

## **The Music Encoding Initiative (MEI) DTD and the OCVE Perry Roland, University of Virginia Library**

### **Introduction**

The purpose of the Music Encoding Initiative (MEI) DTD is two-fold: to provide a standardized, universal XML encoding format for music content (and its accompanying meta-data) and to facilitate interchange of the encoded data. MEI is not designed to be an input code per se, like the Plaine and Easie code; however, it is intended to be human-readable and easily understood and applied. Because of its emphasis on comprehensiveness and software independence, MEI may also function as an archival data format.

This white paper describes the features of MEI and the advantages of its use as the encoding standard for the Online Chopin Variorum Edition.

### **Details of the Standard**

#### *MEI is XML-based*

MEI aims to encode music in such a way that it can serve as the standard format that sits at the center of an interchange hub between systems and functionality. For any encoding standard to serve such a purpose, almost by definition the standard must be XML-based.

Why XML? The key requirement for a music-encoding standard is that it must support the encoding of the visual, gestural, and analytical domains of notation in a way that is flexible, yet formal and verifiable. By definition, XML encodes the structure of a document, following the formalized and verifiable rules that are explicitly set forth in a DTD (Document Type Definition). The DTD additionally provides the instructions for the interpretation of those rules when programmatically interacting with an XML file. While it cannot model semantics, a DTD formally declares the syntax of the representation. A representation based on a formal grammar like that embodied in a DTD can describe a broad range of music.

In addition, XML is an inherently hierarchical encoding structure, nesting components and sections of a document within a tree structure. There are several benefits in representing musical notation hierarchically, such as the isolation of individual components to limit interactions between components and to specify the scope of the operators that act upon them. A hierarchical structure also allows any component of a score to be treated in exactly the same manner as any other, regardless of its size or position in the hierarchy. Unnecessary complexity is not introduced by using a hierarchical structure instead of the list structure commonly used to represent music.

As an XML-based music encoding solution, MEI can include relationships between elements, support navigation within the music structure as well as to external multimedia entities, allow definition of custom symbols, and support cooperative creation and editing of music markup.

MEI has a significant advantage over other proposed XML standards that define an entirely new terminology because it uses familiar names for elements and attributes. For example, the element for encoding a note is called <note> and the simultaneous sounding of multiple notes is called a <chord>. Using common music notation terminology has the benefit of making MEI files more human-readable,

and makes clear the correspondence between MEI-encoded data and music notation. That being said, however, MEI may also be extended in such a way as to allow alternate element names. These alternative names may support longer descriptive names for new users, shorter mnemonic names for experienced users and reduction of storage requirements, names in different languages, or an extension mechanism, sometimes called architectural form, based on concept name.

Why a DTD and not a Schema? The simplest reason is that DTDs are a stable part of XML technology and are widely supported in authoring software. More importantly, schemas do not include support for external entities, and the lack of external entities makes a schema difficult to extend and modularize without rewriting, which negates the concept of a core encoding standard.

### *Comprehensiveness*

Simple codes that represent only the information necessary for a particular application seem efficient because they require less development effort and less complex processing software; however, more comprehensive code is better able to capture the interdependency present in the elements within music. A comprehensive coding scheme is inherently preferable because of its support for a multiplicity of functional and interchange possibilities. When the encoding formats are limited and specific software is required, converting between just 6 different representations necessitates 30, that is,  $(6*(6-1))$ , different translation programs, one for each pair of representations. However, using a more general representation as a “hub” requires only 12, that is,  $(6*2)$ , translators, one from each representation to the central hub and one from the hub to each representation. Typically, format conversion is “lossy” even under the best conditions. Therefore, the fewer conversions needed the better.

The fact that MEI uses the terminology of music notation does not limit its usefulness for encoding abstract music. MEI’s elements are named for structures found in music notation in order to make the process of encoding more comprehensible, but an MEI instance does not have to include any layout information. MEI can be used to encode one or more fixed forms of a composition, but can also be used to encode notation, performance, and analytical information on a more abstract level.

