**XSLT Orientation (3/30/2018)**

**General Overview**

XSLT is used to transform [FITS xml output](https://projects.iq.harvard.edu/fits/fits-xml) into the master.xml file. It takes two stylesheets to do this. The first cleans up the FITS xml to make it easier to work with. The second transforms the cleaned up FITS output into the master.xml, which is mostly PREMIS. There are two versions of this second stylesheet, one for if the AIP contains multiple files and the other for it contains a single file.

FITS produces a separate xml document for every file within a AIP. The aip-finish\_script includes steps to create a single xml document from these separate files prior to running the stylesheets.

**XSLT we are using**

* **fits-cleanup.xsl:**  makes FITS output easier to work with.
  + Removes empty elements so that each element does not need to be checked for if it has content before making the PREMIS element for it.
  + Makes the order of children consistent to simplify the selection of a sibling. If the order is not consistent, then have to check for both a preceding and following sibling.
  + Creates one FITS identity section for each possible version of a format so all format information can be treated the same way when transforming it to PREMIS.
  + Adds structure to the creating application and inhibitor information to make it easier to remove duplicates and to keep related information together when different tools have different results.
* **fits-to-master\_multifile.xsl and fits-to-master\_singlefile.xsl:**  turn FITS into master.xml (PREMIS and 2 Dublin Core fields) for AIPs with multiple files or a single file respectively. The aip-finish\_script determines which stylesheet is used. The main differences are that the single file master.xml does not include the filelist section and has a premis:objectCategory of file in the aip section instead of representation.

**FITS Overview**

FITS was selected to generate the technical metadata because it produces detailed information, the output is structured (XML) and therefore can be manipulated, and it runs from the command line so it could be integrated into the script. FITS natively includes all the information needed for the master.xml except for AIP title and identifiers. This information was added to the AIP folder title so that it would be in the file path, which FITS does capture, so the master.xml could be automatically generated from the FITS output without requiring staff to manually edit the documents.

FITS combines the outputs from several tools and so there are many times when tools disagree on an identification. They may suggest different format names, different versions of the format, different creating application dates, etc. The UGA Libraries does not currently have the staff time or expertise to analyze these conflicts and determine which are more accurate so the conflicting data is all kept in the master.xml to capture the most complete information available about these files.

**Document Structure**

Both stylesheets set FITS as the xpath-default-namespace (set in the xsl:stylesheet element) so that the fits: prefix does not need to be used in XPaths. This shortens the XPaths and makes them easier to read, and eliminates a common error (leaving off the prefix) that staff were making.

**fits-cleanup.xsl**

The first template, <xsl:template match="node()|@\*">, copies the entire combined-fits XML input into the output document. The other templates match components of the XML and replace it with something else. Because of template priority rules, if there is a template matching a specific component, it is applied. If there is not, the more general first template is applied. There is a template that matches all of the sections in the combined-fits. The first template is maintaining the overall structure (combined-fits and fits elements).

For the most part, the templates are applied in place: elements are copied over, maybe with some changes, in the same location in the output that they are in the input. This is different from the fits-to-master.xsl, which has to call or apply templates to indicate where the output should be.

However, there is one template in the fits-cleanup.xsl that is applied in several locations to avoid repeating the same XSLT over and over. This is a template that only copies an element if it contains a value. The template matches all of the elements that need this test: <xsl:template match="version | filepath | size | md5checksum | valid | well-formed>. Then the template can be applied for any one of those elements, e.g. <xsl:apply-templates select="valid">.

**fits-to-master.xsl**

The first template, <xsl:template match="/">, creates the organization of the output document by calling or applying templates, supplying wrapper elements such as master and premis:object, and inserting the value of a variable for aip-title. The rest of the stylesheet is the templates called or applied in this root template and additional templates and variables used within those templates.

The stylesheet is broken up into many templates to make them easier to maintain. For the most part, each template corresponds to a single main PREMIS element. If a PREMIS element is complex, there may be additional templates to create components of it. This way, when changes need to be made for an element, only a small portion of the XSLT needs to altered and tested.

For PREMIS elements that need to be calculated for each file (most of the filelist section), apply-templates is used. The template has a match attribute which gives the XPath for what the template works on. For example, <xsl:template match="size"> will act on the size elements. Then an <xsl:apply-templates> with a select attribute that has a matching XPath indicates where the result of the template should be in the output document. Apply-templates has two advantages for repeating elements that keep the code simpler: it already acts on each instance of an element (eliminating the need for a <xsl:for-each>) and it only acts on an element when it is present (eliminating the need to test for the presence of optional elements).

For PREMIS elements that only have a single value for the master.xml document, for example the aip-id or the aip-size, call-templates is used. The template has a name attribute and contains the XSLT to construct that element. Then <xsl:call-template> with a matching name attribute is used to indicate where the result of that template should be in the output document. All of the templates for the aip section, as well as file-id and relationship-aip in the filelist section, use this structure.

**Regular Expressions (regex)**

To be able to make master.xml files for a batch of AIPs automatically, we needed a way to calculate the collection, aip, and file identifiers and the aip title from the files themselves. This was accomplished by including the information in the folder title (structured aip-id\_Title of the AIP). The FITS output includes the file path, so the correct portion of the filepath is selected using regular expressions and assigned to a variable. Then the variables can be called wherever they are needed throughout the XSLT.

* Collection identifier is the first part of the aip-id.
* AIP identifier is before the underscore.
* AIP title is after the underscore.
* File identifier is the entire file path starting with the aip-id.

Regex is also used to change date created values into the required format based on their pattern.

