Opal Toolkit Reference Guide

Reference Guide for the Opal Toolkit version 2.5 Edition

Opal Toolkit Reference Guide :

Reference Guide for the Opal Toolkit version 2.5 Edition
Published 2012

Table of Contents

Chapter 1. Overview

Opal Toolkit version: 2.5

The Opal toolkit enables wrapingp existing scientific applications as Web services, and exposing them to various clients. The implementation provides features such as **scheduling** (e.g. using Condor, PBS and SGE via DRMAA), **security** using GSI-based certificates, **job and data management** by executing every job in a separate working directory, **state management** by storing the service state via Hibernate in a database such as HSQL, PostgreSQL, MySQL or DB2).

The application developer specifies a configuration for a scientific application and deploys the application as a web service. Now this application is available via service interface to remote end users.

The service interface is described in Web Services Description Language (WSDL) and is defined in *wsdl/opal.wsdl* file. Stub generators provided by Web service toolkits are used to generate the client and server-side bindings for the services. The services are implemented in Java using the Apache Axis toolkit, while the clients can be written in any language.

The WSDL defines the following operations:

- **getSystemInfo** - get system information. This operation returns basic system information for the Opal server, including the number of total and free CPUs, number of jobs running and queued, the job manager type, etc.
- **getAppMetadata** - get application metadata. This operation returns application metadata which includes usage information, along with any number of arbitrary application-specific metadata specified as an array of `info` elements, e.g. description of the various options that are passed to the application binary. Ideally, these elements should be arbitrary XML tags (specified by `xsd:any`). Due to the limitations of certains toolkits in other languages (e.g. ZSI in Python), we specify them as *strings*. You may embed XML tags inside them using *CDATA* sections. The metadata can include an optional `types` element, which describes the application command-line arguments. This element is described in detail in the Advanced Submission Form section.
- **launchJob** - launch job. This operation requires a list of arguments as a string, and an array of structures representing the input files. Each structure contains the *name* of the input file and either the *contents* in Base64 encoded binary form, a MIME *attachment*, or a *location* (URL) to the associated file. The operation returns a Job ID that can be used to retrieve jobs' status and outputs.
- **launchJobBlocking** - launch blocking job. This operation requires a list of arguments as a string, and an array of structures representing the input files. The operation blocks until the remote execution is complete, and returns job outputs (as described above) as the response. This operation is appropriate only for jobs that are short running.
- **queryStatus** - query job status. This operation expects a Job ID to query the status of a running job and returns a status code, message, and URL of the working directory for the job.
- **getJobStatistics** - get job statistics. This operation returns basic job statistics including start time, activation time and completion time for a given Job ID.
- **getOutputs** - get job outputs. This operation returns the outputs from a job that is identified by a Job ID. The output consists of the URLs for the stdout and stderr, and an array of structures representing the output files. The structure contains the *name* of the output file and the *url* from where it can be downloaded.
- **getOutputAsBase64ByName** - get output file by name. This operation returns the contents of an output file as Base64 binary. The input is a data structure that contains the Job ID for a particular job, and the name of the file to be retrieved.
- **destroy** - destroy job. This operation destroys a running job identified by a Job ID.

Please see Opal use cases examples[1] for more info.

Notes

1. http://www.nbcr.net/data/docs/opal/deployment.html

Chapter 2. Opal Server Installation

2.1. Prerequisites

2.1.1. Common Prerequisites for Server and Client

1. **Java 1.6.x or higher**
   - Download and install J2SE from http://www.oracle.com/technetwork/java/javase/downloads[1]
   - Add the installation bin directory to the environment variable *PATH*.
   - Set the environment variable *JAVA_HOME* to point to the top level directory of the java installation.
2. **Ant 1.7.1 or higher**
   - Download and install ant from http://ant.apache.org[2]
   - Set the environment variable *ANT_HOME* to point to your ant installation.

2.1.2. Additional Prerequisites for Server

1. **Tomcat 6.0 or 7.0**. Tomcat is a servlet container that hosts Opal Web services. You can download the source distribution or the binary version of Tomcat 6.0 or 7.0 from http://tomcat.apache.org/. In this reference guide we will refer to the location of the Tomcat installation as *CATALINA_HOME*.

2.1.3. Optional Prerequisites for Server

1. **MPI**: If you plan to run your application in parallel, you will need a version of MPI. We have run parallale applications using OpenMPI[4]. Make sure that the SSH keys (for the account that runs tomcat) are set up correctly in order to be able to run MPI jobs without being prompted for a password.
2. **Relational Database**: By default, we use a java-based in-memory and disk-based HSQLDB[5] database for persistent services states. For production use we recommend using a real database such as:
   - Postgres[6] (v.8.2.4 or higher),
   - MySQL[7] (tested with v.5.1.61), or
   - DB2[8] (tested with v.8.2).
3. **Batch Job Scheduler Tools**. Opal can be set up to submit jobs directly to schedulers such as Condor or TORQUE/PBS simply by writing a shell script. Another alternative is to access schedulers via the DRMAA[9] API. The Opal services can submit jobs to schedulers as long as they support the DRMAA API.

   We have tested:

- Condor[10] (serial jobs)
- SGE via DRMAA API[11] (serial and parallel jobs)
- TORQUE/PBS[12] (serial jobs)

2.2. Installation Instructions

The Opal source distribution is available for download from source forge[13] and can be used to install a server and a command-line client on Unix and Windows platforms.

Extract the source from the downloaded tarball using the GNU tar (or other similar utilities for Windows):

```
tar zxvf opal-ws-2.5.tar.gz
```

This should create a new directory called opal-ws-2.5/ where all the sources are expanded. Henceforth, we refer to this directory as *OPAL_HOME*.

1. Edit **$OPAL_HOME/etc/opal.properties** to configure the static container properties correctly. The template file looks similar to the following:

   ```
   # the base URL for the tomcat installation
   # this is required since Java can't figure out the IP
   # address if there are multiple network interfaces.
   tomcat.url=http://localhost:8080

   # the path relative to $CATALINA_HOME (or absolute)
   # where opal will look for the xml file to authomatically
   # deploy
   opal.deploy.path=deploy

   # parallel parameters
   num.procs=1
   mpi.run=/path/to/your/mpi

   # zip up input/output files, if set to true
   # data.archive=true

   # location of working directory relative to $CATALINA_HOME/webapps.
   # this could be a symbolic link to another location (which should be
   # NFS mounted if this is on a cluster). if this is a symlink, copy
   # etc/opal.xml to $CATALINA_HOME/conf/Catalina/localhost/opal.xml. if
   # the name of the symlink is changed to something other than "opal-jobs",
   # modify the opal.xml accordingly
   # working.dir=opal-jobs

   # by default, opal doesn't allow the use of absolute paths in the command-line
   # set the following parameter to allow some exceptions
   ```

```
# allowed.path.prefixes=/db/, /whatever/dir/, /another/one/

# use this key to display how long to save user data on server
opal.datalifetime=4 days

# specify in seconds the hard limit for how long a job can run
# only applicable if either DRMAA or Globus is being used, and if
# the scheduler supports it (some SGE versions ignore this parameter)
# Please be aware that your application will be killed by the scheduler
# once it reaches the specified limit (in same cases without any log)
opal.hard_limit=3600

# full qualified class name (FQCN) of the job manager being used
opal.jobmanager=edu.sdsc.nbcr.opal.manager.ForkJobManager
# opal.jobmanager=edu.sdsc.nbcr.opal.manager.DRMAAJobManager
# opal.jobmanager=edu.sdsc.nbcr.opal.manager.CondorJobManager
...
```

- Set the IP adress and port in **tomcat.url** to the correct values `http://your.ip.address:yourport` of your server.
- If the Opal installation needs to support parallel applications, set the **num.procs** to the number of processors available, and the **mpi.run** to the location of the mpirun on your host.
- By default, all new working directories for job executions are created inside the *$CATALINA_HOME/webapps/ROOT* directory. If this needs to change to another location on your system (when installing on a cluster, should be NFS mounted) uncomment the property **working.dir and set it accordingly**.

Once done with customizing, run the following commands:

```
cp $OPAL_HOME/etc/opal.xml $CATALINA_HOME/conf/Catalina/localhost/
cd $CATALINA_HOME/webapps
ln -s /path/to/working_dir/on/nfs opal-jobs
```

Note that the above use of symbolic links will only work on Unix systems.

2. If you would like to install a database for persistent service state, please refer to Opal State Database Installation chapter.
3. If you would like to setup scheduler support, refer to Scheduler Support Using Job Managers chapter.
4. You can optionally set up secure access to your services by consulting Configuring GSI-based Security section.
5. Edit the **$OPAL_HOME/build.properties** file to ensure that the build properties are set correctly.
   - Set `catalina.home` to point to the location of your Tomcat installation (i.e. $CATALINA_HOME)
   - Set **tomcat.port** to the port number that the Tomcat server is running on.
6. Install the Opal toolkit into the Tomcat installation, by executing:

   ```
   ant install
   ```
7. If you are using Tomcat 5.5.X, you have to enable directory listing to display jobs outputs properly. Find the definition of the main servlet in the file *$CATALINA_HOME/conf/web.xml*:

   ```
   <servlet>
       <servlet-name>default</servlet-name>
   ```

```
        <servlet-class>
          org.apache.catalina.servlets.DefaultServlet
        </servlet-class>
        <init-param>
            <param-name>debug</param-name>
            <param-value>0</param-value>
        </init-param>
        <init-param>
            <param-name>listings</param-name>
            <param-value>false</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
```

And make sure that the **<param-name>** *listings* is set to true:

```
            <param-name>listings</param-name>
            <param-value>true</param-value>
```

8. Set your JAVA_OPTS to "-Djava.awt.headless=true" to enable Tomcat to run in headless mode. Run the Tomcat server by changing to the $CATALINA_HOME/bin directory, and running the appropriate command:

```
./startup.sh (on Unix)
./startup.bat (on Windows)
```

9. Validate that the Opal has been installed correctly by following http://yourhost:8080/opal2/happyaxis.jsp. If you deployed Tomcat on another port, you will have to change the port number above. If all the **Needed Components** are found, Axis has been deployed fine. You can ignore the warnings about the **Optional Components**.
10. If the above steps have been executed successfully, the Opal server is ready for deployment of applications as Web services.