As an example, an MEI file can be transformed into a Mup text file using XSLT, and the file that results from the transformation can itself be transformed into Postscript and a PDF. Notational representation requires extensive data; the fact that MEI can encode enough information to generate accurate notation when compared with existing notation is an indication of the level of detail and extensibility of the MEI standard. The fact that an MEI file can also be transformed into a MIDI file, and that an MEI file can be derived from a MusicXML file are additional proof of its promise as a comprehensive interchange format between systems and functions.

### *Software Independence*

XML is a license-free standard and is not tied to any platform or application. XML provides a human- or machine-readable syntax for encoding structured data in a way that allows it to be manipulated and displayed using simple and increasingly standardized and ubiquitous tools available for the Mac, Windows, or UNIX/Linux environments. No special tools are required for creating, manipulating, or rendering XML data. Opting for XML is like choosing SQL for databases – one has to build a custom database and programs and procedures that manipulate it, but the fact that one can do these things provides infinite flexibility in designing a solution that meets the needs of the problem. As an XML solution, MEI is flexible, portable, and customizable, making it easy to exchange information regardless of platforms or systems.

### *Formal Extension/Restriction Mechanisms*

A music encoding solution cannot aim to eliminate all variation. Instead, it must remain flexible enough to accommodate variations in the source material. MEI does this by providing a large set of general-use elements and by allowing the user to choose from multiple methods of encoding based on his use for the data. Much of the encoding is optional so that the encoder is not required to mark up data that is not necessary for the problem at hand.

Since no representation can anticipate all of its uses, extensibility is also required. MEI can be extended, primarily via internal parameter entities, as needed to support the requirements of the corpus being encoded.

### *Rich Meta-data Header*

MEI provides a clear separation between data and meta-data. Separating these allows the meta-data to be shared with other entities, both internal and external, to which it applies. The <meihead> element contains bibliographic (descriptive, administrative, and technical) meta-data for the encoded file and for the source(s) from which the data was taken, while the <work> element contains the encoded music data. In addition to standard bibliographic meta-data, the header may also contain a declaration of editorial practice and a file revision history.

The concepts for the sub-elements of <meihead> are based on standard cataloging standards, such as ISBD(G), AACR2, and Dublin Core. By drawing on these sources, MEI is able to provide richer meta-data than that found in other XML music encoding schemes, which often provide little more than composer and title elements.

Most elements found within the header have 2 attributes, analog and label, which facilitate the use of the data in a bibliographic context. The analog attribute is intended to contain an indication of the element's correspondence to some other bibliographic standard, such as MARC or Dublin Core, while the label attribute allows for presentational labeling of the element's content. This makes conversion to another format or inclusion in a separate meta-data catalog easier.

The <filedesc> sub-element contains descriptive meta-data such as title, agent, and publication status, administrative meta-data, such as vendor, price, and rights management info, and technical information, such as system requirements, for the file. An agent is defined as a person or corporate body directly involved in the creation of the source or its electronic representation.

The <sourcedesc> sub-element contains descriptions of each source for the file. <source> elements employ the same bibliographic elements as <filedesc>, but also include physical description elements, such as medium, dimensions, provenance, inscription, condition, etc., which are important in the description of original, especially unpublished manuscript, sources. A header may have multiple <source> elements, such as those for manuscript versus published versions of the work, and may additionally include descriptive data about the provenance and condition of the source material. The source elements may be referenced through an attribute in a number of other elements, including <section>, <ending>, <part>, <measure>, <staff>, <layer>, <rdg>, and <ossia>. This is especially valuable in the encoding of critical editions, where any of these structures might refer back to its source.

### *No Constraints on the Concept of a Work*

The design principles behind MEI are based on those that guided the creation of the Text Encoding Initiative (TEI). TEI is a comprehensive, yet extensible standard for the encoding and transmission of textual documents in electronic form. The large and varied body of texts that have been encoded demonstrates the success of TEI. At first glance, TEI appears to have an overly broad scope; however, the scope of the TEI is limited in a subtle, yet significant way. The TEI is primarily concerned with markup of those expressions can be expressed as text; that is, those expressions which take a *written* form. Furthermore, the TEI is mute regarding the “proper” way to compose text, or in defining what a text might be.

