut, mir nur auch
=10 =10 =10 =10 =10 =10 =10 =10 =10 =10
4BB- 8r 8.ee wohl wärst bli- Li- nie ern- Flü-
. 8B- 8d 8gL . . . . . .
. . 16dd und du -tzes- -lien- A- -sten -gel
8r 8B- 8d 8gJ 8dd weh, da! -schnell, schnee, de! Blicks hätt'!
=11 =11 =11 =11 =11 =11 =11 =11 =11 =11
4C 8r 8cc wenn wärst Aeug- rei- Schei- mach' Geh'
. 8B- 8c 8eL 8g ich du -lein -zen- -den ihm nur
8r 8B- 8c 8eJ 8a dich mir so -de thut den zu
=12 =12 =12 =12 =12 =12 =12 =12 =12 =12
4FF 4F 4A 4c 4f 4f seh'! nah'! hell. Fee! weh. Knix! Bett!
8r 8r 8r . . . . . .
== == == == == == == == == ==
*- *- *- *- *- *- *- *- *- *-
!!!SMS: Deutscher Liederschatz, Band 1, Ludwig Erk, ed.
!!!ENC: Craig Stuart Sapp
!!!END: 2004/05/16/
!!!EED: 2011/02/05/
!!!EFL: 001/651
!!muse2ps: s^(.){(..)}^z21jv200,114,144
!!CharacterEncoding: UTF-8
```

Additional input format via Humdrum

Verovio with Humdrum enabled supports some additional input formats that can be used with --input-from :

- **MuseData** with option md , musedata , or musedata-hum
- **EsAC** with esac

For more information about these input formats, see the Verovio Humdrum Viewerdocumentation.

MusicXML

Verovio has two converters for importing MusicXML data. The first one directly converts MusicXML into MEI. The second one first converts to Humdrum and then converts the Humdrum to MEI. By default, the first importer is used. It is also the one triggered when the value xml is passed to the --input-from option.

Compressed MusicXML files

Verovio supports MusicXML compressed (MXL) files. It only loads basic single-file MusicXML MXL files containing the index file (META-INF/container.xml) and the MusicXML file, with the extension .xml . The input process searches for the META-INF/container.xml file from which the filename of the MusicXML file is extracted. The filename extracted is the first ./rootfile@full-path listed in /container/rootfiles .

Input of MXL files is auto detected and the xml value does not have to be passed to --input-format . However, when using the JavaScript toolkit, you need to make sure your data is an ArrayBuffer or a base64 string, and use loadZipDataBuffer() or loadZipDataBase64() respectively to load it instead loadData() . Here is an example using the JavaScript Fetch API, loading the file as an ArrayBuffer :

JAVASCRIPT

```
fetch( mxlUrl )
    .then( response => response.arrayBuffer() )
    .then( data =>
    {
        vrvToolkit.loadZipDataBuffer( data )
        // Do anything else you want with the file here
    } ).catch( e =>
    {
        console.log( e );
    } );
```

Importing MusicXML via Humdrum

The MusicXML import via Humdrum is available only for Verovio builds where Humdrum support has been enabled specifically at build time. For the JavaScript toolkit, this is not the default and it is important to make sure that the appropriate build is being used. See the related section for more information about this. With the command-line tool and the Python toolkit, Humdrum support is enabled by default.

With Verovio builds that support Humdrum, the MusicXML import via Humdrum can be triggered by setting the --input-from option to musicxml-hum . For example:

TERMINAL

```
verovio -f musicxml-hum -t hum file.xml
```

The MusicXML import via Humdrum can itself be made the default MusicXML importer with the build option MUSICXML_DEFAULT_HUMDRUM . See the command-line section for more information on how to change build options. With this, MusicXML files will be loaded via the Humdrum importer without having to specify musicxml-hum for the option --input-from . The direct importer can still be used by passing the value xml to --input-from .

Plaine and Easie

The Plaine & Easie Code is a library standard that enables entering music incipits in modern or mensural notation. It is mostly used by the Répertoire International des Sources Musicales (RISM) for inventorying the music incipits of the manuscripts. More information about the syntax is available on the IAML website.

Plaine and Easie input in Verovio is a text file (or string) with a list of the following @key:value lines:

- @clef – the initial clef
- @keysig – the initial key signature
- @timesig – the initial time signature
- @data – the incipit content

From version 3.7, the content can be structured as a JSON object with a clef , keysig , timesig and data key. Verovio will auto detect both as Plaine & Easie format. Internaly, text files with @key:value lines are converted into a JSON object.

> The structure of this input format is not part of the PAE specification but only a convention put in place for Verovio

Examples

Beams and tuplets

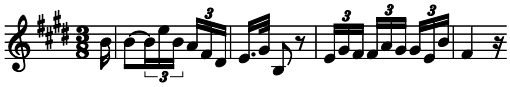
Text file input

```
@clef:G-2
@keysig:xFCGD
@timesig:3/8
@data:'6B/{8B+(6B''E'B)}({AFD})/{6.E3G},8B-/({6'EGF})({FAG})({GEB})/4F6-
```

JSON input

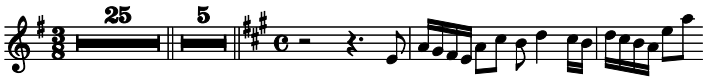
JSON

```
{
    "clef": "G-2",
    "keysig": "xFCGD",
    "timesig": "3/8",
    "data": "'6B/{8B+(6B''E'B)}({AFD})/{6.E3G},8B-/({6'EGF})({FAG})({GEB})/4F6-"
}
```



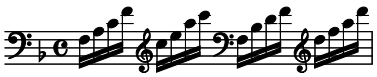
Measure rests and key and time signature changes

```
@clef:G-2
@keysig:xF
@timesig:3/8
@data:=25//=5//$xFCG @c 2-4.-'8E/{6AGFE}{8A"C}'B"4D{6C'B}/{"DC'BA}{"8EA}
```



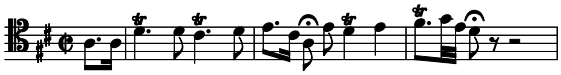
Clef changes

```
@clef:F-4
@keysig:bB
@timesig:c
@data:,6{FA'CF}%G-2 {"6CEA"'C}%F-4 {,6FB'DF}%G-2 {"6DFA"'D}/
```



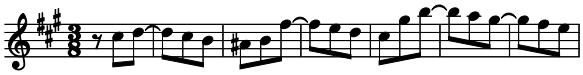
Trills and fermata

```
@clef:C-4
@keysig:xFC
@timesig:c/
@data:,{8.A6A}/'4.Dt8D4.Ct8D/{8.E6C},8(A)'E4DtE/{8.Ft3GE}8(D)-2-/
```



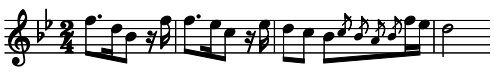
Ties

```
@clef:G-2
@keysig:xFCG
@timesig:3/8
@data:"8-{CD+}/{DC'B}/{xAB"F+}/{FED}/{CGB+}/{BAG+}/{GFE}/
```



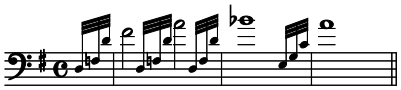
Grace notes (acciaccaturas)

```
@clef:G-2
@keysig:bBE
@timesig:2/4
@data:"{8.F6D'8B}6-"F/{8.F6E8C}6-E/{8DC}{'8B"gC'gBgAgB"6FE}/2D
```



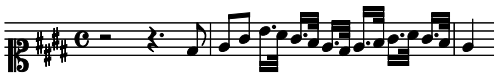
Grace notes (appoggiaturas)

```
@clef:F-4
@keysig:xF
@timesig:c
@data:qq,3{DnF'D}r/2Fqq,3{DnF'D}r2Aqq3,{DF'D}r/1bB,qq3{EG'C}r/1A//
```



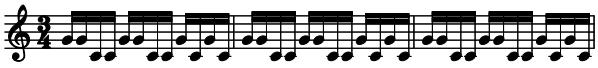
Rhythmic patterns

```
@clef:C-1
@keysig:xFCGD
@timesig:c
@data:'2-4.-8D/{EG}6.3{BA}{GF}{ED}{EF}{GA}{GF}/4E
```



Abbreviated writing

```
@clef:G-2
@keysig:
@timesig:3/4
@data:'6!{GGCC}!f{GCGC}/i/i/
```



PAE Validation

The toolkit can be used to validate Plaine & Easie input data with the ValidatePAE or ValidatePAEFile methods. The methods load the PAE data passed as a string or from a file respectively. They both return a stringified JSON object with validation error or warning messages.

The JSON object can contain one or more validation messages. When a global input error is encountered (e.g, data is missing in the input), a single object is returned. Otherwise, the object is structured with keys corresponding to the JSON input keys (clef , keysig , timesig and data). Each key can have one single validation message, except for data that contains an array of one or more messages. Only keys for which a validation message is given will exist in the validation object. In non-pendantic mode, syntax problems are marked as warning as long as parsing can continue.

Each validation message is structured as follow:

JSON

```
{
  "column": 0,
  "row": 0,
  "code": 1,
  "text": "A description of the validation problem",
  "type": "error"
}
```

Description of the values:

- The column indicates the position where the problem occurs in the input string. It is always 0 for clef , keysig and timesig . It can be -1 in data when no position can be indicated.
- The row is always 0 .
- The type can be error or warning .
- The code corresponds to a numeric error code that can be used to map the errors into another system and (for example) to translate the messages.

Whenever the error message contains a string interpolation %s , then the json message also contains a value key with the value to be used for the interpolation.

Here is an example of invalid input data and the object returned by the validation call:

JSON

```
{
    "clef": "GG2",
    "keysig": "bB",
    "data": "=1/4-''DC'tB/''tCC"
}
```

JSON

```
{
  "clef": {
    "column": 0,
    "row": 0,
    "code": 43,
    "text": "Unexpected second character in clef sign",
    "type": "warning"
  },
  "data": [
    {
      "column": 10,
      "row": 0,
      "code": 17,
      "text": "Invalid t not after a note",
      "type": "warning"
    },
    {
      "column": 15,
      "row": 0,
      "code": 17,
      "text": "Invalid t not after a note",
      "type": "warning"
    }
  ]
}
```

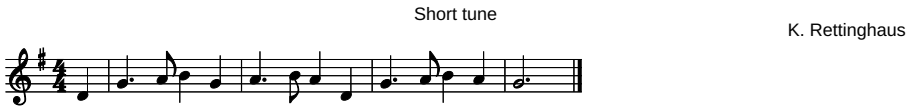

ABC

Abc is a text-based music notation system originally designed for use with folk and traditional tunes and used throughout the web. You can find the documentation on the ABC notation website.

Examples

Let's start with a simple little tune.

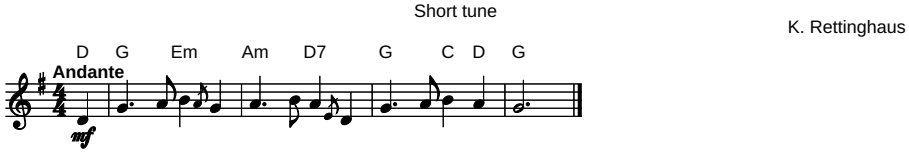
```
X: 99
T:Short tune
C:K. Rettinghaus
M:4/4
L:1/4
K:G
D|G>ABG|A>BAD|G>ABA|G3|]
```



Verovio takes several information fields into account, e.g. the reference number `X` , the tune title `T` , the meter `M` , the unit note length `L` , the key `K` . As you can see, Verovio prints the header as expected by default. You may suppress this behaviour with the `--header none` option.

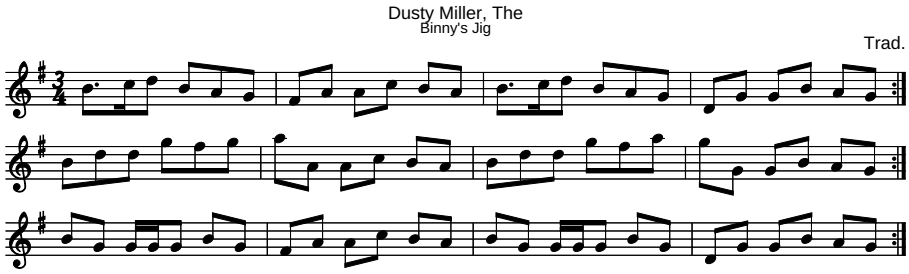
Now let's add a literal tempo as well as some grace notes and chord symbols. Dynamics are also very important! Note that chord symbols are put above the melody.

```
X: 99
T:Short tune
C:K. Rettinghaus
M:4/4
L:1/4
Q:"Andante"
K:G
"D"!mf!D|"G"G>A "Em"B {/A/}G|"Am"A>B "D7"A{/E/}D|"G"G>A "C"B "D"A|"G"G3|]
```



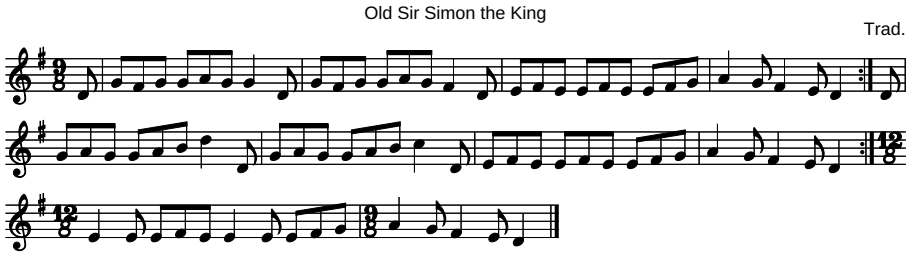
With the option `--breaks: 'encoded'` Verovio keeps the encoded layout, as you can see on this page. The default value is `'auto'` , which lets Verovio to decide where to put a line-break.