2.3. Application Deployment

1. Every application that is deployed as an Opal service requires an Opal configuration file. A sample configuration file `$OPAL_HOME/configs/date.xml` for the command */bin/date* is shown below:

```
<appConfig xmlns="http://nbcr.sdsc.edu/opal/types"
          xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <metadata appName=date>
    <usage><![CDATA[date [-u] mmddhhmm[[cc]yy]]]></usage>
    <info xsd:type="xsd:string">
        <CDATA[
             NAME
                date - print or set the system date and time
             SYNOPSIS
                date [OPTION]... [+FORMAT]
                date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]
             DESCRIPTION
                Display the current time in the given FORMAT, or set the system date.
           ]]>
    </info>
    <types> </types>
  </metadata>
  <binaryLocation>/bin/date</binaryLocation>
  <defaultArgs></defaultArgs>
  <validateArgs>false</validateArgs>
  <jobManagerFQCN>edu.sdsc.nbcr.opal.manager.ForkJobManager</jobManagerFQCN>
  <parallel>false</parallel>
</appConfig>
```

The configuration consists of:

- *appConfig*, a top level element which contains all other elements.
- *metadata*, consists of:
  - *usage*, a string specifying how the application is invoked.
  - *info*, a string specifying an array of optional application information.
  - *types*, an optional element. More information about this element can be found from the Advanced Submission Form section).
- *binaryLocation*, specifies the location of the application's binary. Note, this is a full path of an executable, and no arguments are allowed here.
- *defaultArgs*, specifies the default arguments that need to be used for every run.
- *parallel*, specifies if an application is parallel or not.
- *validateArgs* is optional. If set to true, it instructs Opal to use the optional command-line specification (within types) to validate arguments prior to execution.
- *jobManagerFQCN* is optional. It overwrites the default job manager, and can be set to the fully qualified classname of an Opal job manager. The default FQCNs for job managers provided by Opal are:
  - `edu.sdsc.nbcr.opal.manager.ForkJobManager` - for fork job
  - `edu.sdsc.nbcr.opal.manager.DRMAAJobManager` - SGE job
  - `edu.sdsc.nbcr.opal.manager.GlobusJobManager` - Globus job on a local cluster
  - `edu.sdsc.nbcr.opal.manager.RemoteGlobusJobManager` - Globus job on a remote cluster
  - `edu.sdsc.nbcr.opal.manager.CondorJobManager` - Condor job
  - `edu.sdsc.nbcr.opal.manager.CSFJobManager` - Community Scheduler Framework job

Use the *$OPAL_HOME/configs/pdb2pqr_config.xml* as a guideline to write configurations for your applications. For a more detailed application configuration, including specification of the command-line arguments, please see Sample Application Configuration File section.

2. Deploy the services inside Tomcat, by copying the *date.xml* file inside the directory specified in *$OPAL_HOME/etc/opal.properties* with the keyword *opal.deploy.path*. If you have not changed this property the default directory is *$CATALINA_HOME/deploy*.

```
cp configs/date.xml $CATALINA_HOME/deploy
```

If the service is deployed successfully, it can be accessed by the following methods:

- programmatically at URL *http://host:port/opal2/services/serviceName*
- via the Opal Dashboard at URL: *http://FQDN:port/opal2/dashboard*

3. You can undeploy your service at any time simply by deleting the file:

```
rm $CATALINA_HOME/deploy/date.xml
```

If you plan on using the service with large inputs and outputs, it is a good idea to increase the heap size being used by the JVM. This can be done by setting the *JAVA_OPTS* environment variable to *-Xmx1024m*, and restarting Tomcat. This increases the heap size to 1GB. If all went well until this step, the services are running and ready to be used. Test them by running the client, described in the following section.

The service creates a new working directory for every execution. These working directories are not deleted automatically, so they must be periodically removed. You may use the script *$OPAL_HOME/etc/cleanup.sh* as a template. You will have to modify it to point to your Tomcat installation and (optionally) modify the number of days to retain working directories.

Notes

1. http://www.oracle.com/technetwork/java/javase/downloads
2. http://ant.apache.org/
3. http://tomcat.apache.org/
4. http://www.open-mpi.org
5. http://hsqldb.org/
6. http://www.postgresql.org/
7. http://dev.mysql.com/downloads/
8. http://www-01.ibm.com/software/data/db2/
9. http://drmaa.org/
10. http://research.cs.wisc.edu/htcondor
11. http://www.oracle.com/us/products/tools/oracle-grid-engine-075549.html
12. http://www.clusterresources.com/products/torque-resource-manager.php
13. http://sourceforge.net/project/showfiles.php?group_id=211778&package_id=297015
14. http://yourhost:8080/opal2/happyaxis.jsp

Chapter 3. Command-line Client Installation

3.1. Prerequisites

The source distribution can be used to install the server and the client on Unix and Windows platforms. See Common Prerequisites for Server and Client.

3.2. Installation Instructions

1. Download the Opal2 source distribution from source forge[1].
2. Extract the downloaded tarball using the GNU tar (or similar utilities for Windows):

   ```
   tar zxvf opal-ws-2.5.tar.gz
   ```

   This should create a new directory called opal-ws-2.5/ where all the sources are expanded. Henceforth, we refer to this directory as *OPAL_HOME*.
3. A generic Java client for Opal services lets you run any application exposed as an Opal service, and retrieve its status and outputs. It is implemented by the GenericServiceClient class located in *$OPAL_HOME/src/edu/sdsc/nbcr/opal*. To compile the client, run the following command from inside the *$OPAL_HOME* directory:

   ```
   $ANT_HOME/bin/ant jar
   ```

3.3. Using the Command-line Client

- **Set your classpath** before running the client using the etc/classpath.bat, etc/classpath.csh or etc/classpath.sh script depending on your OS/shell. For example, for tcsh on Unix, set your classpath using the following command:

  ```
  source etc/classpath.csh
  ```

- **Get the client usage information** by using the following command:

  ```
  java edu.sdsc.nbcr.opal.GenericServiceClient
  ```

- **Launch a job** for the Date service mentioned in Application Deployment by using the following command:

  ```
  java edu.sdsc.nbcr.opal.GenericServiceClient \
               -l http://localhost:8080/opal2/services/DateService \
               -r launchJob \
               -a \""-v1d -v3m -v0y -v-1d -u"\"
  ```

  The command displays the resulting job id, along with the preliminary status.

  If your arguments contain characters "-", you will have to put your whole set of arguments within quotes that are escaped (\"..\"). If your arguments don't have special characters you can use regular quotes ("..").

- **Retrieve job status** by running the following command:

  ```
  java edu.sdsc.nbcr.opal.GenericServiceClient \
               -l http://localhost:8080/opal2/services/DateService \
               -r queryStatus \
  ```

```
                  -j <job_id>
```

- **Retrieve output** once the job has finished:

  ```
  java edu.sdsc.nbcr.opal.GenericServiceClient \
                -l http://localhost:8080/opal2/services/DateService \
                -r getOutputs \
                -j <job_id>
  ```

  You may need to change the above URL if you used a different port, or are running the client from another machine.

Notes

1. http://sourceforge.net/project/showfiles.php?group_id=211778&package_id=297015

Chapter 4. Opal State Database Installation

By default, Opal uses an in-memory HSQL[1] database to store its state. For production systems for a better performance you may wish to store the Opal state in a real external database. This chapter describes how to configure Opal to to store its state using PostgreSQL, MySQL or DB2 database.

4.1. Using Postgres

1. Install a PostgreSQL Postgres[2] database version 8.2.4.
2. Create a database called *opal2_db*, and a user called *opal_user* with a password. Grant all permissions on *opal2_db* to the *opal_user*. Configure the database to accept remote JDBC connections (consult the database documentation).
3. Edit the *etc/hibernate-opal.cfg.xml* configuration file and comment out the properties that are HSQL-specific as follows:

```
        <!-- Database connection settings for HSQL -->
<!--         <property name="connection.driver_class">org.hsqldb.jdbcDriver</property> -->
<!--         <property name="connection.url">jdbc:hsqldb:file:data/opaldb</property> -->
<!--         <property name="connection.username">sa</property> -->
<!--         <property name="connection.password"></property> -->
<!--         <property name="dialect">org.hibernate.dialect.HSQLDialect</property> -->
```

4. Uncomment the properties that are PostgreSQL specific as follows:

```
        <!-- Database connection settings for PostgreSQL -->
        <property name="connection.driver_class">org.postgresql.Driver</property>
        <property name="connection.url">jdbc:postgresql://localhost/opal2_db</property>
        <property name="connection.username">opal_user</property>
        <property name="connection.password">opal_passwd</property>
        <property name="dialect">org.hibernate.dialect.PostgreSQLDialect</property>
```

   Note that you can change the property *connection.url* to point to a database that is running on remote host as long as the remote host can accept JDBC connection from the host running the Opal services. In this case change *localhost* to the remote host FQDN.

5. Reinstall Opal by running the following command:

```
ant install
```

6. Restart Tomcat for the changes to take effect.

4.2. Using MySQL

1. Install a MySQL[3] database (tested with version 5.1.61).

2. Create a database called *opal2_db*, and a user called *opal_user* with a password. Grant all permissions on *opal2_db* to the *opal_user*. Configure the database to accept remote JDBC connections (consult the database documentation).
3. Edit the *etc/hibernate-opal.cfg.xml* configuration file and comment out the properties that are HSQL-specific as follows:

```
        <!-- Database connection settings for HSQL -->
<!--         <property name="connection.driver_class">org.hsqldb.jdbcDriver</property> -->
<!--         <property name="connection.url">jdbc:hsqldb:file:data/opaldb</property> -->
<!--         <property name="connection.username">sa</property> -->
<!--         <property name="connection.password"></property> -->
<!--         <property name="dialect">org.hibernate.dialect.HSQLDialect</property> -->
```

4. Uncomment the properties that are MySQL specific as follows:

```
        <!-- Database connection settings for MySQL -->
        <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="connection.url">
              jdbc:mysql://localhost/opal2_db?autoReconnect=true</property>
        <property name="connection.username">opal_user</property>
        <property name="connection.password">opal_passwd</property>
        <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
```

   Note that you can change the property *connection.url* to point to a database that is running on a remote host as long as the remote host can accept JDBC connection from the host running the Opal services. In this case change *localhost* to the remote host FQDN.

5. Reinstall Opal by running the following command:

```
ant install
```

6. Restart Tomcat for the changes to take effect.

4.3. Using DB2

1. Install a DB2 database (tested with version 8.2).
2. Create a database called *opal2_db*, and a user called *opal_user* with a password. Grant all permissions on *opal2_db* to the *opal_user*. Configure the database to accept remote JDBC connections (consult the database documentation).
3. Edit the *etc/hibernate-opal.cfg.xml* configuration file and comment out the properties that are HSQL-specific as follows:

```
        <!-- Database connection settings for HSQL -->
<!--         <property name="connection.driver_class">org.hsqldb.jdbcDriver</property> -->
<!--         <property name="connection.url">jdbc:hsqldb:file:data/opaldb</property> -->
<!--         <property name="connection.username">sa</property> -->
<!--         <property name="connection.password"></property> -->
<!--         <property name="dialect">org.hibernate.dialect.HSQLDialect</property> -->
```

4. Uncomment the properties that are DB2 specific as follows:

```
        <!-- Database connection settings for DB2 -->
        <property name="connection.driver_class">com.ibm.db2.jcc.DB2Driver</property>
        <property name="connection.url">jdbc:db2://localhost:60000/opaldb</property>
        <property name="connection.username">opal_user</property>
        <property name="connection.password">opal_passwd</property>
        <property name="dialect">org.hibernate.dialect.DB2Dialect</property>
```

Note that you can change the property *connection.url* to point to a database that is running on a remote host as long as the remote host can accept JDBC connection from the host running the Opal services. In this case change *localhost* to the remote host FQDN.