Similarly, MEI is agnostic regarding the definition of the term *musical work* – it is simply the “thing” being encoded. It can be a monophonic song or a complex symphony. The underlying assumption, however, is that the original work being encoded is or can be expressed in a *written* form.

A musical work in MEI terms may be a collection of works, such as a printed collected edition or an electronic database of related works. The <group> element facilitates creation of a collection of <work> elements that share a header, such as a collection of songs by different composers issued under a single title, but that also require separate encodings. Basic meta-data for each work may be encoded in its own front matter or in <source> elements in the file header. The <group> element is recursively nest-able to any level so that sub-groups may be created. The <meicorpus> element is preferable when a complete bibliographic header is required for each member of a collection.

### *Handling of Work-level Text*

Since MEI takes a broad view of what a music document is, the DTD has <front> and <back> matter elements, like those found in the TEI DTD, that provide logical and presentational text markup functionality. Critical editions and collections of works often contain extensive text, such as a table of contents, an introduction, commentary, a biography, an index, etc. Accommodating this text within MEI rather than embedding MEI within a text markup scheme such as TEI or DocBook gives control of the text and the notation to the encoder of the MEI instance and to the music markup community rather than the creators and maintainers of the text standard.

In addition, to <front> and <back> elements, MEI can encode the introductory or explanatory text sometimes found between sections of a musical work. As an extreme example, one might encode a music treatise not as a text which includes musical examples, but rather as a set of musical examples which have text between them.

### *Segmentation*

The <music> element encodes the musical content of the work. It contains one or more discrete, linear segments, called <mdiv> (“musical division”). An <mdiv> is the highest-level indication of the structure of the composition. For example, a single <mdiv> indicates a single-movement work; however, when a musical work can be broken into several top-level segments, the <music> element may contain multiple <mdiv> elements. The <mdiv> element is a generic one which may be typed – a symphony, for example, usually consists of movements while operas are made up of acts. A part or score may be divided into linear segments or sections. <section> elements usually function as a scoping mechanism for clef signs, key and meter signatures, plus metronome, tempo, and expression markings. The use of section elements also minimizes the need for backward scanning to

establish context when the starting point for access is not at the beginning of the score. Section elements may also be used for other user-defined, i.e., analytical or editorial, purposes, and arbitrarily nested to any desired level. There is also an <ending> element, a specialized instance of <section> element that may not be recursively nested.

### *Separate Score and Parts views*

The <mdiv> element may contain one or both of two possible views. The <score> element contains a traditional, full open score while the <parts> element contains each performer's view of the work. Score and parts views are intended to accommodate different methods of organizing the markup – no particular presentation is implied, and software may render a collection of parts as a score or a score as a collection of parts. The explicit encoding of two views is necessary because it is not always possible to derive one view from the other. In addition, separating scores and parts can eliminate a great deal of markup complexity.

A <part> element contains an individual performer's view of the score, a mini-score requiring all the encoding features of a full score. The encoding of individual parts is practical when they do not share visual characteristics, such as typeface or page layout, with the full score. <part> elements in MEI have little to do with voice leading, which can be encoded using the next attribute available on all event-type elements.

### *Encoding of Variant Readings*

MEI facilitates encoding of multiple versions of the music content in the same file. Using multiple files increases the amount of work to encode the file, because significant portions of the content in each version are often the same, and creates a problem of synchronizing, that is, locating the same point, say the beginning of measure 3, in all the files.

An <app> (apparatus) element contains at least two <rdg> (reading) elements, each of which may be linked back to different <source> elements in the header. A reading may contain <app> sub-elements so that variants of variants may be described. This process may be used recursively to any depth necessary. Each reading may be assigned an order, such as for selection or rendering purposes, other than the encoded order. Readings may have different musical parameters, such as different keys or a different number of measures. <app> can be used when readings do not synchronize with each other, such as the encoding of a sketch and a published version with a different number of measures in a particular section.

The <ossia> element performs a similar function similar to <app>, but at the measure level – representing an alternative present in the source being transcribed.

