Navigating and Processing MEI Data with XPath and XSLT

Martha E. Thomae
Universidade NOVA de Lisboa

Perry Roland
University of Virginia

Content

- Introduction to XPath (hands-on)
- XSLT: Conversion from MEI to MEI, metadata example
- XSLT: Conversion from MEI to HTML, metadata example
- XSLT: Extracting (pulling) information, music example
- XSLT: Putting this information into a TSV table (extra example)

https://github.com/martha-thomae/XSLT-examples-for-MEI

XPath

XPath

- **XPath** uses path expressions to select nodes in an XML document
- The node is selected by following a path (or steps)
  - `/step/step/…`
    - **Example:** `/mei/meiHead/fileDesc/titleStmt/title`

XPath

- **XPath** uses path expressions to select nodes in an XML document
- The node is selected by following a path (or steps)
  - `/step/step/…`
    - **Example:** `/mei/meiHead/fileDesc/titleStmt/title`

- Each step consists of:
  - **Axis** (optional): represents a relationship to the context (current) node and is used to locate nodes relative to that node on the tree
    - *`ancestor`*, *`descendant`*, *`ancestor-or-self`*, *`descendant-or-self`*, *`preceding-sibling`*, *`following-sibling`*, *`parent`*, *`child`*, *`preceding`*, *`following`*, *`self`*, *`attribute`*
  - **Node test**: identifies a node within an axis
  - **Predicates** (optional): conditions to further refine the selected node set

```
axisname::nodetest[predicate]
```

Let’s try a few things together!

https://github.com/martha-thomae/XSLT-examples-for-MEI

File to process (input):
Bach-JS_Ein_feste_Burg.mei

Examples: Paths and Nodes

- `/` (this is called the “document node”)
- `/mei`
- `/mei/meiHead/fileDesc`
- `//fileDesc`
- `/mei/meiHead/fileDesc/titleStmt/title`
- `//title`
- `//titleStmt/title`

Examples: Nodes and Axes

- `//fileDesc/descendant::persName`
- `//persName/ancestor::composer`
- `//persName/preceding-sibling::persName`
- `//persName/following-sibling::persName`

One can find nodes by their name (as shown so far), or by their type:

- `//element()`
- `//attribute()`
- `//text()`
- `//processing-instruction()`

Examples: Attributes and Predicates

Let’s try these:

- `//note`
- `//note/@pname` (also `//note/attribute::pname`)
- `//note[@pname='c’]`
- `//note[@pname='c']/@dur`

Tasks:

- Find the octave of the notes
- Find the pitch name of all whole notes

Functions and Operators

- Functions
  - Node set functions: `count()`, `position()`, `last()`
  - String functions: `normalize-space()`, `concat()`, `substring()`
  - Get properties of nodes: `local-name()`
- Operators:
  - Logical: `and`, `or` (also `|`), `not()`
  - Arithmetic: `+`, `-`, `*`, `div`, `mod`
  - Comparison: `eq`, `=`, `ne`, `!=`, `>`, `<`, `>=`, `<=`

➢ `count(//note[@pname='c' and @oct='4’])`

XPath is how you find things

XSLT is how you manipulate things

XPath is used in XSLT to
find the things you want to work on

XSLT

XSLT

- Extensible Stylesheet Language Transformations (XSLT)
- An XML-based language used, in conjunction with specialized processing software, for the transformation of XML documents (like MEI and HTML)

How to process XSLT?

- Requires a **processor** (as all programming languages)
  - Command line processors (e.g., Saxon, Xalan)
  - Built into a browser (with XSLT 1.0)
  - Other programming language (e.g., Java, JavaScript, Python)
  - Some editors (like **Oxygen**) have built in processors
    - Saxon-HE 9.9.1.7 (Home Edition) → free
    - Saxon-PE 9.9.1.7 (Professional Edition)
    - Saxon-EE 9.9.1.7 (Enterprise Edition)

XSLT

Conversion from MEI to MEI

Metadata Example

Resources

https://github.com/martha-thomae/XSLT-examples-for-MEI

- **File to process (input):** CNW01.xml
- **Steps:** 1_MEI-to-MEI folder
- **Final XSLT file:** cnwFix_complete.xsl

- **Situation:** Update the MEI file CNW01 (which was encoded with MEI version 4.0.1) to be compliant with version 5.0 and correct some unconventional practices in the encoding

Summary: cnwFix_complete.xsl

1. Start template
2. Copy template
3. Change an attribute’s value
4. Remove empty attributes
5. Move an element (copy-paste + delete original)
6. Rewrite the text of an element
7. Unwrap content
8. Add an attribute
9. Add an element (as child)

1. Starting Template

Default behaviour: returning all the text that is in the document

```
<xsl:template match="/">
  <xsl:apply-templates/>
</xsl:template>
```

2. Copy Template

```
<xsl:template match="element() | text() | processing-instruction() | comment() | @*">
  <xsl:copy>
    <xsl:apply-templates select="@*"/>
    <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>
```

2. Copy Template

```
<xsl:template match="/">
   <xsl:apply-templates/>
</xsl:template>
```

- The **starting template** says start applying templates,
- but the only template is the **copy template → copies everything!**

```
<xsl:template match="element() | text() | processing-instruction() | comment() | @*">
  <xsl:copy>
    <xsl:apply-templates select="@*"/>
    <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>
```

2. Copy Template

- The <xsl:copy> does a shallow copy of the matches
- But the matches are everything: elements, text, processing instructions, comments, and attributes
- And then it applies that template—that does the shallow copy—sequentially to (1) elements' attributes and (2) their children

```
<xsl:template match="element() | text() | processing-instruction() | comment() | @*">
  <xsl:copy>
    <xsl:apply-templates select="@*"/>
    <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>
```

2. Copy Template

- The **<xsl:copy>** does a **shallow copy** of the matches
- But the matches are everything: elements, text, processing instructions, comments, and attributes
- And then it applies that template—that does the shallow copy—sequentially to (1) elements' attributes and (2) their children

```
<xsl:template match="element() | text() | processing-instruction() | comment() | @*">
  <xsl:copy>
    <xsl:apply-templates select="@*"/>
    <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>
```

2. Copy Template

- The <xsl:copy> does a **shallow copy** of the matches
- But the matches are everything: elements, text, processing instructions, comments, and attributes
- And then it applies that template—that does the shallow copy—sequentially to (1) elements' attributes and (2) their children

```
<xsl:template match="element() | text() | processing-instruction() | comment() | @*">
  <xsl:copy>
    <xsl:apply-templates select="@*"/>
    <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>
```