5. Reinstall Opal by running the following command:

```
ant install
```

6. Restart Tomcat for the changes to take effect.

Notes

1. http://hsqldb.org/
2. http://www.postgresql.org/
3. http://dev.mysql.com/downloads/

Chapter 5. Scheduler Support Using Job Managers

In this chapter, we describe how to configure Opal to submit jobs to a batch job scheduler using the provided job managers. We also give a brief introduction how to write your own job manager.

Prerequisite: a scheduler such as Condor, PBS or SGE installed on your compute cluster. Please consult the appropriate documentation for the installation.

Access to schedulers is provided in a number of ways - in particular, via the DRMAA API, and via the various scheduler plugins. Starting with release 2.1, we support Condor natively without using DRMAA or Globus. In release 2.3, we introduced direct support for TORQUE/PBS.

5.1. Using DRMAA

We have only tested job submissions for SGE 6.x.

1. Ensure that your scheduler supports job submission via the DRMAA API, consult your scheduelr documentation. Ensure that *libdrmaa.so* is in your library path (set the LD_LIBRARY_PATH environment variable).
2. Set the following properties inside the opal.properties file:
   - Set job manager to SGE

     ```
     opal.jobmanager=edu.sdsc.nbcr.opal.manager.DRMAAJobManager
     ```

   - Optional: set the name of the parallel environment (PE) to be used for parallel jobs. PE must be defined in your scheduelr queue. You can ignore drmaa.pe if you do not plan on supporting parallel jobs.

     ```
     drmaa.pe=mpich
     ```

   - Optional: set an SGE queue for job submission.

     ```
     drmaa.queue=all.q
     ```

   Note that the `drmaa.pe` and the `drmaa.queue` can be overridden on a per-application basis within the application configuration file, for example:

   ```
   <appConfig xmlns="http://nbcr.sdsc.edu/opal/types"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema">
     <metadata appName=...
         ...
     </metadata>
     <drmaaQueue>si</drmaaQueue>
     <drmaaPE>orte</drmaaPE>
     <parallel>true</parallel>
   </appConfig>
   ```

3. Reinstall Opal by running the following command:

   ```
   ant install
   ```

4. Restart Tomcat for the changes to take effect.

5. Notes:
   - When using DRMAA job manager we do not stage input and output files from the Opal server to the submit host. The machines should use a shared file system for the job manager to work correctly.
   - DRMAA support will not work correctly if you have another version of the drmaa.jar inside Tomcat's *common/lib* or *server/lib* directories, since these vers ions will be loaded first by Tomcat's class loader.

5.2. Using Condor

1. Ensure that you have a working Condor pool by following the Condor manual[1]. We have tested with version 7.6.2, but we expect that it will work with most recent Condor versions. In particular, verify that you can submit and monitor jobs from the command-line with commands such as *condor_submit*, *condor_status*, etc., and that these commands are in the Opal user's PATH.
2. Set the following properties inside the opal.properties file:
   - Set job manager to Condor

     ```
     opal.jobmanager=edu.sdsc.nbcr.opal.manager.CondorJobManager
     ```
   - Set the script used by Condor to submit parallel jobs. You can ignore `mpi.script` if you do not plan to support parallel jobs.

     ```
     mpi.script=/opt/condor/etc/examples/mp1script
     ```
   - To enable server-specific condor expressions put them in a file. Use a regular condor submit file syntax (key = value).

     ```
     condor.expr.file=/opt/opal/etc/condor.expr
     ```
3. Reinstall Opal by running the following command:

   ```
   ant install
   ```
4. Restart Tomcat for the changes to take effect.
5. Notes:
   - We have not tested advanced Condor features such as flocking and Condor-G.
   - Job directory does not have to be on NFS-mounted filesystem.

5.3. Using TORQUE/PBS

1. Ensure that you have a working TORQUE/PBS installation, by following the instructions on the TORQUE website[2]. In particular, verify that you can submit and monitor jobs from the command-line with commands such

as *qsub* and *qstat*. Note that we have only tested with TORQUE v.2.3.0 and PBS v.PBSPro_11.3.0.121723.

2. Set the following properties inside the opal.properties file:
   - Set job manager to PBS.

     ```
     opal.jobmanager=edu.sdsc.nbcr.opal.manager.PBSJobManager
     ```
   - To enable server-specific PBS expressions put them in a file (use regular PBS submit syntax in a file).

     ```
     pbs.expr.file=/opt/opal/etc/pbs.expr
     ```
   - Limit jobName length:

     ```
     pbs.name.limit=15
     ```

     Opal creates long jobs names (longer than 15 characters). As a result, need to set default job name limit per your server configuration. Job submit files will have *#PBS -N jobName* line and the jobName will have a specified by limit length.

3. Reinstall Opal by running the following command:

   ```
   ant install
   ```
4. Make sure that all the PBS binaries such as *qsub* and *qstat* are in your PATH.
5. Restart Tomcat for the changes to take effect.

5.4. Using Meta Service

The Opal meta service plugin can be used to send jobs to remote Opal services. You can define several remote hosts for a particular web service, and the web service will run on a random host chosen from this set of remote hosts. In a future Opal release, we may enhance the meta scheduling algorithm.

The *<metaServiceConfig>* tag in the application configuration file is used to define the location of the meta service configuration file for the particular service. For example,

```
<metaServiceConfig>/home/opaluser/meta/pdb2pqr.txt</metaServiceConfig>
```

The meta service configuration file contains one or more lines, where each line lists a remote Opal service URL followed by the number of processors on that remote Opal service (separated by a space). Please use 1 as the number of processors for serial services. The sample file below defines the remote hosts for the Pdb2pqr meta service.

```
http://kryptonite.nbcr.net/opal2/services/pdb2pqr_1.7 1
http://ws.nbcr.net/opal2/services/Pdb2pqrOpalService 1
```

5.5. Writing Your Own Job Manager

If the job managers provided by the Opal toolkit are not sufficient for your purposes, you can write your own job manager to submit jobs to your favorite scheduler.

To write your own Opal job manager, you must implement the *edu.sdsc.nbcr.opal.manager.OpalJobManager* interface. One job manager is instantiated per job instance - i.e. for every run of an application, a new job manager is created. The OpalJobManager interface has 6 methods that must be implemented:

1. **initialize**: initialize the job manager by setting a list of properties (key/value pairs), application configuration, and an optional handle. All job manager specific properties should be put inside *$OPAL_HOME/etc/opal.properties*. Opal will parse all the properties from opal.properties and make them available to the job manager.

```
    /**
     * @param props the properties file containing the value to configure this plugin
     * @param config the opal configuration for this application
     * @param handle manager specific handle to bind to, if this is a resumption.
     *               NULL,if this manager is being initialized for the first time.
     *
     * @throws JobManagerException if there is an error during initialization
     */
    public void initialize(Properties props,
      AppConfigType config,
      String handle)
 throws JobManagerException;
```

2. **destroyJobManager**: cleanup when the job manager is destroyed - all resources held should be freed here.

```
    /**
     * @throws JobManagerException if there is an error during destruction
     */
    public void destroyJobManager()
 throws JobManagerException;
```

3. **launchJob**: launch a job with given arguments. The input files are already staged in by the service implementation, and the job manager implementation can assume that they are already in the right location.

```
    /**
     * Launch a job with the given arguments.
     *
     * @param argList a string containing the command line used to launch the application
     * @param numproc the number of processors requested. Null, if it is a serial job
     * @param workingDir String representing the working dir of this job on the local system
     *
     * @return a plugin specific job handle to be persisted by the service implementation
     * @throws JobManagerException if there is an error during job launch
     */
    public String launchJob(String argList,
       Integer numproc,
       String workingDir)
 throws JobManagerException;
```

4. **waitForActivation**: block until the job has begun execution. Opal uses this information to collect job statistics.

```
    /**
     * @return status for this job after blocking
     * @throws JobManagerException if there is an error while waiting for the job to be ACTIVE
     */
    public StatusOutputType waitForActivation()
 throws JobManagerException;
```

5. **waitForCompletion**: block until the application has finished execution.

```
    /**
     * @return final job status
     * @throws JobManagerException if there is an error while waiting for the job to finish
     */
    public StatusOutputType waitForCompletion()
 throws JobManagerException;
```

6. **destroyJob**: destroy an executing job.

```
    /**
     * @return final job status
     * @throws JobManagerException if there is an error during job destruction
     */
    public StatusOutputType destroyJob()
 throws JobManagerException;
```

More information about the Java classes can be obtained from the Javadocs, which you can generate from $OPAL_HOME as follows:

```
ant api-docs
```

This will generate documentation of the Opal API, which will be available in HTML from at *$OPAL_HOME/docs/api/index.html*.

We recommend that you implement your particular job manager inside the directory *$OPAL_HOME/src/edu/sdsc/nbcr/opal/manager*. There you will find other job managers that are available by default - you might want to look at the *ForkJobManager* as an example. You can compile your job manager with the rest of Opal, and install it within Tomcat by running the following commands:

```
ant compile
ant install
```

Notes

1. http://research.cs.wisc.edu/htcondor/manual/v7.6.2/
2. http://www.clusterresources.com/products/torque-resource-manager.php

