Meeting Minutes 24 October
Notes On Website Generation

Author:
Warren Hedley (Bioengineering Institute, University of Auckland)

1 Introduction

After practically grinding his nose clean off while at Physiome — luckily there's a lot of it so there was no real danger of it disappearing — in the rush to get the `cellml.org` website up and running, Warren found that some substantial changes were needed to the generation system to really enable it to be used properly at both Physiome and Auckland without a lot of hacking each time synchronisation took place. This document details the nature of the changes, the best way of keeping the sites in synch and some best practices for adding new documents to the website.

2 The New Process

The main problem with the old setup was the need to maintain different copies of the files `site_setup.xml` and `uri_config.xml` at each location, and the fact that most of the contents of `uri_config.xml` are duplicated from `site_setup.xml` anyway (or they should be.) So what do we have now?

2.1 Edit the XML Makefiles

The website generation is handled by a standard UNIX makefile which is generated from a number of XML makefile documents using an XSLT transformation contained in the `xml_makefile.xsl` stylesheet. The majority of the XML makefile documents are contained in the `xml/makefiles` directory, and each contains a small part of the whole makefile. The files are:

- **`config.xml`** — declares a number of parameters (mostly directory aliases) that will be used in other parts of the makefile, as well as directory aliases specifically for use in the actual website transformation stylesheets and the website documents.
- **`transforms.xml`** — declares the processes that are used to transform website documents from XML into the HTML and PDF forms that are available on the website. Each transformation contains a number of discrete steps, which are executed sequentially on the command line, where the environment is retained between commands by making use of the && operator.
- **`scripts_end.xml`** — declares some scripts that are used at the end of the website generation process (currently there is no `scripts_start.xml`, but there could be!) These scripts perform tasks like the insertion of blank HTML pages into every directory in the website and the creation of gzip archives (although technically gzip archives should be generated with dependencies.)
- **`cellml_public.xml`** — lists the input documents involved in the creation of the public part of the CellML website and each of the outputs for each input with the transformation needed to produce each output. This document includes references to the transformations needed to produce each image referenced in the output documents and declares a parameter for the filename of each output image, which may then be referenced in the input XML documents.
- **`cellml_examples.xml`** — the list of input and output documents and images that make up the examples part of the CellML website.

- **`cellml_private.xml`** — the list of input and output documents and images that make up the private part of the CellML website.

The makefile documents in the `xml/makefiles` directory are linked together (using the XML external-parsed-entity mechanism) by the only site-specific file in the entire website: `site_setup.xml` in the `site_specific` directory. `site_setup.xml` defines a number of directory alias parameters which are used in remainder of the makefile documents in the creation of paths to both input and output documents.

2.2 Create the Makefile and Document Configuration File

Using the executable shell script `generate_makefile_and_config.sh` in the `bin` directory, we generate `Makefile` in the `bin` directory and `uri_config.xml` in the `site_specific` directory. All of the information in these files comes from the XML makefile documents.

`Makefile` is the standard unix makefile used to build the website. The actual build process is split up into a number of rules based on the values of the type attributes on each of the targets in the makefiles, which might have values like `html_public` and `pdf_private`; `all` will update all targets if necessary and execute all scripts. The makefile documents declare dependencies on various other targets to ensure that everything is built in order — for instance, the creation of the PDF documents requires that encapsulated postscript versions of all images referenced by that document already exist. Note that the makefiles do not list dependencies on files used in the transformation process such as XSLT files however. For this reason, every rule declared in the makefile has a matching clean rule which removes all targets associated with the given rule, allowing them to be completely rebuilt from scratch — `clean_all` removes all output targets.

`uri_config.xml` contains directory aliases for each of the parameters listed in the makefile input documents, and this file is used to resolve aliases when they are used in document transformation process. In website documents and the XSLT stylesheets used in website generation, directory aliases can be used in for instance the **`relref`** attribute on **`<document_reference>`** elements or in the filenames of images as shown below (where paths may generally be a combination of aliases and text):

```
<figure name="data">
  <image>
    <html_image alt="alternate text" xref="$BMP_PRVT_DATA" />
    <latex_image xref="$EPS_EXMPL_DATA" />
  </image>
  <caption>
    This was talked about in the
    <document_reference xref="$HTML_PRVT_SCOPE">
    scope</document_reference> document.
  </caption>
</figure>
```

The aliases in the `uri_config.xml` file are generated directly from the parameter mappings declared in the makefile documents. So given the following parameter declarations in `site_setup.xml` and `config.xml`,

```
<parameter name="DEST_FILE_ROOT" value="/www/sites/physiome/cellml" />
<parameter name="CELLML_PRIVATE" value="@DEST_FILE_ROOT/private" />
```

we get the following in `uri_config.xml`.

```
<uri alias="DEST_FILE_ROOT" href="/www/sites/physiome/cellml" />
<uri alias="CELLML_PRIVATE" href="/www/sites/physiome/cellml/private" />
```

Note that many of the URI aliases that appear in `uri_config.xml` will not actually be referenced in any of the document files, but it seemed simpler to just generate an alias for every parameter than to add some flag to the makefile documents indicating which parameters should become aliases.