2. Copy Template

- The <xsl:copy> does a **shallow copy** of the matches
- But the matches are everything: elements, text, processing instructions, comments, and attributes
- And then it **applies that template—that does the shallow copy—sequentially** to **(1) elements' attributes** and (2) their children

```
<xsl:template match="element() | text() | processing-instruction() | comment() | @*">
  <xsl:copy>
    <xsl:apply-templates select="@*"/>
    <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>
```

2. Copy Template

- The <xsl:copy> does a **shallow copy** of the matches
- But the matches are everything: elements, text, processing instructions, comments, and attributes
- And then it **applies that template—that does the shallow copy—sequentially** to (1) elements' attributes and **(2) their children**

```
<xsl:template match="element() | text() | processing-instruction() | comment() | @*">
  <xsl:copy>
    <xsl:apply-templates select="@*"/>
    <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>
```

Identity Transform
= Starting + Copy Templates

```
<xsl:template match="/">
   <xsl:apply-templates/>
</xsl:template>
```

```
<xsl:template match="element() | text() | processing-instruction() | comment() | @*">
  <xsl:copy>
    <xsl:apply-templates select="@*"/>
    <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>
```

Finish with an exact copy of the document we started with

IMPORTANT NOTE

- Default behaviour:
  - Walk through the tree
  - And (with the copy template) copy everything

- New templates: Overwrite that default behaviour

Add other templates to overwrite the default behaviour (copy) for certain situations

3. Change the value of an attribute

Situation:

- Initial file: @meiversion = 4.0.1
- Change @meiversion to match the one of the associated schema (5.0)

3. Change the value of an attribute

Situation:

- Initial file: @meiversion = 4.0.1
- Change @meiversion to match the one of the associated schema (5.0)

Template:

```
<xsl:template match="@meiversion">
  <xsl:attribute name="meiversion">5.0</xsl:attribute>
</xsl:template>
```

- Finds the attribute (the match)
- Creates a new attribute with the same name
- And assigns it a value

3. Change the value of an attribute

Situation:

- Initial file: @meiversion = 4.0.1
- Change @meiversion to match the one of the associated schema (5.0)

Template:

```
<xsl:template match="@meiversion">
  <xsl:attribute name="meiversion">5.0</xsl:attribute>
</xsl:template>
```

- Finds the attribute (the match)
- Creates a new attribute with the same name
- And assigns it a value

3. Change the value of an attribute

Situation:

- Initial file: @meiversion = 4.0.1
- Change @meiversion to match the one of the associated schema (5.0)

Template:

```
<xsl:template match="@meiversion">
  <xsl:attribute name="meiversion">5.0</xsl:attribute>
</xsl:template>
```

- Finds the attribute (the match)
- Creates a new attribute with the same name
- And assigns it a value

3. Change the value of an attribute

Situation:

- Initial file: @meiversion = 4.0.1
- Change @meiversion to match the one of the associated schema (5.0)

Template:

```
<xsl:template match="@meiversion">
  <xsl:attribute name="meiversion">5.0</xsl:attribute>
</xsl:template>
```

- Finds the attribute (the match)
- Creates a new attribute with the same name
- And assigns it a value

3. Change the value of an attribute

```
<xsl:template match="@meiversion">
  <xsl:attribute name="meiversion">5.0</xsl:attribute>
</xsl:template>
```

This doesn’t copy the attribute

As the program walks through the XML tree, by default copying everything

It eventually runs into @meiversion

When this happens (it matches @meiversion),
it creates a new attribute with that same name and with a new value

And then it continues walking through the tree

4. Remove empty attributes

Situation: Many elements have an empty attribute @label

```
<p xml:id="idm829">
    At the end of 1896, when Nielsen had finished the choral work <ref
    xmlns:xl="http://www.w3.org/1999/xlink" xml:id="idm830"
    target="document.xq?doc=cnw0100.xml" xl:show="replace" label=""> <rend
    xml:id="idm831" fontstyle="italic">Hymnus Amoris</rend> (CNW 100)</ref>, he began to plan
    an opera. The text for <rend xml:id="idm832" fontstyle="italic">Saul og David</rend>, written
    by Einar Christiansen, was ready in 1899. A few months seem to have passed before Nielsen
    started in earnest on the composition. Part of the opera was composed during his stay in Italy
    from December 1899 to June 1900. The work was finished in April 1901 and accepted for
    performance at The Royal Theatre in September. Parts of the work were performed at a
    concert in November 1900.
</p>
```

```
<bibl label="" xml:id="bibl_d1e46064520">
  <title xml:id="title_N20E2C">CNB</title>
  <biblScope xml:id="biblScope_N20E2E">I/655</biblScope>
</bibl>
```

4. Remove empty attributes

Situation: Many elements have an empty attribute @label

```
<xsl:template match="@*[normalize-space(.) = ']"/>
```

- Example of a predicate = condition
  - *normalize-space*(.) = '
- Empty template
  - If it matches → don't do anything
  - Therefore, the match does not make it to the output file (this template overwrites the default: copy to the output file)

4. Remove empty attributes

Situation: Many elements have an empty attribute @label

```
<xsl:template match="@*[normalize-space(.) = '']"/>
```

- Example of a predicate = condition
  - *normalize-space*(.) = ''
- Empty template
  - If it matches → don't do anything
  - Therefore, the match does not make it to the output file (this template overwrites the default: copy to the output file)

5. Move an element

Situation:

```
<pubStmt xml:id="pubStmt_N2029F">
  <respStmt xml:id="respStmt_eb89e9f0">
    <resp xml:id="resp_N202A2">Publisher</resp>
    <corpName xml:id="corpName_f7008792">
      <abbr xml:id="abbr_N202A6">DCM</abbr>
      <expan xml:id="expan_cde0339b">Danish Centre for Music Editing</expan>
      <address xml:id="address_N202AA">
        <addrLine xml:id="addrLine_b24889b6">Royal Danish Library</addrLine>
        <addrLine xml:id="addrLine_N202AD">Søren Kierkegaards Plads 1</addrLine>
        …
      </address>
    </corpName>
    <persName role="editor" xml:id="persName_28d8cc37">Niels Bo Foltmann</persName>
    …
  </respStmt>
  <date isodate="2014" xml:id="date_N202DF">2014</date>
</pubStmt>
```