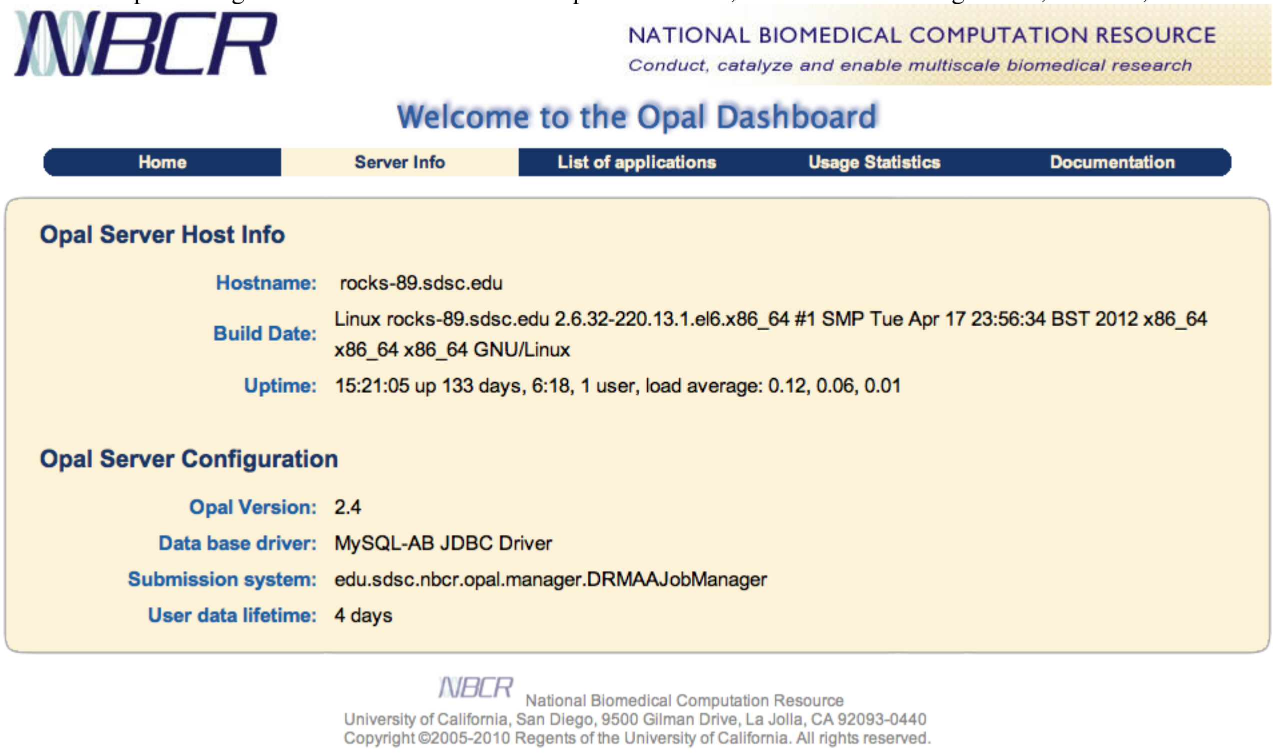
Chapter 6. Opal Dashboard

6.1. Overview

The Opal Dashboard is a Web-based graphical user interface that allows a user to monitor an Opal service, to view statistics regarding its usage and to execute deployed applications from within a standard Web browser. It provides for dynamic creation of customized applications forms for job submission. The dashboard is automatically deployed along with Opal, and it can be accessed at http://servername:8080/opal2/dashboard[1].

The Opal dashboard has the following tabs:

- **Home**: provides general information about Opal, and an overview of the rest of the functionality of the Dashboard.
- **Server Info**: provides general information about the Opal installation, such as server configuration, versions, etc.


NBCR
NATIONAL BIOMEDICAL COMPUTATION RESOURCE
Conduct, catalyze and enable multiscale biomedical research
Welcome to the Opal Dashboard
Home
Server Info
List of applications
Usage Statistics
Documentation

Opal Server Host Info

Hostname:	rocks-89.sdsc.edu
Build Date:	Linux rocks-89.sdsc.edu 2.6.32-220.13.1.el6.x86_64 #1 SMP Tue Apr 17 23:56:34 BST 2012 x86_64 x86_64 x86_64 GNU/Linux
Uptime:	15:21:05 up 133 days, 6:18, 1 user, load average: 0.12, 0.06, 0.01

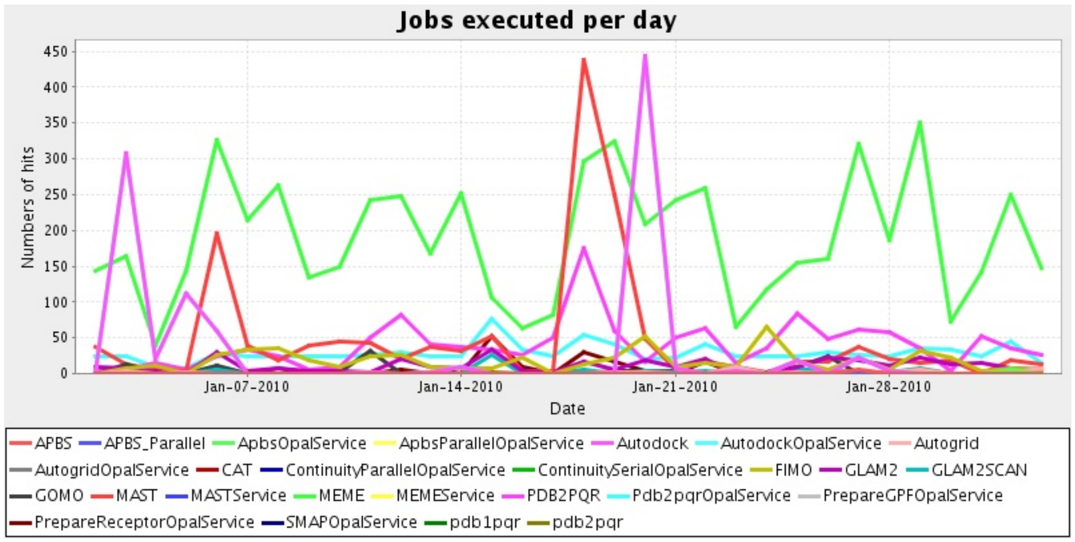
Opal Server Configuration

Opal Version:	2.4
Data base driver:	MySQL-AB JDBC Driver
Submission system:	edu.sdsc.nbcr.opal.manager.DRMAAJobManager
User data lifetime:	4 days

NBCR National Biomedical Computation Resource
University of California, San Diego, 9500 Gilman Drive, La Jolla, CA 92093-0440
Copyright ©2005-2010 Regents of the University of California. All rights reserved.

- **Usage Statistics**: provides charts with usage statistics, errors and average execution times. The charts are created dynamically based on user choice of timeframe and a list of services. If HSQL database is used, then all services that have ever been deployed will be displayed. For other databases the charts will include only services that are currently active (services that have been undeployed will not be displayed).

  The following screenshot displays the number of jobs executed per day over a specified timeframe.



- **List of Applications**: provides a list of all installed applications and an atom feed for the list of applications. Users can search for applications based on specified metadata, and click on the applications to access submission forms.
- **Documentation**: provides links to Opal documentation, websites, tutorials, publications, etc.

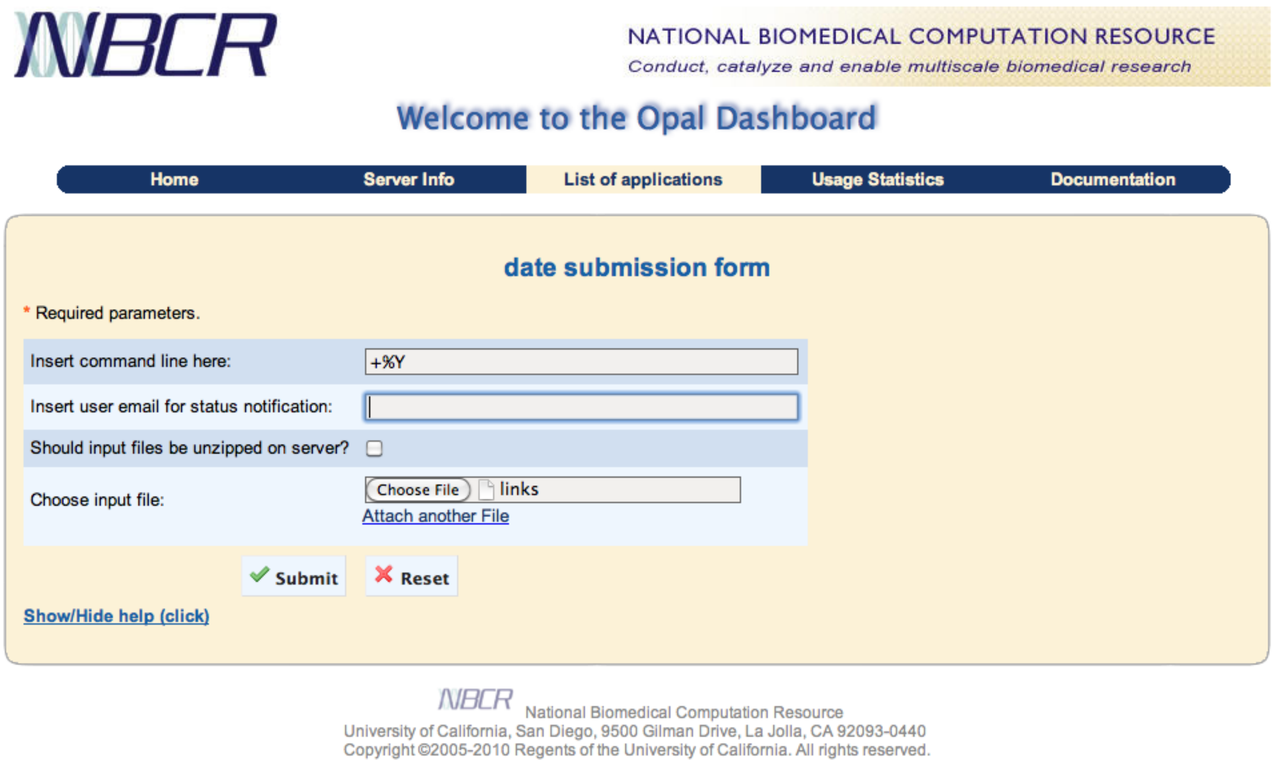
The list of installed applications is generated dynamically by fetching real time data from the Apache Axis engine. When the user clicks on one of the available services, the Opal dashboard displays a submisison form. The submission form type depends on the configuration of the specific application:

- Simple Submission Form
- Advanced Submission Form

6.2. Simple Submission Form

When an application has been deployed using a basic application configuration, such as the one used by the **/bin/date** example shown in Application Deployment, the Opal Dashboard will display a simple form with an input text field and an input file box. In this case, the application configuration file does not present any metadata describing the format of the application input arguments.

The user should type the desired command-line arguments into the input text box and, optionally, upload an input file. When a user specifies one file to upload, a form presents an *Attach another File* link which enables a button to upload more files. The user can indicate via a check button if the files should be uncompressed on the server.



* Required parameters.

Insert command line here:	+%Y
Insert user email for status notification:	
Should input files be unzipped on server?	☐
Choose input file:	links

National Biomedical Computation Resource
University of California, San Diego, 9500 Gilman Drive, La Jolla, CA 92093-0440
Copyright ©2005-2010 Regents of the University of California. All rights reserved.

6.3. Advanced Submission Form

One can add an optional element *types* in the application configuration file described at Application Deployment. This element contains a description of the command line input arguments accepted by the application. The Opal dashboard will render an advanced submission form with the customized inputs arguments.

6.3.1. The <types> section

The <types> section of the application configuration file can be composed of four optional elements. Each element, when present, has one or more nested elements.

1. **<flags>**: optional, contain a set of <flag> elements. The flags are not ordered.

   - **<flag>**: is used to define an input argument (or a flag) for the application. These arguments are passed using one or more characters prefixed with one or more dashes. Usually they are used to activate or deactivate some application functionality, for example *--verbose*. The following elements can be nested inside <flag>:

     - **<id>**: required element, contains an internal unique identifier which represents this flag.
     - **<tag>**: required element, contains a string that is used as a tag for this argument.
     - **<default>**: optional element, specifies if this flag should be automatically selected or not. Can be only *true* or *false*.

- **<textDesc>**: optional element, contain an explanation of the flag's functionality.

```
<types>
      <!-- list of flags -->
      <flags>
          <flag>
              <id>verbose</id>
              <tag>--verbose</tag>
              <textDesc>Provide verbose output</textDesc>
              <default>true</default>
          </flag>

          ...
      </flags>
```

2. **<taggedParams>**: optional, contains a set of <parameter> elements. They can appear in any order on the command line.