2.3 Create The Website

Once the Makefile and configuration file have been generated, the website may be generated by changing into the `bin` directory where the Makefile and the scripts are located and typing `make all`. The current website generation process relies on a number of tools, which are listed below.

- **saxon** — Saxon is a freeware XSLT processor developed by Michael Kay at ICL. It has consistently been the processor that best conforms to the specification and is still being actively worked on as Michael receives bug reports and feature suggestions from users. At the time of writing the most recent version is 5.5.1, although every version since 5.4 has been used in website generation. The Saxon homepage is at http://users.iclway.co.uk/mhkay/saxon/index.html.
- **RelativePath.java** — In order to make the website as portable as possible, all of the links from documents on the website to other documents and images are relative. This requires the standard XSLT functionality to be extended by calling an external class written by Warren called RelativePath. Note that the mechanism by which the class is called is not completely portable, so the use of this extension limits us to the Saxon processor. The class is in the `com.whedley.utils` package (for want of a better name), and should be in the classpath of the user generating the website.
- **latex, ghostscript** — In order to generate the PDF versions of all the website documents, we use XSLT to transform the input XML documents into LaTeX, compile them into DVI using the LaTeX application, convert them into postscript using `dvips` (also part of LaTeX), and finally convert them into PDF using the `ps2pdf` shell script which in turn calls `ghostscript`. In the near future Warren intends to make this process a bit more efficient by generating LaTeX files from the XML that can be transformed directly into PDF using the `pdftex`, another LaTeX tool.
- **xfig, ImageMagick** — XFig is an X-Windows based vector graphics application that allows the user to draw simple diagrams that can be transformed into extremely attractive looking encapsulated postscript suitable for inclusion in PDF documents. The encapsulated project can be converted into a good-looking bitmap graphic in the GIF format using the `convert` command line application that is part of the ImageMagick library. To obtain good quality renderings of the XFig line art and text, a large graphic is created from the encapsulated postscript, and then this is reduced using linear sampling.

3 Synchronization

In the unlikely event that someone at a location other than Auckland contributes a significant amount of content to the CellML website, we need some way to be able to keep the input document trees in synch. The output documents can then be generated with the appropriate configuration for each site. (Recall that all of the site specific website details have hopefully been concentrated in the `site_specific/site_setup.xml` file.) To this end there is a script called `make_update_bundle.sh` in the `bin` directory which generates an archive containing all files that have been modified since the `.last_update` file was last modified. It then updates the `.last_update` file. The resulting file can be transferred to other sites where the input document tree is maintained, and expanded.

4 Creating New Documents

This section contains some best practice tips for creating new websites. Currently there is a complete lack of documentation for the XML makefile language, the website document language (which was recently christened Funky Ass ML, a name which may stick unless someone comes up with something better), and

the process by which HTML and LaTeX documents are created from FAML documents. But as such a wealth of data already exists, full documentation shouldn't be needed, as it is generally easiest to learn XML vocabularies by example.

Note that the preferred tool for editing XML documents is `nedit`, a powerful freeware text editor available for most UNIX platforms from http://www.nedit.org/. If the potential author grabs a copy of Warren's `.nedit` file (which contains a set of custom preferences) they will obtain an attractive XML syntax highlighting scheme and a number of macros that ease the creation of FAML documents (mapped to the function keys.)

Adding a document to the website involves three basic steps:

1. Creating the document itself. There is a template document called `cwml_template.xml` in the `xml/templates` directory, which contains the basics of document structure. This template should be copied into an appropriate location and given a concise but descriptive filename, where this should probably be what would appear in the website menu. Make sure to document the author and contributors, even if the document is intended for the public website, where they don't appear visibly. If the document contains **`<embedded_xml>`** elements, then add an internal DTD subset defining the elements and attributes that will be used within the document. (Warren prefers that all input documents be run through the `xmlCheck` tool — a wrapper for `nsgmls` — without error.)
2. Add a link to the document in the appropriate menu file in the `xml/menus` directory. It should be pretty obvious how to do this.
3. Add information about the input document and output documents to the appropriate makefile document. This should only involve some minor cutting-and-pasting from an existing input section. If the input document references images, the pages to the target images should be declared as parameters in the makefile, so that a directory alias to the file will be available in the configuration to be used by the transformation.
4. Using the `generate_makefile_and_config.sh` in the `bin` directory, regenerate the unix makefile and directory alias configuration file.
5. Type `make all` in the `bin` directory.

As more documentation on this topic emerges, this document and related documentation will be placed in their own *Website Maintenance* section. One of the things I'd like to add is a FAQ. Unfortunately it all works for me. Questions for the FAQ would be much appreciated.