Simtk.org

Developer’s Manual

Last Updated: 04/04/2013, Joy P. Ku

Table of Contents

1 Introduction 5

1.1 Overview of Simbios and Simtk.org 5

1.2 About This Document 5

2 Basic Structure 6

2.1 XSLT 7

2.1.1 XML Objects 7

2.1.2 The Trouble with Ampersands 7

2.2 JavaScript 7

2.3 The Database 8

3 Simtk Areas of Functionality 8

3.1 Project Statistics 8

3.1.1 Geography of Use Page 8

3.1.2 Reporting Graphs 9

3.2 File Uploads 9

3.3 Search Functionality 9

3.4 Subversion Interface 9

3.5 Discussion Forums 9

3.5.1 Forum Administration 9

3.5.2 Forum Architecture 10

3.6 Project Keywords and Ontology Terms 10

3.7 RSS feeds 11

3.8 Biositemaps RDF 11

4 Cookbook 11

4.1 Adding a Project Overview Field: an Example 11

4.2 Development Pipeline and Subversion 12

4.2.1 Branching in Subversion 13

4.2.2 Merging in Subversion 13

4.2.3 Publishing Changes to the Production Server 14

# Introduction

## Overview of Simbios and Simtk.org

Simbios is one of the National Centers for Biomedical Computation, funded by the National Institutes of Health since 2005. Simbios stands for Physics-Based **Sim**ulation of **Bio**logical **S**tructures. Its mission is to provide infrastructure, software, and training to help biomedical researchers understand biological form and function as they create novel drugs, synthetic tissues, medical devices, and surgical interventions.

As part of its mission, Simbios has developed and maintained Simtk.org, a web portal for the development and sharing of software, models, data, and other biocomputational resources. It allows researchers to register their projects, upload their software and data for sharing with other researchers, and explore what other people are doing in related spaces. It also enables projects developed by Simbios and its collaborators to be distributed more easily.

The code for Simtk.org is open-source. It is forked from GForge 4.0.2, and as such, is available under the GPL license for others to utilize. Simtk.org runs a heavily modified version of the PHP-based GForge. Additional functionality is provided by phpBB for the discussion forums, Mailman for the mailing lists, MoinMoin for the wikis. It utilizes PostgreSQL for the underlying database. See the table below for the software versions currently being used.

|  |  |
| --- | --- |
| Software | Release in Use |
| Python | 2.3.4 |
| PHP | 5.1.6 |
| GForge | 4.0.2 |
| Mailman | 2.1.5 |
| phpBB | 3.0.8 |
| MoinMoin | ?? |
| PostgreSQL | 8.1.5 |
| Subversion | ?? |
| WebSVN | 2.0 beta 8 |

Current software dependencies for the Simtk.org website

## About This Document

This document is intended to provide information for web developers working on Simtk.org. It uses the following conventions:

* File and directory names: use a smaller font-size (Helvetica 9), like this test.doc
* Commands: use the Courier New font

# Basic Structure

The GForge code has been heavily modified by a series of programmers to make it suitable to Simbios’ purposes. It relies on XML getting piped to a series of server-side XSL templates, which then generate the final page that the user sees. However, this means that there are a lot of pages which potentially need to be modified in order to implement a feature or fix a bug.

The www subdirectory is obviously where most of the website files live. However, almost every page refers to some object whose class definition lies somewhere within the common subfolder. This includes the separate objects whose sole purpose is to generate XML from the main GForge objects (such as projects, users, files, etc).

Within the www directory, you should be able to find most of what you need pretty easily. A few pointers for often used or special items, however:

* Most of the string literals for things such as feedback, error messages, email text, etc. are kept in the file www/include/languages/Base.tab.
* All of the XSLT is kept in the www/templates/xml/xsl sub-folder.
* For some unfathomable reason, the CSS, Javascript, and several key image files are also kept in their respective folders under www/templates/xml as well.