5. Move an element

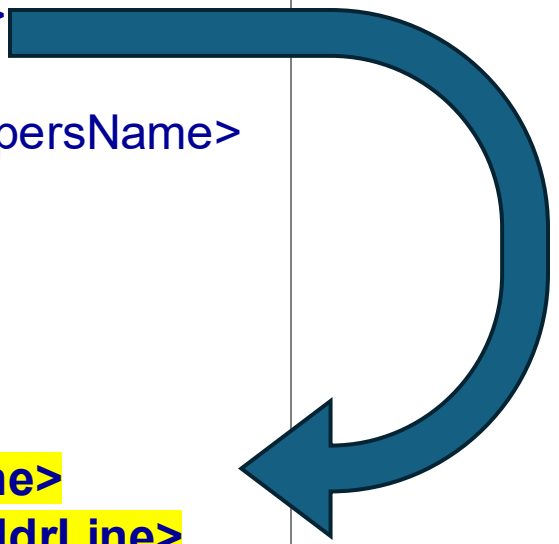
Situation:

```
<pubStmt xml:id="pubStmt_N2029F">
  <respStmt xml:id="respStmt_eb89e9f0">
    <resp xml:id="resp_N202A2">Publisher</resp>
    <corpName xml:id="corpName_f7008792">
      <abbr xml:id="abbr_N202A6">DCM</abbr>
      <expan xml:id="expan_cde0339b">Danish Centre for Music Editing</expan>
      <address xml:id="address_N202AA">
        <addrLine xml:id="addrLine_b24889b6">Royal Danish Library</addrLine>
        <addrLine xml:id="addrLine_N202AD">Søren Kierkegaards Plads 1</addrLine>
        …
      </address>
    </corpName>
    <persName role="editor" xml:id="persName_28d8cc37">Niels Bo Foltmann</persName>
    …
  </respStmt>
  <date isodate="2014" xml:id="date_N202DF">
</pubStmt>
```

Outside of <respStmt>
Inside a new element <pubPlace>
(still inside of <pubStmt>)

5. Move an element

```
<pubStmt xml:id="pubStmt_N2029F">
  <respStmt xml:id="respStmt_eb89e9f0">
    <resp xml:id="resp_N202A2">Publisher</resp>
    <corpName xml:id="corpName_f7008792">
      <abbr xml:id="abbr_N202A6">DCM</abbr>
      <expan xml:id="expan_cde0339b">Danish Centre for Music Editing</expan>
    </corpName>
    <persName role="editor" xml:id="persName_28d8cc37">Niels Bo Foltmann</persName>
    …
  </respStmt>
  <date isodate="2014" xml:id="date_N202DF">2014</date>
  <pubPlace>
    <address xml:id="address_N202AA">
      <addrLine xml:id="addrLine_b24889b6">Royal Danish Library</addrLine>
      <addrLine xml:id="addrLine_N202AD">Søren Kierkegaards Plads 1</addrLine>
      …
    </address>
  </pubPlace>
</pubStmt>
```



5. Move an element

Solution would be a cut and paste, which is equivalent to:

Step 1: Copy & paste

Step 2: Delete original

5. Move an element

Step 1: Copy & paste <address> to a new place

```
<!-- Copy <address> from <respStmt> to <pubPlace> -->
<xsl:template match="*:fileDesc/*:pubStmt[*:respStmt/*:corpName/*:address]">
  <xsl:copy>
    <xsl:apply-templates/>
    <pubPlace>
      <xsl:copy-of select="*:respStmt/*:corpName/*:address"/>
    </pubPlace>
  </xsl:copy>
</xsl:template>
```

5. Move an element

Step 1: Copy & paste <address> to a new place

```
<!-- Copy <address> from <respStmt> to <pubPlace> -->
<xsl:template match="*:fileDesc/*:pubStmt[*:respStmt/*:corpName/*:address]">
  <xsl:copy>
    <xsl:apply-templates select="@*"/>
    <xsl:apply-templates/>
    <pubPlace>
      <xsl:copy-of select="*:respStmt/*:corpName/*:address"/>
    </pubPlace>
  </xsl:copy>
</xsl:template>
```

- Refer to the last element (before the predicate)
- Shallow copy vs. deep copy

5. Move an element

Step 1: Copy & paste <address> to a new place

```
<!-- Copy <address> from <respStmt> to <pubPlace> -->
<xsl:template match="*:fileDesc/*:pubStmt[*:respStmt/*:corpName/*:address]">
  <xsl:copy>
    <xsl:apply-templates select="@*"/>
    <xsl:apply-templates/>
    <pubPlace>
      <xsl:copy-of select="*:respStmt/*:corpName/*:address"/>
    </pubPlace>
  </xsl:copy>
</xsl:template>
```

Take pubStmt, copy the element (shallow copy), then copy everything inside (by applying the copy template), and add the new pubPlace element at the end (of pubStmt) and make it contain a deep copy of everything in address.

5. Move an element

Step 2: Delete original <address>

```
<!-- Delete <address> from <corpName> -->
<xsl:template match="*:corpName[*:address]">
  <xsl:copy>
    <xsl:copy-of select="@*"/>
    <xsl:apply-templates select="*[not(local-name() eq 'address')]"/>
  </xsl:copy>
</xsl:template>
```

5. Move an element

Step 2: Delete original <address>

```
<!-- Delete <address> from <corpName> -->
<xsl:template match="*:corpName[*:address]">
  <xsl:copy>
    <xsl:copy-of select="@*"/>
    <xsl:apply-templates select="*[not(local-name() eq 'address')]"/>
  </xsl:copy>
</xsl:template>
```

- Refer to the last element (before the predicate)
- Shallow copy vs. deep copy

5. Move an element

Step 2: Delete original <address>

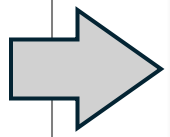
```
<!-- Delete <address> from <corpName> -->
<xsl:template match="*:corpName[*:address]">
  <xsl:copy>
    <xsl:copy-of select="@*"/>
    <xsl:apply-templates select="*[not(local-name() eq 'address')]"/>
  </xsl:copy>
</xsl:template>
```

Take corpName, copy the element (shallow copy), make a deep copy of all its attributes (i.e., attributes and values) and apply the copy template to all its child elements except for <address>

6. Rewrite the text of an element

Situation:

```
<bibl label="" xml:id="bibl_ec2da090">
  <genre xml:id="genre_N234FE">article</genre>
  <genre xml:id="genre_N23500">book</genre>
  <author …>…</author>
  <title>…</title>
  <editor>…</editor>
  …
</bilb>
```



