

NPSGD User Guide

Thomas Dimson

December 29, 2010

Abstract

NPSGD is an online framework that makes it easy for scientists to expose their models online. This document provides an general overview of how the system works with some administrative tips along the way. Notably, it does not talk about the inner workings of how to create models as this is documented in the guide called “Adding a new model to NPSGD”.

Contents

1	System Requirements	2
2	System Overview	2
2.1	Daemons	2
2.2	Request Flow	3
2.3	Fault Tolerance	3
3	Software Layout	4
4	Development Environment	4
5	A word on models	5
6	Templates	5
6.1	HTML Templates	6
6.2	Email Templates	6
6.3	LaTeX Templates	7
7	Static Files	7
7.1	Javascript	7
7.2	CSS	7

1 System Requirements

A basic configuration will require:

- Python 2.5 or higher <http://www.python.org/>
- Facebook's Tornado Web server for Python <http://www.tornadoweb.org/>
- L^AT_EX distribution of some form.
- If you are going to use some Matlab scripts, some version of Matlab is required
- UNIX-like operating system. Cygwin may work but has not been tested.

2 System Overview

2.1 Daemons

NPSGD is a distributed system which communicates internally and externally via http. One of the design goals of the system was to allow multiple machines to perform the function of worker so as not to overload any particular system. To facilitate this, the framework is split into three daemons:

- **npsgd_web**: The frontend to NPSGD. The job of this daemon is to communicate with all external clients over http. Since no state is kept inside the daemon, you can have as many frontends as desired (though one should be sufficient unless you are under heavy load). By default, a client visiting <http://localhost:8000> will hit this daemon.
- **npsgd_queue**: The manager of NPSGD. The job of this daemon is to keep track of the state of particular model runs, send out e-mail confirmations, and to distribute jobs to worker machines. There can only ever be one queue within an NPSGD system. By default, it runs on port 9000 but should not be accessed externally (in particular, it should be firewalled off).
- **npsgd_worker**: The model processor of NPSGD. The job of these daemons are to poll the queue for model requests and process them. Generally, this takes the form of a finite state machine that spawns some external process to actually perform the scientific simulation. After processing, it sends results to the requestor's e-mail address.

If NPSGD is being configured for long term access, all of these daemons should have startup scripts associated on the end machine. Sample scripts are available in the **startup/** directory alongside the NPSGD distribution.

Each model that is implemented for NPSGD must be available for the queue, the web frontend and the worker machines. A NFS mount for the models is particularly handy for this purpose (though not necessary).

2.2 Request Flow

A sample model of NPSGD is available at <http://www.npsg.uwaterloo.ca/models/ABMU.php>. It would be helpful to try this out in order to gain understanding of the system. The typical request flow would be something like this:

1. User visits the model site via <http://localhost:8000/models/example>, which is served using `npsgd_web`.
2. User submits a request for the example model, `npsgd_web` makes a request to `npsgd_queue` to submit a confirmation code.
3. `npsgd_queue` e-mails a confirmation code to the user's e-mail address.
4. User clicks the link in the e-mail, usually something like http://localhost:8000/confirm_submission/code.
5. `npsgd_web` forwards the confirmation onto `npsgd_queue`. At this point the model request is ready to be processed.
6. `npsgd_worker` polls the queue for a task. Since there is a task, it receives a task response.
7. `npsgd_worker` prepares execution, spawns a subprocess to open up the model run.
8. Model execution completes. `npsgd_worker` creates a \LaTeX document with the results.
9. \LaTeX document and other attachments are sent to the user's e-mail. `npsgd_worker` tells the queue that it has executed successfully.

2.3 Fault Tolerance

NPSGD has been designed to be fault tolerant. Whenever a web request is made, `npsgd_web` will check with the queue to ensure it is up and that it has workers on the line that have polled for a request recently. If one of these conditions is not met, a simple error message is displayed to the requestor.

When processing a task, `npsgd_workers` communicate back to the queue with a heartbeat request to make sure that the queue knows that the worker is still alive. If this heartbeat has not been heard for a long while, the queue daemon will put the request back into the queue.

All requests are tried for a fixed number of times before a complete failure (by default, the queue will retry a request three times before declaring it failed). Upon failure, an error message is e-mailed to the user.

If the e-mail system is ever down, NPSGD will keep trying to send e-mails out until they have successfully completed.

One particular "gotcha" at this point is that the queue server is not persistent: all requests are kept in memory. This makes it particularly important to make sure it does not go down as requests could be lost if it does it midway through processing. A persistent queue file is a large possibility for future development.

3 Software Layout

NPSGD ships in exactly the same way it was developed and isn't easily converted into a form that matches the standard UNIX layouts. When you clone the main repository for NPSGD, you will see the following files and directories:

- `config.example`: An example configuration file for NPSGD
- `epydoc.cfg`: Config file for generating an HTML representation of the code using epydoc.
- `LICENSE`: License agreement for distribution
- `README`: Easy readme guide to NPSGD
- `npsgd_web.py`: Web daemon for handling client requests.
- `npsgd_queue.py`: Daemon for queueing NPSGD model runs.
- `npsgd_worker.py`: Worker for actually processing NPSGD model runs.
- `doc/`: Directory containing NPSGD documentation.
- `models/`: Default directory for storing user-defined models
- `npsgd/`: Helper package containing modules required to run the daemons.
- `startup/`: Directory containing some sample startup scripts for Ubuntu (must be modified to specify paths correctly).
- `static/`: Directory containing static files served by the web server (Javascript, css).
- `templates/`: Directory containing templates used by NPSGD

4 Development Environment

Since NPSGD consists of so many moving parts, creating a quick development environment can be quite a task. First of all, ensure that all the system requirements are installed. After that, you will want to clone NPSGD from the main development site and then make a custom copy of the config file.

```
git clone git://github.com/cosbynator/NPSGD.git
cp config.example config.cfg
```

You will need to edit the config file to make a few changes, most notably:

- Change `npsgbase` to point at your development directory
- Change `pdflatexPath` to point at your `pdflatex` binary.
- Change the `email` section to specify a valid username/password. The defaults are approximately set up for a gmail account (obviously with an invalid username/password).