A brief description of the other directories is given below:

* **common:** This is where most of the class objects are kept. The core objects such as User, Group, Role, Permission, etc. are kept in the include subfolder, while most of the other objects are kept in their respective subdirectories (e.g., forum, mail, publications, search). Most XML-generating objects will be kept in the xml subdirectory, but not all of them
* **cronjobs**: This folder is unremarkable from a web perspective except for one thing: there’s a /dav-svn subdirectory containing files which actually are referenced from web-acessible code, so be careful when modifying anything there.
* **dart:** This contains code for the old automated build and dashboard system and probably could be deleted.
* **doc:** This directory contains both user and developer documentation for Simtk.org.
* **etc:** This directory contains the local.inc, which needs to be updated to reflect the server name and database it uses.
* **geoip:** This folder contains our IP-to-geolocation database, which is leased from MaxMind.
* **htdig:**  Files likely for the software htDig but probably unused.
* **jpgraph:** Files for the JGraph library are located here. JGraph is used to generate the statistics graphs available for each project on Simtk.org.
* **mailman:** Files for the mailing lists available to each Simtk.org project.
* **plugins:** As the name suggests, this directory contains files for plugins. The only plugin currently is for WebSVN for browsing subversion repositories.
* **unused:** The unused directory is, as one might guess, unused. It seems to contain files from the original GForge suite which have since become irrelevant, and can probably be deleted with no adverse effect.
* **upload:** This directory contains scripts used when an individual uploads a file to the site.
* **utils:** Finally, the **utils** directory contains a number of utility scripts for such things as generating a Biositemaps file, script and mailing list administration, and Python libraries for Simtk objects. None of these should be executed from web-accessible code.

## XSLT

The XSLT files are numerous and complicated. The major one to pay attention to is template\_elements.xsl because it contains templates frequently referenced by the rest of the XSLT such as the sidebar, page header, and page footer. Several of the other template files are simply shells around a core template found in another file. For example, most of the logic in both create-user.xsl and edit-user.xsl is done by template\_edit\_user.xsl. If you’re looking at an often-used page element, check the template\_\*.xsl files first.

Another thing to watch for is that not all portions of the site use XSL. Some sections such as forums, bug tracking, and many of the site administrative functions still have their HTML generated directly by the PHP scripts. Therefore, if you can’t find an appropriate XSLT file in the templates directory, check the PHP script itself.

### XML Objects

The basic XmlObject that most other XML classes inherit from is roughly structured as follows:

• data

• session

• user (contains authorization info)

• section (contains information specific to the area of the website the user is in)

• (Page-specific info goes here)

Because of this, and also because different pages call different XML-generating objects, things that show up on one page may be different than similar things showing up on another page. This was the case with the Downloads information section in the left-hand sidebar. Because the project’s Overview page furnished different and more detailed information than the general project information, the Downloads section would show up differently in that page. Make sure that any new XML is accessible from all the pages it needs to show up on.

### The Trouble with Ampersands

Ampersands can cause a lot of trouble in XML if they aren’t escaped. They occur frequently in URLs and other pesky places where they might not be expected. One solution is to escape all final input using the escapeOnce() function, but the preferred way is to surround all data in the GForge XML objects with a <[! CDATA[]]> clause if possible.

If an XML error occurs on the production server, it will just show a blank page; however, an XML error on any of the test servers will show an error message in a very basic format.

## JavaScript

Simtk.org has several JavaScript widgets, which are stored in www/templates/xml/js. Most of these draw on the Yahoo! User Interface library (YUI), portions of which have been copied onto the server for consistent access (there have been times when they’ve been unreachable from Yahoo!). The YUI files referenced from production scripts should all be in the yui directory and end in "-min.js”, as these are the JavaScript libraries that have been specially compiled for low-latency transfers.

There is one JavaScript widget, the ontology textbox found in the project edit page, which relies on jQuery. jQuery and YUI have a history of not getting along very well, so be careful when extending widgets on that page.