Change to **chapter**

6. Rewrite the text of an element

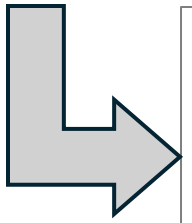
Solution:

```
<!-- Replace "article" with "chapter" when genre = "book" -->
<xsl:template match="*:bibl/*:genre[. = 'article']">
  <xsl:if test="../*:genre = 'book'">
    <genre>
      <xsl:copy-of select="@*"/>
      <xsl:text>chapter</xsl:text>
    </genre>
  </xsl:if>
</xsl:template>
```

7. Unwrap content

Situation: One contributor, with multiple persNames

```
<work xml:id="work_d1e45064422">
  …
  <contributor xml:id="respStmt_N20DEC">
    <persName role="composer" xml:id="persName_054b457c">Carl Nielsen</persName>
    <persName role="author" xml:id="persName_fb8a90b3">Einar Christiansen</persName>
  </contributor>
  …
</work>
```



Remove <contributor> but keep its children within <work> (unwrap them), with a slight change for v5.0:

```
  <composer xml:id="persName_054b457c">Carl Nielsen</composer>
  <author xml:id="persName_fb8a90b3">Einar Christiansen</author>
```

7. Unwrap content

- Example of xsl:for-each

```
<!-- When <contributor> contains more than one <persName>,
    create role-based elements -->
<xsl:template match="*:contributor[count(*:persName) > 1]">
  <xsl:for-each select="*:persName">
    <xsl:element name="{@role}">
      <xsl:copy-of select="@*[not(local-name() eq 'role')]"/>
      <xsl:apply-templates/>
    </xsl:element>
  </xsl:for-each>
</xsl:template>
```

7. Unwrap content

- Example of xsl:for-each

```
<!-- When <contributor> contains more than one <persName>,
    create role-based elements -->
<xsl:template match="*:contributor[count(*:persName) > 1]">
  <xsl:for-each select="*:persName">
    <xsl:element name="{@role}">
      <xsl:copy-of select="@*[not(local-name() eq 'role')]"/>
      <xsl:apply-templates/>
    </xsl:element>
  </xsl:for-each>
</xsl:template>
```

- Refer to the last element (before the predicate)
- We match <contributor> (but we don't copy it)
- We create an element, with the name of the value of @role (e.g., composer or author)
- And do a deep-copy of the rest of the attributes of persName into this new element

8. Add attribute

- Situation:
  - Multiple titles
  - Some have @type="subordinate"
  - **Add @type="main" to the main titles**

```
<work>
  …
  <title xml:id="title_3cac2ce0" xml:lang="da">Saul og David</title>
  <title xml:id="title_3143a854" xml:lang="en">Saul and David</title>
  <title type="subordinate" xml:id="title_d86e4bbb" xml:lang="da">Opera i fire akter</title>
  <title type="subordinate" xml:id="title_ef212608" xml:lang="en">Opera in Four Acts</title>
  …
</work>
```

8. Add attribute

- Solution:

```
<!-- Add type="main" to work titles without @type -->
<xsl:template match="*:work/*:title[not(@type)]">
  <xsl:copy>
    <xsl:attribute name="type">main</xsl:attribute>
    <xsl:apply-templates select="@*"/>
    <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>
```

- Refer to the last element (before the predicate)
- Shallow copy of the element (<title>)
- Create new attribute: @type=main
- And apply the copy-template to all its attributes and children

9. Add element as a child

- **Situation:** Finally, add a <change> element to the end of <revisionDesc> summarizing the modifications done. Should look like this

```
<change isodate="2025-05-30+02:00" xml:id="change_d1e11469">
  <respStmt>
    <name>MEC 2025 XPath/XSLT tutorial</name>
  </respStmt>
  <changeDesc>
    <p>Adjustments for MEI 5.0 validation</p>
  </changeDesc>
</change>
```

9. Add element as child

```
<!-- Add <change> to <revisionDesc> -->
<xsl:template match="*:revisionDesc">
  <xsl:copy>
    <xsl:apply-templates select="@*"/>
    <xsl:apply-templates/>
    <change isodate="{current-date()}" xml:id="{concat('change_', generate-id())}">
      <respStmt>
        <name>MEC 2025 XPath/XSLT tutorial</name>
      </respStmt>
      <changeDesc>
        <p>Adjustments for MEI 5.0 validation</p>
      </changeDesc>
    </change>
  </xsl:copy>
</xsl:template>
```

Summary: cnwFix_complete.xsl

1. Start template
2. Copy template
3. Change an attribute’s value
4. Remove empty attributes
5. Move an element (copy-paste + delete original)
6. Rewrite the text of an element
7. Unwrap content (example of xsl:for-each)
8. Add an attribute
9. Add an element (as child)

XSLT

Conversion from MEI to HTML

(Same) Metadata Example

The output of the previous MEI to MEI processing

Resources

https://github.com/martha-thomae/XSLT-examples-for-MEI

- **File to process (input):** Output of the MEI-to-MEI conversion (CNW01_fixed.xml)
- **Steps:** 2_MEI-to-HTML folder
- **Final XSLT file:** cnwWork2Html_complete.xsl

Saul and David: Opera in Four Acts

Saul og David: Opera i fire akter

Identifier(s)

- 1 (CNW)
- I/4–5 (CNU)
- 330 (CNS)
- 25 (FS)

Composer

Carl Nielsen

Author

Einar Christiansen

Creation

1899–1901

History

At the end of 1896, when Nielsen had finished the choral work *Hymnus Amoris* (CNW 100), he began to plan an opera. The text for *Saul og David*, written by Einar Christiansen, was ready in 1899. A few months seem to have passed before Nielsen started in earnest on the composition. Part of the opera was composed during his stay in Italy from December 1899 to June 1900. The work was finished in April 1901 and accepted for performance at The Royal Theatre in September. Parts of the work were performed at a concert in November 1900.

