

Scientific Application Web Server (SAWs) Users Manual

Mathematics and Computer Science Division

About Argonne National Laboratory

Argonne is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC under contract DE-AC02-06CH11357. The Laboratory's main facility is outside Chicago, at 9700 South Cass Avenue, Argonne, Illinois 60439. For information about Argonne and its pioneering science and technology programs, see www.anl.gov.

Availability of This Report

This report is available, at no cost, at <http://www.osti.gov/bridge>. It is also available on paper to the U.S. Department of Energy and its contractors, for a processing fee, from:

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831-0062
phone (865) 576-8401
fax (865) 576-5728
reports@adonis.osti.gov

Disclaimer

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor UChicago Argonne, LLC, nor any of their employees or officers, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of document authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof, Argonne National Laboratory, or UChicago Argonne, LLC.

Scientific Application Web Server (SAWs) Users Manual

by
Matt Otten, J. Brown, and B. Smith
Mathematics and Computer Science Division, Argonne National Laboratory

September 30, 2013

Scientific Application Web Server (SAWs) Users Manual

Matthew Otten
Department of Physics
Cornell University
Ithaca, NY, USA
mjo98@cornell.edu

Jed Brown and Barry Smith
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL, USA
{jedbrown,bsmith}@mcs.anl.gov.

September 3, 2013

For simulations that run for an extended period of time one often wishes to know the current state of the simulation without constantly logging large amounts of information to files. The Scientific Application Web server, or SAWs, provides dynamic access to data (and the altering of data) in a simulation as it runs. Using a lightweight web server, the SAWs C library is linked with an existing C/C++/Fortran program and provides RESTful URLs for user selected variables in the simulation. Accessing those URLs returns JSON-represented values of the variables which may then be processed in a multitude of ways. SAWs includes some Javascript code, in ./js/jsSAWs.js, that calls the RESTful interface and displays the results in a browser. SAWs also provides a sample bash script that access the RESTful interface, in ./bin. Tools like jshon can be used to write sophisticated shell scripts that processes the SAWs data.

1 Simplest Use Case

The simplest SAWs program has only three calls:

```
#include <SAWs.h>
SAWs_Initialize();
SAWs_Add_Variable("variableName",&variableData,dataLen,SAWS_READ,SAWS_DOUBLE);
/*Run program, view data via client*/
SAWs_Finalize();
```

Then point your browser to <http://localhost:8080/> to view the data.

Sometimes the simulation code may be run on a system that is not publicly reachable from the internet so you cannot connect directly to it with your browser, fortunately if you can ssh to the system, you can tunnel the HTTP connection and still attach to the simulation. For example, if the simulation is running on saws.xxx.edu and you need to ssh first to login.xxx.edu and then to saws.xxx.edu, you can run “ssh -L 8080:saws.xxx.edu:8080 login.xxx.edu” on your local machine to set up the tunnel. After the tunnel is created, connect to port 8080 on your local machine and it will automatically connect to the server running on saws.xxx.edu. The iPhone/iPad and Android have several apps that allow setting up ssh tunnels.

2 Installation

To install, download, or clone the git repository from <https://bitbucket.org/saws/saws>, run

```
./configure [--enable-shared] [--prefix=installdirectory] [--with-cc=ccompiler]
make
[make install]
```

This generates lib/libSAWs.a and include/SAWs.h in the prefix directory. If a prefix directory is not provided, the results are stored in ./build.

If you prefer not to make a library, you can simply copy the three source files and three include files in ./src to your project's directory and list the source in your makefile.

3 Problems

To report problems or request help please use <https://bitbucket.org/saws/saws/issues>.

4 C API – Serving Your Variables

The interface to the C API is provided in SAWs.h. SAWs has four sets of functions in the C API for

- setting SAWs options,
- starting and stopping SAWs,
- indicating which variables should be served by SAWs, and
- locking and unlocking the SAWs variables.

All SAWs functions return an integer error code, which is 0 on no error. The error codes are provided in SAWs.h. We strive to make SAWs tolerant to failure so that SAWs will never crash the simulation code even if SAWs itself cannot run.

4.1 Setting SAWs Options

The first set of options must be called before SAWs_Initialize().