## The Database

The GForge database is a mixture of confusing terms and questionable design decisions. For starters, very few tables have foreign keys tying them to other tables to which they refer. This makes figuring out how the tables are related very difficult for an outsider. A reference diagram can be found in the file Simtk\_db\_graph\_full.jpg and the corresponding code for creating the foreign keys is in the file simtk-structure.sql, located in the documentation and db folders of the website code, respectively.

As can be seen in the aforementioned diagram, the two tables most often referenced are users and groups. "Groups" is the term that GForge uses for projects, and the nomenclature can get muddled sometimes. These tables hold what you would expect them to, and serve as the ultimate reference for most other tables.

Other tables of note include:

• frs\_package, frs\_release, and frs\_file, which contain information about file packages, releases, and files themselves.

• frs\_dlstats\_file and frs\_dlstats\_filetotal\_agg contain information about the download statistics for files.

• news\_bytes contains the news items for a project.

• publications contains publication information.

• related\_projects contains information about how groups are related to one another.

• related\_links contains all the links a project administrator has chosen to connect to a project.

• Search categories still rely on the trove\_cat table for category information, and are linked to groups by the trove\_cat\_group table.

• project\_keywords and project\_bro\_resources contain information about keywords and BRO ontology terms assigned to individual projects.

In addition to the tables, there are also quite a number of table views which don’t show up in the aforementioned diagram. These can be seen in Postgres by issuing the command \dv. The most commonly used views are frs\_frpg\_view and frs\_dlstats\_\*.

# Simtk Areas of Functionality

## Project Statistics

### Geography of Use Page

The "Geography of Use" page for each project (accessed by going to the project Overview page and then clicking Overview -> Geography of Use) is driven by a PHP script which in turn generates Javascript for placing each marker on a Google map. This PHP script lives at www/stats/usagemap\_js.php and will need to be modified if any different data need to be graphed geographically. Google Map used here is “Version 1” and is deprecated. We are in the process of migrating to use Google Map Version 3. Google Map Version 3 does not use keys.

A geolocation database is used for mapping IP addresses to real-world locations for all the projects’ "geography of use" sections. It’s leased from MaxMind (maxmind.com) and was last updated on 2009-11-10. Whenever the database gets updated again, it should be an easy replacement: simply drop the GeoIPCity.dat file into the same location as the original and everything should be automatically updated.

### Reporting Graphs

The reporting graphs (accessed by going to the project Overview page and then clicking Overview -> Geography of Use) are all generated by files under the reporting folder, but they call out to the JGraph library which sits on the same level as www.

## File Uploads

One thing to watch out for is that users uploading a file will trigger a Perl script to get executed as CGI, which may figure into features addressing uploads. The script is located in upload/cgi-bin.

## Search Functionality

Search is a bit tricky because search modules have two aspects, engines and renderers (under the respective folders in www/search), which don’t necessarily get divided up how you would think. Most of the actual search logic for the general search takes place in common/xml/ProjectSearchXml.class.php anyway, and 90 percent of the time that will be the file you need to modify.

## Subversion Interface

If somebody takes a Subversion repository’s URL and puts it in the location bar of their browser, it will trigger an XSLT download that renders a web page showing instructions on how to navigate Subversion. These files are svnindex.xsl and svnindex.css in the www directory; we have a separate Websvn installation for browsing Subversion online.

Websvn gives users a better way to browse Subversion repositories online than the default Subversion plugin. It runs as a site plugin to simtk.org at /var/www/html/simtk/plugins/scmsvn/websvn2 and can be reached by going to https://simtk.org/websvn/wsvn/[projectname]**.**  It’s been fairly good about running self-sufficiently, although many sections will ask for authentication for no good reason. The standard administrator credentials for the site should get around these authentication challenges, however. Note: If a project requested does not exist due to typo or path omission, credentials are requested also. Hence, please also check the path names requested.

## Discussion Forums

New discussion forum software was added onto Simtk.org in September 2011. The forums run a somewhat-modified version of the open-source phpBB 3.0.8. Data for the forums is stored in PostgreSQL as tables prefixed by “phpbb.” Many of these are self-explanatory (phpbb\_posts, phpbb\_topics, etc.)