```
X:1
T:Dusty Miller, The
T:Binny's Jig
C:Trad.
R:DH
M:3/4
K:G
B>cd BAG|FA Ac BA|B>cd BAG|DG GB AG:|
Bdd gfg|aA Ac BA|Bdd gfa|gG GB AG:|
BG G/2G/2G BG|FA Ac BA|BG G/2G/2G BG|DG GB AG:|
```



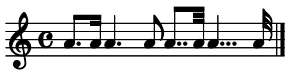
Alternatively it is always possible to suppress score line-breaks. Meter changes are also supported.

```
X:2
T:Old Sir Simon the King
C:Trad.
S:Offord MSS          % from Offord manuscript
N:see also Playford   % reference note
M:9/8
R:SJ                  % slip jig
N:originally in C     % transcription note
K:G
D|GFG GAG G2D|GFG GAG F2D|EFE EFE EFG|A2G F2E D2:|
D|GAG GAB d2D|GAG GAB c2D|[1 EFE EFE EFG|A2G F2E D2:|\\\ % no line-break in score
M:12/8                % change of meter
[2 E2E EFE E2E EFG|\\\  % no line-break in score
M:9/8                 % change of meter
A2G F2E D2|]
```



Broken rhythm markers

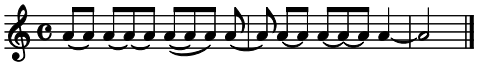
```
X:1
T:Broken rhythm markers
M:C
K:C
A>A A2>A2 A>>A A2>>>A2|]
```



Ties and slurs

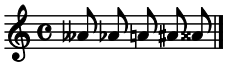
Verovio correctly differentiates between ties and slurs.

```
X:1
T:Ties and Slurs
M:C
K:C
(AA) (A(A)A) ((AA)A) (A|A) A-A A-A-A A2-|A4|]
```



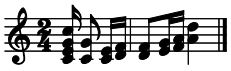
Accidentals

```
X:1
T:Accidentals
M:C
K:C
__A _A =A ^A ^^A|]
```



Chords

```
X:1
T:Chords
M:2/4
K:C
[CEGc] [C2G2] [CE][DF]|[D2F2][EG][FA] [A4d4]|]
```



Known limitations:

- Tuplets are not supported
- User defined symbols are not supported
- Multi-voice music is not supported

CMME

The CMME format is an XML-based format developped for the Computerized Mensural Music Editing project. It can be imported into Verovio as of version 4.4. CMME files can be imported with `-f cmme.xml` but are also auto-detected.

A CMME file is imported as an MEI mensural file with one single `mdiv/score`. Sections in CMME are imported as distinct `section` elements within the `score`.

Everything in imported as `mensural` notation, including sections marked as `Plainchant` in CMME since there is no difference in the way the music is encoded in those sections. Where there is less voices in a section than in the rest of the score, empty and invisible staff elements `<staff visibility="false"/>` are added to the section.

The initial `staffDef` elements have no `@clef.*` and no `@keyisg` and these are given in the layer instead. Every `staffDef` contains a `mensur` indicating that all level of divisions are binary by default.

MEI header

The `GeneralData` element of the CMME file is used to populate the MEI header. It uses:

- The `Title` as `title`
- The `Section` as `title@type="subordinate"`
- The `Composer` as `composer`

Durations

The CMME import forces the use of the `--duration-equivalence` option to `minima` when importing a CMME file. A warning is shown if the option was previously different - which is the case by default. This could potentially have side effects for subsequent calls since it is not set back to its original value.

Proportions

Proportions in CMME can be encoded as `Proportion` element, or as `TempoChange` within `Mensuration`. Encoding a proportion as a tempo change is arguably not the proper way to do it, but as a matter of fact, it is a used practice the importer needs to deal with. The importer tries to disentangle proportions and tempo changes. One complication is that proportions and tempo changes in CMME act differently. Proportions are cumulated with the previous ones, whereas tempo changes are not.

Ideally proportions encoded in a mensur indications that are actual tempo changes should be ignored in the conversion because they do not represent a proportion in the notation. Furthermore, when converting the imported data to CMN, this will yield measure content that hardly makes sense since tuplets will be used to represent the proportions. However, when a proportion encoded in a mensur indication is a real proportion, it must be taken into account and preserved.

The importer ignores, if possible, the proportions inserted in the mensuration signs indicating a tempo change. This is done on the basis of the presence of an identical proportion for all voices. When tempo change proportions are occurring at all voices and with an identical `Num` and `Den`, they are preserved as an MEI `proportion` with a `@type="cmme_tempo_change"`. Verovio ignores when performing that alignment of the data, including when converting to CMN.

When the proportion is not identical in all voices, it has to be preserved and taken into account for the alignment of the data. We can here distinguish two cases. The first is when the tempo change does not occur in all voices. In this case, it quite likely corresponds to a true proportion. It is preserved as an MEI proportion but with a @type="reset" to indicate that it should not be cumulated with a previous one - which is the default behavior of Verovio.

The second case is where all voices have a proportion, but it differs. In this case, there is no straightforward way of knowing which proportion is a tempo change, and which is a mixture of the two. The conversion will maintained an MEI proportion with @type="reset" at all voices. Even though the conversion result will be properly aligned in mensural MEI, it will yield a combination of corresponding tuplets when converted into CMN that will not be fully satisfactory. The best thing to do would be to correct the CMME files and to remove the tempo changes, or to separate the proportions and the tempo changes.

Finally, a CMME Proportion will converted to MEI proportion with no @type , impliying that the proportion has to be cumulated with a previous one.

Coloration and color change

Coloration in CMME is encoded with Colored on Note , and is converted to MEI @colored .

The CMME ColorChange (i.e., the change of the color of the ink) is converted by applying the color (non-black) to note@color and rest@color . The possible colors are black (assumed to be the default and not encoded), red , yellow , green or blue .

Output formats

SVG

For more information about the SVG output in Verovio, see the Internal structure and the Controlling the SVG output sections in the previous chapter.

Font limitation

Firefox on Linux (Ubuntu), uses “DejaVu Serif” as default font, which can cause some text layout problems when displaying the SVG files generated with Verovio.

MEI

With its MEI output, Verovio can serve as a converter to MEI. This can be useful for converting data from another input format supported by Verovio (e.g., MusicXML, ABC) to MEI. It can also be used to upgrade files encoded in an older version of MEI to the one supported by the version of Verovio that being used. Another typical use-case where outputting MEI from Verovio can be desirable is for transposing content.

When converting other formats to MEI, it is important to keep in mind that the output produced by Verovio will only include the MEI features (elements and attributes) currently supported by the Verovio version being used. It is also important to remember that the MEI produced by Verovio is only one way to express things in MEI and that MEI will often offer other valid and recommendable ways to represent the same things. Choosing between them depends on the goal being pursued. It is possible that Verovio is the appropriate solution but not necessary.

When converting from an older version of MEI, it is important to remember that Verovio will not perform any upgrade of the data encoded in the MEI header, with the exception of the MEI version. This means that using Verovio for upgrading MEI data is probably appropriate only for encodings that feature a very basic header and is not recommended with rich ones. It is recommended to check what has changed in MEI for the header between the versions. In any case, it is strongly recommended to check the header by validating the output files produced by Verovio. Regarding the content, Verovio will upgrade only the features that used to be supported in the previous version. See the section on MEI in the Input formats in the previous section for more detail about what is upgraded.

Unsupported elements and attributes

When loading MEI data into Verovio and outputting MEI, the following is to expect regarding MEI elements and attributes that are currently not supported by Verovio. Because elements that are not supported by Verovio are ignored and are not loaded into memory, they will not be preserved in the MEI output. This includes the element themselves, but also any descendant they might have. As described in the section about the Input formats, a warning will appear in the console about these. There is one exception with the <annot> elements for which all the content will be preserved, including MEI element descendants that are not supported elsewhere in Verovio. Regarding attributes, Verovio will preserve in the output all attributes, including the one that are not supported or that have not relevance for the rendering.

Analytical markup

When loading MEI data into Verovio, some analytical markup is converted into standard markup.

The attributes that are converted are:

- @fermata
- @tie

For example:

Original data

XML

```
<note t="i" xml:id="n1"/>
<note t="t" xml:id="n2"/>
```

Output data

XML

```
<tie startid="#n1" endid="#n2"/>
```

Original data

XML

```
<mRest fermata="above" xml:id="mr1"/>
```

Output data

XML

```
<fermata startid="#mr1" place="above"/>
```

By default, the analytical markup is not preserve in the MEI output. It can be with the option --preserve-analytical-markup .

Articulations

Articulations in MEI can be encoded with multiple values within a @artic attribute. Verovio implementation is based on single valued @artic attributes. When loading MEI data, multiple valued attributes are transformed into corresponding single valued ones by duplicating the <artic> element. This remain as such in the MEI output. For example:

Original data

XML

```
<artic artic="marc ten" place="above"/>
```

Output data

XML

```
<artic artic="marc" place="above"/>
<artic artic="ten" place="above"/>
```

Page-based MEI

> The MEI page-based model is not part of MEI. It was put in place for the development of Verovio and can still change in the future. It will be documented as input format once it is stabilized.

MIDI

Verovio provides a basic MIDI output feature that can be used from the command-line tool or from the JavaScript toolkit. The MIDI output can be written to a file for further processing or for building application with MIDI playback, including in online environments. However, since MIDI is not supported in web-browsers in a standard way, an additional player will be required in such cases.

When a file is loaded in the toolkit only to render a MIDI file, then setting the --breaks option to none is more efficient because it will avoid the unecessary step of calculating the page layout of the document. The command-line version of the toolkit does this automatically.

The MIDI output takes into account:

- Tempo indication (@midi.bpm) provided in the first scoreDef and in tempo elements.
- The sounding accidental values provided by @accid.ges on notes and accid .
- The sounding octave values provided by @oct.ges on note .
- Transposing instrument information provided by @trans.semi on staffDef .
- Tie elements referring to notes with @startid and @endid .

Verovio uses the Midifile library for generating the MIDI output.

Usage

With the command-line tool, for generating a MIDI file with the default options, you need to do:

TERMINAL

```
verovio -t midi -o output.midi input-file.mei
```

With the JavaScript toolkit, the MIDI output is available through the renderToMIDI() method. This returns a base64-coded MIDI file as string, which can be passed to a player or made available for download.

Timemap

The timemap is an array of JSON objects, with each entry having these keys:

- tstamp: this is the time in millisecond from the start of the music to the start of the current event (real time)
- qstamp: the time in quarter notes from the start of the music to the start of the current event entry (score time)
- tempo: when the tempo changes the new tempo will be given for the current event. Also the tempo changes are only allowed to occur at the starts of measures in the current code for creating MIDI files, and this is the same limitation for the timemap file. The tempo and qstamp values can be used to re-calculate a new set of tstamp values if the tempo changes.
- on: This is an array of note ids that start at the current event time. This list will not be given if there are no note ons at the current event.
- off: This is an array of note ids that end at the current event time. This list will not be given if there are

no note offs at the current event.

When a file is loaded in the toolkit only to render the timemap file, then setting the --breaks option to none is more efficient because it will avoid the unecessary step of calculating the page layout of the document. The command-line version of the toolkit does this automatically.

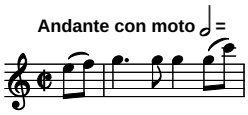
Usage

With the command-line tool, for generating a timemap JSON with the default options, you need to do:

TERMINAL

```
verovio -t timemap -o output.json input-file.mei
```

Examples



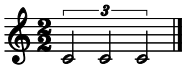
XML

```
<measure type="upbeat">
  <staff n="1">
    <layer n="1">
      <beam>
        <note xml:id="m0_s2_e1" dur="8" oct="5" pname="e"/>
        <note xml:id="m0_s2_e2" dur="8" oct="5" pname="f"/>
      </beam>
    </layer>
  </staff>
  <tempo staff="1" tstamp="1" midi.bpm="70">Andante con moto <rend glyph.auth="smufl">𝅗𝅥</rend> = 70
</tempo>
  <slur startid="#m0_s2_e1" endid="#m0_s2_e2"/>
</measure>
<measure n="1">
  <staff n="1">
    <layer n="1">
      <note dots="1" dur="4" oct="5" pname="g"/>
      <note dur="8" oct="5" pname="g"/>
      <note dur="4" oct="5" pname="g"/>
      <beam>
        <note xml:id="m1_s2_e4" dur="8" oct="5" pname="g"/>
        <note xml:id="m1_s2_e5" dur="8" oct="6" pname="c"/>
      </beam>
    </layer>
  </staff>
  <slur startid="#m1_s2_e4" endid="#m1_s2_e5"/>
</measure>
```

JSON

```
[
  {
    "tstamp": 0,
    "qstamp": 0,
    "tempo": 70,
    "on": [
      "m0_s2_e1"
    ]
  },
  {
    "tstamp": 428.571429,
    "qstamp": 0.5,
    "on": [
      "m0_s2_e2"
    ],
    "off": [
      "m0_s2_e1"
    ]
  },
  {
    "tstamp": 857.142857,
    "qstamp": 1,
    "on": [
      "note-0000001938389898"
    ],
    "off": [
```

```
    "m0_s2_e2"
   ]
  },
  {
   "tstamp": 2142.857143,
   "qstamp": 2.5,
   "on": [
    "note-0000001651747389"
   ],
   "off": [
    "note-0000001938389898"
   ]
  },
  {
   "tstamp": 2571.428571,
   "qstamp": 3,
   "on": [
    "note-0000001917733971"
   ],
   "off": [
    "note-0000001651747389"
   ]
  },
  {
   "tstamp": 3428.571429,
   "qstamp": 4,
   "on": [
    "m1_s2_e4"
   ],
   "off": [
    "note-0000001917733971"
   ]
  },
  {
   "tstamp": 3857.142857,
   "qstamp": 4.5,
   "on": [
    "m1_s2_e5"
   ],
   "off": [
    "m1_s2_e4"
   ]
  },
  {
   "tstamp": 4285.714286,
   "qstamp": 5,
   "off": [
    "m1_s2_e5"
   ]
  }
]
```

JSON