### *Support for Mensural Music and Non-aligning Bar Lines*

In addition to measured music, MEI can be used for unmeasured music. Options include encoding the work in one or more <staff> elements, which may directly contain events, such as notes, rests, etc., or as multiple parts by using the <part> element. Part-by-part encoding using the <part> element is also useful for the encoding of music with non-aligning barlines for print purposes, where the barlines in the varying parts can additionally be identified as aligning or non-aligning through the use of an attribute on the <measure> element. Some software also requires staff-by-staff encoding which can be accommodated through separate parts. When assembly of a group of parts into a score is desired and

there are non-aligning barlines, barlines that indicate points of alignment across all the parts should be marked as ‘controlling’.

### *Flexible Staff/Measure or Measure/Staff Organization*

Within a <section> one may choose a staff-by-staff organization with nested measures or a measure-by-measure approach with nested staves, depending upon the problem to be solved. For example, staff-by-staff organization would be preferable for layout encoding, while measure-by-measure would be preferable for automated analysis.

Within the <measure> element, <staff> and <layer> elements are provided in order to indicate the organization of the measure’s contents – multiple staves may connote separate data streams without implying any physical layout. Alternatively, the measure’s contents may be encoded as a series of events where each event may be related to a particular stream of data via its own staff and layer attributes.

### *Events*

In general, events are modeled, not symbols. Events are the typical, time-based, discrete atoms of musical data, such as notes, chords, rests, etc. While events may have visual properties, modeling symbols places too much emphasis on presentational qualities and makes the markup less generally useful as a “music” markup standard.

XML attributes are typically used to describe the musical attributes of events for several reasons. First, attribute values may be constrained and defaulted using a DTD while element content cannot. Second, sub-elements are useful when the data requires structure, when there may be more than one value at a time, or when the data should make sense with the markup removed. None of these conditions usually apply to musical attributes. Third, the use of attributes for musical attributes makes the distinction between a feature of the encoding, such as a note, and that feature’s properties, such as the note’s stem length, clear. Finally, the use of attributes helps reduce the file size and enforces a one-to-one relationship between the data object and the property.

The event class, %m.events, is user-definable. It also has an attribute class, %a.event, associated with it.

### *Control Events*

Control events, such as dynamics, ties, phrase marks, pedal marks, etc., depend upon other events, such as notes or rests, for their existence. They often do not fit the principal hierarchy of sections, measures and staves. They cannot always be treated as properties of first-order data objects or forced to conform to the same markup hierarchy as the first-order objects. Therefore, a second class of events, %m.controlevents, exists in MEI. This class of events is also user-definable and has its own associated attribute class, %a.controlevent.

In MEI, multiple control events of the same type, e.g., pedal indications, may be associated with the same set of events, such as notes, through differentiated type attributes. This is particularly useful when the principal events are the same across multiple sources but the control events are different.

## *Hyperlinks and References*

A number of elements in MEI may contain references to other parts of the file or hyperlinks to external documents. The <bibref>, <extptr>, <extref>, <pb>, <section>, <rdg>, <ending>, <measure>, and <annot> elements may link to external objects, such as page images, texts, and sound files. The <ptr>, <ref>, <section>, <rdg>, <ending>, <measure> and <annot> elements may link to other parts of the file. One example is the <annot> element, which can be used to relate a portion of the work to front or back matter, a source, or to an external resource. Multiple <annot> elements can create one-to-many links between concepts. In another example, measures or pages might be linked to images of those sections.

## *MIDI Output Suggestions*

A synthesized performance suitable for many purposes may be mechanically derived from the notation, and the flexibility and comprehensiveness of the MEI DTD allow for any additional information necessary to complete this process to be encoded. The midi event class, %m.midievents, is also user-definable.

## *Realized Ornamentation*

Ornamentation may be encoded as one or more symbols, but it can also be encoded in a number of ways to represent one or more preferred realizations. Realized ornamentation can be encoded as a variant reading, as a separate staff, or as additional notes within a <note> element.