### Forum Administration

Administration of the Simtk.org forums is handled through phpBB's “Administration Control Panel” (ACP). Tasks that require access to the ACP include clearing the forum's cache, or editing or adjusting any user or group information. User and group information in the forums is populated at the time a user or group is created. ***At this time there is no live updating of information after creation***. If a group name or other information must be changed, you can edit it using the “Manage Forums” option in the ACP.

### Forum Architecture

User accounts in the forums correspond to individual users on Simtk.org. Simtk.org projects receive two types of entries in the forums. One is a “group,” which is used in phpBB to control access permissions. The second is a “forum,” where the actual discussion takes place.

Group names in Simtk's implementation of phpBB begin with the group ID of the project on Simtk.org and a semicolon. This is to facilitate the easy matching of groups with projects. Groups consist of users belonging to the project on Simtk.org. Membership in phpBB groups is updated whenever project team information is changed using the Simtk.org administration pages.

All Simtk.org project forums are sub-forums of a master “Projects” forum in phpBB. The system has the ability to group forums using parent forums. This could be useful in organizing discussion forums by biological area or other factors.

## Project Keywords and Ontology Terms

All Simtk.org projects are tagged with keywords and optionally, ontology terms taken from the Biomedical Resource Ontology (BRO).

The BRO is one of a great number of ontologies kept by the National Center for Biomedical Ontology (NCBO) (found online at their site <http://bioportal.bioontology.org/ontologies/1104>). It provides a useful way of classifying projects submitted to Simtk.org, and as such we do a periodic dump of their terms into our database.

Every week, a pair of Python scripts from the ctmeyer account check the copy of the BRO stored on simtk.org against the latest copy at the NCBO. If any have been added, changed, or deprecated, an email is sent to [webmaster@simtk.org](mailto:webmaster@simtk.org) to alert admins that action needs to be taken.

If a new copy exists, it's generally pretty straightforward to update: in most cases, running a command similar to the one that reported the differences (/home/ctmeyer/Python-2.4.6/python ~ctmeyer/BROimport.py -i -a [[your@email](mailto:your@email)]) will do the trick—the difference between this and the cron command is changing -m to -i, which makes the script update the database automatically.

Note that the Python script has to be run using Python 2.4.6 or above, since it uses instructions that aren't available in the default simtk version of Python, 2.3.4.

If for some reason a term fails to update (you keep seeing it again and again as being changed), then you'll need to go in and update it manually. This is a slightly more involved process. For each term you'll need to check the attributes in the database (the bro\_resources table) against those on the web site. You'll also need to check the parents and children of the ontology term against what's stored in the bro\_inheritance table. If there are any differences, you'll need to rectify those differences manually in the database using SQL commands to insert, delete, or change information in the above-named tables to match the information from the BRO.

## RSS feeds

Simtk.org features both incoming and outgoing RSS feeds on its news page. For incoming RSS feeds, a library called SimplePie is used and modified to achieve this purpose. The news feed on the right-hand side of the Simtk.org news page entitled “Simulation Community News” at <https://simtk.org/news.php> is fed by feeds accumulated and processed by a SimplePie script on that page. Configuration is fairly straightforward, with presentation and feed options available inline.

The Simtk.org outgoing RSS feed is available at <https://simtk.org/news_rss.php>. It is generated by SimplePie based on custom coding in that page to create XML.

## Biositemaps RDF

The biositemaps file is an index of projects available on simtk.org, set down in a standard XML-based format according to guidelines given by the NCBC and used within the Resource Discovery System.

It’s generated every night by the script /var/www/html/simtk/utils/biositemap\_NCBC \_Simbios.php. After it fetches information from the server and generates an RDF for the biositemaps, the resulting file is passed through the script /home/ctmeyer/check\_biositemap.py (see note below), which checks the consistency of the file. If the file is good, it gets written to a temporary file and then gets copied by a cron job to the web directory later on.