```
[
  {
    "tstamp": 0,
    "qstamp": 0,
    "tempo": 120,
    "on": [
      "note-0000002010789077"
    ]
  },
  {
    "tstamp": 666.666667,
    "qstamp": 1.333333,
    "on": [
      "note-0000001595005340"
    ],
    "off": [
      "note-0000002010789077"
    ]
  },
  {
    "tstamp": 1333.333333,
    "qstamp": 2.666667,
    "on": [
      "note-0000000084354770"
    ],
    "off": [
      "note-0000001595005340"
    ]
  },
  {
    "tstamp": 2000,
    "qstamp": 4,
    "off": [
      "note-0000000084354770"
    ]
  }
]
```


Expansionmap

The expansionmap is a JSON object that is generated when the xml:id of an expansion element is passed to Verovio with the --expand option. The expansionmap represents the relationship between the original score parts and the repeated (expanded, unfolded) sections created through the expansion process.

While expanding, Verovio generates new, predictable xml:id s for all expanded elements (section , ending , rdg , lem), appending a -rendX to each xml:id , with X being the number of occurrences, starting from 2 for the first repetition of a given element. Thus, xml:id="A-rend3" would refer to the third occurence (or the second repetition) of element "A" .

The keys in the expansionmap JSON object are the xml:id s of expanded elements in the score (both original and unfolded). The values contain lists of related (original and unfolded) elements, e.g. ["A", "A-rend2", "A-rend3"] . The original score element (e.g., "A") is always the first value in the list.

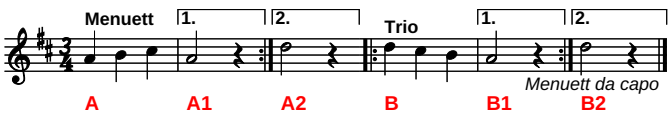
Usage

To generate an expansionmap JSON with the default options on the command line:

TERMINAL

```
verovio -t expansionmap --expand expansion-default -o output.json expansion-001.mei
```

Examples



XML

```
<section xml:id="all">
  <expansion xml:id="expansion-default" plist="#A #A1 #A #A2 #B #B1 #B #B2 #A #A2"/>
  <expansion xml:id="expansion-minimal" plist="#A #A2 #B #B2 #A #A2"/>
  <expansion xml:id="expansion-maximal" plist="#A #A1 #A #A2 #B #B1 #B #B2 #A #A1 #A #A2"/>
  <section xml:id="A"/>
  <ending xml:id="A1" n="1."/>
  <ending xml:id="A2" n="2."/>
  <section xml:id="B"/>
  <ending xml:id="B1" n="1."/>
  <ending xml:id="B2" n="2."/>
</section>
```

The beginning of the expansionmap from this example with the “default” expansion

XML

```
<expansion xml:id="expansion-default" plist="#A #A1 #A #A2 #B #B1 #B #B2 #A #A2"/>
```

looks like this:

JSON

```
{
    "A": [
        "A",
        "A-rend2",
        "A-rend3"
    ],
    "A-rend2": [
        "A",
        "A-rend2",
        "A-rend3"
    ],
    "A-rend3": [
        "A",
        "A-rend2",
        "A-rend3"
    ],
    "A2": [
        "A2",
        "A2-rend2"
    ],
    "A2-rend2": [
        "A2",
        "A2-rend2"
    ],
    "B": [
        "B",
        "B-rend2"
    ],
    "B-rend2": [
        "B",
        "B-rend2"
    ],
    "dbf3mxc": [
        "dbf3mxc",
        "dbf3mxc-rend2",
        "dbf3mxc-rend3"
    ],
    "dbf3mxc-rend2": [
        "dbf3mxc",
        "dbf3mxc-rend2",
        "dbf3mxc-rend3"
    ],
    "dbf3mxc-rend3": [
        "dbf3mxc",
        "dbf3mxc-rend2",
        "dbf3mxc-rend3"
    ],
    "c1k0s711": [
        "c1k0s711",
        "c1k0s711-rend2",
        "c1k0s711-rend3"
    ],
    "c1k0s711-rend2": [
        "c1k0s711",
        "c1k0s711-rend2",
        "c1k0s711-rend3"
    ],
    "c1k0s711-rend3": [
        "c1k0s711",
        "c1k0s711-rend2",
        "c1k0s711-rend3"
    ],
```


Plaine and Easie

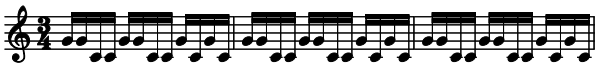
The output format for the Plaine and Easie output in Verovio uses the same file structure with key:value lines as described in the section in the Input formats. See also there for the features supported.

Note that:

- duration is given explicitly for every note
- no abbreviated writing is used in the Plaine and Easie output

For example, let's consider the following example passed as input to Verovio:

```
@clef:G-2
@keysig:
@timesig:3/4
@data:'6!{GGCC}!f{GCGC}/i/i/
```



Verovio will produce the following Plaine and Easie output:

```
@keysig:b
@timesig:3/4
@clef:G-2
@data:{6'G6G6C6C}{6G6G6C6C}{6G6C6G6C}/{6G6G6C6C}{6G6G6C6C}{6G6C6G6C}/{6G6G6C6C}{6G6G6C6C}{6G6C6G6C}/
```

Humdrum

The Humdrum output format for Verovio is available only from MusicXML input and only if the Humdrum importer is used when loading the data into Verovio. See this section for more information about the MusicXML import via Humdrum.

With this in hand, you can convert MusicXML to Humdrum from the command-line with:

TERMINAL

```
verovio -f musicxml-hum -t hum file.xml
```

If the MusicXML importer via Humdrum is the default, you can simply do:

TERMINAL

```
verovio -t hum file.xml
```

Toolkit methods

This section documents the methods available from the Verovio toolkit. The methods are public methods of the C++ vrv::Toolkit class. They are all available in the Python and JavaScript bindings, unless specified otherwise. For examples, all the methods reading a file or writing to a file are not available in the JavaScript version of the toolkit.

The names of the methods is also indentical across the different versions of the toolkit except for the capitalisation. The original C++ method names are UpperCamelCased in C++ but lowerCamelCased in the Python and JavaScript bindings. This is only to make the bindings follow more idiomatic capitalisation.

For the methods taking parameters as stringified JSON objects (or returning one), the objects are not stringified in the JavaScript version of the toolkit. That is, a JSON object is passed or returned as is. The same applied for the Python toolkit where the object is passed or returned as a Python dictionary.

ConvertHumdrumToHumdrum

Filter Humdrum data.

Returns

std::string – The Humdrum data as a string

Parameters

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

Original header

C++

```
std::string vrv::Toolkit::ConvertHumdrumToHumdrum(const std::string &humdrumData)
```

Example call

PYTHON

```
result = toolkit.convertHumdrumToHumdrum(humdrumData)
```

ConvertHumdrumToMIDI

Convert Humdrum data to MIDI.

Returns

std::string – The MIDI file as a base64-encoded string

Parameters

Name	Type	Default	Description

humdrumData	const std::string &	∅

Original header

C++

```
std::string vrv::Toolkit::ConvertHumdrumToMIDI(const std::string &humdrumData)
```

Example call

PYTHON

```
result = toolkit.convertHumdrumToMIDI(humdrumData)
```

ConvertMEIToHumdrum

Convert MEI data into Humdrum data.

Returns

std::string – The Humdrum data as a string

Parameters

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

Original header

C++

```
std::string vrv::Toolkit::ConvertMEIToHumdrum(const std::string &meiData)
```

Example call

PYTHON

```
result = toolkit.convertMEIToHumdrum(meiData)
```

Edit

Edit the MEI data - experimental code not to rely on.

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 - experimental code not to rely on.

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

GetDefaultOptions

Return a dictionary of all the options with their default value.

Returns

std::string – A stringified JSON object

Original header

C++

```
std::string vrv::Toolkit::GetDefaultOptions() const
```

Example call

PYTHON

```
result = toolkit.getDefaultOptions()
```

GetDescriptiveFeatures

Return descriptive features as a JSON string.

The features are tailored for implementing incipit search.

Returns

std::string – A stringified JSON object with the requested features

Parameters

Name	Type	Default	Description
jsonOptions	const std::string &	∅	A stringified JSON object with the feature extraction options

Original header

C++

```
std::string vrv::Toolkit::GetDescriptiveFeatures(const std::string &jsonOptions)
```

Example call

PYTHON

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

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. If the element is not found, the method returns an empty JSON object.

GetElementsAtTime

Return 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

Return 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 method is not available in the JavaScript distributed 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

Original header

C++

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

Example call

PYTHON

result = toolkit.getHumdrumFile(filename)

GetID

Return the ID of the Toolkit instance.

> This method is not available in the JavaScript distributed version of the toolkit

Returns

std::string – The ID as as string

Original header

C++

std::string vrv::Toolkit::GetID()

Example call

PYTHON

result = toolkit.getID()

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; basic: true or false; false 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

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

Returns

std::string – An ID string

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)
```

GetOptionUsageString

Get all usage for all option categories as string.

Returns

std::string

Original header

C++

```
std::string vrv::Toolkit::GetOptionUsageString() const
```

Example call

PYTHON

```
result = toolkit.getOptionUsageString()
```

GetOptions

Return a dictionary of all the options with their current value.

Returns

std::string – A stringified JSON object

Original header

C++

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

Example call

PYTHON

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

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)
```

GetResourcePath

Get the resource path for the Toolkit instance.

> This method is not available in the JavaScript distributed version of the toolkit

Returns

std::string – A string with the resource path

Original header

```
C++
std::string vrv::Toolkit::GetResourcePath() const
```

Example call

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

GetScale

Get the scale option.

> This method is not available in the JavaScript distributed version of the toolkit

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)

GetVersion

Return the version number.

Returns

std::string – the version number as a string

Original header

C++

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

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.

> This method is not available in the JavaScript distributed version of the toolkit

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)
```

PrintOptionUsage

Print formatted option usage for specific category (with max/min/default values) to output stream.

Returns

void

Parameters

Name	Type	Default	Description

category	const std::string &	∅
output	std::ostream &	∅

Original header

C++

void vrv::Toolkit::PrintOptionUsage(const std::string &category, std::ostream &output) const

Example call

PYTHON

toolkit.printOptionUsage(category, output)

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

Parameters

Name	Type	Default	Description
jsonOptions	const std::string &	""	A stringified JSON object with the action options resetCache: true or false; true by default;

Original header

C++

void vrv::Toolkit::RedoLayout(const std::string &jsonOptions="")

Example call

PYTHON

toolkit.redoLayout(jsonOptions)

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()

RenderData

Render the first page of the data to SVG.

This method is a wrapper for setting options, loading data and rendering the first page. It will return an empty string if the options cannot be set or the data cannot be loaded.

Returns

std::string – The SVG first page as a string

Parameters

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

Original header

C++

std::string vrv::Toolkit::RenderData(const std::string &data, const std::string &jsonOptions)

Example call

PYTHON

```
result = toolkit.renderData(data, jsonOptions)
```

RenderToExpansionMap

Render a document's expansionMap, if existing.

Returns

std::string – The expansionMap as a string

Original header

C++

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

Example call

PYTHON

```
result = toolkit.renderToExpansionMap()
```

RenderToExpansionMapFile

Render a document's expansionMap and save it to a file.

> This method is not available in the JavaScript distributed version of the toolkit

Returns

bool

Parameters

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

Original header

C++

```
bool vrv::Toolkit::RenderToExpansionMapFile(const std::string &filename)
```

Example call

PYTHON

```
result = toolkit.renderToExpansionMapFile(filename)
```

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 method is not available in the JavaScript distributed 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

Original header

C++

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

Example call

```
PYTHON
result = toolkit.renderToMIDIFile(filename)
```

RenderToPAE

Render a document to Plaine & Easie code.

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 & Easie code and save it to the file.

Only the top staff / layer is exported.

> This method is not available in the JavaScript distributed 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

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 method is not available in the JavaScript distributed 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
pageNo	int	1	The page to render (1-based)

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

Parameters

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

Original header

C++

```
std::string vrv::Toolkit::RenderToTimemap(const std::string &jsonOptions="")
```

Example call

PYTHON

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

JSON Options

"includeMeasures": <boolean>	Include measures in the timemap (false by default)
"includeRests": <boolean>	Include rests in the timemap (false by default)

Examples:

C++

```
std::string jsonOptions = "{'includeMeasures': true}";
```

PYTHON

```
jsonOptions = {'includeMeasures': True}
```

RenderToTimemapFile

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

> This method is not available in the JavaScript distributed 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 objects with the timemap options

Original header

C++

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

Example call

PYTHON

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

ResetOptions

Reset all options to default values.

Returns

void

Original header

C++

```
void vrv::Toolkit::ResetOptions()
```

Example call

PYTHON

```
toolkit.resetOptions()
```

ResetXmlIdSeed

Reset the seed used to generate MEI @xml:id attribute values.

Passing 0 will seed the @xml:id generator with a random (time-based) seed value. This method will have no effect if the xml-id-checksum option is set.

Returns

void

Parameters

Name	Type	Default	Description
seed	int	∅	The seed value for generating the @xml:id values (0 for a time-based random seed)

Original header

C++

```
void vrv::Toolkit::ResetXmlIdSeed(int seed)
```

Example call

PYTHON

```
toolkit.resetXmlIdSeed(seed)
```

SaveFile

Get the MEI and save it to the file.

