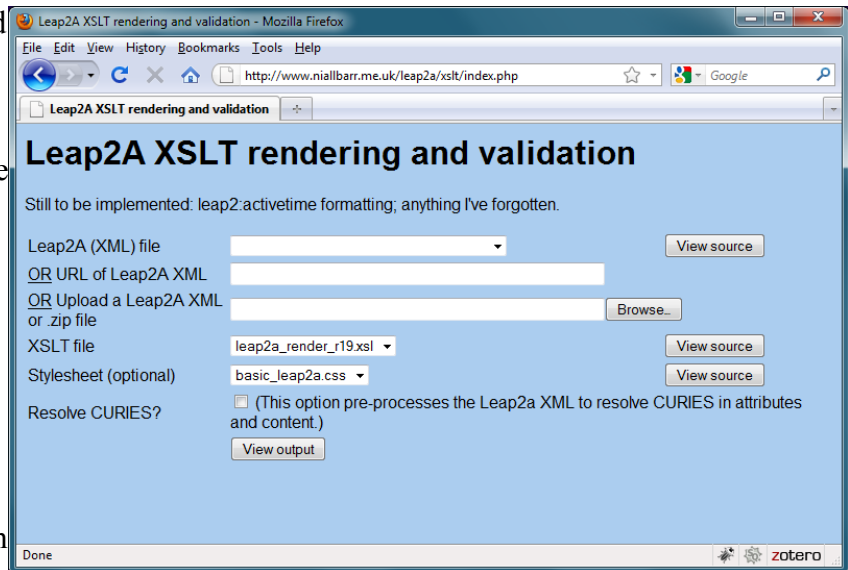


The test site

A test website has been developed that allows users to run the rendering and validation toolkit on a number of example Leap2a files as well as their own files. The test site renders the output of the validator to the user's web browser. To make use of this site the user either selects one of the demonstration files from the top drop-down list, or puts the full URL of a publicly available Leapfile into the text box immediately below the drop list, or selects an XML or zip file from the local disk using the file select



below that. Next the user is able to select an XSL file to perform rendering and validation, and a CSS file to style the rendered output using drop lists. A final option allows the Leap2a XML to be preprocessed to resolve CURIRES. The user then clicks on the view output button, rendering and validation XSL is used to process and display the Leap2a input.

If an XML file has been used the input, only the Leap2a XML is extracted (for security reasons), and so other files within that the package will not be displayed.

There are three buttons labelled "view source" which allows users to see the source of example Leap2a, XSL and CSS files held on the system.

Installation.

The demonstration website is written in PHP 5, and has been tested under Apache and IIS servers. To install it on a different server, the server requires to have an up-to-date installation of PHP with XSLT and ZIP enabled. A temporary directory that is writeable by the web server process is also required.

The software needs to be copied onto the web server with directory structure from the distribution. It can be located in a subdirectory at any depth. A single line of code in the file index.php needs to be edited to set the path to a temporary directory that must be writable by the web server process. (Because the location of this directory will be visible to users it should be one that does not give out any information about the web server or its users. For example '/var/leap2a_tmp' would be an appropriate directory, however '/home/myname/Desktop/tmp' would not be recommended.)

The demonstration site is currently hosted at <http://www.niallbarr.me.uk/leap2a/xslt/index.php>

Using the XSLT offline

The XSL rendering and validation XSLT can be used offline by adding a reference to it at the top of the Leap2a XML file:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="leap2a_render_r19.xsl"?>
<feed ...
```

This allows any modern web browser to use the XSLT to render the XML.

The XSLT file, leap2a_render_r19.xsl, needs to be held in the same location as the Leap2a file, as the W3C specification for using XSLT to transform XML for browsers requires the XML and XSLT to be in the same domain.

Changing the appearance of the output

Any major change to the output will require major rewriting of the XSLT, however significant changes can be made by editing the CSS file that is used for the demonstration site, basic_leap2a.css.

The HTML output includes a number of class names that can be used to alter the appearance of the output. These are:

- .intro
- .leap_notes
- .parthead
- .error
- .warning
- .persondata
- .orgdata
- .info
- .severewarning

In addition, the of the HTML table, tr and td elements can be styled.

The XSL generator (µtron)

To speed the development of the rendering and validation toolkit, a simple code generator was developed loosely based on Schematron, however with very much simpler rules, and with the ability to generate and framework structure for XSL that would carry out both rendering and validation at the same time. The code generator is written in PHP and runs on a web server, however it would be very simple to create a commandline version. The input for the code generator is a very simplified version of RelaxNG (which has been named xdef) that defines each element that is to be provided with its own xsl:template section. Elements that are used within other elements, but are not declared individually or processed in line with in the parent element's template section. In addition to the usual modifiers of *, + and ? meaning zero more, one or more and optional there is a fourth modifier -, which is used to indicate that an element is deprecated. Name spaces that will be used in the document are declared at the top of the xdef input file. Below the element declarations sections of XSL code that carry out more specific tasks than can be carried out by automatically generated code are placed in #define sections. Each define section starts with an identifier consisting of the element that is processed by the template where the XSL fragment will be placed, followed by either the child element that is being processed or an identifier for the location. The best way to identify these define names is to generate XSL, which will indicate where defined fragments of XSL can be included, and the appropriate define name in comments within the XSL output. The names can include trailing wildcards, in which case they are used as a default if a more precise definition does not exist. The when wildcards are used defined code can also include a few PHP style variable names that customizes the XSLT fragment to the specific context where it is used.

Possible defines:

```
#define MISSINGELEMENT_parent/child
```

Defines a section of XSLT for a missing required child in element. A * wildcard can be used to replace either the child element name or both the parent and child. The PHP variables {Selement} and {\$child} can be used to display the names of the parent and child elements.

```
#define UNEXPECTEDCHILD_parent
```

Defines a section of XSLT for an unexpected child element in parent. A * wildcard can be used to replace the parent element name. The PHP variables {\$element} and {\$child} can be used to display the names of the parent and child elements.

```
#define element@attribute
```

Defines a section of XSLT to be used for attribute of element. A * wildcard can be used to make it match any attribute, and the PHP variable {\$attribute} can be used to represent the attribute name.

```
#define parent/child
```

XSLT fragment for the element child inside the template for parent. (The XSLT will need to be appropriate for the cardinality of child.) A * wildcard can be used to replace either the child element name or both the parent and child. The PHP variables {\$element} and {\$child} can be used to display the names of the parent and child elements.

```
#define parent_before_child
```

```
#define parent_after_child
```

Sections to be used inside the template for parent just before and just after applying the template for child. (Both parent and child should be defined with element declarations.)

```
#define elementname_start
```

A fragment of XSLT that will go at the start of the template for element <elementname>.

```
UNEXPECTEDCONTENT_elementname
```

Defines the XSLT fragment used where content is found in an element <elementname> that is expected to be empty.

```
#define OtherSectionsOfXSL
```

/OtherSectionsOfXSL allows extra templates that are not generated by microtron to be included.

It is strongly recommended that rather than editing the very large Leap2a rendering and validation XSL file, the xdef file is modified, and the rendering and validation XSL is regenerated from that.

The code generator is available online at

http://www.niallbarr.me.uk/wizards/xsl_validation/index.php

Licence

The Leap2a rendering and validation toolkit is copyright © 2010, The University of Glasgow, and is released under the Apache 2 licence. The code is able to call the GPL licenced Geshi library for syntax highlighting, however this must be obtained separately, and saved in a subfolder called geshi in the folder containing index.php.