Bibliography

- Jürgen Balzer. Den dramatiske musik. in Carl Nielsen: i hundredåret for hans fødsel, Jürgen Balzer, ed. Copenhagen: Nyt Nordisk Forlag Arnold Busck, 1965, 71–96.
- Jürgen Balzer. The Dramatic Music. in Carl Nielsen: Centenary Essays, Jürgen Balzer, ed. Copenhagen: Nyt Nordisk Forlag Arnold Busck, 1965, 75–101.
- Gunnar Colding-Jørgensen. Carl Nielsens særpræg som dramatisk komponist dokumenteret gennem en analyse af "Saul og David"

We will create a webpage with the information contained in the <work> (same MEI file)

- Title
- Identifiers
- Composer
- Author
- Creation
- History
- Biography (only books)

Saul and David: Opera in Four Acts

Saul og David: Opera i fire akter

Identifier(s)

- 1 (CNW)
- I/4–5 (CNU)
- 330 (CNS)
- 25 (FS)

Composer

Carl Nielsen

Author

Einar Christiansen

Creation

1899–1901

History

At the end of 1896, when Nielsen had finished the choral work *Hymnus Amoris* (CNW 100), he began to plan an opera. The text for *Saul og David*, written by Einar Christiansen, was ready in 1899. A few months seem to have passed before Nielsen started in earnest on the composition. Part of the opera was composed during his stay in Italy from December 1899 to June 1900. The work was finished in April 1901 and accepted for performance at The Royal Theatre in September. Parts of the work were performed at a concert in November 1900.

Bibliography

- Jürgen Balzer. Den dramatiske musik. in Carl Nielsen: i hundredåret for hans fødsel, Jürgen Balzer, ed. Copenhagen: Nyt Nordisk Forlag Arnold Busck, 1965, 71–96.
- Jürgen Balzer. The Dramatic Music. in Carl Nielsen: Centenary Essays, Jürgen Balzer, ed. Copenhagen: Nyt Nordisk Forlag Arnold Busck, 1965, 75–101.
- Gunnar Colding-Jørgensen. Carl Nielsens særpræg som dramatisk komponist dokumenteret gennem en analyse af "Saul og David"

We will create a webpage with the information contained in the <work> (same MEI file)

- Title
- Identifiers
- Composer
- Author
- Creation
- History
- Biography (only books)

Some Initial Changes:

- XML namespace (@xmlns) in <xsl:stylesheet>
  - For MEI output: xmlns = "http://www.music-encoding.org/ns/mei"
  - For HTML output: xmlns = "http://www.w3.org/1999/xhtml"

- The @method in <xsl:output>
  - For MEI output: method = "xml"
  - For HTML output: method = "xhtml"

Saul and David: Opera in Four Acts

Saul og David: Opera i fire akter

Identifier(s)

- 1 (CNW)
- I/4–5 (CNU)
- 330 (CNS)
- 25 (FS)

Composer

Carl Nielsen

Author

Einar Christiansen

Creation

1899–1901

History

At the end of 1896, when Nielsen had finished the choral work *Hymnus Amoris* (CNW 100), he began to plan an opera. The text for *Saul og David*, written by Einar Christiansen, was ready in 1899. A few months seem to have passed before Nielsen started in earnest on the composition. Part of the opera was composed during his stay in Italy from December 1899 to June 1900. The work was finished in April 1901 and accepted for performance at The Royal Theatre in September. Parts of the work were performed at a concert in November 1900.

Bibliography

- Jürgen Balzer. Den dramatiske musik. in Carl Nielsen: i hundredåret for hans fødsel, Jürgen Balzer, ed. Copenhagen: Nyt Nordisk Forlag Arnold Busck, 1965, 71–96.
- Jürgen Balzer. The Dramatic Music. in Carl Nielsen: Centenary Essays, Jürgen Balzer, ed. Copenhagen: Nyt Nordisk Forlag Arnold Busck, 1965, 75–101.
- Gunnar Colding-Jørgensen. Carl Nielsens særpræg som dramatisk komponist dokumenteret gennem en analyse af "Saul og David"

We will create a webpage with the information contained in the <work> (same MEI file)

- Title
- Identifiers
- Composer
- Author
- Creation
- History
- Biography (only books)

We Will Make a Work Page

Only going to process the <work> element, so:

- We need to apply templates to that (to work)

```
<xsl:template match="/">
  <xsl:apply-templates select="//*:work"/>
</xsl:template>
```

```
<xsl:template match="/">
  <xsl:apply-templates/>
</xsl:template>
```

old starting template

- And, therefore, we also need a template that matches the work

```
<xsl:template match="*:work">
  …
</xsl:template>
```

We Will Make a Work Page

Only going to process the <work> element, so:

- We need to apply templates to that (to work)

```
<xsl:template match="/">
  <xsl:apply-templates select="//*:work"/>
</xsl:template>
```

```
<xsl:template match="/">
  <xsl:apply-templates/>
</xsl:template>
```

old starting template

- And, therefore, we also need a template that matches the work

```
<xsl:template match="*:work">
  …
</xsl:template>
```

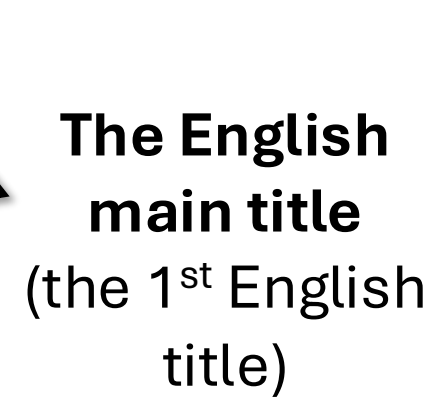
What goes inside this template?

Fill in the Template

```
<xsl:template match="*:work">
  ...
</xsl:template>
```

```
<xsl:template match="*:work">
    <html>
      <head></head>
      <body></body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

```
<xsl:template match="*:work">
    <html>
      <head>
        <title> TEXT FOR TITLE </title>
      </head>
      <body></body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```



```
<xsl:template match="*:work">
    <html>
      <head>
        <title>
          <xsl:value-of select="*:title[@xml:lang = 'en'][position() = 1]"/>
        </title>
      </head>
      <body></body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

```
<xsl:template match="*:work">
    <html>
      <head>
        <title>
          <xsl:value-of select="*:title[@xml:lang = 'en'][position() = 1]"/>
        </title>
      </head>
      <body></body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Now, let's fill in the <body>

Saul and David: Opera in Four Acts

Saul og David: Opera i fire akter

Identifier(s)

- 1 (CNW)
- I/4–5 (CNU)
- 330 (CNS)
- 25 (FS)

Composer

Carl Nielsen

Author

Einar Christiansen

Creation

1899–1901

History

At the end of 1896, when Nielsen had finished the choral work *Hymnus Amoris* (CNW 100), he began to plan an opera. The text for *Saul og David*, written by Einar Christiansen, was ready in 1899. A few months seem to have passed before Nielsen started in earnest on the composition. Part of the opera was composed during his stay in Italy from December 1899 to June 1900. The work was finished in April 1901 and accepted for performance at The Royal Theatre in September. Parts of the work were performed at a concert in November 1900.