- int SAWs_Set_Use_HTTPS(const char* certificate) – With this option, SAWs will use the HTTPS protocol instead of the default HTTP protocol. Users are responsible for generating the SSL certificate they wish to use.
- int SAWs_Set_Port(int portnumber) – This option sets the port number for SAWs to use; the default is 8080.
- int SAWs_Set_Use_Logfile(const char *filename) – With this option, SAWs will log all requests to the file (if the filename is NULL the SAWs will log to SAWs.log).
- int SAWs_Set_Document_Root(const char *directory) – By default SAWs will not serve any files from the filesystem. If this option is provided then SAWs will look in this directory to resolve any requests it cannot handle. Often . is the appropriate root to use. One should be careful not to use a root directory that contains private files.

SAWs has two optional sets of parameters that may be set or changed at any time. These provide a way for the application code to supply HTML that they want served by SAWs in addition to the RESTful, [4], requests. By default “index.html” is served with a simple HTML page that provides a GUI to display the available variables.

- `int SAWs_Set_Header(const char *URL,const char *htmltext)` – the HTML `<HEAD>` portion of the page for that URL page. Generally the default provided by SAWs is fine, and this does not need to be used unless one is providing additional Javascript code that needs to be defined in HEAD.
- `int SAWs_Set_Body(const char* URL,int part,const char *htmltext)` – the HTML `<BODY>` portion of the page. The displayed body is obtained by concatenating three provided body parts. If certain body parts are not provided, the defaults are used when the URL is “index.html”; otherwise empty defaults are used.
- `int SAWs_Set_Local_JSHeader(void)` – causes SAWs to reference its Javascript libraries in the `js` subdirectory of the root directory set with `SAWs_Set_Document_Root()` rather than locating them over the internet. This is for situations when the SAWs server and client are being used without internet access. Note that you must copy the `js` directory from SAWs to this location. You must also provide a document root directory for this to work correctly. This option is ignored if you have called `SAWs_Set_Header()` for “index.html” and any other BODYs that you are providing, since the locations of the Javascript, in that case, is determined by what you have provided in the header.

4.2 Starting and Stopping SAWs

- `int SAWs_Initialize()` – This routine may be called multiple times with intervening calls to `SAWs_Finalize()`.
- `int SAWs_Finalize()` – This routine automatically deletes any registered variables that have not yet been deleted. Any options set by the user will remain in effect if `SAWs_Initialize()` is called again.

4.3 Registering Variables

- `int SAWs_Register(const char* URL,void *addr,int len,SAWs_Memory_Type mtype,SAWs_Data_Type dtype)` – registers an array of length `len` whose entries are of type `dtype`. The user is responsible for ensuring that `addr` remains valid until the call to `SAWs_Delete()` is called. `mtype` must be either `SAWs_READ` or `SAWs_WRITE`, and `dtype` must be one of `SAWs_CHAR`, `SAWs_BOOLEAN`, `SAWs_INT`, `SAWs_FLOAT`, `SAWs_DOUBLE`, or `SAWs_STRING`. Once a variable is registered, its URL is of the form `[dir1/[dir2/][.../]]variablename` with any number of nested directories. By convention all variables that begin with an underscore are not displayed in the SAWs Javascript GUIs.
- `int SAWs_Delete(const char* URL)` – makes the variable no longer available to the server.

4.4 Locking the Variables

When the server thread is accessing the registered variables and packing them to send to the requester it, the main simulation process may be changing the values, and hence inconsistent values may be sent to the client. For example, some of the entries in the array may be new, while others may be old. To prevent this situation, the main simulation process should “lock” the server thread while it changes any registered variables and then immediately unlock the server thread when it is finished changing values. While SAWs is locked the server thread cannot access the variables; thus the simulation should lock the variables for as short a time as possible.

- int SAWs_Lock()
- int SAWs_Unlock()

5 RESTful API

SAWs makes use of HTTP (or HTTPS) through the RESTful API. In order to access the data, a GET request is sent to the proper URL. The URL to get a specific named variable is /SAWs/dir1/dir2/.../variable. SAWs also allows obtaining an entire directory of variables with /SAWs/dir or obtaining all variables with /SAWs/. The response is sent back as JSON, [3]. In order to update variables on the server, a POST request is sent to the same URL where the data was obtained; for example, to update variables in directoryname, a POST is sent to that directory.