> This method is not available in the JavaScript distributed 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)
```

Select

Set the value for a selection.

The selection will be applied only when some data is loaded or the layout is redone. The selection can be reset (cancelled) by passing an empty string or an empty JSON object. A selection across multiple mdivs is not possible.

Returns

bool – True if the selection was successfully parsed or reset

Parameters

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

Original header

C++

```
bool vrv::Toolkit::Select(const std::string &selection)
```

Example call

PYTHON

```
result = toolkit.select(selection)
```

Selection parameter

The JSON object can have a `measureRange` , a `start` and `end` , or can be empty for re-setting the selection. The measure range in `measureRange` is 1-based. The position value is the index position of the measure and not the `measure@n` value. The values `start` and `end` can be used as range to specify the beginning and respectively the end of the document. When specifying `start` or `end` keys, the values must refer to the `measure@xml:id` .

Examples of parameters:

JSON

```
{ "measureRange": "2-3" }
{ "measureRange": "82-end" }
{ "measureRange": "38" }
{ "start": "measure-L337", "end": "measure-L355" }
{}
```

See also: Score content selection

SetInputFrom

Set the input from option.

> This method is not available in the JavaScript distributed version of the toolkit

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)
```

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.

> This method is not available in the JavaScript distributed version of the toolkit

Returns

bool – True if the option was successfully set

Parameters

Name	Type	Default	Description
outputTo	std::string const &	∅	the value to output as string

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 and any extra fonts.

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

> This method is not available in the JavaScript distributed version of the toolkit

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.

> This method is not available in the JavaScript distributed version of the toolkit

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)
```

See also: Scaling

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)
```

ValidatePAE

Validate the Plaine & Easie code passed in the string data.

A single JSON object is returned when there is a global input error. When reading the input succeeds, validation is grouped by input keys. The methods always returns errors in PAE pedantic mode. No data remains loaded after the validation.

Returns

std::string – A stringified JSON object with the validation warnings or errors

Parameters

Name	Type	Default	Description
data	const std::string &	∅	A string with the data in JSON or with PAE @ keys

Original header

C++

```
std::string vrv::Toolkit::ValidatePAE(const std::string &data)
```

Example call

PYTHON

```
result = toolkit.validatePAE(data)
```

See also: Plaine and Easie

ValidatePAEFile

Validate the Plaine & Easie code from a file.

The method calls Toolkit::ValidatePAE.

> This method is not available in the JavaScript distributed version of the toolkit

Returns

std::string – A stringified JSON object with the validation warnings or errors

Parameters

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

Original header

C++

```
std::string vrv::Toolkit::ValidatePAEFile(const std::string &filename)
```

Example call

PYTHON

```
result = toolkit.validatePAEFile(filename)
```

See also: Plaine and Easie

Toolkit options

For the Python toolkit, options have to be passed asstringified JSON objects. For the JavaScript toolkit, they have to be passed as JSON objects directly.

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 only and are not used in the JavaScript or Python toolkits.

-a, --all-pages

Output all pages

-h, --help <string>

Display this message
(default: “”)

-f, --input-from <string>

Select input format from: "abc", "cmme.xml", "darms", "esac", "humdrum", "mei", "pae", "volpiano", "xml" (musicxml), "musicxml-hum" (musicxml via humdrum) or "mei-pb-serialized"
(default: “mei”)

See also: Input formats

-l, --log-level <string>

Set the log level: "off", "error", "warning", "info", or "debug"
(default: “warning”)

See also: Environment functions

-o, --outfile <string>

Output file name (use "-" as file name for standard output)
(default: “svg”)

-t, --output-to <string>

Select output format to: "mei", "mei-pb", "mei-facs", "mei-basic", "svg", "midi", "timemap", "expansionmap", "humdrum", "pae" or "mei-pb-serialized"
(default: “svg”)

See also: Output formats

-p, --page <integer>

Select the page to engrave (default is 1)

-r, --resource-path <string>

Path to the directory with Verovio resources
(default: “/usr/local/share/verovio”)

See also: SetResourcePath | Environment functions | Resources for versions built locally

---

-s, --scale <integer>

Scale of the output in percent (100 is normal size)
(default: 100; min: 1; max: 1000)

See also: Scaling

- , --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 <integer>

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

The option functions in a similar manner to --adjust--page-height . It shrinks the width of the music in the case where there is only a single system of music, and the music does not completely fill the full width specified by --page-width .

--breaks <string>

Define page and system breaks layout
(default: “auto”; other values: [‘none’, ‘auto’, ‘line’, ‘smart’, ‘encoded’])

See also: Output layout

--breaks-smart-sb <decimal>

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)

--condense <string>

Control condensed score layout
(default: “auto”; other values: [’none’, ‘auto’, ‘encoded’])

--condense-first-page

When condensing a score also condense the first page

--condense-not-last-system

When condensing a score never condense the last system

--condense-tempo-pages

When condensing a score also condense pages with a tempo change

--even-note-spacing

Align notes and rests without adding duration based space

--footer <string>

Control footer layout
(default: “auto”; other values: [’none’, ‘auto’, ‘encoded’, ‘always’])

--header <string>

Control header layout
(default: “auto”; other values: [’none’, ‘auto’, ‘encoded’])

--hum-type

Include type attributes when importing from Humdrum

--incip

Read <incip> elements as data input

--justify-vertically

Justify spacing vertically to fill the page

See also: Vertical justification

--landscape

Swap the values for page height and page width

--min-last-justification <decimal>

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)

--move-score-definition-to-staff

Move score definition (clef, keySig, meterSig, etc.) from scoreDef to staffDef

--neume-as-note

Render neumes as note heads instead of original notation

--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 <integer>

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 characters as XML entities instead of hex byte codes

--page-height <integer>

The page height
(default: 2970; min: 100; max: 60000)

See also: Controlling the SVG output

--page-margin-bottom <integer>

The page bottom margin
(default: 50; min: 0; max: 500)

--page-margin-left <integer>

The page left margin
(default: 50; min: 0; max: 500)

--page-margin-right <integer>

The page right margin
(default: 50; min: 0; max: 500)

--page-margin-top <integer>

The page top margin
(default: 50; min: 0; max: 500)

--page-width <integer>

The page width
(default: 2100; min: 100; max: 100000)

See also: Controlling the SVG output

--pedal-style <string>

The global pedal style
(default: “auto”; other values: [‘auto’, ‘line’, ‘pedstar’, ‘altpedstar’])

--preserve-analytical-markup

Preserves the analytical markup in MEI

--remove-ids

Remove XML IDs in the MEI output that are not referenced

--scale-to-page-size

Scale the content within the page instead of scaling the page itself

See also: Scaling

--set-locale

Changes the global locale to C (this is not thread-safe)

--show-runtime

Display the total runtime on command-line

--shrink-to-fit

Scale down page content to fit the page height if needed

--smufl-text-font <string>

Specify if the smufl text font is embedded, linked, or ignored
(default: “embedded”; other values: [‘embedded’, ‘linked’, ‘none’])

See also: Music symbols in text

--staccato-center

Align staccato and staccatissimo articulations with center of the note

--svg-additional-attribute <string> *

Add additional attribute for graphical elements in SVG as "data-*", for example, "note@pname" would add a "data-pname" to all note elements

--svg-bounding-boxes

Include bounding boxes in SVG output

--svg-content-bounding-boxes

Include content bounding boxes in SVG output

--svg-css <string>

CSS (as a string) to be added to the SVG output
(default: “”)

--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 <decimal>

The MEI unit (1⁄2 of the distance between the staff lines)
(default: 9.0; min: 4.5; max: 12.0)

See also: Units and page dimensions | Scaling

--use-brace-glyph

Use brace glyph from current font

--use-facsimile

Use information in the <facsimile> 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

--xml-id-checksum

Seed the generator for XML IDs using the checksum of the input data

General layout options

--bar-line-separation <decimal>

The default distance between multiple barlines when locked together
(default: 0.8; min: 0.5; max: 2.0)

--bar-line-width <decimal>

The barline width
(default: 0.3; min: 0.1; max: 0.8)

--beam-french-style

For notes in beams, stems will stop at first outermost sub-beam without crossing it

--beam-max-slope <integer>

The maximum beam slope
(default: 10; min: 0; max: 20)

--beam-mixed-preserve

Mixed beams will be drawn even if there is not enough space

--beam-mixed-stem-min <decimal>

The minimal stem length in MEI units used to draw mixed beams
(default: 3.5; min: 1.0; max: 8.0)

--bracket-thickness <decimal>

The thickness of the system bracket
(default: 1.0; min: 0.5; max: 2.0)

--breaks-no-widow

Prevent single measures on the last page by fitting it into previous system

--dashed-bar-line-dash-length <decimal>

The dash length of dashed barlines
(default: 1.14; min: 0.1; max: 5.0)

--dashed-bar-line-gap-length <decimal>

The gap length of dashed barlines
(default: 1.14; min: 0.1; max: 5.0)

--dynam-dist <decimal>

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

--dynam-single-glyphs

Don't use SMuFL's predefined dynamics glyph combinations

--engraving-defaults <string>

Json describing defaults for engraving SMuFL elements

--extender-line-min-space <decimal>

Minimum space required for extender line to be drawn
(default: 1.5; min: 1.5; max: 10.0)

--fingering-scale <decimal>

The scale of fingering font compared to default font size
(default: 0.75; min: 0.25; max: 1.0)

--font <string>

Set the music font
(default: “Leipzig”)

See also: SMuFL fonts

--font-add-custom <string> *

Add a custom music font as zip file

--font-fallback <string>

The music font fallback for missing glyphs
(default: “Leipzig”; other values: [‘Leipzig’, ‘Bravura’])

--font-load-all

Load all music fonts

--font-text-liberation

Use the Liberation text font

--grace-factor <decimal>

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 <decimal>

The hairpin size in MEI units
(default: 3.0; min: 1.0; max: 8.0)

--hairpin-thickness <decimal>

The thickness of the hairpin
(default: 0.2; min: 0.1; max: 0.8)

--handwritten-font <string> *

Fonts that emulate hand writing and require special handling

--harm-dist <decimal>

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

--justification-brace-group <decimal>

Space between staves inside a braced group justification
(default: 1.0; min: 0.0; max: 10.0)

See also: Vertical justification

--justification-bracket-group <decimal>

Space between staves inside a bracketed group justification
(default: 1.0; min: 0.0; max: 10.0)

See also: Vertical justification

--justification-max-vertical <decimal>

Maximum ratio of justifiable height to page height that can be used for the vertical justification
(default: 0.2; min: 0.0; max: 1.0)

--justification-staff <decimal>

The staff justification
(default: 1.0; min: 0.0; max: 10.0)

See also: Vertical justification

--justification-system <decimal>

The system spacing justification
(default: 1.0; min: 0.0; max: 10.0)

See also: Vertical justification

--ledger-line-extension <decimal>

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 <decimal>

The thickness of the ledger lines
(default: 0.25; min: 0.1; max: 0.5)

--lyric-elision <string>

The lyric elision width
(default: “regular”; other values: [‘unicode’, ‘narrow’, ‘regular’, ‘wide’])

--lyric-height-factor <decimal>

The lyric verse line height factor
(default: 1.0; min: 1.0; max: 20.0)

--lyric-line-thickness <decimal>

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 <decimal>

The lyrics size in MEI units
(default: 4.5; min: 2.0; max: 8.0)

--lyric-top-min-margin <decimal>

The minmal margin above the lyrics in MEI units
(default: 2.0; min: 0.0; max: 8.0)

--lyric-verse-collapse

Collapse empty verse lines in lyrics

--lyric-word-space <decimal>

The lyric word space length
(default: 1.2; min: 0.0; max: 10.0)

--measure-min-width <integer>

The minimal measure width in MEI units
(default: 15; min: 1; max: 30)

--mnum-interval <integer>

How frequently to place measure numbers

--multi-rest-style <string>

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

Description of the values:

- auto : changes to block style if the number of measures exceeds four. It takes the block attribute into account.
- default : same as auto, but ignoring the block attribute
- block : always displays block style except for single measure rests
- symbols : always display symbols except for large numbers of measures (>30)

--multi-rest-thickness <decimal>

The thickness of the multi rest in MEI units
(default: 2.0; min: 0.5; max: 6.0)

--octave-alternative-symbols

Use alternative symbols for displaying octaves

--octave-line-thickness <decimal>

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

--octave-no-spanning-parentheses

Do not enclose octaves that are spanning over systems with parentheses.