   - **<param>**: these elements are usually formed by a prefix (tag), a separator, and an input value. The 'separator' is a character used to separate the tag from the input value. For example, for *-input=<FileName1>* the separator is '=', for *-input <FileName2>* the separator is ' '. The following elements can be nested inside <param>:

     - **<id>**: required, contains an internal unique identifier which represents this parameter
     - **<tag>**: required, contains the string that is used as a tag for this argument. NOTE: do not use this element if a parent element <param> is inside <untaggedParams> element.
     - **<default>**: optional element, contains the default value for a given parameter. When present, the default value is copied into the corresponding input box on the application form.
     - **<paramType>**: required, specifies the type for this parameter. Its values can be: INT, BOOL, FLOAT, STRING, FILE and URL.
     - **<ioType>**: required to be present when the paramType tag is FILE; specifies if the file is an input or an output. Possible values are: INPUT, OUTPUT and INOUT
     - **<required>**: optional, specifies if a parameter is mandatory. When not present, the default is false. Can contain values *true* or *false*.
     - **<value>**: optional, can be repeated multiple times. If present it means that this parameter can assume only the values enumerated in these tags.
     - **<textDesc>**: optional, contains an explanation of the argument functionality.

```
<!-- list of tagged parameters -->
<taggedParams>
    <separator>=</separator>
    <param>
        <id>forcefield</id>
        <tag>--ff</tag>
        <paramType>STRING</paramType>
        <required>true</required>
        <value>AMBER</value>
        <value>CHARMM</value>
        <value>PARSE</value>
```

```
                    <value>TYL06</value>
                    <default>AMBER</default>
                    <textDesc>The forcefield to use --
                      currently AMBER, CHARMM, PARSE, and TYL06 are supported.
                    </textDesc>
                </param>
                      ...
            </taggedParams>
```

3. **<untaggedParams>**: optional, contains a set of <parameter> elements. Each <parameter> element defines an input argument when its order on the command line is relevant. Hence, the oder of specified parameters in the application configuration file must follow the order of the arguments expected by the application. Each <parameter> element has the same set of nested elements as the <parameter> for <taggedParams> element, accept <tag>, see NOTE above.

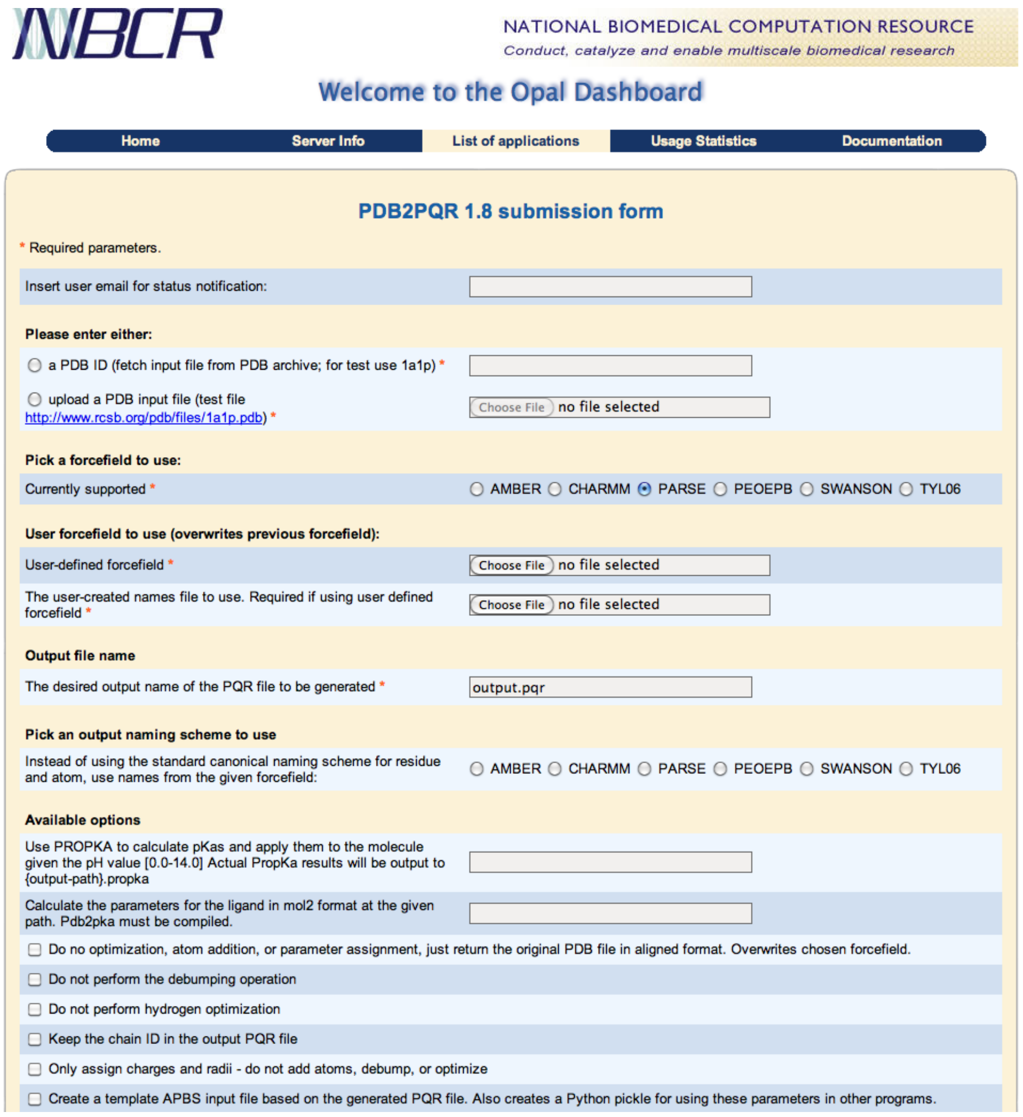
```
<!-- list of untagged parameters, in order -->
<untaggedParams>
    <param>
        <id>inFile</id>
        <paramType>FILE</paramType>
        <ioType>INPUT</ioType>
        <!--  <required>true</required>   -->
        <textDesc>The PDB input file.</textDesc>
    </param>
          ...
</untaggedParams>
```

4. **<groups>**: optional, contains a set of <group> elements.

- **<group>**: optional, defines groups of several parameters that have a similar functionality. Each defined group will be displayed together on the submission form. It contains the following nested elements:
  - **<name>**: required, contains the internal name for this group
  - **<elements>**: required, contains the list of parameters and flags that are part of this group
  - **<required>**: optional. Can be only *true* or *false*. If this element is true, it means that all the elements within this group are mandatory. When absent, the default is false.
  - **<exclusive>**: optional, specifies that parameters in this group are mutually exclusive. Default is false. Can be only *true* or *false*.
  - **<textDesc>**: olptional, contains a description that will be rendered on the submission form as a header for this group of parameters.

```
<groups>
  <group>
    <name>inputParam</name>
    <elements>inFile inId</elements>
     <required>true</required>
     <exclusive>true</exclusive>
     <textDesc>Input file to be used (choose one of the two options)</textDesc>
  </group>
         ...
</groups>
```

For the Opal application configuration described in Sample Application Configuration File, the following advanced submission form is generated:



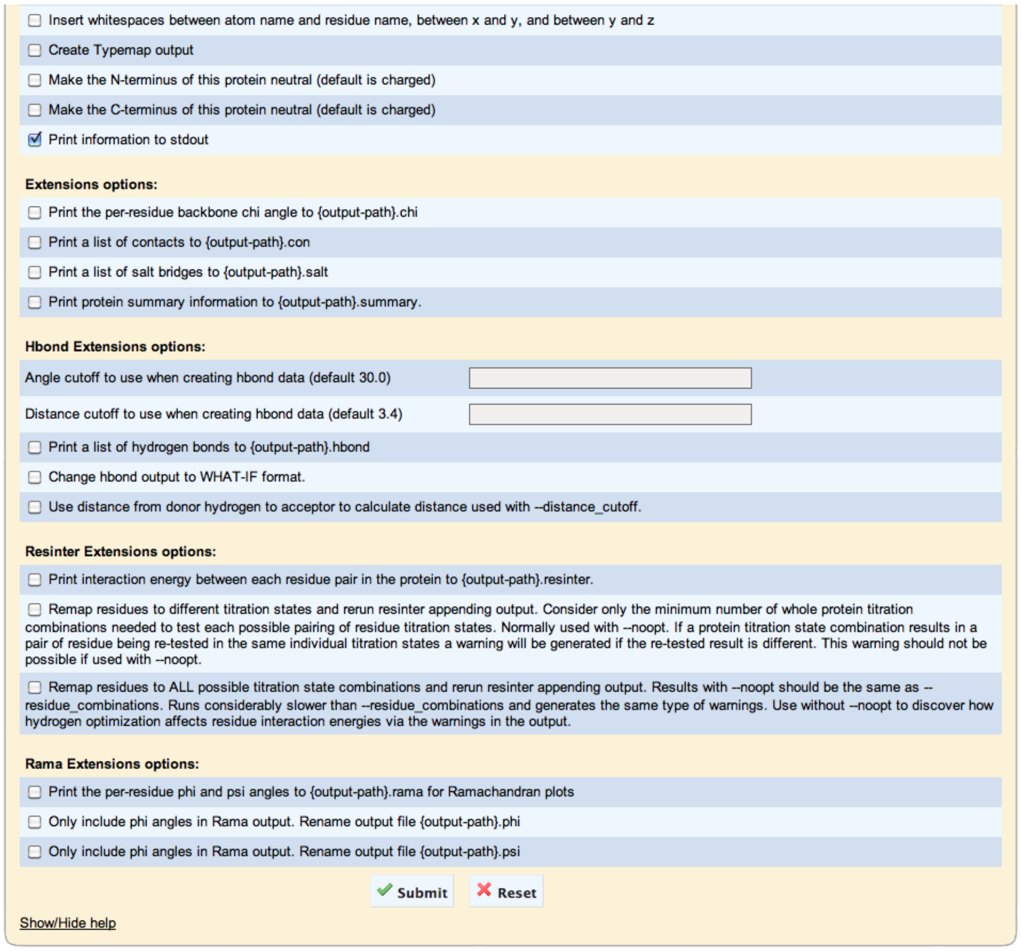
NBCR
NATIONAL BIOMEDICAL COMPUTATION RESOURCE
Conduct, catalyze and enable multiscale biomedical research
Welcome to the Opal Dashboard
Home
Server Info
List of applications
Usage Statistics
Documentation

PDB2PQR 1.8 submission form

* Required parameters.