SAWs can also server webpages from the filesystem if SAWs_Set_Root_Directory() is provided. A sample is provided in ./examples/index.html.

6 JavaScript API

SAWs provides a small number of useful Javascript routines for communicating with the server. We expect many clients to provide their own additional Javascript code specific to their application. The Javascript is in js/jsSAWs.js.

- SAWs.getDirectory(URL,callback,callbackdata) – sends a GET request to the server passing the URL. When the JSON data is received back from the server, the function callback is called with two arguments. The first argument is the data received from the server, and the second is the callbackdata passed in. The call to the callback function is asynchronous.
- SAWs.displayDirectory(directory,divEntry) – takes the directory information (received from the server) and displays it in the provided div.
- SAWs.getAndDisplayDirectory(URL,divEntry) – sends a GET request to the server, receives the response, and displays the response in the provided div.
- SAWs.postDirectory(directory) – sends a directory to the server with a POST request and any SAWS_WRITE variables on the server are replaced with the provided ones.
- SAWs.updateDirectoryFromDisplay(divEntry) – takes any values changed in the div object, puts them in the appropriate directory object, and passes that up to the server with SAWs.postDirectory().

7 JSON API

The JSON that is transmitted between the client and the server is of the following form:

```
{
  "directories": {
    "SAWs_ROOT_DIRECTORY": {
      "variables": {},
      "directories": {
        "dir1": {
          "variables": {},
          "directories": {
```

```

"dir2": {
  "variables": {,
    "variable1":{"data":["thedata"],"dtype":"thedtype",
                  "length":"thelength","mtype":"themtype","alternatives":[]}
  }
  "directories":{
    "dir3": {
      "variables": {
        "variable1":{"data":["thedata"],"dtype":"thedtype",
                      "length":"thelength","mtype":"themtype",
                      "alternatives":[]}
        "variable2":{"data":["thedata"],"dtype":"thedtype",
                      "length":"thelength","mtype":"themtype",
                      "alternatives":[]}
      }
    }
  }
}
}}}

```

8 How It Works

SAWs.Initialize() starts up a lightweight web server known as Mongoose [1] that runs on a different pthread from the main simulation. When started, Mongoose is passed a function pointer that is called with the data received from any client.

If the requested URL begins with SAWs, then the handler uses the received URL to locate the registered directory or variable that is requested. If a GET request was received it then converts the directory or variable value(s) to JSON and passes the resulting JSON back to the requester. In the case of a POST request, the handler parses the received JSON and updates the variable values within the directory based on the received JSON data. SAWs.Register() maintains a tree data structure with all the variables that have been registered, the directory and variable names, and the address pointers. The handler can then simply search down through the tree data structure to match the URL that has arrived with a path in the tree.

For all URL requests that do not begin with SAWs the URL is resolved in two ways. If SAWs.Set_Document_Root() was called, then SAWs will look at the file system for the given URL (filename); if found, the file is sent back to the client. If the file is not found or the root directory was not set, then SAWs will look for a header or body that was provided by SAWs.Set_Header() or SAWs.Set_Body() with the requested URL name and transmit that text to the client. Otherwise it transmits a "not found." The default HEAD and BODY text may be found in the source code SAWs.c. The file ./examples/index.html gives an example of their forms when providing a file to serve the page.

The Javascript implementation uses jQuery's AJAX methods to communicate with the RESTful interface of the server; that is, it sends HTTP requests to the server, receives the JSON response and then converts the received JSON to a Javascript object. In addition, jQuery is used to display

data to the div objects defined on the HTML webpages. This code may be found in ./js/jsSAWs.js.

References

- [1] Mongoose: Sergey Lyubka, Available from: <http://code.google.com/p/mongoose/>.
- [2] cJSON: Dave Gamble. Available from: <http://sourceforge.net/projects/cjson/>.
- [3] JSON. Described at: <http://www.json.org/>.
- [4] RESTful. http://en.wikipedia.org/wiki/Representational_state_transfer



Mathematics and Computer Science Division

Argonne National Laboratory
9700 South Cass Avenue, Bldg. 240
Argonne, IL 60439-4847

www.anl.gov



Argonne National Laboratory is a U.S. Department of Energy
laboratory managed by UChicago Argonne, LLC