- Change `matlabPath` if you will be testing/editing matlab models

After completing these sections you will be able to run all three daemons on a local terminal. Specifically, run the following in separate terminal windows:

```
python npsgd_web.py
python npsgd_queue.py
python npsgd_worker.py
```

Logging will be printed to standard error unless otherwise specified. After running `npsgd_web` you should be able to browse to `http://localhost:8000/models/example` and see some sample output. Try performing a complete model run to make sure the system is operating correctly.

5 A word on models

Models are the key component to customizing NPSGD. The scope of this document does not cover the *implementation* of custom models (see “Adding a new model to NPSGD” for that). Still, it is important to understand how they fit into the system.

Models are really just Python files that inherit from `ModelTask` in order to provide complete flexibility for underlying implementation. These python files often act as wrappers to existing models that are implemented in some other language (we have models implemented in both Matlab and C++). Just because they are Python files does not force the implementation to be in Python: Python just acts as a facilitator between NPSGD and the underlying model.

Each daemon is configured to monitor the model directory, by default under `models/`. Python files in this directory act as “plugins” to the system - models are implemented in them. The daemons periodically scan the directory and look at all the python files, those that inherit from `ModelTask` are imported into the system. When importing, a hash of the file is used to tag the model with a “version”. Duplicate versions of models are ignored (not imported into the system), while new versions are stored. Old versions are retained in the system in case there are any queued requests that are still acting on the old version. For this reason, it is important to reload the page after updating any models.

Models are self-contained: they have all the information necessary for the queue, web and worker daemons to operate. In particular, they contain the parameters necessary for the web interface to configure the models and the execution cycle necessary in order for the workers to run a model with a particular parameter set. The code for the models must be shared across the daemons: this is done manually, or via a NFS mount.

6 Templates

Templates have many purposes in NPSGD:

1. To create the HTML that is served by the web daemon
2. To create the e-mail sent by the queue and worker daemons

3. To create \LaTeX markup to send results to the user

All templates for npsgd are stored in the `templates/` subdirectory. This is meant to be user defined. The default templates shipped with NPGSD are quite specific to our own models.

Templates use a syntax created for the Tornado web server. This syntax is documented at the Tornado site: <http://www.tornadoweb.org/>, under templates.

6.1 HTML Templates

NPSGD ships with two sets of html templates, one for an embedded site (served directly without things like title tags), and one for a basic site. These templates are used by the web daemon to display html directly to the user. Both are available under the `templates/html/` subdirectory. These files contain:

1. `base.html`: Template that all other templates inherit from. Could contain things like HTML headers.
2. `model.html`: The template that displays a model to the user.
3. `model_error.html`: The template that shows an error when a something goes wrong in the model.
4. `confirm.html`: Template that shows after a model request has been made (displays that an e-mail will be dispatched to the user).
5. `confirmed.html`: Template that shows after a confirmation code has been entered correctly.
6. `already_confirmed.html`: Template that shows after a confirmation code has been entered for a second time. This displays a message saying that the model will not be rerun.

6.2 Email Templates

Emails are used as communication to the user after a request has been performed via HTML. Each template comes in two parts: one for the email subject, and one for the email body. The naming convention is “name_body.txt” for body and “name_subject.txt” for subject. In particular, these templates are available:

1. `results_email`: The email used to display model run results
2. `failure_email`: The email used to notify the user that we could not complete a model request.
3. `confirm_email`: The email used to communicate the confirmation code for a particular model request.

6.3 LaTeX Templates

There is only one LaTeX template, namely `result_template.text`. This is used to declare all the packages required for a model results PDF. Inside, there is an empty variable where the details of a model run will go. Models themselves specify the model details, but it is always wrapped in this template.

7 Static Files

In addition to templates, NPSGD needs the use of certain static files to customize the in-browser behaviour. All static files are available in the `static/` subdirectory.

7.1 Javascript

NPSGD uses a lot of javascript to do parameter verification and provide UI widgets for the user. In the `static/js` subdirectory, you will find

1. `npsgd.js`: our particular javascript files
2. `jquery-version.min.js`: JQuery, a javascript library that makes writing javascript simpler: <http://jquery.com/>
3. `jquery.qtip.min.js`: A library for creating popup tooltips. This is used for model's helper text. <http://craigsworks.com/projects/qtip/>
4. `jquery-ui-version.min.js`: JQuery UI, a set of UI widgets for JQuery: <http://jqueryui.com/>. This is used to give a nice user experience for things like sliders and range selectors.
5. `jquery.validate.min.js`: JQuery validation plugin. We use this to perform parameter verification in client side, before requests are sent to the server: <http://bassistance.de/jquery-plugins/jquery-plugin-validation/>

7.2 CSS

In addition to Javascript, some basic CSS needs to be shared across all templates. These are in the `static/css` subdirectory.

1. `npsgd.css`: This provides just a couple of CSS includes that are needed across all templates.
2. `smoothness/`: This subdirectory contains the default CSS files for the JQuery UI project. It is used to display the widgets of JQuery UI.

7.3 Images

Finally, there are some images that NPSGD needs to display available in the `static/images` subdirectory. These should mostly be self explanatory.