Insert user email for status notification:	
Please enter either:	
○ a PDB ID (fetch input file from PDB archive; for test use 1a1p) *	
○ upload a PDB input file (test file http://www.rcsb.org/pdb/files/1a1p.pdb) *	Choose File no file selected
Pick a forcefield to use:	
Currently supported *	AMBER CHARMM [✓ PARSE] PEOEPB SWANSON TYL06
User forcefield to use (overwrites previous forcefield):	
User-defined forcefield *	Choose File no file selected
The user-created names file to use. Required if using user defined forcefield *	Choose File no file selected
Output file name	
The desired output name of the PQR file to be generated *	output.pqr
Pick an output naming scheme to use	
Instead of using the standard canonical naming scheme for residue and atom, use names from the given forcefield:	AMBER CHARMM PARSE PEOEPB SWANSON TYL06
Available options	
Use PROPKA to calculate pKas and apply them to the molecule given the pH value [0.0-14.0] Actual PropKa results will be output to {output-path}.propka	
Calculate the parameters for the ligand in mol2 format at the given path. Pdb2pka must be compiled.	

- ☐ Do no optimization, atom addition, or parameter assignment, just return the original PDB file in aligned format. Overwrites chosen forcefield.
- ☐ Do not perform the debumping operation
- ☐ Do not perform hydrogen optimization
- ☐ Keep the chain ID in the output PQR file
- ☐ Only assign charges and radii - do not add atoms, debump, or optimize
- ☐ Create a template APBS input file based on the generated PQR file. Also creates a Python pickle for using these parameters in other programs.



- [ ] Insert whitespaces between atom name and residue name, between x and y, and between y and z
- [ ] Create Typemap output
- [ ] Make the N-terminus of this protein neutral (default is charged)
- [ ] Make the C-terminus of this protein neutral (default is charged)
- [✓] Print information to stdout

Extensions options:

- [ ] Print the per-residue backbone chi angle to {output-path}.chi
- [ ] Print a list of contacts to {output-path}.con
- [ ] Print a list of salt bridges to {output-path}.salt
- [ ] Print protein summary information to {output-path}.summary.

Hbond Extensions options:

Angle cutoff to use when creating hbond data (default 30.0)	
Distance cutoff to use when creating hbond data (default 3.4)	

- [ ] Print a list of hydrogen bonds to {output-path}.hbond
- [ ] Change hbond output to WHAT-IF format.
- [ ] Use distance from donor hydrogen to acceptor to calculate distance used with --distance_cutoff.

Resinter Extensions options:

- [ ] Print interaction energy between each residue pair in the protein to {output-path}.resinter.
- [ ] Remap residues to different titration states and rerun resinter appending output. Consider only the minimum number of whole protein titration combinations needed to test each possible pairing of residue titration states. Normally used with --noopt. If a protein titration state combination results in a pair of residue being re-tested in the same individual titration states a warning will be generated if the re-tested result is different. This warning should not be possible if used with --noopt.
- [ ] Remap residues to ALL possible titration state combinations and rerun resinter appending output. Results with --noopt should be the same as --residue_combinations. Runs considerably slower than --residue_combinations and generates the same type of warnings. Use without --noopt to discover how hydrogen optimization affects residue interaction energies via the warnings in the output.

Rama Extensions options:

- [ ] Print the per-residue phi and psi angles to {output-path}.rama for Ramachandran plots
- [ ] Only include phi angles in Rama output. Rename output file {output-path}.phi
- [ ] Only include phi angles in Rama output. Rename output file {output-path}.psi

Notes

1. http://localhost:8080/opal2/dashboard

Chapter 7. Feedback and Acknowledgements

Feedback

- To report bugs and feature requests, please use Sourceforge Opal Toolkit Tracker[1]
- Email opaltoolkit-users @ lists.sourceforge.net

Acknowledgements

The organizations involvement in the development of Opal:

- National Biomedical Computation Resource [2]
- San Diego Supercomputer Center [3]
- California Institute for Telecommunications and Information Technology (Calit2) [4]

Opal development has been supported by the following funding agencies:

- National Institutes of Health[5]
- National Science Foundation[6]
- Gordon and Betty Moore Foundation[7]

Notes

1. http://sourceforge.net/tracker/?group_id=211778
2. http://www.nbcr.net
3. http://www.sdsc.edu
4. http://www.calit2.net/
5. http://www.nih.gov/
6. http://www.nsf.gov/
7. http://www.moore.org/

Chapter 8. Appendix

8.1. Configuring up GSI-based Security

This section explains how to configure Opal to use GSI to autheticate clients.

1. Create a globus-based PEM server certificate and unencrypted private key for the tomcat server (consult the documentation of your CA software).
2. Make sure that Opal has been installed successfully. If not, do so by running the following command:

   ```
   ant install
   ```

3. To enable GSI HTTPS in Tomcat there are two different procedures depending on your version of Tomcat. If you are using 5.0.X, you can start from the sample etc/server.xml provided (works for version 5.0.30). The following snippets are responsible for enabling https:

   ```
   <Service name="Catalina">
    ...
    <Connector className="org.globus.tomcat.coyote.net.HTTPSConnector"
               port="8443" maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
        enableLookups="false" disableUploadTimeout="true"
        acceptCount="100" clientAuth="true"
        debug="3" scheme="https"
               autoFlush="true" encryption="true"
               cert="/path/to/certificate/file"
               key="/path/to/private/key/file"
               cacertdir="/path/to/ca/certificates/directory" />
    ...
    <Engine name="Catalina" defaultHost="localhost" debug="0">
      ....
      <Valve className="org.globus.tomcat.coyote.valves.HTTPSValve"/>
      ....
    </Engine>
   </Service>
   ```

   Make sure that the cert and key points correctly to the server certificate and key generated in Step 1, and that the cacertdir points to your list of trusted CAs.

   If you are using Tomcat 5.5.X you should modify the server.xml in the following way:

   ```
   <Service name="Catalina">
    ...

     <Connector
         className="org.globus.tomcat.coyote.net.HTTPSConnector"
         port="8443" maxThreads="150"
         minSpareThreads="25" maxSpareThreads="75"
         autoFlush="true" disableUploadTimeout="true"
         scheme="https" enableLookups="true"
         acceptCount="10" debug="0"
         protocolHandlerClassName="org.apache.coyote.http11.Http11Protocol"
   ```

```
        socketFactory="org.globus.tomcat.catalina.net.BaseHTTPSServerSocketFactory"
        proxy="/path/to/proxy/file" cert="/path/to/certificate/file"
        key="/path/to/private/key/file"
        cacertdir="/path/to/ca/certificates/directory"
        encryption="true"/>

    ...
    <Engine name="Catalina" defaultHost="localhost" debug="0">
     ....
     <Valve className="org.globus.tomcat.coyote.valves.HTTPSValve55"/>
     ....
    </Engine>
```

The parameters proxy, cert, key and cacertdir should point to your local files. Furthermore, if you are using a proxy, do not use the cert/key combination - in other words, they are mutually exclusive. The encryption attribute is also optional (defaults to true if not set).

4. To enable grid-map authorization of clients, add the following XML fragment inside the `<requestFlow/>` element of the `<globalConfiguration/>` in $CATALINA_HOME/webapps/opal2/WEB-INF/server-config.wsdd.

   ```
   <handler type="java:edu.sdsc.nbcr.common.GridMapAuthHandler">
    <parameter name="gridmap" value="/path/to/grid-mapfile"/>
   </handler>
   ```

   Make sure that the value points to a valid grid-map file. To authorize a client to use the service, add an entry into the grid-map file with a mapping between the client's DN and a local user. Since all jobs are being launched as the app_user, map all client DN's to the generic app_user, e.g the following is an entry in a grid-map file:

   ```
   "/C=US/O=nbcr/OU=sdsc/CN=app_user" app_user
   ```

   Instead, if you would like to authorize based on a list of acceptable CAs, then you must enable the ca-map authorization of clients. To do so, add the following XML fragment inside the `<requestFlow/>` element of the `<globalConfiguration/>` in $CATALINA_HOME/webapps/opal/WEB-INF/server-config.wsdd.

   ```
   <handler type="java:edu.sdsc.nbcr.common.CAAuthHandler">
    <parameter name="ca-map" value="/path/to/ca-mapfile"/>
   </handler>
   ```

   Make sure that the value points to a valid ca-map file. To authorize a client to use the service, add an entry into the ca-map file with the DN for the client's CA, e.g. the following is an entry in a ca-map file:

   ```
   "C=US,O=nbcr,OU=sdsc,CN=Certificate Manager" NBCR
   ```

5. Restart the Tomcat server for the configurations to take effect.

6. Create a globus-based PEM certificate for the client, and create a limited-lifetime proxy by performing a "grid-proxy-init". Before invoking the client, make sure that the X509_USER_PROXY system property is set correctly to the location of the generated proxy. You may launch a job using GSI HTTPS as follows:

   ```
     java -DX509_USER_PROXY=$X509_USER_PROXY edu.sdsc.nbcr.opal.GenericServiceClient \
                 -l https://localhost:8443/opal2/services/Pdb2pqrServicePort \
                 -r launchJob \
                 -a "--ff=amber sample.pdb output.pqr" \
                 -f samples/sample.pdb
   ```

You may need to ensure that both the client and the server trust each others' CA's (by adding entries into the .globus/certificates and /etc/grid-security/certificates directories respectively, if need be). The GenericServiceClient class shows how the user credentials can be set programmatically inside a client stub in order to enable GSI HTTPS.

7. Note that the Opal dashboard will not function out of the box when GSI-based mutual authentication is being used. This is because the Opal server will reject clients that are not authenticated. You will need to import your client certificate or proxy into your Web browser to be able to authenticate to the Opal server. The procedure for this varies from one browser to another. Please follow the documentation for your own specific Web browsers.

8.2. Automatic WSDL Generation

Starting with version 2.2, Opal optionally supports automatic WSDL generation based on the `<types>` element in the application metadata. This code has been contributed to the Opal SVN by Anthony Bretaudeau from IRISA[1]. If automatic WSDL generation is enabled, Opal services can be accesed using their regular non-typed standard WSDL or the newly autogenerated strongly-typed WSDL. This will enable all legacy clients to continue to use the old-style WSDL, while clients that prefer a stronger typed WSDL can use the autogenerated version.

Similarly to the automatic interface generation using the `<types>` element in the application metadata, an updated WSDL is generated using a simple rule-based translation during service deployment time using XSLT. For instance, an argument specified as a `flag` below:

```
<flag>
    <id>verbose</id>
    <tag>--verbose</tag>
    <textDesc>Provide verbose output</textDesc>
    <default>true</default>
</flag>
```

is translated into a boolean XML element as follows:

```
<xsd:element maxOccurs="1" minOccurs="0" name="verbose" type="xsd:boolean" default="true">
           <xsd:annotation>
               <xsd:documentation>Provide verbose output</xsd:documentation>
           </xsd:annotation>
</xsd:element>
```

Untagged parameters are translated into an `xsd:sequence` of appropriately typed elements. For example, an untagged parameter element `<paramType>`STRING`</paramType>` will be converted into an XML element of type `xsd:string`.