The general structure for using regular expressions in XSLT is:

<xsl:analyze-string select="XPath" regex="regular expression">

<xsl:matching-substring>xslt for what to do with the match</xsl:matching-substring>

</xsl:analyze-string>

**Variables**

Variables are used to calculate the value of something once (for example, using a regular expression or an XPath) and then it can be referred to throughout the stylesheet. Variables can apply to the entire stylesheet by setting their value at the same level as the templates (global). Variables can also just apply to a single template by setting their value within that template (local).

A variable's value is set with <xsl:variable name="variablename">xslt to calculcate</xsl:variable>. Then it is referred to with the syntax $variablename.

Global variables are used for the identifiers and aip-title and are located towards the top of the fits-to-master stylesheets. Local variables are used for two reasons:

* To compare an element to another one and match them by related criteria. For example, in <xsl:template match="identity">, the templates for premis:formatNote element for valid and well-formed should only be applied if they were generated by the same tool that identified the format. So a variable $tool is set to be the tool that identified a format, and then the valid and well-formed templates are only applied if their tool is equal to the value of $tool:

<xsl:apply-templates select="../following-sibling::filestatus/valid[@toolname=$tool]" />

* To select the value of an XPath that needs to be referred to throughout a template to make it easier to read. This may be a longer XPath but most often is the value of the current node. Current node can be represented with a period, but that can be hard to see within an XPath. For example, in <xsl:template match="md5checksum">, XPaths are formatted like "$md5/toolname" rather than ./toolname.

**Parameters**

A parameter works like a variable except that the value for the parameter is supplied from outside of the stylesheet. The name of the parameter is set using <xsl:param name="param-name"/>. Then it is referred to with the same syntax as a variable, $param-name.

In fits-to-master.xsl, a parameter is supplied from the aip-finish\_script with the department name, which corresponds to the group identifier in the ARCHive. This is used to construct the ARCHive uri's used in PREMIS identifiers as the objectIdentifierType and it is used to determine which regular expressions to use to extract the value of the identifiers and aip-title from the filepath. This allows Hargrett and Russell to use the same stylesheets, which simplifies maintenance.

**XPath Common Navigation**

* Use brackets to select an element that contains a child, attribute, and/or just has a value. This is used instead of doing an if test to see if these are present for simpler XSLT.
  + inhibitor[inhibitorType] selects element inhibitor if it has a child inhibitorType
  + externalIdentifier[@type='puid'][string()] selects element externalIdentifier if it has a type attribute with the value puid and it has content, tested by string().
  + tool[not(@toolname='')] selects element tool that does not have an attribute toolname which is empty (that is 2 single quotes, not a double quote).
* Navigate up one step in the XML document.
  + ../@format selects the attribute format of the parent of the current node.

**Built-in error detection**

The FITS output can have a lot of variation, both from the number of tools it compiles information from and the wide variety of formats that we analyze. The XSLT accounts for all variations observed so far, but whenever possible we want the XSLT to alert us when a new variation occurs. Otherwise, it may produce information in the master.xml that does not make sense or it may not include useful information in the master.xml because it didn't match a template.

To do this, when we anticipate it is likely that there could be new variations, the XSLT is configured to create an invalid master.xml if a new variation appears, generally by including an empty PREMIS element. This way, the AIP is flagged as having a problem and staff know to research it and update the XSLT if needed. Examples:

Within <xsl:template match="identity">, <xsl:if test="not(tool)"><premis:formatNote /></xsl:if> makes an empty premis:formatNote if there is no identifying tool present.

Within <xsl:template match="created">, there is an <xsl:choose> that selects how dates of different formats should be handled. This includes an <xsl:otherwise> to create an invalid premis:dateCreatedByApplication element if there are any dates that do not match any of the <xsl:when> patterns:

<xsl:otherwise>

<premis:dateCreatedByApplication>New Date Format Identified: Update Stylesheet</premis:dateCreatedByApplication>

</xsl:otherwise>

A manual quality control process is still necessary to look for variations that were not anticipated.

**For-Each-Group**

Both stylesheets make use of <xsl:for-each-group> to bring together elements based on shared criteria and then act on them as a group, which is new with XSLT 2.0. [See this orientation](https://www.xml.com/pub/a/2003/11/05/tr.html) for how it works in general. In fits-cleanup.xsl, it is used to group sibling creating application or inhibitor elements identified by the same tool into a new parent element. In fits-to-master.xsl, it is used to create unique lists of file formats, creating applications, and inhibitors.

The structure used is slightly different between the two stylesheets because they are doing different things. In fits-cleanup.xsl, the group is made by selecting multiple sibling elements and grouping them by the same single criteria, the tool name. Then each group can be acted on:

<xsl:for-each-group select="xpath1 | xpath 2" group-by="criteria">

<new-parent>

<xsl:for-each select="current-group()">

<xsl:sort select="name()"/>

xslt for what to do with the group

</xsl:for-each>

</new-parent>

</xsl:for-each-group>

In fits-to-master.xsl, the group is made by selecting a single element and then grouping them by multiple criteria (for example, the same name and version). Then when the children of each group is acted on, there is only one output per group, which results in removing duplicates. It will give an error if anything selected has more than one element that matches the criteria, which is one of the main reasons the fits-cleanup stylesheet is necessary.

<xsl:for-each-group select="XPath" group-by="concat(criteria1,criteria2)">

<xsl:sort select="current-grouping-key()" />

xslt for what to do with the group's children goes here (XPath in select is the current node)

</xsl:for-each-group>