Bibliography

- Jürgen Balzer. Den dramatiske musik. in Carl Nielsen: i hundredåret for hans fødsel, Jürgen Balzer, ed. Copenhagen: Nyt Nordisk Forlag Arnold Busck, 1965, 71–96.
- Jürgen Balzer. The Dramatic Music. in Carl Nielsen: Centenary Essays, Jürgen Balzer, ed. Copenhagen: Nyt Nordisk Forlag Arnold Busck, 1965, 75–101.
- Gunnar Colding-Jørgensen. Carl Nielsens særpræg som dramatisk komponist dokumenteret gennem en analyse af "Saul og David"

We will create a webpage with the information contained in the <work> (same MEI file)

- Title
- Identifiers
- Composer
- Author
- Creation
- History
- Biography (only books)

Saul and David: Opera in Four Acts

<h1> *Title in English*

Saul og David: Opera i fire akter

<h2> *Title in Danish*

Identifier(s)

<h2>

1 (CNW)
...

- 1 (CNW)
- I/4–5 (CNU)
- 330 (CNS)
- 25 (FS)

Composer

Carl Nielsen

Author

Einar Christiansen

Creation

1899–1901

<h2>Composer</h2>
Carl Nielsen

<h2>Author</h2>
Einar Christiansen

<h2>Creation</h2>
1899–1901

<h2> + with formatting
(for links <a> and italics <i>)

History

At the end of 1896, when Nielsen had finished the choral work *Hymnus Amoris* (CNW 100), he began to plan an opera. The text for *Saul og David*, written by Einar Christiansen, was ready in 1899. A few months seem to have passed before Nielsen started in earnest on the composition. Part of the opera was composed during his stay in Italy from December 1899 to June 1900. The work was finished in April 1901 and accepted for performance at The Royal Theatre in September. Parts of the work were performed at a concert in November 1900.

Bibliography

<h2> and list (<ul> with <li> children, with various information)

- Jürgen Balzer. Den dramatiske musik. in Carl Nielsen: i hundr
Forlag Arnold Busck, 1965, 71–96.
- Jürgen Balzer. The Dramatic Music. in Carl Nielsen: Centenary Essays, Jürgen Balzer, ed. Copenhagen: Nyt Nordisk Forlag Arnold Busck, 1965, 75–101.
- Gunnar Colding-Jørgensen. Carl Nielsens særpræg som dramatisk komponist dokumenteret gennem en analyse af "Saul og David"

We will create a webpage with the information contained in the <work>
(same MEI file)

- Title
- Identifiers
- Composer
- Author
- Creation
- History
- Biography (only books)

Saul and David: Opera in Four Acts

<h1> *Title in English*

Saul og David: Opera i fire akter

<h2> *Title in Danish*

Identifier(s)

<h2>

- 1 (CNW)
- I/4–5 (CNU)
- 330 (CNS)
- 25 (FS)

```
<ul>
  <li>1 (CNW)</li>
  ...
</ul>
```

Composer

Carl Nielsen

Author

Einar Christiansen

Creation

1899–1901

```
<h2>Composer</h2>
<p>Carl Nielsen</p>

<h2>Author</h2>
<p>Einar Christiansen</p>

<h2>Creation</h2>
<p>1899–1901</p>
```

History

<h2> + <p> with formatting (for links <a> and italics <i>)

At the end of 1896, when Nielsen had finished the choral work *Hymnus Amoris* (CNW 100), he began to plan an opera. The text for *Saul og David*, written by Einar Christiansen, was ready in 1899. A few months seem to have passed before Nielsen started in earnest on the composition. Part of the opera was composed during his stay in Italy from December 1899 to June 1900. The work was finished in April 1901 and accepted for performance at The Royal Theatre in September. Parts of the work were performed at a concert in November 1900.

Bibliography

<h2> and list (<ul> with <li> children, with various information)

- Jürgen Balzer. Den dramatiske musik. in Carl Nielsen: i hundr
  Forlag Arnold Busck, 1965, 71–96.
- Jürgen Balzer. The Dramatic Music. in Carl Nielsen: Centenary Essays, Jürgen Balzer, ed. Copenhagen: Nyt Nordisk Forlag Arnold Busck, 1965, 75–101.
- Gunnar Colding-Jørgensen. Carl Nielsens særpræg som dramatisk komponist dokumenteret gennem en analyse af "Saul og David"

We will create a webpage with the information contained in the <work> (same MEI file)

- Title
- Identifiers
- Composer
- Author
- Creation
- History
- Biography (only books)

```
<work xml:id="work_d1e45064422">
 <identifier label="CNW" xml:id="identifier_N20DCD">1</identifier>
 <identifier label="CNU" xml:id="identifier_N20DD0">I/4–5</identifier>
 <identifier label="CNS" xml:id="identifier_N20DD3">330</identifier>
 <identifier label="FS" xml:id="identifier_N20DD6">25</identifier>
 <title type="main" xml:id="title_3cac2ce0" xml:lang="da">Saul og David</title>
 <title type="main" xml:id="title_3143a854" xml:lang="en">Saul and David</title>
 <title type="subordinate" xml:id="title_d86e4bbb" xml:lang="da">Opera i fire akter</title>
 <title type="subordinate" xml:id="title_ef212608" xml:lang="en">Opera in Four Acts</title>
 <composer xml:id="persName_054b457c">Carl Nielsen</composer>
 <author xml:id="persName_fb8a90b3">Einar Christiansen</author>
 <creation xml:id="creation_d30e7e3c">
   <date notafter="1901" notbefore="1899" xml:id="date_N20DF9">1899</date>
 </creation>
 <history xml:id="history_fec11c69">
  <p xml:id="idm829">At the end of 1896, when Nielsen had finished the choral work <ref
        xmlns:xl=http://www.w3.org/1999/xlink xml:id="idm830"
        target="document.xq?doc=cnw0100.xml" xl:show="replace"><rend xml:id="idm831"
        fontstyle="italic">Hymnus Amoris</rend> (CNW 100)</ref>, he began to plan an opera. The text
        for <rend xml:id="idm832" fontstyle="italic">Saul og David</rend>, written by Einar Christiansen,
        was ready in 1899. A few months seem to have passed before Nielsen started in earnest on the
        composition. Part of the opera was composed during his stay in Italy from December 1899 to
        June 1900. The work was finished in April 1901 and accepted for performance at The Royal
```

```
<creation xml:id="creation_d30e7e3c">
  <date notafter="1901" notbefore="1899" xml:id="date_N20DF9">1899</date>
</creation>
<history xml:id="history_fec11c69">
  <p xml:id="idm829">At the end of 1896, when Nielsen had finished the choral work <ref
      xmlns:xl=http://www.w3.org/1999/xlink xml:id="idm830"
      target="document.xq?doc=cnw0100.xml" xl:show="replace"><rend xml:id="idm831"
      fontstyle="italic">Hymnus Amoris</rend> (CNW 100)</ref>, he began to plan an opera. The text
      for <rend xml:id="idm832" fontstyle="italic">Saul og David</rend>, written by Einar Christiansen,
      was ready in 1899. A few months seem to have passed before Nielsen started in earnest on the
      composition. Part of the opera was composed during his stay in Italy from December 1899 to
      June 1900. The work was finished in April 1901 and accepted for performance at The Royal
      Theatre in September. Parts of the work were performed at a concert in November 1900.</p>
</history>
<bibList xml:id="listBibl_9531b4bb-3110-496d-8f1d-dfc21413f3904482">
  …
  <bibl …>
    <genre>book</genre>
    …
  </bibl>
  …
<bibList>
</work>
```

