SAPTF V1.6
Sequence analysis plugin tool framework

April 21, 2010

SAPTF provides a simple way of placing CGI-based sequence analysis tools on a web server. Each tool is configured via an XML file and tools can easily be added or removed without writing new HTML pages or CGI scripts. SAPTF also deals with slow, long-running, programs by providing an AJAX means of checking for the final data avoiding server timeouts.

1 Building and installing

1.1 Configure Perl

Perl must be installed with the XML::DOM and CGI modules.

Under Fedora Linux, the CGI module is installed by default and the XML::DOM module can be installed by simply typing:

```
yum install perl-XML-DOM
```

Otherwise do:

```
perl -MCPAN -e shell
install CGI
install XML::DOM
exit
```

1.2 Configure Apache

Your web server must be configured either to allow `.htaccess` files to control the behaviour of files in the directory in which you install SAPTF (and its temporary files), or to recognize files with the extension `.cgi` as being CGI scripts and to allow CGI scripts to be executed in the directory in which you install SAPTF.

The simple option is to ensure you have the line

```
AllowOverride All
```

in the configuration file where it applies to the document root directory. Installing SAPTF will install a `.htaccess` file which will set the required options.

Alternatively, add the lines:

```
AddHandler cgi-script .cgi
Options ExecCGI
DirectoryIndex index.html index.php analyze.cgi
```

for the SAPTF and temporary directories in the HTML tree.

Note that the SAPTF system will be accessed through a CGI script called `analyze.cgi` The `.htaccess` file is set up to use this as an index file for the directory if you don't have an `index.html` or `index.php` file. Thus, suppose you have installed SAPTF such that the script is accessed using the URL:

```
http://www.myserver.com/saptf/analyze.cgi
```

it may also be accessed using the URL

```
http://www.myserver.com/saptf/
```

If you have modified your Apache configuration file, restart Apache.

1.3 Install Overlib

Install the overlib JavaScript package from

```
http://www.bosrup.com/web/overlib/
```

Typically you would install this in `$DOCROOT/js/overlib/` (where `$DOCROOT` is your Apache web server document root directory). Create that directory and unpack the Overlib `.zip` file inside it.

1.4 Run the install.pl script

First, this will ask you where Perl lives on your system. NOTE: you must have the CGI and DOM Perl modules installed.

Next it will ask where your HTML document root is (typically something like `/var/www/html`) and then the name of the subdirectory where you wish to install SAPTF. We will call this directory `$SAPTF` in the rest of this document. It will also ask for an HTML subdirectory where SAPTF can store temporary files — these will be CGI scripts used in 'slow' (asynchronous) mode.

Next it will ask where you wish tools to live. This is only really relevant to running the demonstration, as you specify where each tool lives in the configuration file. Simply accept the default to run the demonstration code. If you don't want to run the demonstration, then it doesn't really matter what you put in here.

You will then bed asked for the server's path to the Overlib library — that is the relative URL which you would use to access it. If you installed Overlib in `$DOCROOT/js/overlib` then this would be `/js/overlib`.

Other options are self-explanatory. You can set the default colours and size for the help popups, but these can be set in the configuration file.

Note that a `.config` file will be written in the current directory when you run the install script. This will remember the values that that you have entered so you can run the install again without having to re-enter them.

Also, note that if you decide to install the code by hand, you will need to set the Perl executable at the top of `analyze.cgi` and doanalyze.cgi and will need to set the values for the variables in `config.pm`. These are normally set during the installation process.

1.5 Run the demo

Point your web browser at

```
http://MYHOST/MYPATH/analyze.cgi
```

(replacing `MYHOST` with your web server and `MYPATH` with the appropriate path depending on the HTML directory in which you installed SAPTF (typically simply something like `/saptf`).

2 Further Configuration

The above steps should get the demonstration of SAPTF up and running and will install and populate the configuration files. This section gives you more information on modifying these for your own needs:

2.1 $SAPTF/config.pm

`$SAPTF/config.pm` contains configuration information, mostly in the form of paths to where files live. The `install.pl` script will set these depending on the values you give, but you can modify them later as required.

- `$::xmlfile` must point to the XML file where you specify your sequence analysis tools. You can call this file anything you like and it can live anywhere that can be read by the Apache web server. Typically it will be `$SAPTF/saptf.xml`

  The format of this file is described in detail below.

- `$::overlib` must point to the URL for the directory in which you installed overlib. So if you installed it in `$DOCROOT/js/overlib/` then this would be `/js/overlib`

- `$::docroot` is the path to the HTML Apache document root (something like `/var/www/html`).