To enable automatic interface generation, do the following:

1. Uncomment the *typedservices.dir* property within the build.properties as follows:

```
# location of the typed services directory relative to $CATALINA_HOME/webapps.
# uncomment if you want to generate typed WSDL for services
typedservices.dir = opal2/typedservices
```

2. Enable the SOAP filter that performs the conversion from the regular untyped Opal WSDL to the new WSDL. Uncomment the following lines in the *webapps/opal2/WEB-INF/web.xml*

```
<filter>
  <filter-name>OpalSOAPRequestFilter</filter-name>
  <filter-class>org.inria.genouest.opal.tools.soaprequest.filter.OpalSOAPRequestFilter</filte
</filter>
<filter-mapping>
  <filter-name>OpalSOAPRequestFilter</filter-name>
  <servlet-name>AxisServlet</servlet-name>
</filter-mapping>
```

3. Re-deploy Opal by running:

   ```
   ant install
   ```

4. Restart Tomcat.

Now when you deploy Opal services using the **ant deploy** command, it will print the URL of the autogenerated WSDL that you can share with clients. Note that it makes sense to autogenerate WSDL's only if the application configuration has the correct types specified. If the types have not been specified, continue to use the regular legacy-style non-typed WSDLs.

8.3. Configuring job limit for Fork Job Manager

Beginning with version 2.3, Opal enables jobs launching using the Fork Job Manager. This feature is particularly useful if the jobs are allowed to use a finite amount of memory or processors, thus preventing a large number of jobs running at the same time. Note that this is not a replacement for a traditional batch scheduler - it is simply a way to limit the total number of jobs that are concurrently being executed. We have not done any scalability testing of this feature because the Fork Job Manager is not supposed to be used for large number of concurrent jobs.

By default, the job limits are turned off. To turn it on, please set the **fork.jobs.limit** to the number of jobs that can execute concurrently. For instance, the specific lines for configuring Fork Job Manager are as follows:

```
# full qualified class name (FQCN) of the job manager being used
opal.jobmanager=edu.sdsc.nbcr.opal.manager.ForkJobManager
...
## BEGIN: information for Fork job manager
## ---------------------------------------------------------
# number of jobs that can be in execution simultaneously
fork.jobs.limit=1
## ---------------------------------------------------------
## END: information for For job manager
```

For the changes to take effect, reinstall Opal using the **ant install** command, and restart Tomcat.

8.4. Configuring IP-based Restrictions for Job Submission

Beginning with version 2.2, Opal provides basic support for restricting number of job submissions per IP. Appropriate properties for configuring this functionality are set in the **etc/opal.properties** file:

- Set **opal.ip.processing** to **true**. By default, the IP-based restrictions are turned off.
- Use **opal.ip.limit** to configure the number of jobs that are allowed per IP address per hour. If the Opal server receives more than these many jobs from any IP address per hour, it will reject job submissions by throwning an appropriate exception.
- Use **opal.ip.blacklist** to block all jobs from a particular IP address. Note that the blacklist takes precedence over the whitelist - i.e. if an IP address is present on the blacklist, all jobs from this address will be blocked even if it is present in the whitelist.
- Use **opal.ip.whitelist** to allow all jobs from a particular IP address. Note that you should add localhost (127.0.0.1) to the whitelist, if you want the Opal Dashboard to be able to submit jobs always.

An example configuration for IP-based restrictions feature:

```
## BEGIN: information for the per IP limits on job submission
## -------------------------------------------------------
# boolean switch to turn processing on or off
opal.ip.processing=true

# number of jobs per IP per hour
opal.ip.limit=10

# block all jobs from this IP - comma separated entries (optional)
opal.ip.blacklist=66.102.7.104, 137.223.43.127

# always allow jobs from this IP - comma separated entries (optional)
opal.ip.whitelist=66.102.7.105, 127.0.0.1
## -------------------------------------------------------
## END: information for the per IP limits on job submission
```

After editing **etc/opal.properties** reinstall Opal using the **ant install** command, and restart Tomcat for the changes to take effect.

8.5. Configuring Email Notifications

Opal provides an option to notify users of the job completion status by email. Appropriate properties for configuring this feature are set in the **etc/opal.properties** file. By default, the email notifications are turned off. To turn it on, set the **mail.enable** to true and update the rest of the properties appropriately:

```
## BEGIN: email configuration for server
##        To be used for notifying users
## ------------------------------------------
# set to true if email notification is to be turned ON
```

```
mail.enable=true
# url for smtp server
mail.smtp.host=smtp.gmail.com
# whether the server needs authentication
mail.smtp.auth=true
# turn debugging on or off
mail.smtp.debug=false
# from address in notification email - could be a "no-reply" address
mail.smtp.from=foo@bar.com
# credentials for logging into smtp server
mail.smtp.user=foo@bar.com
mail.smtp.password=foobar
## ------------------------------------------
## END: email configuration for server
```

After editing **etc/opal.properties**, reinstall Opal using the **ant install** command, and restart Tomcat for the changes to take effect.

Users who want to receive email notifications will need to specify their email address via the Dashboard (this field is present on every form), or via the command-line client.