--ossia-staff-size <decimal>

The ossia staff size in relation to the staff size
(default: 0.5; min: 0.75; max: 1.0)

--pedal-line-thickness <decimal>

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

--repeat-bar-line-dot-separation <decimal>

The default horizontal distance between the dots and the inner barline of a repeat barline
(default: 0.36; min: 0.1; max: 1.0)

--repeat-ending-line-thickness <decimal>

Repeat and ending line thickness
(default: 0.15; min: 0.1; max: 2.0)

--slur-curve-factor <decimal>

Slur curve factor - high value means rounder slurs
(default: 1.0; min: 0.2; max: 5.0)

--slur-endpoint-flexibility <decimal>

Slur endpoint flexibility - allow more endpoint movement during adjustment
(default: 0.0; min: 0.0; max: 1.0)

--slur-endpoint-thickness <decimal>

The endpoint slur thickness in MEI units
(default: 0.1; min: 0.05; max: 0.25)

--slur-margin <decimal>

Slur safety distance in MEI units to obstacles
(default: 1.0; min: 0.1; max: 4.0)

--slur-max-slope <integer>

The maximum slur slope in degrees
(default: 60; min: 30; max: 85)

--slur-midpoint-thickness <decimal>

The midpoint slur thickness in MEI units
(default: 0.6; min: 0.2; max: 1.2)

--slur-symmetry <decimal>

Slur symmetry - high value means more symmetric slurs
(default: 0.0; min: 0.0; max: 1.0)

--spacing-brace-group <integer>

Minimum space between staves inside a braced group in MEI units
(default: 12; min: 0; max: 48)

--spacing-bracket-group <integer>

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 <decimal>

Specify the linear spacing factor
(default: 0.25; min: 0.0; max: 1.0)

See also: Content spacing

--spacing-non-linear <decimal>

Specify the non-linear spacing factor
(default: 0.6; min: 0.0; max: 1.0)

See also: Content spacing

--spacing-ossia <decimal>

Specify the factor of an ossia spacing in relation to staff spacing
(default: 0.35; min: 0.1; max: 1.0)

--spacing-staff <integer>

The staff minimal spacing in MEI units
(default: 12; min: 0; max: 48)

See also: Staff and system spacing

--spacing-system <integer>

The system minimal spacing in MEI units
(default: 4; min: 0; max: 48)

See also: Staff and system spacing

--staff-line-width <decimal>

The staff line width in MEI units
(default: 0.15; min: 0.1; max: 0.3)

--stem-width <decimal>

The stem width
(default: 0.2; min: 0.1; max: 0.5)

--sub-bracket-thickness <decimal>

The thickness of system sub-bracket
(default: 0.2; min: 0.1; max: 2.0)

--system-divider <string>

The display of system dividers
(default: “auto”; other values: [‘none’, ‘auto’, ‘left’, ‘left-right’])

--system-max-per-page <integer>

Maximum number of systems per page

--text-enclosure-thickness <decimal>

The thickness of the line text enclosing box
(default: 0.2; min: 0.1; max: 0.8)

--thick-barline-thickness <decimal>

The thickness of the thick barline
(default: 1.0; min: 0.5; max: 2.0)

--tie-endpoint-thickness <decimal>

The Endpoint tie thickness in MEI units
(default: 0.1; min: 0.05; max: 0.25)

--tie-midpoint-thickness <decimal>

The midpoint tie thickness in MEI units
(default: 0.5; min: 0.2; max: 1.0)

--tie-min-length <decimal>

The minimum length of tie in MEI units
(default: 2.0; min: 0.0; max: 10.0)

--tuplet-angled-on-beams

Tuplet brackets angled on beams only

--tuplet-bracket-thickness <decimal>

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 <string> *

Set the xPath query for selecting <app> child elements, for example: "./rdg[contains(@source, 'source-id')]"; by default the <lem> or the first <rdg> is selected

--choice-x-path-query <string> *

Set the xPath query for selecting <choice> child elements, for example: "./orig"; by default the first child is selected

--expand <string>

Expand all referenced elements in the expansion <xml:id>
(default: “”)

--expand-always

Expand for all outputs, using selected, first, or generated expansion

--expand-never

Expand for no output, including MIDI and timemap

--load-selected-mdiv-only

Load only the selected mdiv; the content of the other is skipped

--mdiv-all

Load and render all <mdiv> elements in the MEI files

--mdiv-x-path-query <string>

Set the xPath query for selecting the <mdiv> to be rendered; only one <mdiv> can be rendered
(default: “”)

--ossia-hidden

Hide ossias when rendering

--subst-x-path-query <string> *

Set the xPath query for selecting <subst> child elements, for example: "./del"; by default the first child is selected

--transpose <string>

Transpose the entire content
(default: “”)

See also: Transposition

--transpose-mdiv <string>

Json mapping the mdiv ids to the corresponding transposition

--transpose-selected-only

Transpose only the selected content and ignore unselected editorial content

By default, Verovio loads a single mdiv . However, transposition applies to the entire file loaded, i.e., to all mdiv elements. By setting --transpose-selected-only , only the selected mdiv will be transposed.

--transpose-to-sounding-pitch

Transpose to sounding pitch by evaluating @trans.semi

Element margins

--bottom-margin-artic <decimal>

The margin for artic in MEI units
(default: 0.75; min: 0.0; max: 10.0)

--bottom-margin-harm <decimal>

The margin for harm in MEI units
(default: 1.0; min: 0.0; max: 10.0)

--bottom-margin-header <decimal>

The margin for header in MEI units
(default: 2.0; min: 0.0; max: 24.0)

--bottom-margin-octave <decimal>

The margin for octave in MEI units
(default: 1.0; min: 0.0; max: 10.0)

--default-bottom-margin <decimal>

The default bottom margin
(default: 0.5; min: 0.0; max: 5.0)

--default-left-margin <decimal>

The default left margin
(default: 0.0; min: 0.0; max: 2.0)

--default-right-margin <decimal>

The default right margin
(default: 0.0; min: 0.0; max: 2.0)

--default-top-margin <decimal>

The default top margin
(default: 0.5; min: 0.0; max: 6.0)

--left-margin-accid <decimal>

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

--left-margin-bar-line <decimal>

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

--left-margin-beat-rpt <decimal>

The margin for beatRpt in MEI units
(default: 2.0; min: 0.0; max: 2.0)

--left-margin-chord <decimal>

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

--left-margin-clef <decimal>

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

--left-margin-key-sig <decimal>

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

--left-margin-left-bar-line <decimal>

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

--left-margin-m-rest <decimal>

The margin for mRest in MEI units
(default: 0.0; min: 0.0; max: 2.0)

--left-margin-m-rpt2 <decimal>

The margin for mRpt2 in MEI units
(default: 0.0; min: 0.0; max: 2.0)

--left-margin-mensur <decimal>

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

--left-margin-meter-sig <decimal>

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

--left-margin-multi-rest <decimal>

The margin for multiRest in MEI units
(default: 0.0; min: 0.0; max: 2.0)

--left-margin-multi-rpt <decimal>

The margin for multiRpt in MEI units
(default: 0.0; min: 0.0; max: 2.0)

--left-margin-note <decimal>

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

--left-margin-rest <decimal>

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

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

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

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

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

--right-margin-accid <decimal>

The right margin for accid in MEI units
(default: 0.5; min: 0.0; max: 2.0)

--right-margin-bar-line <decimal>

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

--right-margin-beat-rpt <decimal>

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

--right-margin-chord <decimal>

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

--right-margin-clef <decimal>

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

--right-margin-key-sig <decimal>

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

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

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

--right-margin-m-rest <decimal>

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

--right-margin-m-rpt2 <decimal>

The right margin for mRpt2 in MEI units
(default: 0.0; min: 0.0; max: 2.0)

--right-margin-mensur <decimal>

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

--right-margin-meter-sig <decimal>

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

--right-margin-multi-rest <decimal>

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

--right-margin-multi-rpt <decimal>

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

--right-margin-note <decimal>

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

--right-margin-rest <decimal>

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

--right-margin-right-bar-line <decimal>

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

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

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

--top-margin-artic <decimal>

The margin for artic in MEI units
(default: 0.75; min: 0.0; max: 10.0)

--top-margin-harm <decimal>

The margin for harm in MEI units
(default: 1.0; min: 0.0; max: 10.0)

--top-margin-pg-footer <decimal>

The margin for footer in MEI units
(default: 2.0; min: 0.0; max: 24.0)

Midi options

--midi-no-cue

Skip cue notes in MIDI output

--midi-tempo-adjustment <decimal>

The MIDI tempo adjustment factor
(default: 1.0; min: 0.2; max: 4.0)

Mensural options

--duration-equivalence <string>

The mensural duration equivalence
(default: “brevis”; other values: [‘brevis’, ‘semibrevis’, ‘minima’])

--ligature-as-bracket

Render ligatures as bracket instead of original notation

--ligature-oblique <string>

Ligature oblique shape
(default: “auto”; other values: [‘auto’, ‘straight’, ‘curved’])

--mensural-responsive-view

Convert mensural content to a more responsive view reduced to the seleceted markup

--mensural-score-up

Score up the mensural voices by providing a dur.quality to the notes

--mensural-to-cmn

Convert mensural sections to CMN measure-based MEI

MEI supported elements

> Note that, for the MEI attribute classes listed here, some attributes may not be implemented and that not all possible attribute values are supported.

<abbr>

att.labelled, att.source, att.typed

<accid>

att.accid.log, att.accidental, att.accidental.ges, att.color, att.coord, att.enclosingChars, att.extSym.auth, att.extSym.names, att.facsimile, att.labelled, att.linking, att.placementOnStaff, att.placementRelEvent, att.staffLoc, att.staffLoc.pitched, att.typed, att.visualOffsetHo, att.visualOffsetVo

<add>

att.labelled, att.source, att.typed

<anchoredText>

att.altSym, att.color, att.labelled, att.linking, att.placementRelStaff, att.typed, att.visualOffsetHo, att.visualOffsetVo

<annot>

att.altSym, att.color, att.labelled, att.linking, att.partIdent, att.plist, att.staffIdent, att.startEndId, att.startId, att.timestamp.log, att.timestamp.log, att.typed, att.visualOffsetHo, att.visualOffsetVo

<app>

att.labelled, att.typed

<arpeg>

att.altSym, att.arpeg.log, att.arpeg.vis, att.color, att.enclosingChars, att.labelled, att.linking, att.partIdent, att.plist, att.staffIdent, att.startId, att.timestamp.log, att.typed, att.visualOffsetHo, att.visualOffsetVo

<artic>

att.articulation, att.articulation.ges, att.color, att.coord, att.enclosingChars, att.extSym.auth, att.extSym.names, att.facsimile, att.labelled, att.linking, att.placementRelEvent, att.typed, att.visualOffsetHo, att.visualOffsetVo

<bTrem>

att.coord, att.facsimile, att.labelled, att.linking, att.numberPlacement, att.numbered, att.tremForm, att.tremMeasured, att.typed

<barLine>

att.barLine.log, att.barLine.vis, att.color, att.coord, att.facsimile, att.labelled, att.linking, att.nNumberLike, att.typed, att.visibility

<beam>

att.beamRend, att.beamedWith, att.color, att.coord, att.cue, att.facsimile, att.labelled, att.linking, att.typed

<beamSpan>

att.altSym, att.beamRend, att.beamedWith, att.color, att.labelled, att.linking, att.partIdent, att.plist, att.staffIdent, att.startEndId, att.startId, att.timestamp.log, att.timestamp.log, att.typed, att.visualOffsetHo, att.visualOffsetVo

<beatRpt>

att.beatRpt.log, att.beatRpt.vis, att.color, att.coord, att.facsimile, att.labelled, att.linking, att.typed

<bracketSpan>

att.altSym, att.bracketSpan.log, att.color, att.labelled, att.lineRend, att.lineRend.base, att.linking, att.partIdent, att.staffIdent, att.startEndId, att.startId, att.timestamp.log, att.timestamp.log, att.typed, att.visualOffsetHo, att.visualOffsetVo

<breath>

att.altSym, att.color, att.labelled, att.linking, att.partIdent, att.placementRelStaff, att.staffIdent, att.startId, att.timestamp.log, att.typed, att.visualOffsetHo, att.visualOffsetVo

<caesura>

att.altSym, att.color, att.extSym.auth, att.extSym.names, att.labelled, att.linking, att.partIdent, att.placementRelStaff, att.staffIdent, att.startId, att.timestamp.log, att.typed, att.visualOffsetHo, att.visualOffsetVo

<choice>

att.labelled, att.typed

<chord>

att.augmentDots, att.beamSecondary, att.chord.vis, att.color, att.coord, att.cue, att.duration.ges, att.duration.log, att.duration.quality, att.duration.ratio, att.facsimile, att.fermataPresent, att.graced, att.labelled, att.linking, att.staffIdent, att.stems, att.stems.cmn, att.tiePresent, att.typed, att.visibility

<clef>

att.clef.log, att.clefShape, att.color, att.coord, att.enclosingChars, att.extSym.auth, att.extSym.names, att.facsimile, att.labelled, att.lineLoc, att.linking, att.octave, att.octaveDisplacement, att.staffIdent, att.typed, att.typography, att.visibility, att.visualOffsetHo, att.visualOffsetVo

<corr>

att.labelled, att.source, att.typed

<course>

att.accidental, att.nNumberLike, att.octave, att.pitch

<cpMark>

att.altSym, att.color, att.labelled, att.linking, att.partIdent, att.placementRelStaff, att.staffIdent, att.startEndId, att.startId, att.timestamp.log, att.timestamp.log, att.typed, att.visualOffsetHo, att.visualOffsetVo

<custos>

att.color, att.coord, att.extSym.auth, att.extSym.names, att.facsimile, att.labelled, att.linking, att.note.ges, att.octave, att.pitch, att.pitch.ges, att.staffLoc, att.staffLoc.pitched, att.typed, att.visualOffsetHo, att.visualOffsetVo

<damage>

att.labelled, att.source, att.typed

<del>

att.labelled, att.source, att.typed

<dir>

att.altSym, att.color, att.extender, att.labelled, att.lang, att.lineRend.base, att.linking, att.partIdent, att.placementRelStaff, att.staffIdent, att.startEndId, att.startId, att.timestamp.log, att.timestamp.log, att.typed, att.verticalGroup, att.visualOffsetHo, att.visualOffsetVo

att.typed

<divLine>

att.color, att.coord, att.divLine.log, att.extSym.auth, att.extSym.names, att.facsimile, att.labelled, att.linking, att.nNumberLike, att.typed, att.visibility, att.visualOffsetHo, att.visualOffsetVo

<dot>

att.color, att.coord, att.dot.log, att.facsimile, att.labelled, att.linking, att.staffLoc, att.staffLoc.pitched, att.typed, att.visualOffsetHo, att.visualOffsetVo

<dynam>

att.altSym, att.color, att.enclosingChars, att.extender, att.labelled, att.lineRend.base, att.linking, att.midiValue, att.midiValue, att.partIdent, att.placementRelStaff, att.staffIdent, att.startEndId, att.startId, att.timestamp.log, att.timestamp.log, att.typed, att.verticalGroup, att.visualOffsetHo, att.visualOffsetVo

<ending>

att.labelled, att.lineRend, att.lineRend.base, att.nNumberLike, att.typed

<expan>

att.labelled, att.source, att.typed

<expansion>

att.plist, att.typed

<f>

att.extender, att.labelled, att.partIdent, att.staffIdent, att.startEndId, att.startId, att.timestamp.log, att.timestamp.log, att.typed

<fTrem>

att.coord, att.fTrem.vis, att.facsimile, att.labelled, att.linking, att.tremMeasured, att.typed

<facsimile>

att.typed

<fb>

<fermata>

att.altSym, att.color, att.enclosingChars, att.extSym.auth, att.extSym.names, att.fermata.vis, att.labelled, att.linking, att.partIdent, att.placementRelStaff, att.staffIdent, att.startId, att.timestamp.log, att.typed, att.visualOffsetHo, att.visualOffsetVo