- `$::htmltmpurl` is the server's view of the temporary directory. This would be appended onto `$::docroot` to obtain the physical directory (typically something like `/saptftmp`).

- `$::perlexe` is the full path to your Perl executable

- `$::debug` should normally be set to 0. You can set it to 1 to see some additional information from SAPTF as to what programs it tries to run. Some of this will go to the HTML output and some to your Apache error log. It places a delay into the scripts generated in 'slow' mode for testing purposes.

2.2 $SAPTF/html.pm

The `$SAPTF/html.pm` module provides 4 functions which print HTML. This is used to provide the header and footer on the pages as well as printing error messages. You can customize these as much as you wish to fit in with your required layout. Of course you can build the `$SAPTF/html.pm` module from a template file, for example using the Perl Template Toolkit, to include standard headers, menus, etc. Section 5 tells you about some sample code which is provided for creating a menu-type layout in this way.

1. `PrintHTMLHeader()` — prints an HTML header – something of the form:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
   "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
<link rel='stylesheet' href='/saptf/saptf.css' />
<title>Simple SAPTF HTML file</title>
</head>
<body>
```

Note that this includes a reference to a CSS file to style the SAPTF tool selector and anything that you might format with CSS in your tool output.

NOTE! The mechanism used for revealing and hiding sequence entry boxes is not compatible with the JavaScript Prototype library, so make sure that the Prototype library is not included by the HTML Header.

2. `PrintHTMLFooter()` — ends the HTML – something of the form:

```
</div> <!-- maincontent -->
</div> <!-- main -->
</body>
</html>
```

Note that the footer closes 2 divs which will be created by the SAPTF scripts.

3. `ErrorMessage()` — takes 2 parameters and prints an error message – something of the form:

```
<h2>Error</h2>
<h3>$msg1</h3>
<p>$msg2</p>
```

`$msg1` is a title for the error and `$msg2` is more descriptive text

4. `ErrorPage()` — takes 2 parameters and prints a complete page for an error. As in the example, this should call the `PrintHTMLHeader()` function, then open a '`main`' div, print a first-level heading, open a '`maincontent`' div, call the `ErrorMessage()` function and finally call the `PrintHTMLFooter()` function.

3 CSS — Styling SAPTF

3.1 Images for the help button

You can replace the `i_help.gif` file with whatever image you would like for a help icon.

Note that you should ensure that the `<img>` tag –or– the `saptfimg` class (or both) is set to produce no border around the image:

```
img {border: none;}
.saptfimg {border: none;}
```

3.2 DIVs, IDs and CLASSes

SAPTF provides the following CSS DIVs, IDs and CLASSes

IDs are only used in relation to DIVs, though some DIVs use CLASSes

The complete content is placed in a `<div id='main'>`. This is useful for frame-like layouts. `<h1>` title (default: `<h1>Sequence Analysis Tools</h1>`) is placed in this. The 'main' div then contains a `<div id='maincontent'>`. Again useful for frame-like layouts. This is then followed immediately by a `<div id='saptf'>` which allows you to format your 'saptf' classes differently from classes with the same name used elsewhere in your HTML. This layout applies to both the `analyze.cgi` script which shows the set of tools and the `doanalyze.cgi` script which displays the results of the selected analysis.

The list of tools in wrapped in a `<div class='buttonlist'>`.

Each item in the list of tools is a list item with `class='buttonlistitem'`

Parameters for an analysis tool appear in a `<div class='buttonlistparams'>`

The set of sequence input boxes is wrapped in a `<div class='seqinput'>`

Each sequence input box, together with its title, is placed in a `<div id='xxx'>` where `'xxx'` is the name for the input box specified in your XML configuration file. You don't normally want to style these — they are used to switch on and off the display of the boxes depending what analsysis tool is selected.

All textarea boxes used to input sequence data have `class='seqbox'`.

Thus, overall, the `analyze.cgi` HTML looks something like:

```
<div id='main'>
   <h1>TITLE</h1>

   <div id='maincontent'>
      <div id='saptf'>
         <h2>SUBTITLE</h2>

         <h4>TOOL LIST HEADER</h4>

         <div class='buttonlist'>
            <ul>
               <li class='buttonlistitem'>...</li>
               ...
            </ul>
         </div> <!-- buttonlist -->