**Note**: the script /home/ctmeyer/check\_biositemap.py is in an odd location because it requires at least Python 2.4.6, which is also built in /home/ctmeyer. The version of Python on the server is lower, and upgrading the live server is not something to be taken lightly, therefore the biositemap script has been run entirely within that self-contained system.

# Cookbook

## Adding a Project Overview Field: an Example

Suppose you want to add a field to be displayed on the project overview. First you would likely have to add a column in the database; let’s call it "primary\_document". (This assumes proficiency with SQL and glosses over the necessary commands for adding columns to a table; consult the web for a refresher if necessary.)

Leaving aside the issue of taking user input and storing it in the database, the next step is to get the new field to show up in the XML for the project overview. You could equally well add this to either of two files: the file you might look at first is common/xml/ProjectOverviewXml.class.php, but that only exposes a little information. For information that might get used places other than the overview, we need to modify common/xml/ProjectXml.class.php. We’ll work with the latter.

Find where the information is taken from the database. In our case there’s a "SELECT \*" statement practically first thing in the file, so it’s already being grabbed. Further, the call to db\_row\_to\_xml ensures that it will automatically be transformed into an XML field named "primary\_document". However, this is boring, so let’s assume there needs to be some formatting done (such as turning URLs into hyperlinks). To do this we need to get the result from the database, format it, and then inject it into the XML stream bracketed by XML elements. The final code would look something like:

$primary\_document = db\_result( $resGroup, 0, "primary\_document" );

$xmlData .= "<long\_description\_with\_links\_and\_breaks>";

$xmlData .= escapeOnce( util\_make\_links( $primary\_document ) );

$xmlData .= "</long\_description\_with\_links\_and\_breaks>";

The first line gets the proper field from the first (and only, in this case) row of the database results. The second and fourth lines set up the element tags that will contain this new information. The third line adds the information retrieved from the database after running it through a function to transform URLs into hyperlinks and subsequently another function to ensure that characters that could cause trouble for XML are properly escaped. *Important: make sure you always use the escapeOnce() function when you’re injecting text into XML unless you really, really know what you’re doing!*  See Section 9.2.2 (The Trouble with Ampersands) above for more information.

Finally, the field needs to be referenced in the XSLT; otherwise it will never show up on the web page. In the file www/templates/xml/xsl/overview.xsl we see everything dictating how the project overview page is displayed. In order to insert our new field, simply add the following lines somewhere around the part of the code where the audience, synopsis, goals, and downloads are given:

<xsl:if test="string-length(string(/data/project/primary\_document)) &gt; 0" >

<div>

<span class="textIntro">Primary Document: </span>

<xsl:value-of select="/data/project/primary\_document" />

</div>

</xsl:if>

The first xsl statement (the "if" clause) tests to make sure that there’s data to show–otherwise we’d end up with a blank entry if the user didn’t input anything. The div tags ensure that this is encapsulated in a block-level element (HTML-speak meaning the item will have carriage returns before and after, essentially). The span gives us the short heading telling the user what the field is, and the xsl statement outputs the actual data. Now we’ve added a new field!

A more complex example would be adding permission controls so that (for instance) only the project admin could see the primary\_document field. This would be a matter of testing for the proper permissions (under /data/session/user/ in the XSLT DOM) alongside the string length in the code above.

## Development Pipeline and Subversion

The Simtk.org code is developed within a Subversion repository: <https://simtk.org/svn/website/trunk>. For development purposes, you should check out a copy of the code to work on locally. You may also create a branch of the code (see sections below) to work on locally.

The following process occurs in order for your changes to go live:

1. When you have finished making your changes for a particular feature or bug, have others test the changes using your version of the code.
2. Once your developments have been approved, you should check the code in.
3. The code will then be checked out and tested on a staging server to ensure that all files and changes have been checked in and work properly outside of your personal development environment.
4. After successfully being tested on the staging server, your code changes will then be promoted to the live server.

The sections below provide more instructions on developing using Subversion.

### Branching in Subversion

Creating a branch is a handy thing to do when faced with several bugs and features, any of which could leap in importance. A branch will keep all changes to its code base from affecting the main trunk as well as other branches, and will only become part of the site code when it’s merged into the trunk.