## *Symbol Placement Specification*

MEI assumes that the placement of elements for layout purposes will be handled by some downstream layout processor, such as Music Publisher (mup), and that placement will follow some regular conventions. In those cases where the encoding of the layout of the work requires notation of an exception, MEI may be used to encode layout offsets. Since the proportions of staves in layouts vary with the rendering software, MEI does not encode any specific measures (pixels, picas, ems) or absolute locations, instead specifying offsets in 1/2 staff inter-line units that can be interpreted as appropriate by the software used. Absolute locations can be added via extension.

## *Annotations*

Annotations may be used to group participating events and provide a label for the group, such as a melodic fragment derived from another found in a sketch. Annotations may also serve as a pointer to an editorial or analytical observation, which may be encoded elsewhere in the document or even in an external file. Multiple <annot> elements can create one-to-many links between concepts.

## **Testing of MEI for the OCVE**

### *Process*

MEI has been tested for use in the OCVE by encoding page 9 of the Barcarolle, op. 60. A MusicXML file was initially produced using SharpEye; the MusicXML file was then hand-edited by Craig Sapp of Stanford University. An MEI file was derived via XSLT from the edited MusicXML file. The MEI

file was verified against an image of the printed version, which was supplied by the “Chopin Early Editions” project at the University of Chicago Library. Some hand-editing was necessary to add some features that SharpEye didn’t catch, such as pedal and tempo markings, and to correct some errors introduced by SharpEye, such as missing notes and incorrect phrase mark attachments. A second XSLT process was used to transform the MEI file into a mup file. The mup file required some hand-editing – MIDI tempo indications were added, and the placement of staccato dots in m. 6 and m. 12 was corrected. The mup software was then used to generate both a PostScript notation file and a MIDI file. The DTD and the sample files from the test are available online at <http://www.lib.virginia.edu/digital/resndev/mei/>.

### *Evaluation*

The XSLT used for the MusicXML-to-MEI and the MEI-to-mup transformations were created for this proof-of-concept test and not for full production. Errors and deficiencies found in the MEI file and the mup file could be remedied through upgrades to the XSLT used for the transformation processes.

The conversion of MusicXML to MEI was somewhat complex due to the fact that MusicXML takes an event-centric view of notation. For example, beams are treated as properties of a note rather than as a higher-level structure. Had this same approach been used in the MEI instance the transformation from MusicXML would have been easier. The downside is that one would still be left with an event-centric view.

An additional future experiment might attempt to develop a transformation directly from the SharpEye format (.mro) to MEI without the intermediary MusicXML transformation. The data necessary for precise placement of the phrase marks did not exist in the MusicXML file, but was in the SharpEye file. This situation is a good example of data loss during multiple conversions. Despite the fact that direct conversion may result in a better transformation, there is a potential technological limitation in that XSLT 1.0, the current version, cannot perform a transformation on a text file. XSLT 2.0, currently under review by the W3C, would alleviate this limitation. Of course, the transformation process does not require XSLT and could be carried out using a general programming language such as PERL.

The result of the MEI-to-mup transformation could have been written so that mup used its default placement rules for phrase marks. This would alleviate the need for the placement data, but in earlier tests was found to result in badly-formed phrase marks.

Verification against a printed version is still desirable. Hand-editing may still be needed in the immediate future, but the possibility of full automation of the process is a distinct possibility.

Linking from an individual encoded measure to an image of the measure is a trivial task and was not tested. Even so, a minimal MEI instance could function as a “database” of links to individual measure images for the superimposition, juxtaposition, and combination/interpolation operations suggested in the image-based portion of the original proposal. A fully encoded instance could be created as time allowed. Then the image operations mentioned above could be performed on images of individual measures derived from the complete encoding.

### *Future Potential*

The true potential of MEI is that a single file can be used to encode multiple versions of a musical work and generate multiple outputs. The creation of the MEI standard is an ongoing, collaborative

process, fed by projects such as the use of MEI for the OCVE. MEI is currently available in version 1.5b; 1.6b is under development, driven in part by the research needed to develop this white paper. Future development of the core MEI DTD and extensions will likely require a coordinating group that will collaborate on the registration, review, and release of updates that can keep the standard fresh and up-to-date for the foreseeable future.