8.6. Sample Application Configuration File

This example shows how to write the Opal job submission configuration file for the PDB2PQR[2] program. PDB2PQR is designed to automate many of the common tasks of preparing structures for continuum electrostatics calculations, by providing a platform-independent utility for converting protein files in PDB format to PQR format. An example file $OPAL_HOME/configs/pdb2pqr_config.xml includes advanced features such as specification of a complex set of the command-line arguments.

```
<appConfig xmlns="http://nbcr.sdsc.edu/opal/types"
           xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <metadata appName="PDB2PQR 1.8">
    <usage><![CDATA[PDB2PQR is a Python software package that automates many of
              the common tasks of preparing structures for continuum electrostatics
              calculations, providing a platform-independent utility for converting
              protein files in PDB format to PQR format. Version 1.8.
              <BR><A HREF=http://www.nbcr.net/ws_help/PDB2PQR/>
              Tutorial: http://www.nbcr.net/ws_help/PDB2PQR/</A>
            ]]>
    </usage>

    <info xsd:type="xsd:string">
    <![CDATA[
pdb2pqr  (Version 1.8)
Usage: pdb2pqr.py [options] PDB_PATH PQR_OUTPUT_PATH

This module takes a PDB file as input and performs optimizations before
yielding a new PQR-style file in PQR_OUTPUT_PATH. If PDB_PATH is an ID it will
automatically be obtained from the PDB archive.

Options:
```

```
--version             show program’s version number and exit
-h, --help            show this help message and exit

Mandatory options:
  One of the following options must be used.

  --ff=FIELD_NAME     The forcefield to use - currently AMBER, CHARMM,
                      PARSE, TYL06, PEOEPB and SWANSON are supported.
  --userff=USER_FIELD_FILE
                      The user created forcefield file to use. Requires
                      --usernames overrides --ff
  --clean             Do no optimization, atom addition, or parameter
                      assignment, just return the original PDB file in
                      aligned format. Overrides --ff and --userff

General options:
  --nodebump          Do not perform the debumping operation
  --noopt             Do not perform hydrogen optimization
  --chain             Keep the chain ID in the output PQR file
  --assign-only       Only assign charges and radii - do not add atoms,
                      debump, or optimize.
  --ffout=FIELD_NAME  Instead of using the standard canonical naming scheme
                      for residue and atom names, use the names from the
                      given forcefield - currently AMBER, CHARMM, PARSE,
                      TYL06, PEOEPB and SWANSON are supported.
  --usernames=USER_NAME_FILE
                      The user created names file to use. Required if using
                      --userff
  --apbs-input        Create a template APBS input file based on the
                      generated PQR file.  Also creates a Python pickle for
                      using these parameters in other programs.
  --ligand=PATH       Calculate the parameters for the ligand in mol2 format
                      at the given path. Pdb2pka must be compiled.
  --whitespace        Insert whitespaces between atom name and residue name,
                      between x and y, and between y and z.
  --typemap           Create Typemap output.
  --neutraln          Make the N-terminus of this protein neutral (default
                      is charged). Requires PARSE force field.
  --neutralc          Make the C-terminus of this protein neutral (default
                      is charged). Requires PARSE force field.
  -v, --verbose       Print information to stdout.

Propka options:
  --with-ph=PH        Use propka to calculate pKas and apply them to the
                      molecule given the pH value. Actual PropKa results
                      will be output to <output-path>.propka.
  --reference=REFERENCE
                      setting which reference to use for stability
                      calculations [neutral/low-pH]

Extension options:
  --chi               Print the per-residue backbone chi angle to {output-
                      path}.chi
  --summary           Print protein summary information to {output-
                      path}.summary.
  --contact           Print a list of contacts to {output-path}.con
  --salt              Print a list of salt bridges to {output-path}.salt

Hbond extension options:
  --hbond             Print a list of hydrogen bonds to {output-path}.hbond
  --whatif            Change hbond output to WHAT-IF format.
  --angle_cutoff=ANGLE_CUTOFF
                      Angle cutoff to use when creating hbond data (default
                      30.0)
  --distance_cutoff=DISTANCE_CUTOFF
                      Distance cutoff to use when creating hbond data
                      (default 3.4)
  --old_distance_method
                      Use distance from donor hydrogen to acceptor to
                      calculate distance used with --distance_cutoff.

Resinter extension options:
  --resinter          Print interaction energy between each residue pair in
                      the protein to {output-path}.resinter.
  --residue_combinations
                      Remap residues to different titration states and rerun
                      resinter appending output. Consider only the minimum
                      number of whole protein titration combinations needed
                      to test each possible pairing of residue titration
                      states. Normally used with --noopt. If a protein
                      titration state combination results in a pair of
                      residue being  re-tested in the same individual
                      titration states a warning will be generated if the
                      re-tested result is different. This warning should not
                      be possible if used with --noopt.
  --all_residue_combinations
                      Remap residues to ALL possible titration state
                      combinations and rerun resinter appending output.
                      Results with --noopt should be the same as
                      --residue_combinations. Runs considerably slower than
                      --residue_combinations and generates the same type of
                      warnings.  Use without --noopt to discover how
                      hydrogen optimization affects residue  interaction
                      energies via the warnings in the output.

Rama extension options:
  --rama              Print the per-residue phi and psi angles to {output-
                      path}.rama for Ramachandran plots
  --phi_only          Only include phi angles in output. Rename output file
                      {output-path}.phi
  --psi_only          Only include psi angles in output. Rename output file
                      {output-path}.psi
  ]]>
  </info>

  <types>
```

```
<!-- list of flags -->
<flags>
    <flag>
        <id>verbose</id>
        <tag>--verbose</tag>
        <textDesc>Print information to stdout</textDesc>
        <default>true</default>
    </flag>
    <flag>
        <id>nodebump</id>
        <tag>--nodebump</tag>
        <textDesc>Do not perform the debumping operation</textDesc>
    </flag>
    <flag>
        <id>noopt</id>
        <tag>--noopt</tag>
        <textDesc>Do not perform hydrogen optimization</textDesc>
    </flag>
    <flag>
        <id>chain</id>
        <tag>--chain</tag>
        <textDesc>Keep the chain ID in the output PQR file</textDesc>
    </flag>
    <flag>
        <id>assign-only</id>
        <tag>--assign-only</tag>
        <textDesc>Only assign charges and radii - do not add atoms, debump, or optimize</tex
    </flag>
    <flag>
        <id>clean</id>
        <tag>--clean</tag>
        <textDesc>Do no optimization, atom addition, or parameter assignment,
            just return the original PDB file in aligned format. Overwrites chosen forcefiel
        </textDesc>
    </flag>
    <flag>
        <id>apbs-input</id>
        <tag>--apbs-input</tag>
        <textDesc>Create a template APBS input file based on the generated PQR file.
                  Also creates a Python pickle for using these parameters in other programs.
        </textDesc>
    </flag>
    <flag>
        <id>whitespace</id>
        <tag>--whitespace</tag>
        <textDesc>Insert whitespaces between atom name and residue name,
                   between x and y, and between y and z </textDesc>
    </flag>
    <flag>
        <id>typemap</id>
        <tag>--typemap</tag>
        <textDesc>Create Typemap output </textDesc>
    </flag>
```

```
            <flag>
                <id>neutraln</id>
                <tag>--neutraln</tag>
                <textDesc>Make the N-terminus of this protein neutral (default is charged)</textDesc
            </flag>
            <flag>
                <id>neutralc</id>
                <tag>--neutralc</tag>
                <textDesc>Make the C-terminus of this protein neutral (default is charged)</textDesc
            </flag>

            <flag>
                <id>chi</id>
                <tag>--chi</tag>
                <textDesc>Print the per-residue backbone chi angle to {output-path}.chi</textDesc>
            </flag>
            <flag>
                <id>rama</id>
                <tag>--rama</tag>
                <textDesc>Print the per-residue phi and psi angles to {output-path}.rama
                     for Ramachandran plots</textDesc>
            </flag>
            <flag>
`               <id>psi_only</id>
                <tag>--psi_only</tag>
                <textDesc>Only include phi angles in Rama output. Rename output file {output-path}.p
            </flag>
            <flag>
                <id>phi_only</id>
                <tag>--phi_only</tag>
                <textDesc>Only include phi angles in Rama output. Rename output file {output-path}.p
            </flag>
            <flag>
`               <id>contact</id>
                <tag>--contact</tag>
                <textDesc>Print a list of contacts to {output-path}.con</textDesc>
            </flag>
            <flag>
`               <id>salt</id>
                <tag>--salt</tag>
                <textDesc>Print a list of salt bridges to {output-path}.salt </textDesc>
            </flag>
            <flag>
`               <id>summary</id>
                <tag>--summary</tag>
                <textDesc>Print protein summary information to {output-path}.summary.</textDesc>
            </flag>
            <flag>
                <id>hbond</id>
                <tag>--hbond</tag>
                <textDesc>Print a list of hydrogen bonds to {output-path}.hbond</textDesc>
            </flag>
            <flag>
```

```
        <id>whatif</id>
        <tag>--whatif</tag>
        <textDesc>Change hbond output to WHAT-IF format.</textDesc>
    </flag>
    <flag>
        <id>old_distance_method</id>
        <tag>--old_distance_method</tag>
        <textDesc>Use distance from donor hydrogen to acceptor to calculate distance
             used with --distance_cutoff.
        </textDesc>
    </flag>
    <flag>
        <id>resinter</id>
        <tag>--resinter</tag>
        <textDesc>Print interaction energy between each residue pair in
              the protein to {output-path}.resinter.</textDesc>
    </flag>
    <flag>
        <id>residue_combinations</id>
        <tag>--residue_combinations</tag>
        <textDesc>Remap residues to different titration states and rerun
                  resinter appending output. Consider only the minimum
                  number of whole protein titration combinations needed
                  to test each possible pairing of residue titration
                  states. Normally used with --noopt. If a protein
                  titration state combination results in a pair of
                  residue being  re-tested in the same individual
                  titration states a warning will be generated if the
                  re-tested result is different. This warning should not
                  be possible if used with --noopt.
        </textDesc>
    </flag>
    <flag>
        <id>all_residue_combinations</id>
        <tag>--all_residue_combinations</tag>
        <textDesc>Remap residues to ALL possible titration state
                  combinations and rerun resinter appending output.
                  Results with --noopt should be the same as
                  --residue_combinations. Runs considerably slower than
                  --residue_combinations and generates the same type of
                  warnings.  Use without --noopt to discover how
                  hydrogen optimization affects residue  interaction
                  energies via the warnings in the output.
        </textDesc>
    </flag>

</flags>

<!-- list of tagged parameters -->
<taggedParams>
    <separator>=</separator>
    <param>
        <id>forcefield</id>
        <tag>--ff</tag>
        <paramType>STRING</paramType>
        <required>true</required>
        <value>AMBER</value>
        <value>CHARMM</value>
        <value>PARSE</value>
        <value>PEOEPB</value>
        <value>SWANSON</value>
        <value>TYL06</value>
        <default>PARSE</default>
        <textDesc>Currently supported</textDesc>
    </param>
    <param>
        <id>user-forcefield</id>
        <tag>--userff</tag>
        <paramType>FILE</paramType>
        <ioType>INPUT</ioType>
        <required>false</required>
        <textDesc>User-defined forcefield </textDesc>
    </param>
    <param>
        <id>usernames</id>
        <tag>--usernames</tag>
        <paramType>FILE</paramType>
        <ioType>INPUT</ioType>
        <required>false</required>
        <textDesc>The user-created names file to use. Required if using user defined forcefi

    </param>
    <param>
        <id>ffout</id>
        <tag>--ffout</tag>
        <paramType>STRING</paramType>
        <value>AMBER</value>
        <value>CHARMM</value>
        <value>PARSE</value>
        <value>PEOEPB</value>
        <value>SWANSON</value>
        <value>TYL06</value>
        <textDesc>Instead of using the standard canonical naming scheme for residue and atom
                  use names from the given forcefield:
        </textDesc>
    </param>
    <param>
        <id>with-ph</id>
        <tag>--with-ph</tag>
        <paramType>FLOAT</paramType>
        <textDesc>Use PROPKA to calculate pKas and apply them to the molecule given the pH v
                  Actual PropKa results will be output to {output-path}.propka
        </textDesc>
    </param>
    <param>
        <id>ligand</id>
```

```
            <tag>--ligand</tag>
            <paramType>FILE</paramType>
            <ioType>OUTPUT</ioType>
            <textDesc>Calculate the parameters for the ligand in mol2 format at the given path.
                      Pdb2pka must be compiled.
            </textDesc>

        </param>
        <param>
            <id>angle_cutoff</id>
            <tag>--angle_cutoff</tag>
            <paramType>FLOAT</paramType>
            <textDesc>Angle cutoff to use when creating hbond data (default 30.0)</textDesc>
        </param>
        <param>
            <id>distance_cutoff</id>
            <tag>--distance_cutoff</tag>
            <paramType>FLOAT</paramType>
            <textDesc>Distance cutoff to use when creating hbond data (default 3.4)</textDesc>
        </param>
    </taggedParams>

    <!-- list of untagged parameters, in order -->
    <untaggedParams>
        <param>
            <id>inId</id>
            <paramType>STRING</paramType>
            <ioType>INPUT</ioType>
            <!--  <required>true</required>   -->
            <textDesc>a PDB ID (fetch input file from PDB archive; for test use 1a1p)</textDesc>
        </param>
        <param>
            <id>inFile</id>
            <paramType>FILE</paramType>
            <ioType>INPUT</ioType>
            <!--  <required>true</required>   -->
            <textDesc><![CDATA[ upload a PDB input file (test file
                      <a href="http://www.rcsb.org/pdb/files/1a1p.pdb">http://www.rcsb.org/pdb/f
            </textDesc>
        </param>
        <param>
            <id>output-file</id>
            <paramType>FILE</paramType>
            <ioType>OUTPUT</ioType>
            <required>true</required>
            <default>output.pqr</default>
            <textDesc>The desired output name of the PQR file to be generated</textDesc>
        </param>
    </untaggedParams>
    <groups>
      <group>
        <name>inputParam</name>
        <elements>inId inFile</elements>
```

```
  <required>true</required>
  <exclusive>true</exclusive>
  <textDesc>Please enter either:</textDesc>
</group>
<group>
  <name>ForceFieldGroup</name>
  <elements>forcefield </elements>
  <required>true</required>
  <textDesc>Pick a forcefield to use:</textDesc>
</group>
<group>
  <name>UserForceFieldGroup</name>
  <elements>user-forcefield usernames</elements>
  <required>true</required>
  <textDesc>User forcefield to use (overwrites previous forcefield):</textDesc>
</group>
<group>
  <name>outputFile</name>
  <elements>output-file</elements>
  <required>true</required>
  <textDesc>Output file name</textDesc>
</group>
<group>
  <name>ffoutGroup</name>
  <elements>ffout</elements>
  <required>false</required>
  <textDesc>Pick an output naming scheme to use</textDesc>
</group>
<group>
  <name>AvailableOptionsGroup</name>
  <elements>clean nodebump noopt chain assign-only with-ph apbs-input
            ligand whitespace typemap neutraln neutralc verbose </elements>
  <required>false</required>
  <textDesc>Available options </textDesc>
</group>
<group>
  <name>ExtensionOptionsGroup</name>
  <elements>chi contact salt summary </elements>
  <textDesc>Extensions options:</textDesc>
</group>
<group>
  <name>HbondExtensionOptionsGroup</name>
  <elements>hbond whatif angle_cutoff distance_cutoff old_distance_method </elements>
  <textDesc> Hbond Extensions options:</textDesc>
</group>
<group>
  <name>ResinterExtensionOptionsGroup</name>
  <elements>resinter residue_combinations all_residue_combinations </elements>
  <textDesc>Resinter Extensions options:</textDesc>
</group>
<group>
  <name>RamaExtensionOptionsGroup</name>
  <elements>rama phi_only psi_only </elements>
```

```
            <textDesc>Rama Extensions options:</textDesc>
          </group>
        </groups>
    </types>

  </metadata>

  <binaryLocation>/opt/pdb2pqr_1.8/pdb2pqr.py</binaryLocation>
   <parallel>false</parallel>
</appConfig>
```

8.7. Sample Script for Generating Proxy

This sample script `newcert.sh` gets the certificate from the proxy. This is useful if the Opal job managers are configured to use a proxy to submit jobs on behalf of the user. Note that the **-t 168** means that the certificate will be valid for 168 hours or 7 days. The default certificate life time is only 12 hours.

```
#!/bin/bash

export GLOBUS_LOCATION=/opt/gt4
. $GLOBUS_LOCATION/etc/globus-user-env.sh
. $GLOBUS_LOCATION/globus-devel-env.sh

echo $password | $GLOBUS_LOCATION/bin/myproxy-logon -t 168 -s myproxy.teragrid.org
     -S &> logfile.log
echo running `date`>>logfile.log

cat logfile.log >> get_cert.log
```

You can add the above script to crontab to run `newcert.sh` at the beginning of every hour.

```
0 * * * * $path_to_newcert_sh
```

Notes

1. http://www.irisa.fr
2. http://pdb2pqr.sourceforge.net/