<fig>

att.horizontalAlign, att.labelled, att.typed, att.verticalAlign

<fing>

att.altSym, att.color, att.labelled, att.linking, att.nNumberLike, att.partIdent, att.placementRelStaff, att.staffIdent, att.startId, att.timestamp.log, att.typed, att.visualOffsetHo, att.visualOffsetVo

<gliss>

att.altSym, att.color, att.labelled, att.lineRend, att.lineRend.base, att.linking, att.nNumberLike, att.partIdent, att.staffIdent, att.startEndId, att.startId, att.timestamp.log, att.timestamp.log, att.typed, att.visualOffsetHo, att.visualOffsetVo

<graceGrp>

att.color, att.coord, att.facsimile, att.graceGrp.log, att.graced, att.labelled, att.linking, att.typed

<graphic>

att.height, att.pointing, att.typed, att.width

<grpSym>

att.color, att.grpSym.log, att.staffGroupingSym, att.startEndId, att.startId

<hairpin>

att.altSym, att.color, att.hairpin.log, att.hairpin.vis, att.labelled, att.lineRend.base, att.linking, att.partIdent, att.placementRelStaff, att.staffIdent, att.startEndId, att.startId, att.timestamp.log, att.timestamp.log, att.typed, att.verticalGroup, att.visualOffsetHo, att.visualOffsetVo, att.visualOffsetHo, att.visualOffsetVo

<halfmRpt>

att.color, att.coord, att.facsimile, att.labelled, att.linking, att.typed, att.visualOffsetHo, att.visualOffsetVo

<harm>

att.altSym, att.color, att.labelled, att.lang, att.linking, att.nNumberLike, att.partIdent, att.placementRelStaff, att.staffIdent, att.startEndId, att.startId, att.timestamp.log, att.timestamp.log, att.typed, att.visualOffsetHo, att.visualOffsetVo

<instrDef>

att.channelized, att.labelled, att.midiInstrument, att.nNumberLike

<keyAccid>

att.accidental, att.color, att.coord, att.enclosingChars, att.extSym.auth, att.extSym.names, att.facsimile, att.labelled, att.linking, att.note.ges, att.octave, att.pitch, att.pitch.ges, att.staffLoc, att.staffLoc.pitched, att.typed

<keySig>

att.accidental, att.color, att.coord, att.facsimile, att.keyMode, att.keySig.log, att.keySig.vis, att.labelled, att.linking, att.pitch, att.typed, att.visibility

<label>

<labelAbbr>

<layer>

att.cue, att.nInteger, att.typed, att.visibility

<layerDef>

att.labelled, att.nInteger, att.typed

<lb>

att.labelled, att.typed

<lem>

att.labelled, att.source, att.typed

<ligature>

att.coord, att.facsimile, att.labelled, att.ligature.vis, att.linking, att.typed

<liquescent>

att.color, att.coord, att.facsimile, att.labelled, att.linking, att.note.ges, att.octave, att.pitch, att.pitch.ges, att.staffLoc, att.staffLoc.pitched, att.typed, att.visualOffsetHo, att.visualOffsetVo

<lv>

att.altSym, att.color, att.curvature, att.labelled, att.lineRend.base, att.linking, att.partIdent, att.staffIdent, att.startEndId, att.startId, att.timestamp.log, att.timestamp.log, att.typed, att.visualOffsetHo, att.visualOffsetVo, att.visualOffsetHo, att.visualOffsetVo

<mNum>

att.altSym, att.color, att.labelled, att.lang, att.linking, att.partIdent, att.placementRelStaff, att.staffIdent, att.startId, att.timestamp.log, att.typed, att.typography, att.visualOffsetHo, att.visualOffsetVo

<mRest>

att.color, att.coord, att.cue, att.cutout, att.facsimile, att.fermataPresent, att.labelled, att.linking, att.staffLoc, att.staffLoc.pitched, att.typed, att.visibility, att.visualOffsetHo, att.visualOffsetVo

<mRpt>

att.color, att.coord, att.facsimile, att.labelled, att.linking, att.numberPlacement, att.numbered, att.typed

<mRpt2>

att.color, att.coord, att.facsimile, att.labelled, att.linking, att.typed

<mSpace>

att.coord, att.facsimile, att.labelled, att.linking, att.typed

<mdiv>

att.labelled, att.nNumberLike, att.typed

<measure>

att.barring, att.coord, att.coord, att.facsimile, att.measure.log, att.meterConformanceBar, att.nNumberLike, att.pointing, att.typed

<mensur>

att.color, att.coord, att.cue, att.duration.ratio, att.facsimile, att.labelled, att.linking, att.mensur.vis, att.mensuralShared, att.slashCount, att.staffLoc, att.typed

<meterSig>

att.color, att.coord, att.enclosingChars, att.extSym.names, att.facsimile, att.labelled, att.linking, att.meterSig.log, att.meterSig.vis, att.typed, att.typography, att.visibility

<meterSigGrp>

att.basic, att.coord, att.facsimile, att.labelled, att.linking, att.meterSigGrp.log, att.typed, att.visibility

<mordent>

att.altSym, att.color, att.enclosingChars, att.extSym.auth, att.extSym.names, att.labelled, att.linking, att.mordent.log, att.ornamentAccid, att.partIdent, att.placementRelStaff, att.staffIdent, att.startId, att.timestamp.log, att.typed, att.visualOffsetHo, att.visualOffsetVo

<multiRest>

att.color, att.coord, att.facsimile, att.labelled, att.linking, att.multiRest.vis, att.numberPlacement, att.numbered , att.staffLoc, att.staffLoc.pitched, att.typed, att.width

<multiRpt>

att.coord, att.facsimile, att.labelled, att.linking, att.numbered, att.typed

<nc>

att.augmentDots, att.beamSecondary, att.color, att.coord, att.curvatureDirection, att.duration.ges, att.duration.log, att.duration.quality, att.duration.ratio, att.facsimile, att.fermataPresent, att.intervalMelodic, att.labelled, att.linking, att.ncForm, att.note.ges, att.octave, att.pitch, att.pitch.ges, att.staffIdent, att.staffLoc, att.staffLoc.pitched, att.typed, att.visualOffsetHo, att.visualOffsetVo

<neume>

att.color, att.coord, att.facsimile, att.labelled, att.linking, att.typed, att.visualOffsetHo, att.visualOffsetVo

<note>

att.altSym, att.augmentDots, att.beamSecondary, att.color, att.coloration, att.coord, att.cue, att.duration.ges, att.duration.log, att.duration.quality, att.duration.ratio, att.extSym.auth, att.extSym.names, att.facsimile, att.fermataPresent, att.graced, att.harmonicFunction, att.labelled, att.linking, att.midiVelocity, att.note.ges, att.noteHeads, att.noteVisMensural, att.octave, att.pitch, att.pitch.ges, att.staffIdent, att.staffLoc, att.staffLoc.pitched, att.stems, att.stems.cmn, att.stringtab, att.tiePresent, att.typed, att.visibility, att.visualOffsetHo, att.visualOffsetVo

<num>

att.labelled, att.typed

<octave>

att.altSym, att.color, att.extender, att.labelled, att.lineRend, att.lineRend.base, att.linking, att.nNumberLike, att.octaveDisplacement, att.partIdent, att.staffIdent, att.startEndId, att.startId, att.timestamp.log, att.timestamp.log, att.typed, att.visualOffsetHo, att.visualOffsetVo

<orig>

att.labelled, att.source, att.typed

<oriscus>

att.color, att.coord, att.facsimile, att.labelled, att.linking, att.note.ges, att.octave, att.pitch, att.pitch.ges, att.staffLoc, att.staffLoc.pitched, att.typed, att.visualOffsetHo, att.visualOffsetVo

<ornam>

att.altSym, att.color, att.labelled, att.linking, att.ornamentAccid, att.partIdent, att.placementRelStaff, att.staffIdent, att.startId, att.timestamp.log, att.typed, att.visualOffsetHo, att.visualOffsetVo

<ossia>

att.typed

<pb>

att.facsimile, att.nNumberLike, att.typed

<pedal>

att.altSym, att.color, att.extSym.auth, att.extSym.names, att.labelled, att.linking, att.partIdent, att.pedal.log, att.pedal.vis, att.placementRelStaff, att.staffIdent, att.startEndId, att.startId, att.timestamp.log, att.timestamp.log, att.typed, att.verticalGroup, att.visualOffsetHo, att.visualOffsetVo

<pgFoot>

att.formework, att.typed

<pgFoot2>

att.horizontalAlign, att.typed

<pgHead>

att.formework, att.typed

<pgHead2>

att.horizontalAlign, att.typed

<phrase>

att.altSym, att.color, att.curvature, att.labelled, att.layerIdent, att.lineRend.base, att.linking, att.partIdent, att.staffIdent, att.startEndId, att.startId, att.timestamp.log, att.timestamp.log, att.typed, att.visualOffsetHo, att.visualOffsetVo, att.visualOffsetHo, att.visualOffsetVo

<plica>

att.coord, att.facsimile, att.labelled, att.linking, att.plica.vis, att.typed

<proport>

att.coord, att.duration.ratio, att.facsimile, att.labelled, att.linking, att.typed

<quilisma>

att.color, att.coord, att.facsimile, att.labelled, att.linking, att.note.ges, att.octave, att.pitch, att.pitch.ges, att.staffLoc, att.staffLoc.pitched, att.typed, att.visualOffsetHo, att.visualOffsetVo

<rdg>

att.labelled, att.source, att.typed

<ref>

att.labelled, att.typed

<reg>

att.labelled, att.source, att.typed

<reh>

att.altSym, att.color, att.labelled, att.lang, att.linking, att.partIdent, att.placementRelStaff, att.staffIdent, att.startId, att.timestamp.log, att.typed, att.verticalGroup, att.visualOffsetHo, att.visualOffsetVo

<rend>

att.color, att.extSym.auth, att.horizontalAlign, att.labelled, att.lang, att.nNumberLike, att.textRendition, att.typed, att.typography, att.verticalAlign, att.whitespace

<repeatMark>

att.altSym, att.color, att.extSym.auth, att.extSym.names, att.labelled, att.linking, att.partIdent, att.placementRelStaff, att.repeatMark.log, att.staffIdent, att.startId, att.timestamp.log, att.typed, att.visualOffsetHo, att.visualOffsetVo

<rest>

att.altSym, att.augmentDots, att.beamSecondary, att.color, att.coord, att.cue, att.duration.ges, att.duration.log, att.duration.quality, att.duration.ratio, att.enclosingChars, att.extSym.auth, att.extSym.names , att.facsimile, att.fermataPresent, att.labelled, att.linking, att.restVisMensural, att.staffIdent, att.staffLoc, att.staffLoc.pitched, att.typed, att.visualOffsetHo, att.visualOffsetVo

<restore>

att.labelled, att.source, att.typed

<sb>

att.facsimile, att.nNumberLike, att.typed

<score>

att.labelled, att.nNumberLike, att.typed

<scoreDef>

att.barring, att.distances, att.durationDefault, att.endings, att.lyricStyle, att.measureNumbers, att.midiTempo, att.mmTempo, att.multinumMeasures, att.octaveDefault, att.optimization, att.pianoPedals, att.spacing, att.systems, att.timeBase, att.tuning, att.typed

<section>

att.nNumberLike, att.section.vis, att.typed

<sic>

att.labelled, att.source, att.typed

<slur>

att.altSym, att.color, att.curvature, att.labelled, att.layerIdent, att.lineRend.base, att.linking, att.partIdent, att.staffIdent, att.startEndId, att.startId, att.timestamp.log, att.timestamp.log, att.typed, att.visualOffsetHo, att.visualOffsetVo, att.visualOffsetHo, att.visualOffsetVo

<space>

att.augmentDots, att.beamSecondary, att.coord, att.duration.ges, att.duration.log, att.duration.quality, att.duration.ratio, att.facsimile, att.fermataPresent, att.labelled, att.linking, att.staffIdent, att.typed

<staff>

att.coord, att.facsimile, att.nInteger, att.typed, att.visibility

<staffDef>

att.barring, att.distances, att.durationDefault, att.labelled, att.lyricStyle, att.measureNumbers, att.midiTempo, att.mmTempo, att.multinumMeasures, att.nInteger, att.notationType, att.octaveDefault, att.pianoPedals, att.scalable, att.spacing, att.staffDef.log, att.staffDef.vis, att.staffDefVisTablature, att.systems, att.timeBase, att.transposition, att.typed

<staffGrp>

att.barring, att.basic, att.labelled, att.nNumberLike, att.staffGroupingSym, att.staffGrp.vis, att.typed

<subst>

att.labelled, att.typed

<supplied>

att.labelled, att.source, att.typed

<surface>

att.coordinated, att.coordinatedUl, att.typed

<svg>

<syl>

att.coord, att.facsimile, att.labelled, att.lang, att.linking, att.partIdent, att.staffIdent, att.startEndId, att.startId, att.syl.log, att.timestamp.log, att.timestamp.log, att.typed, att.typography, att.visualOffsetHo, att.visualOffsetVo

<syllable>

att.color, att.coord, att.facsimile, att.labelled, att.linking, att.slashCount, att.typed

<symbol>

att.color, att.extSym.auth, att.extSym.names, att.labelled, att.typed, att.typography

<symbolDef>

<symbolTable>

<tabDurSym>

att.coord, att.facsimile, att.labelled, att.linking, att.nNumberLike, att.stringtab, att.typed, att.visualOffsetVo

<tabGrp>

att.augmentDots, att.beamSecondary, att.coord, att.duration.ges, att.duration.log, att.duration.quality, att.duration.ratio, att.facsimile, att.fermataPresent, att.labelled, att.linking, att.staffIdent, att.typed, att.visualOffsetHo, att.visualOffsetVo

<tempo>

att.altSym, att.color, att.extender, att.labelled, att.lang, att.linking, att.midiTempo, att.mmTempo, att.partIdent, att.placementRelStaff, att.staffIdent, att.startEndId, att.startId, att.timestamp.log, att.timestamp.log, att.typed, att.visualOffsetHo, att.visualOffsetVo

<tie>

att.altSym, att.color, att.curvature, att.labelled, att.lineRend.base, att.linking, att.partIdent, att.staffIdent, att.startEndId, att.startId, att.timestamp.log, att.timestamp.log, att.typed, att.visualOffsetHo, att.visualOffsetVo, att.visualOffsetHo, att.visualOffsetVo

<trill>

att.altSym, att.color, att.enclosingChars, att.extSym.auth, att.extSym.names, att.extender, att.labelled, att.lineRend, att.linking, att.nNumberLike, att.ornamentAccid, att.partIdent, att.placementRelStaff,

att.staffIdent, att.startEndId, att.startId, att.timestamp.log, att.timestamp.log, att.typed, att.visualOffsetHo, att.visualOffsetVo

<tuning>

att.tuning.log

<tuplet>

att.color, att.coord, att.duration.ratio, att.facsimile, att.labelled, att.linking, att.numberPlacement, att.tuplet.vis, att.typed

<turn>

att.altSym, att.color, att.enclosingChars, att.extSym.auth, att.extSym.names, att.labelled, att.linking, att.ornamentAccid, att.partIdent, att.placementRelStaff, att.staffIdent, att.startId, att.timestamp.log, att.turn.log, att.typed, att.visualOffsetHo, att.visualOffsetVo

<unclear>

att.labelled, att.source, att.typed

<verse>

att.color, att.coord, att.facsimile, att.labelled, att.lang, att.linking, att.nInteger, att.placementRelStaff, att.typed , att.typography

<zone>

att.coordinated, att.coordinatedUl, att.typed

Environment functions

Verovio includes a few environment-level functions for configuring the global setup in which the toolkit runs. They are namespace-level functions in the C++ codebase. For the Python and JavaScript bindings, they are module-level functions.