To create a branch, simply use the "copy" function of Subversion. For example,

svn cp https://simtk.org/svn/website/trunk https://simtk.org/svn/website/ branches/new\_branch -m "Some message"

creates a branch called "new\_branch" in the "website" Subversion project on simtk.org. Note that the message is required when creating a branch, just as it is for checking in a revision. This can be used to document what the purpose of the new code branch is.

Branches in the website code started out with the nomenclature "[bug/feature]-{report#}". However, it seems that the report number is shared between bugs and features (and every other project on the site), and bugs will sometimes be classified as features and vice-versa, so after about bug #1150 the nomenclature simply shifted to "tracker-{report#}". In the long run, this should make branches easier to find (although it does confuse things somewhat in the short term).

### Merging in Subversion

When a branch has been given the go-ahead for inclusion into the main code trunk, it will need to be merged back into the trunk. The prudent sysadmin will turn this into a multi-step process.

Before anything else: any time you merge any code, it’s very important to keep track of which revisions get merged. For example, if a branch started at revisions 600 but is at revision 607 when merge time comes, record in the comments what the starting and ending revision numbers were. If you have to make changes to the branch later on, this is the only way you’ll be able to know where to pick up.

First, change directories into the trunk and check what the differences will be by using something like

svn merge -r 600:607 https://simtk.org/svn/website/branches/new\_branch –dry-run

which will pretend to merge all the changes from revision 600 to revision 607 from the branch called "new\_branch" into the current code base, whether it’s the trunk or another branch. You’ll see a list of files scroll by along with their status. Make a note of how many files have a "C" by them, because this means that those files will have conflicts which need to be fixed by hand. "M" and "G" are just fine, as are "A" and "D".

If the results of the merge are satisfactory and you can account for all the files, go ahead and issue the same command without the –dry-run flag; this will cause the current code base to update for real.

Now it’s time to take care of all the conflicted files. You can see what files have conflicts at any time by issuing the command svn st. Each conflict will require editing by hand. Note that the base file will have information from both the new and old revisions included in it, so if it’s a PHP script (for example) it won’t execute.

**Never update on a live server if there are file conflicts!**

For each conflicted file, there will be three other new files. One will be the file name with "merge-left" appended, one with "merge-right", and one with "working". Merge-left and -right are the files from the Subversion repository at the first and last revision numbers you gave in the command (so in the example above, "merge-left" would be the file from revision 600 and "merge-right" would be the file from 607), while "working" is the file as it was on the server before you executed the merge. Generally when you compare files, you’ll want to compare the working and merge-right versions.

Once you’ve got a conflicted file sorted out, make sure that the base file contains the changes you’ve made and issue the command svn resolved [filename]. This will tell Subversion to delete the other three files and erase the conflict from its database. It will be sync’ed next time you commit.

Once all conflicts have been dealt with, you can go to the trunk root and check the codebase in just as you normally would. Again, however, make sure to note in the comment which revisions and branch you just merged in, because this is the only record you’ll have if you need to come back later.

### Publishing Changes to the Production Server

Once you’ve resolved any file conflicts, you’re ready to publish changed files to the production server, simtk-dev-a.simtk.org. To do so, log into simtk-dev-a.simtk.org, and change your working directory to /var/www/html/simtk. Before you proceed, run svn status –u to check the status of the files on the server. In the resulting list, a “?” next to an item means that item is on the server, but not in the repository. A “\*” next to an item indicates the item on the repository is more recent than the version on the server. The file or files you are updating should have a “\*” next to them. If they do not, the updated files have not been properly committed to the repository.

When you are ready to publish the updated files to the production server, run sudo svn update /full/path/to/file.ext. You must use the complete path to the file when running this update. If you do not use a full path, and instead just a file name, you will get what may appear to be a confirmation message (“At revision xxxx”), but no action has been taken. When you properly update the file with the complete path, you will receive the message “Updated to revision xxxx.”