         <h4>SEQUENCE DATA HEADER</h4>

         <div class='seqinput'>
            <div id='XXX'>
               <p>SEQUENCE BOX TITLE</p>
               <textarea class='seqbox'></textarea>
            </div> <!-- sequence box -->
            ...
         </div> <!-- seqinput -->
      </div> <!-- saptf -->
   </div> <!-- maincontent -->
</div> <!-- main -->
```

While the output from `doanalyze.cgi` looks something like:

```
<div id='main'>
   <h1>TITLE</h1>
   <div id='maincontent'> 
      <div id='saptf'>
         <h2>RESULTS SUBTITLE</h2>
         <div id='saptf_results'>
            ...
         </div> <!-- saptf_results -->
      </div> <!-- saptf -->
   </div> <!-- maincontent -->
</div> <!-- main -->
```

3.3 CSS

The following is a typical piece of CSS to present these nicely:

```
.buttonlist {
   margin: 5px 0px;
   border: solid black 1px;
   background:  #92aec7;
}
.buttonlist ul {
   margin: 5px 5px 5px 0px;
   padding: 0px 0px 0px 30px;
}
.buttonlist li {
   border: solid black 1px;
   margin: 5px 10px 5px 0px;
   padding: 5px;
   background: #FFFFFF;
}
.buttonlistparams {
   padding: 0px 0px 0px 20px;
}
.seqinput {
   margin: 5px 0px;
   border: solid black 1px;
   background:  #92aec7;
   padding: 5px;
}
.seqinput p {
   margin: 0px;
}
.seqinput textarea {
   margin: 0px 0px 0px 25px;
}
.saptfimg {
   border: none;
}
```

3.4 Styling the output

Output from any analysis tool is placed in a `<div id='saptf_results'>`. Other than that, it will depend what your analysis tool provides in the way of CSS classes or IDs in its output (if it provides HTML). Tools can also provide plain text output in which case SAPTF will wrap them in a `<pre>` tag.

4 The SAPTF XML Configuration file

4.1 The simplest possible example

Normally you would have 2 or more tools to select from. The first tool to appear in the XML file will be the default tool. For a single tool, the simplest XML configuration file would look like:

```
<saptf>
```

```
   <seqbox id='seq'>
      <text>Sequence data:</text>
   </seqbox>

   <seqtool id='myanalysis'>
      <text>Apply my analysis</text>
      <input>seq</input>
      <tool exe='/path/to/myanalysis'
            param='seq' plaintext='1' />
      <description>
         My amazing sequence analysis tool.
      </description>
   </seqtool>
</saptf>
```

4.1.1 The <seqbox> tag

The `<seqbox>` tag specifies a box in which you enter sequence data and an accompanying text label with the embedded `<text>` label. You may create more than one `<seqbox>` each of which must have a unique id.

NOTE! The `id` for each `<seqbox>` *must* start with the letters "`seq`"

4.1.2 The <seqtool> tag

The `<seqtool>` tag specifies a tool for performing sequence analysis. Again it requires a unique id, but there is no restriction on the name. Each tool you specify will be listed on the web page next to a radio button. The text next to the radio button is specified using the `<text>` tag.

4.1.3 The <input> tag

The embedded `<input>` tag specifies which of the `<seqbox>` specified sequence boxes should be displayed when the radio button for this analysis tool is selected. So, in this case, we have specified that the `<seqbox id='seq'>` should be displayed.

4.1.4 The <tool> tag

The `exe=` attribute of the embedded `<tool>` tag specifies the executable program that should be run to perform whatever sequence analysis you require. The `param=` attribute then specifies the sequence input box whose sequence should be passed to the program.

The `plaintext='1'` attribute specifies that the tool is not expected to know anything about HTML and will simply generate a plain text output file. This attribute causes SAPTF to wrap the output of the tool in `<pre></pre>` tags so that formatting of the plain text is preserved.

4.1.5 The <description> tag

The `<description>` tag generates help text when the user clicks on the help button for this tool. This is simply descriptive text that describes what the tool does. Note that any formatting (line breaks, etc.) will be lost.

4.1.6 The executable program

The input sequence will always be cleaned to remove any spaces or newlines and in the default mode shown in this example, the sequence is passed to the executable program on the command line.

So, in this example, if the user were to input the sequence:

```
SASRT YMNPR STALN KHGTG
HFDLN KMEDT HYRTS L
```

then SAPTF would end up running the command:

```
/path/to/myanalysis SASRTYMNPRSTALNKHGTGHFDLNKMEDTHYRTSL
```

In this case the `plaintext='1'` attribute was specified, so the program is not expected to know anything about generating HTML output — it simply creates plain text which is shown on the results web page.

Alternatively, the program can generate HTML output. In this case it should not be creating any overall layout of the page (i.e. no `<html>`, `<head>` or `<body>` tags) — it is simply expected to to generate the HTML required to format the results themselves — for example a `<table>` tag and its contents.

4.2 Two tools with different sequence input boxes

Suppose you have 2 tools which require different sequence input boxes — for example one which expects DNA sequence and one protein sequence. In that case, the configuration file would look like:

```
<saptf>
   <seqbox id='seqprot'>
      <text>Protein sequence data:</text>
   </seqbox>
   <seqbox id='seqdna' hidden='1'>                <!-- NOTE -->
      <text>DNA sequence data:</text>
   </seqbox>

   <seqtool id='proteinanalysis'>
      <text>My analysis (protein)</text>
      <input>seqprot</input>
      <tool exe='/path/to/myproteinanalysis'
            param='seqprot' plaintext='1' />
      <description>
         My amazing sequence analysis tool for proteins.
      </description>
   </seqtool>

   <seqtool id='dnaanalysis'>
      <text>My analysis (DNA)</text>
      <input>seqdna</input>
      <tool exe='/path/to/mydnaanalysis'
            param='seqdna' plaintext='1' />
      <description>
         My amazing sequence analysis tool for DNA.
      </description>
   </seqtool>
</saptf>
```

The configuration file is much the same as we saw before, but we have two `<seqbox>` tags and two `<seqtool>` tags. When the radio button for the DNA analysis tool is selected, the DNA sequence entry box will be displayed and vice versa. Remember that the `id` for each sequence must start with the letters "`seq`".

Note the `hidden='1'` attribute in the second `<seqbox>`. This ensures that it is not displayed when the web page is first loaded. Remember that the default tool is the first one to be listed in the XML file, so you want to ensure that only the sequence entry box(es) required for that tool are shown when the page is loaded.

Of course you might also have the same program take both DNA and protein sequences, but need a flag (perhaps '`-d`') to indicate that it is being supplied with a DNA sequence instead of a protein sequence. In this case, we would provide a configuration file that specifies the '`-d`' in the `param=` attribute:

```
<saptf>
   <seqbox id='seqprot'>
      <text>Protein sequence data:</text>
   </seqbox>
   <seqbox id='seqdna' hidden='1'>
      <text>DNA sequence data:</text>
   </seqbox>

   <seqtool id='proteinanalysis'>
      <text>My analysis (protein)</text>
      <input>seqprot</input>
      <tool exe='/path/to/myanalysis'
            param='seqprot' plaintext='1' />
      <description>
         My amazing sequence analysis tool for proteins.
      </description>
   </seqtool>

   <seqtool id='dnaanalysis'>
      <text>My analysis (DNA)</text>
      <input>seqdna</input>
      <tool exe='/path/to/myanalysis'
            param='-d seqdna' plaintext='1' />    <!-- NOTE -->
      <description>
         My amazing sequence analysis tool for DNA.
      </description>
   </seqtool>
</saptf>
```

The `param=` attribute is handled in an intelligent manner. Anything that matches a sequence entry box name (and, as we will see later, other settable parameter names) will be replaced by the appropriate value. Anything that doesn't match will be passed unchanged. Consequently if your tool required a flag (say '`-type`')followed by the data type (DNA or protein) you could replace the `param=` attributes above with:

```
param='-type protein'
```

and

```
param='-type dna'
```

4.3 A tool with more than one input sequence

You can also have tools which require two or more sequence entry boxes — for example an alignment tool would need two sequences to be input. This is achieved just as you would expect:

```
<saptf>
   <seqbox id='seq1' hidden='1'>              <!-- NOTE -->
      <text>First sequence:</text>
   </seqbox>

   <seqbox id='seq2' hidden='1'>              <!-- NOTE -->
      <text>Second sequence:</text>
   </seqbox>

   <seqtool id='align'>
      <text>Protein sequence alignment</text>
      <input>seq1</input>                     <!-- NOTE -->
      <input>seq2</input>                     <!-- NOTE -->
      <tool exe='/path/to/myalignmentprogram'
            param='seq1 seq2' />              <!-- NOTE -->
      <description>
         Performs a sequence alignment.
      </description>
   </seqtool>
</saptf>
```

Note that there are two `<input>` tags for `<seqtool>` to tell it to display both sequence input boxes and that the `param=` attribute for the `<tool>` tag specifies the two sequence. Again these will both be passed on the command line to the specified executable program.

4.4 Pasing sequence data in a file and manipulating data

Often you may need to pass sequence data to a tool as a file instead of as a sequence on the command line. This is achieved by providing the sequence parameter in parentheses. The sequence data are then written to a file in `/tmp` and the filename is passed as a parameter to the program.

Consequently if, in the above example, the sequence alignment program required the two sequences to be in files, then one would simply replace the `param=` attribute with:

```
param='(seq1) (seq2)'
```

Of course most programs which take sequence data in a file need those data to be in a format like FASTA or PIR. In this case you can provide a function in your XML configuration file to manipulate the sequence into FASTA (or PIR) format. Functions are written in Perl (though any special XML characters have to be replaced with the appropriate XML — for example '`>`' must be replaced with '`>`') and can be used to manipulate the sequence data in any way that you like. Once manipulated, the sequence data are written to a file and the filename is passed to the executable program.

Note that while passing sequences on the command line cleans the sequences by removing whitespace and escaping non-word characters, no such cleanup is done if the sequence is passed via a file. If you wish to perform such cleanup, then you can provide a function to do that for you.

Here are two functions which provide FASTA and PIR formatting respectively:

```
<function name='fasta'>
sub fasta {return("&gt;foo\n@_");}
</function>

<function name='pir'>
sub pir {return("&gt;P1;foo\nfoo - bar\n@_*");}
</function>
```

Note that the Perl subroutine name must match the `name=` attribute of the `<function>` tag.

So if our alignment program required its input in FASTA format, we would simply insert the fasta function in the configuration file and replace our `param=` attribute with:

```
param='fasta(seq1) fasta(seq2)'
```

Functions can also be written to manipulate data without going via a file. In this case, the `nofile='1'` attribute is provided in the `<function>` tag. For example, the following function would remove any characters not found in a DNA sequence, but not pass the results via a file:

```
<function name='dna' nofile='1'>
sub dna
{   my $seq = join('',@_);
    $seq=~s/[\WBD-FH-SU-Zbd-fh-su-z]//g;
    return($seq);
}
</function>
```

4.5 Pull-down menus

Many tools will have options, or parameters, and these can be provided in various ways. The first of these is the pull-down menu. Returning to the protein/DNA example, you might simply provide a single sequence entry box and allow the user to specify whether it is protein or DNA using a pull-down menu. For example:

```
<seqtool id='numbering'>
   <text>My sequence analysis program</text>
   <input>seq</input>
   <tool exe='/path/to/numberingprogram'
         param='-type d seq' />                          <!-- NOTE -->
   <parameter id='d'
              type='pulldown'
              label='Sequence type'
              options='protein,dna'
              default='0' />                             <!-- NOTE -->
   <description>
      My amazing sequence analysis tool.
   </description>
</seqtool>
```

This example specifies a tool which can take a sequence and apply a numbering scheme to that sequence. There are two things to note here.

First, the `<parameter>` tag specifies the pull-down menu that will be provided to the user. This has an `id=` attribute which must be unique within *this* `<seqtool>`. The `type=` attribute specifies that this is a pull-down menu (we will see other types later). The `label=` attribute specifies text that will appear next to the pull-down menu. The `options=` attribute specifies the options that will appear in the pull-down menu as a comma-separated list. The selected one of these will also be passed to the executable program. The `default=` attribute specifies which one of these is the default option. This is the position of the option in the list (counting from zero).

Second, the `param=` attribute of the `<tool>`. Remember that any part of the `param=` attribute that matches the name of a sequence box will be replaced by the sequence given by the user. In the same way, if part of the `param=` attribute matches the `id=` attribute of a `<parameter>` tag, then the value selected by the user will be inserted.

4.6 Radio buttons

Alternatively you could use radio buttons rather than a pull-down menu. In this case you would do:

```
<parameter id='t' type='radio' label='Protein'
           value='protein' default='1' />
<parameter id='t' type='radio' label='DNA'
           value='dna' default='0' />

<tool exe='/path/to/myanalysis'
      param='-type t seq' />
```

Note that there is a `<parameter>` tag for *each* radio button and they have the same `id=` attribue to indicate a mutually-exclusive group of buttons.

The `default=` attribute is used to indicate which radio button is pressed by default.

Radio buttons have one advantage over pull-down menus. In pull-down menus, the value displayed in the menu is passed to the executable program. With radio buttons you specify a label for the button (with the `label=` attribute) and a separate value to be passed to the program (with the `value=` attribute).

4.7 Checkboxes

Alternatively, if the program assumes that the data are protein unless you specify a '`-dna`' flag, then you might want to use a checkbox such that the user enters a protein sequence normally and a DNA sequence when the checkbox has been ticked:

```
<parameter id='t' type='checkbox' label='DNA instead of protein'
           value='-dna' default='0' />
<tool exe='/path/to/mytool'
      param='t seq' />
```

In this case, the `default=` attribute indicates whether or not the box is ticked by default (0: no; 1: yes)

4.8 Numeric inputs

You might also need to specify a numeric option to a program. This is done in the same way using a parameter tag. For example:

```
<parameter id='t' type='numeric' label='Threshold' default='1.0' />
<tool exe='/path/to/mytool'
      param='-t t seq' />
```

The 't' in the param= attribute would be replaced by the user-specified value. So in this case, if the user specified the sequence used above and didn't alter the default, the program would be run as:

```
/path/to/mytool -t 1.0 SASRTYMNPRSTALNKHGTGHFDLNKMEDTHYRTSL
```

Of course you can mix different types of input:

```
<parameter id='t' type='numeric' label='Threshold' default='1.0' />
<parameter id='d' type='pulldown' label='Sequence type'
                  options='protein,dna' default='0' />
<tool exe='/path/to/mytool'
      param='-t t -type d seq' />
```

4.9 Slow programs

If a program runs for a long time, then the user may simply wonder what is happening, or the web server may time out. Adding the `slow='1'` attribute to the `<tool>` tag tells SAPTF to use an AJAX-based mode which will allow the program to run while displaying a 'throbber' (an animated GIF) with a 'wait' message. When the program finishes, this will be replaced by the results of the program. However, this will not deal with programs that are so slow that the web server times out.

With the attribute set to greater than 1 (e.g. `slow='2'`), a different method is used in which the page refreshes (through AJAX) until the results are ready. This deals with programs that run for such a long time that the web server would time out. The number that you specify is reported in the 'wait' message to the user to say that the program may run for the specified number of minutes and a counter is provided that decrements every quarter of a minute. If the counter reaches zero before the program returns a result, then it is deemed to have timed out and a timeout error message will be reported to the user.

This removes a huge amount of coding effort in dealing with such programs.

4.10 Configuration options

The options allow control over the title text provided by SAPTF and colours and size of the help popup (which isn't controlled by CSS).

All options are contained within a `<config>` tag. For example:

```
<config onetool='1'>
   <title>SAPTF Demo</title>
   <subtitle>Sequence analysis</subtitle>
   <resultssubtitle>Your analysis results...</resultssubtitle>
   <toolstitle>Select analysis type</toolstitle>
   <datatitle>Enter sequence data</datatitle>
   <overlib caption='SAPTF Demo Help' width='600'
            fgcolor='#CCCCCC' bgcolor='#AAAAAA'
            closecolor='#3333FF' capcolor='#AA0000' />
</config>
```

The options are as follows:

`<config onetool='1'>`	The `onetool='1'` attribute changes the formatting for cases where there is only one tool. The toolstitle (see below) is not displayed and the bulletted list and radio button are removed
`<title>`	The header-1 title for the input and results pages
`<subtitle>`	The header-2 title for the input page
`<resultssubtitle>`	The header-2 title for the results page
`<toolstitle>`	The header-4 title that appears just above the list of tools
`<datatitle>`	The header-4 title that appears just above the sequence input box(es)
`<overlib caption=''>`	The text to display as a caption in the help boxes
`<overlib width=''>`	The width of the help boxes
`<overlib fgcolor=''>`	The colour of the background behind the main help text
`<overlib bgcolor=''>`	The colour of the background behind the help caption
`<overlib closecolor=''>`	The colour of the help close button
`<overlib capcolor=''>`	The colour of the help caption

5 Advanced Use

The '`advanced`' subdirectory demonstrates how you can use SAPTF as part of a more complex set of web pages. In particular, it shows you how you can use the Perl Template Toolkit to write a more advanced `html.pm` which integrates the SAPTF pages into a menu-based system.

Assuming you have the Perl Template Toolkit installed, then after running the main install script, enter the '`advanced`' directory and simply type:

```
make
make install
```

This will build an HTML index page and the new `html.pm` Perl module (using headers, footers and menus supplied in separate `.tt` files), and will then install them in the HTML directory you specified while doing the install of SAPTF.

You can now point your browser at the index file in the web directory. i.e. `http://MYHOST/MYPATH/`