SetDefaultResourcePath

Specify the path where the resources are located.

> This method is not available in the JavaScript distributed version of the toolkit

Returns

void

Parameters

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

Original header

C++

```
void vrv::SetDefaultResourcePath(const std::string &path)
```

Example call

PYTHON

```
verovio.setDefaultResourcePath(path)
```

EnableLog

Returns

void

Parameters

Name	Type	Default	Description
level	LogLevel	∅	

Original header

C++

```
void vrv::EnableLog(LogLevel level);
```

Example call

PYTHON

```
verovio.enableLog(level)
```

LogLevel

The LogLevel enum includes the following values:

- LOG_OFF : no log

- LOG_DEBUG : log all messages, including debug ones
- LOG_INFO : log all messages
- LOG_WARNING : log error and warning messages (default)
- LOG_ERROR : log error messages only

For both the Python and the JavaScript bindings, the values are available in the modules. This means that the log level can be changed with:

PYTHON

```
import verovio
verovio.enableLog(verovio.LOG_ERROR)
```

HTML / JAVASCRIPT

```
<script>
  document.addEventListener("DOMContentLoaded", (event) => {
      verovio.module.onRuntimeInitialized = () => {
        verovio.enableLog(verovio.LOG_ERROR);
      }
  });
</script>
```

EnableLogToBuffer

Redirect the log messages to a buffer instead of the `std::err` or the console. The messages can be accessed from the toolkit instance via the `GetLog()` method.

Returns

void

Parameters

Name	Type	Default	Description
value	bool	∅	

Original header

C++

```
void vrv::EnableLogToBuffer(bool value);
```

Example call

PYTHON

```
verovio.enableLogToBuffer(value)
```

Installing or building from sources

Command-line version

The Verovio codebase is C++17 compliant and is cross-platform. It has been tested on several operating systems and architectures. This section describes how to build and install the command-line version of the toolkit from the command-line or using some of the most popular IDEs. There are currently no pre-built binaries of the command-line toolkit available except for Homebrew on macOS. However, building it is very straight-forward.

Homebrew on macOS

For macOS users using Homebrew, the command-line version can be installed with:

TERMINAL

```
brew install verovio
```

This also installs the resources and you will be ready to go.

You can also install the latest development source with:

TERMINAL

```
brew install verovio --HEAD
```

Building on macOS or Linux

To build 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

```
mkdir -p build
cd build
cmake ../cmake
make
```

You can increase the building speed by using the `-j` option when running make that specifies the number of jobs to be run in parallel:

TERMINAL

```
make -j 8
```

The generates a `verovio` binary within `./build`. 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 `./build` 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.

To see the current default resource path, look for the “resource path” section in the full help output. You should see something like this:

TERMINAL

```
verovio -h full
...
-r, --resource-path <s>        Path to the directory with Verovio resources (default: "/usr/local/share/verovio"
)
```

Keep in mind that if you have installed, you should not run another version without re-installing it or using the `-r` options to point to a non-default path, 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.

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

Basic usage

To seeing the basic command-line options, run:

TERMINAL

```
verovio --help
```

To see all command-line options, run:

TERMINAL

```
verovio -h full
```

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
```

To build Verovio without Plaine 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_MXL_SUPPORT` for the compressed MusicXML 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

If you have installed Verovio with Homebrew, run:

TERMINAL

```
brew uninstall verovio
```

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
```

Troubleshooting

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
```

Or, when using CMake 3.24 or later, you can simply run:

TERMINAL

```
cd tools
cmake ../cmake --fresh
```

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

Visual Studio Code

Verovio contains simple predefined build tasks in the `tasks.json` file.

You can build Verovio by pressing `Ctrl+Shift+B` / `⇧⌘B` or running **Run Build Task** from the global **Terminal** menu.

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. The recommended version is one built as WebAssembly because it is the fastest and supported by all recent browsers. To use it, the file you need to include is:

verovio-toolkit-wasm.js

If you need Humdrum support, the file to include is:

verovio-toolkit-hum.js

If you need to have support for old browsers, there is an `asm.js` version available. This version is obsolete and is not recommended for new projects. The file to include is:

verovio-toolkit.js

A build for the development version of the `verovio-toolkit-wasm.js` is available through CI, as well as for each release for the Humdrum and the legacy `asm.js` version.

The latest release is always available from:

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

The latest development version is available from:

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

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

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

For instructions on a basic usage of the JavaScript version of the toolkit, see the Getting started section of the Tutorial 1: First steps chapter.

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 with NPM

JAVASCRIPT

```
const verovio = require('verovio');
const fs = require('fs');

/* Wait for verovio to load */
verovio.module.onRuntimeInitialized = function ()
{
    // create the toolkit instance
    const vrvToolkit = new verovio.toolkit();
    // read the MEI file
    mei = fs.readFileSync('hello.mei');
    // load the MEI data as string into the toolkit
    vrvToolkit.loadData(mei.toString());
    // render the fist page as SVG
    svg = vrvToolkit.renderToSVG(1, {});
    // save the SVG into a file
    fs.writeFileSync('hello.svg', svg);
}
```

Usage with ESM

Since version 3.11.0 there is an ESM compatible version of the *npm* package with a modularized build of the Verovio module. This is because we need to wait for the asynchronous module to be ready for usage, and this is now Promise based instead of using the `onRuntimeInitialized` callback function.

Use `.mjs` as file extension when using this directly in Node.js or set `"type": "module"` in your `package.json`.

JAVASCRIPT

```
import createVerovioModule from 'verovio/wasm';
import { VerovioToolkit } from 'verovio/esm';
import fs from 'node:fs';

createVerovioModule().then(VerovioModule => {
  const verovioToolkit = new VerovioToolkit(VerovioModule);
  const score = fs.readFileSync('hello.mei').toString();
  verovioToolkit.loadData(score);
  const data = verovioToolkit.renderToSVG(1, {});
  console.log(data);
});
```

This is the recommended way to use Verovio when creating a website or web app with bundlers like webpack or Vite or when using JavaScript frameworks like React or Vue.js.

Usage with CommonJS

Alternatively this package also exports a version compatible with CommonJS

JAVASCRIPT

```
const createVerovioModule = require('verovio/wasm');
const { VerovioToolkit } = require('verovio/esm');
```

Humdrum support

Since version 3.11.0 the NPM package provides an additional module with Humdrum support:

JAVASCRIPT

```
import createVerovioModule from 'verovio/wasm-hum';
```

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:

TERMINAL

```
./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 3.9, 3.10, 3.11, 3.12 and 3.13. The platforms supported are macOS, 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 the resource path

The Python wheels include the resource directory and it is normally resolved by default when using the module. However, in some cases, it might be not found. This will typically raise some errors about the Bravura and the Leipzig default fonts being not found. In such cases, the resource path must be set explicitly. It should still be possible to use the resources included in the module, with:

PYTHON

```
import verovio
import os

tk = verovio.toolkit(False);
tk.setResourcePath(os.path.join(os.path.dirname(verovio.__file__),"data"));
```

Setting options

The options are set on the toolkit instance. For the Python version of the toolkit, the options (and all other parameters or values return by a function that are a JSON string in the C++ version) are a Python Dictionary. For example, the following code will change the dimensions of the page and redo the layout for the previously loaded data:

PYTHON

```
options = { "pageHeight": 2100, "pageWidth": 2950, "scale": 25 }
tk.setOptions(options)
tk.redoLayout()
tk.renderToSVGFile( "page-scaled.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 Python toolkit can be built with CMake, 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
```

By default, Python 3 is used. If you want to use a specific version of Python, you can do:

TERMINAL

```
cmake ../cmake -B python -DBUILD_AS_PYTHON=ON -DPYTHON_VERSION=3.9
```

Installation with CMake has not be tested yet

Building the toolkit without CMake

The toolkit can be build without CMake. However, SWIG is still needed. It 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 Plaine & Easie support), run:

TERMINAL

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

Building with pip

You can build and install with `pip` directly from a remote repository with:

TERMINAL

```
pip install --global-option=build_ext git+https://github.com/rism-digital/verovio
```

You may specify the branch, commit hash, or tag name after an `@` at the end of the Git url.

If you have a local copy of the repository just run:

TERMINAL

```
pip install <path_to_local_repo>
```

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.

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()
```

Java and Android

Java

To build the Java toolkit you need to have swig and swig-java installed on your machine (seeSWIG) 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.

Android

The simplest way to use Verovio in Android is to use the Java bindings provided by Verovio. You can have it generated and compiled directly in Android Studio. The sample application repository provides a complete example on how to use it, with Verovio included as a submodule in `external/verovio` . You need swig to be installed on your machine.

On Android, it is recommended to use the Liberation font since Times is not available by default. This can be enabled with the `--font-text-liberation` option.

The `app/build.gradle.kts` includes a step to generate the swig binding for Java and to build the toolkit:

KOTLIN

```
////////////////////////////////////////
// Generate the java and cpp files using swig and write them into the project
val swigOutputJava = file("src/main/java/org/verovio/lib")
val swigOutputCpp = file("src/main/cpp/verovio_wrap.cxx")
val swigInterfaceFile = file("${rootDir.absolutePath}/external/verovio/bindings/java/verovio.i")

tasks.register<Exec>("generateSwigBindings") {
    group = "build"
    description = "Generate JNI bindings with SWIG"

    // Adjust working directory to the project root
    workingDir = rootProject.projectDir

    // Ensure output directories exist
    doFirst {
        swigOutputJava.mkdirs()
        swigOutputCpp.parentFile.mkdirs()
    }

    commandLine = listOf(
        "/opt/homebrew/bin/swig", // Change to "/bin/swig" or something else if needed
        "-java",
        "-c++",
        "-package", "org.verovio.lib",
        "-outdir", swigOutputJava.absolutePath,
        "-o", swigOutputCpp.absolutePath,
        swigInterfaceFile.absolutePath
    )
}

// Ensure SWIG runs before compilation
tasks.named("preBuild") {
    dependsOn("generateSwigBindings")
}

tasks.named("clean") {
    doFirst {
        delete("src/main/java/org/verovio/lib/*")
        delete("src/main/cpp/verovio_wrap.cxx")
    }
}
```

The file also includes a step to copy the resource directory:

KOTLIN

```
////////////////////////////////////////
// Copy the verovio resource directory from the submodule to the project
tasks.register<Copy>("copyVerovioData") {
    from(rootDir.resolve("external/verovio/data"))
    into("src/main/assets/verovio/data")
}

tasks.named("preBuild") {
    dependsOn("copyVerovioData")
}
```

Swift and Objective-C

Swift

The Swift binding of Verovio can easily be install with the Swift Package Manager. In xcode, this can be done with adding a package dependency (e.g., through **File > Add Package Dependency…**) and by providing the Verovio GitHub repository. You might need to add an empty `Bridging-Header.h` file to your project.

Once this is done, Verovio is available in Swift with:

SWIFT

```
import VerovioToolkit

var toolkit = VerovioToolkit()
```

Note that the constructor in the Swift binding does not set the resource path. You need to do it explicitly by calling `toolkit.setResourcePath` . The nice thing, however, is that the Swift binding provides a `VerovioResources.bundle` with the Verovio resources. So setting the resources will look like:

SWIFT

```
let bundle = VerovioResources.bundle

if let resourceURL = bundle.url(forResource: "data", withExtension: nil) {
    let resourcePath = resourceURL.deletingLastPathComponent().path
    let _ = toolkit.setResourcePath(resourcePath + "/data")
} else {
    print("Could not find resource URL for 'data'")
}
```

Loading some data and rendering will work with the usual methods of the toolkit:

SWIFT

```
let res = toolkit.loadData(data)
let svg = toolkit.renderToSVG(1, false)
```

Objective-C with CocoaPods

The simplest way to use Verovio in Objective-C is to use CocoaPods to install Verovio by adding it to your Podfile :

```
platform :ios, '16.0'
use_frameworks!
target 'MyApp' do
 pod 'Verovio', :git => 'https://github.com/rism-digital/verovio.git', :branch => 'master'
end
```

Then, run the following command:

TERMINAL