XSLT

Extracting (pulling) information

Music Example

Resources

https://github.com/martha-thomae/XSLT-examples-for-MEI

- File to process (input): Bach-JS_Ein_feste_Burg.mei
- Folder: 3_meiCensus
- Final XSLT file: meiCensus_complete.xsl

- **Extract information from the MEI file (this is called, a pull)**
- And show it as **text** - - - - - - - - - - >

- The @method in <xsl:output>
  - For MEI output: method = "xml"
  - For HTML output: method = "xhtml"
  - **For text output: method = "text"**

MEI Census (Bach-JS_Ein_feste_Burg.mei)

Number of measures: 16
Number of empty measures: 0
Number of barlines: 16
Number of chords: 0
Number of notes: 260
Number of noteheads: 261
Number of rests: 0
Number of ties: 1
Number of beams: 47
Number of beamed chords: 0
Number of beamed notes: 95
Number of beamed rests: 0
Maximum number of staves: 2
Maximum number of layers: 4
Longest note duration: 2
Shortest note duration: 16
Highest note: d♮5
Lowest note: d♮2

Variable (with a simple value)
Example of other XPath functions

Simple counts

Variable containing mini-xml documents + sorting

Two steps: variable containing mini-xml documents + replacement (numeric values) + sorting

MEI Census (Bach-JS_Ein_feste_Burg.mei)

Number of measures: 16
Number of empty measures: 0
Number of barlines: 16
Number of chords: 0
Number of notes: 260
Number of noteheads: 261
Number of rests: 0
Number of ties: 1
Number of beams: 47
Number of beamed chords: 0
Number of beamed notes: 95
Number of beamed rests: 0
Maximum number of staves: 2
Maximum number of layers: 4
Longest note duration: 2
Shortest note duration: 16
Highest note: d♮5
Lowest note: d♮2

Simple counts

MEI Census (Bach-JS_Ein_feste_Burg.mei)

Number of measures: 16
Number of empty measures: 0
Number of barlines: 16
Number of chords: 0
Number of notes: 260
Number of noteheads: 261
Number of rests: 0
Number of ties: 1
Number of beams: 47
Number of beamed chords: 0
Number of beamed notes: 95
Number of beamed rests: 0
Maximum number of staves: 2
Maximum number of layers: 4
Longest note duration: 2
Shortest note duration: 16
Highest note: d♮5
Lowest note: d♮2

```
<xsl:template match="/">
  <xsl:text>Number of measures: </xsl:text>
  <xsl:value-of select="count(//*:measure)"/>
  …
  <xsl:text>&#xa;Number of chords: </xsl:text>
  <xsl:value-of select="count(//*:chord)"/>
  …
  <xsl:text>&#xa;Number of notes: </xsl:text>
  <xsl:value-of select="count(//*:note) - count(//*:tie) - count(//*:note[@tie])"/>
  …
</xsl:template>
```

Variable (with a simple value)
Example of other XPath functions

MEI Census (Bach-JS_Ein_feste_Burg.mei)

Number of measures: 16
Number of empty measures: 0
Number of barlines: 16
Number of chords: 0
Number of notes: 260
Number of noteheads: 261
Number of rests: 0
Number of ties: 1
Number of beams: 47
Number of beamed chords: 0
Number of beamed notes: 95
Number of beamed rests: 0
Maximum number of staves: 2
Maximum number of layers: 4
Longest note duration: 2
Shortest note duration: 16
Highest note: d♮5
Lowest note: d♮2

```
<xsl:variable name="inputFilename">
  <xsl:value-of select="tokenize(base-uri(.), '/')[last()]"/>
</xsl:variable>

<xsl:template match="/">
  <xsl:text>MEI Census (</xsl:text>
  <xsl:value-of select="$inputFilename"/>
  <xsl:text>)&#xa;&#xa;</xsl:text>
  <xsl:text>Number of measures: </xsl:text>
  …
</xsl:template>
```

Variable containing mini-xml documents + sorting

MEI Census (Bach-JS_Ein_feste_Burg.mei)

Number of measures: 16
Number of empty measures: 0
Number of barlines: 16
Number of chords: 0
Number of notes: 260
Number of noteheads: 261
Number of rests: 0
Number of ties: 1
Number of beams: 47
Number of beamed chords: 0
Number of beamed notes: 95
Number of beamed rests: 0
Maximum number of staves: 2
Maximum number of layers: 4
Longest note duration: 2
Shortest note duration: 16
Highest note: d♮5
Lowest note: d♮2

Maximum Number of Staves:

```
<xsl:variable name="measuresStaves">
  <xsl:for-each select="//*:measure">
    <xsl:sort select="count(*:staff)"/>
    <measure>
      <xsl:attribute name="staves">
        <xsl:value-of select="count(*:staff)"/>
      </xsl:attribute>
    </measure>
  </xsl:for-each>
</xsl:variable>
```

<!-- **To find the maximum number of staves and layers:** Variables containing a mini-xml document to apply sorting and find the minimum and maximum number of something -->

```
<xsl:text>&#xa;Maximum number of staves: </xsl:text>
<xsl:value-of select="$measuresStaves/measure[last()]/@staves"/>
```