```
pod install
```

To use Verovio in your iOS project import

OBJECTIVE-C

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

Then you can create an Objective-C wrapper with the following VerovioToolkitWrapper.h/mm files (here with a few sample methods):

OBJECTIVE-C

```
#import <Foundation/Foundation.h>

NS_ASSUME_NONNULL_BEGIN

@interface VerovioToolkitWrapper : NSObject

- (instancetype)init;
- (NSInteger)getPageCount;
- (NSString *)getVersion;
- (BOOL)loadData:(NSString *)data;
- (void)redoLayout;
- (NSString *)renderToSVG:(NSInteger)pageNo;
- (void)setOptions:(nullable NSString *)jsonOptions;

@end

NS_ASSUME_NONNULL_END
```

OBJECTIVE-C

```
#import "VerovioToolkitWrapper.h"

// Include the umbrella header
#import <Verovio/Verovio-umbrella.h>

// Include the C++ Verovio header
#include <verovio/toolkit.h>


@implementation VerovioToolkitWrapper {
    std::unique_ptr<vrv::Toolkit> toolkit;
}

- (instancetype)init {
    self = [super init];
    if (self) {
        NSBundle *verovioBundle = [NSBundle bundleWithIdentifier:@"digital.rism.VerovioFramework"];
        //NSLog(@"Bundle path: %@", verovioBundle.bundlePath);
        NSString *resourcePath = [verovioBundle URLsForResourcesWithExtension:@"xml"
            subdirectory:@"data"]
            .firstObject.URLByDeletingLastPathComponent.path;

        // Obviously it would be good to check that verovioBundle and resourcePath are not nil
        toolkit = std::make_unique<vrv::Toolkit>(false);
        toolkit->SetResourcePath([resourcePath cStringUsingEncoding:NSUTF8StringEncoding]);
    }
    return self;
}

- (NSInteger)getPageCount {
    return toolkit->GetPageCount();
}

- (NSString *)getVersion {
    std::string version = toolkit->GetVersion();
    return [NSString stringWithUTF8String:version.c_str()];
}

- (BOOL)loadData:(NSString *)data {
    std::string input = [data UTF8String];
    return toolkit->LoadData(input);
}

- (NSString *)renderToSVG:(NSInteger)pageNo {
    std::string svg = toolkit->RenderToSVG((int)pageNo);
    return [NSString stringWithUTF8String:svg.c_str()];
}

- (void)redoLayout {
    toolkit->RedoLayout();
}

- (void)setOptions:(nullable NSString *)jsonOptions {
    std::string options = jsonOptions ? [jsonOptions UTF8String] : "";
    toolkit->SetOptions(options);
}

@end
```

Using as a library

Verovio can be built and use as C++ or C library.

Building libverovio.so on Linux or libverovio.dylib on macOS

```
mkdir -p build
cd build
cmake -DBUILD_AS_LIBRARY=ON .
make
```

Running `sudo make install` will copy the library and the headers in `/usr/local/lib` and `/usr/local/include/verovio` respectively.

Building verovio.dll on Windows using Microsoft Visual Studio Build Tools 2022

Open `x64 Native Tools Command Prompt for VS 2022` and enter:

```
cd cmake
cmake -DBUILD_AS_LIBRARY=ON -DCMAKE_BUILD_TYPE=Release -DCMAKE_WINDOWS_EXPORT_ALL_SYMBOLS=TRUE -B build
cmake --build build --config Release
```

Examples

C++ interface

The following code is a minimal example using the C++ Toolkit class:

C++

```
#include <iostream>
using namespace std;

// Include header for the vrv::Toolkit classl
#include "toolkit.h"

int main()
{
    vrv::Toolkit tk(false);
    // Print the version
    cout << tk.GetVersion() << endl;

    return 0;
}
```

The example can be built with:

```
g++ main.cpp -o main --std=c++17 -lverovio -I/usr/local/include/verovio
```

Running `./main` should display the Verovio version.

C function interface

To use Verovio with any language that supports a plain C function interface you will first need to build Verovio as a library. The compiled library (libverovio.so / verovio.dll) will contain callable C symbols. These wrapper symbols are defined in `./tools/c_wrapper.h`

```
#include "stdbool.h"
#include "stdio.h"

// Include the header with the C functions
#include "c_wrapper.h"

int main()
{
    printf("Calling constructor\n");
    void *pointer = vrvToolkit_constructorResourcePath("./path-to-the-resource-dir");
    printf("Pointer value %p\n", pointer);
    const char *options = vrvToolkit_getAvailableOptions(pointer);
    printf("%s", options);
}
```

You can use gcc to compile the example above and link to the pre-built library:

```
gcc main.c -o main -lverovio -I/usr/local/include/verovio
```

Run `./main` or (without having installed the library or changed your default LD_LIBRARY_PATH):

```
LD_LIBRARY_PATH=/path-to-the-library-dir ./main
```

Using Verovio with Qt

Using Verovio with Qt is quite straightforward. It needs to be integrated as a library in the Qt project. This gives access to the C++ `vrv::Toolkit` class. The rendering can be achieved with a Qt `WebView`.

See this demo application repository for a complete example on how to do it.

Troubleshooting

This page contains descriptions and potential fixes for common issues when installing Verovio from source.

C-Compiler error on macOS

When trying to build Verovio from the sources and you've updated the system and the repository, but the CMake file was created a while ago, it might have gone out-of-sync and needs to be regenerated. You might see an error message like:

TERMINAL

```
-- The C compiler identification is AppleClang 14.0.0.14000029
-- The CXX compiler identification is AppleClang 14.0.0.14000029
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - failed
-- Check for working C compiler: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.x
ctoolchain/usr/bin/cc
-- Check for working C compiler: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.x
ctoolchain/usr/bin/cc - broken
CMake Error at /usr/local/Cellar/cmake/3.25.1/share/cmake/Modules/CMakeTestCCompiler.cmake:70 (m
essage):
  The C compiler

    "/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/cc"

  is not able to compile a simple test program.
```

Solution

Rerun cmake with the `--fresh` option (CMake 3.24 and later) or simply delete the `build/CMakeFiles` directory before installing it from sources.

Contributing

Coding guidelines

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

Formatting

Verovio uses a ClangFormat (**19.0**) 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.

Short if-statements should be as single line only with single boolean evaluation.

How to install clang-format on macOS

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

TERMINAL

```
brew install clang-format
```

How to install clang-format on Ubuntu

On Ubuntu clang-format is available in the universe repository. You can install it easily with the command:

TERMINAL

```
sudo apt install clang-format
```

Running clang-format

Please make sure you use at least version 10.0

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 .

Integer data types

Integer numbers should be int , or char but only when this is clearly appropropriate. The use of short is to be avoided unless there are some particular reasons to use it. Variables and class members should not be unsigned numbers unless strictly necessary.

We use int and not size_t , even when working with C++ standard containers. Values such as the one returned by std::vector::size() need to be cast to int when assigned or compared to variables. However, in cases where the scope is limited to local operations on the container and the use of int would yield a warning, size_t is acceptable.

We avoid the use of auto and prefer explicit typing.

Loop variable scope and increment

Variables should be made local to loops when possible (C99).

C++

```
for (int i = 0; i < limit; ++i) {}
```

They should preferably be pre-incremented, especially when not an int or alike.

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 it 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

When contributing to Verovio there are a few steps you can take to help make your contribution easy to understand and evaluate. Verovio uses the GitHub issue tracker and pull requests mechanism to organize these contributions.

These steps are:

1. Provide an short example MEI encoding that demonstrates a bug or a new feature that can be included in our test suite. You can use the Verovio Editor to create your example. This is described in more detail below.
2. Open an issue describing the problem or the new feature, and attach your short example. This provides our developer community with an opportunity to provide feedback on the problem, and determine the appropriate course of action.
3. If you can also provide the solution to the problem by modifying the Verovio source code, then that will speed up the process of getting your issue fixed! If you are a first-time contributor, then please make sure you have read the contributing guidelines. When you are ready, open a Pull Request, making sure to reference the open issue that it solves.

Adding examples to the test-suite

When adding examples to the test-suite, you should keep in mind the following points:

- The example should be as minimal as possible, ideally one or two measures and without un-related MEI / notation features
- The example has to be valid MEI (4.0 or 5.0-dev)
- The header should follow the test-suite style
- The XML should be indented with 3-spaces
- It is not mandatory to have an @xml:id on all MEI elements
- The file name should follow the test-suite style

Example header

Example MEI header for a test-suite example:

XML

```
<meiHead>
   <fileDesc>
      <titleStmt>
      <title>Slur position with cross-staff</title>
      <respStmt>
         <persName role="editor">Laurent Pugin</persName>
         <persName role="encoder">Craig Sapp</persName>
      </respStmt>
      </titleStmt>
      <pubStmt>
      <date isodate="2021-01-06">2021-01-06</date>
      <pubPlace>
         <ref target="https://github.com/rism-digital/verovio/issues/1898" />
      </pubPlace>
      </pubStmt>
      <notesStmt>
      <annot>Slurs with cross-staff should be place identically as in normal situations.</annot>
      </notesStmt>
   </fileDesc>
   <encodingDesc>
      <appInfo>
      <application version="3.1.0" label="2">
         <name>Verovio</name>
      </application>
      </appInfo>
   </encodingDesc>
</meiHead>
```

File names

The test suite examples are grouped by element name, with a very few exceptions. There is a corresponding folder name in the test suite folder. A test-suite example should be saved in the folder corresponding to the MEI element it targets. File names also use the element name and are numbered using the three digits (-001.mei) pattern.

Additional options

In some cases, a test suite example can require specific Verovio options to be set for it to make sense. For instance, it can require a specific layout or spacing parameter, or a specific font. The options can be set in the header of the MEI file as JSON object encoded as `CDATA` in the `<extMeta>` tag.

For example, setting the Bravura font can be triggered by including the following tag in the header of the test suite example:

XML

```
<extMeta><![CDATA[{ "font": "Bravura" }]]></extMeta>
```

> The additional options set in the MEI header are taken into account in both the test-suite page and the test-suite evaluation performed by the GitHub Actions. However, they currently remain ignored in the Verovio Editor.

What to expect with an open issue

When opening an issue, you should be prepared to help shepherd it through the process of getting fixed. If it is a problem with the software itself and you do not know how to fix it, you can still help with testing any potential fixes. You can also help by improving documentation about the new feature by contributing to the Verovio book, as appropriate. **Please do not open an issue unless you are willing to help, in some way, solve it.**

If you open an issue and someone provides a fix that requires no further changes, please respond! A “thank-you” and a note to say that it fixed the problem is always appreciated. You can also close the issue so that we know it has been addressed.

Sometimes an issue may be open for several years. These issues may be particularly complex, or may have been partially but not fully fixed. They usually have a discussion attached with sample encodings. Sometimes these issues have actually been fixed later, but as part of a separate issue. If you open an issue that happens to be fixed later, you can help us by leaving a note on your issue and closing it yourself.

If you are a software developer and can provide a solution, you should mention this in your issue. **For new contributors it is useful to open issues prior to opening pull requests.** Sometimes a change cannot be accepted, so opening an issue first gives an opportunity for the more experienced members of the community to provide feedback before you invest a lot of time in it. 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.

Issues that have a code contribution attached, and which have active participation from the reporter, are typically addressed first and fixed sooner. This is largely due to the community-driven nature of the project, recognizing that the more experienced developers have their own set of priorities. If you can provide a fix, even if it is not 100% correct, then it is easier to review your contribution and provide feedback than it is for someone else to code something from the ground up.

> If Verovio is a critically important part of your project, and you need dedicated help to make changes and contributions, the Verovio project accepts some sponsorship arrangements. Please get in touch to find out more about this.

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/dist` 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.

Generating LibMEI

You can regenerate the LibMEI code with a compiled ODD as input.

Go to the `./libmei` directory and run:

TERMINAL

```
python tools/parseschema2.py -c config.yml ./mei/mei-verovio_compiled.odd
```

Customization

Verovio currently uses an MEI customization that has added or modified a few elements. It is defined in the file ./libmei/mei/develop/mei-verovio.xml . If you want to make changes to it, you can make them there. You will need to regenerate the ODD file ./libmei/mei/develop/mei-verovio_compiled.odd . This can be done with the Edirom MEI Garage. Alternatively, you can also use the MEI command line script. To do so, you need to create 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 the Leipzig font. For adding support for a new SMuFL glyph, the steps are:

1. Add the glyph to the Leipzig font file
2. Generate the Leipzig in various format and the metadata with the script available in Leipzig repository
3. Add the glyph to the list of supported glyph in the XML list and re-generate the fonts.

Make sure you always add glyphs **only** in the Leipzig font. Because conflict solving is quickly very when adding a glyph (in particular with the binary font files), make sure you always pull the latest version of the font file 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 to modify is ./Leipzig.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 simply 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 “FONTLOG” where you also need to add a comment together with the version number. The file can be saved.

Generate other font formats and the metadata

Once the ./Leipzig.sfd has been modify and saved, you have to run the ./generate_font.py script that will generate different font formats and the metadata. You are now ready to make a PR to the Leipzig repository.

Add the glyph to the list of supported glyph in the XML list and re-generate the fonts

Once the PR to the Leipzig repository has been approved and merged, the new glyphs have to be added to the Verovio codebase. The first thing to do is to add them to the list of glyphs supported by Verovio.

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

To do so, you need to copy to ./fonts/Leipzig the new Leipzig files:

- Leipzig.woff2
- Leipzig.ttf
- Leipzig.svg
- leipzig_metadata.json

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 . The CSS font files will also be updated.

When using a custom FontForge binary (e.g. latest AppImage) the specific location needs to be provided to generate.py via the --fontforge <path-to-fontforge-binary> command line argument. When using generate_all.sh , all command line arguments are passed on to the python scripts.

Example: ./generate_all.sh --fontforge /mnt/linux-data/henry/Apps/FontForge-2023-01-01-a1dad3e-x86_64.AppImage

Table of Contents