Maximum Number of Layers?

```
<xsl:variable name="measuresStaves">
  <xsl:for-each select="//*:measure">
    <xsl:sort select="count(*:staff)"/>
    <measure>
      <xsl:attribute name="staves">
        <xsl:value-of select="count(*:staff)"/>
      </xsl:attribute>
    </measure>
  </xsl:for-each>
</xsl:variable>
```

```
<xsl:variable name="measuresLayers">
  <xsl:for-each select="//*:measure">
    <xsl:sort select="count(descendant::*:layer)"/>
    <measure>
      <xsl:attribute name="layers">
        <xsl:value-of select="count(descendant::*:layer)"/>
      </xsl:attribute>
    </measure>
  </xsl:for-each>
</xsl:variable>
```

```
<xsl:text>&#xa;Maximum number of staves: </xsl:text>
<xsl:value-of select="$measuresStaves/measure[last()]/@staves"/>
<xsl:text>&#xa;Maximum number of layers: </xsl:text>
<xsl:value-of select="$measuresLayers/measure[last()]/@layers"/>
```

Two steps: variable containing mini-xml documents + replacement (numeric values) + sorting

MEI Census (Bach-JS_Ein_feste_Burg.mei)

Number of measures: 16
Number of empty measures: 0
Number of barlines: 16
Number of chords: 0
Number of notes: 260
Number of noteheads: 261
Number of rests: 0
Number of ties: 1
Number of beams: 47
Number of beamed chords: 0
Number of beamed notes: 95
Number of beamed rests: 0
Maximum number of staves: 2
Maximum number of layers: 4
Longest note duration: 2
Shortest note duration: 16
Highest note: d♮5
Lowest note: d♮2

Two steps: variable containing mini-xml documents + replacement (numeric values) + sorting

MEI Census (Bach-JS_Ein_feste_Burg.mei)

Number of measures: 16
Number of empty measures: 0
Number of barlines: 16
Number of chords: 0
Number of notes: 260
Number of noteheads: 261
Number of rests: 0
Number of ties: 1
Number of beams: 47
Number of beamed chords: 0
Number of beamed notes: 95
Number of beamed rests: 0
Maximum number of staves: 2
Maximum number of layers: 4
Longest note duration: 2
Shortest note duration: 16
Highest note: d♮5
Lowest note: d♮2

```
<!-- To find shortest and longest durations:
    Variables containing a mini-xml document to apply sorting and find the minimum and maximum
    number of something (also, use of 'replace' function to facilitate the sorting) -->
<xsl:variable name="sortedDurations">
  <xsl:for-each select="//*:note[@dur] | //*:chord[@dur]">
    <xsl:sort select="replace(replace(replace(replace(replace(replace(
                      replace(replace(replace(replace(replace(replace(
                      @dur, 'breve', '.5'),
                            'long$', '.25'),
                            'longa', '.5'),
                            'maxima', '.25'),
                            '^brevis', '1'),
                            'semibrevis', '2'),
                            '^minima', '4'),
                            'semiminima', '8'),
                            '^fusa', '16'),
                            'semifusa', '32')" data-type="number" order="ascending"/>
    <dur dur="{@dur}">
      <!-- breve and long may occur in CMN, others only in Mensural -->
      <xsl:value-of select="replace(replace(replace(replace(replace(replace(
                            replace(replace(replace(replace(replace(replace(
```

```
                    '^minima', '4'),
                    'semiminima', '8'),
                    '^fusa', '16'),
                    'semifusa', '32')" data-type="number" order="ascending"/>
  <dur dur="{@dur}">
    <!-- breve and long may occur in CMN, others only in Mensural -->
    <xsl:value-of select="replace(replace(replace(replace(replace(
                          replace(replace(replace(replace(replace(
                          @dur, 'breve', '.5'),
                                'long$', '.25'),
                                'longa', '.5'),
                                'maxima', '.25'),
                                '^brevis', '1'),
                                'semibrevis', '2'),
                                '^minima', '4'),
                                'semiminima', '8'),
                                '^fusa', '16'),
                                'semifusa', '32')"/>
  </dur>
 </xsl:for-each>
</xsl:variable>
```

New set of elements that only exist in the variable

These are `<dur>` elements that have the original duration and the replaced one:

`<dur dur="brevis">.5</dur>`

The variable `$sortedDurations` is a tree.

Therefore, one can navigate it using XPath expressions to get to the first and last `<dur>` elements (which correspond to the longest and shortest durations given the sorting)

```
<!-- Results -->
<xsl:text>&#xa;Longest note duration: </xsl:text>
<xsl:value-of select="$sortedDurations/*:dur[1]/@dur"/>
<xsl:text>&#xa;Shortest note duration: </xsl:text>
<xsl:value-of select="$sortedDurations/*:dur[last()]/@dur"/>
```

XSLT

Pulling the music information into a TSV Table

Extra Example

Resources

https://github.com/martha-thomae/XSLT-examples-for-MEI

- File to process (input): Bach-JS_Ein_feste_Burg.mei
- Folder: 4_meiPitchDistribution

Applying XSLT **meiPitchDistribution**

to Bach-JS_Ein_feste_Burg.mei

Pitch Name	MIDI key	Pitch Class Number	Pitch Name Count
d♮2	26	2	1
g♮2	31	7	1
a♮2	33	9	5
b♮2	35	11	5
c♮3	36	0	5
d♮3	38	2	12
d♯3	39	3	1
e♮3	40	4	17
f♮3	41	5	18
g♮3	43	7	11
g♯3	44	8	2
a♮3	45	9	14
a♯3	46	10	1
b♮3	47	11	14
c♮4	48	0	13
d♮4	50	2	29
d♯4	51	3	2
e♮4	52	4	25
f♮4	53	5	22
g♮4	55	7	9
g♯4	56	8	3
a♮4	57	9	18
b♮4	59	11	14
c♮5	60	0	8
d♮5	62	2	11

Applying XSLT **meiPitchDistribution_02**

to Bach-JS_Ein_feste_Burg.mei

Pitch Class Num	Pitch Class Name	Pitch Class Count
2	D♮	53
4	E♮/F♭	42
5	E#/F♮	40
9	A♮	37
11	B♮	33
0	C♮/B#	26
7	G♮	21
8	G#/A♭	5
3	D#/E♭	3
10	A#/B♭	1

Applying XSLT **meiPitchDistribution_02**

to Webern_Variations_for_Piano_Op27_No2.mei

Pitch Class Num	Pitch Class Name	Pitch Class Count
9	A♮	6
8	G#/A♭	5
10	A#/B♭	5
5	E#/F♮	4
1	C#/D♭	4
11	B♮	4
2	D♮	4
7	G♮	4
4	E♮/F♭	4
6	F#/G♭	4
0	C♮/B#	4
3	D#/E♭	2

Thank you!