slycat Documentation

Release 1.2.0

Author

March 02, 2017

Contents

This is Slycat™ - a web-based data science analysis and visualization platform, created at Sandia National Laboratories.

Slycat™ is a web-based system for analysis of large, high-dimensional data, developed to provide a collaborative platform for remote analysis of data ensembles. An *ensemble* is a collection of data sets, typically produced through a series of related simulation runs. More generally, an ensemble is a set of samples, each consisting of the same set of variables, over a shared high-dimensional space describing a particular problem domain. Ensemble analysis is a form of meta-analysis that looks at the combined behaviors and features of a group of simulations in an effort to understand and describe the underlying domain space. For instance, sensitivity analysis uses ensembles to examine how simulation input parameters and simulation results are correlated. By looking at groups of runs as a whole, higher level patterns can be seen despite variations in the individual runs.

The Slycat™ system integrates data management, scalable analysis, and visualization via commodity web clients using a multi-tiered hierarchy of computation and data storage. Analysis models are computed local or on the Slycat™ server, and model artifacts are stored in a project database. These artifacts are the basis for visualizations that are delivered to users' desktops through ordinary web browsers. Slycat™ currently provides two types of analysis: canonical correlation analysis (CCA) to model relationships between inputs and output metrics, and time series analysis featuring clustering and comparative visualization of waveforms. *Install Slycat* to try it for yourself!

CHAPTER 1

Design

Slycat™ incorporates several components:

- A Web Server that can load, transform, index, and analyze moderate amounts of data, storing the analysis results for later visualization.
- A web-based user interface that you use to pull your data into the Slycat™ Web Server, compute analyses, and view analysis results. You can use Slycat™ with any modern, standards-compliant browser, including Firefox, Safari, and Chrome. There is no software to install on your workstation.
- A collection of command-line clients that can be used to push data into Slycat™ Web Server and control it remotely, if that suits your workflow better.

The Slycat™ Web Server provides easy collaboration and a graphical user interface for analyses that have broad appeal.

CHAPTER 2

Documentation:

User Manual

Overview

Why Slycat™?

Projects

Models

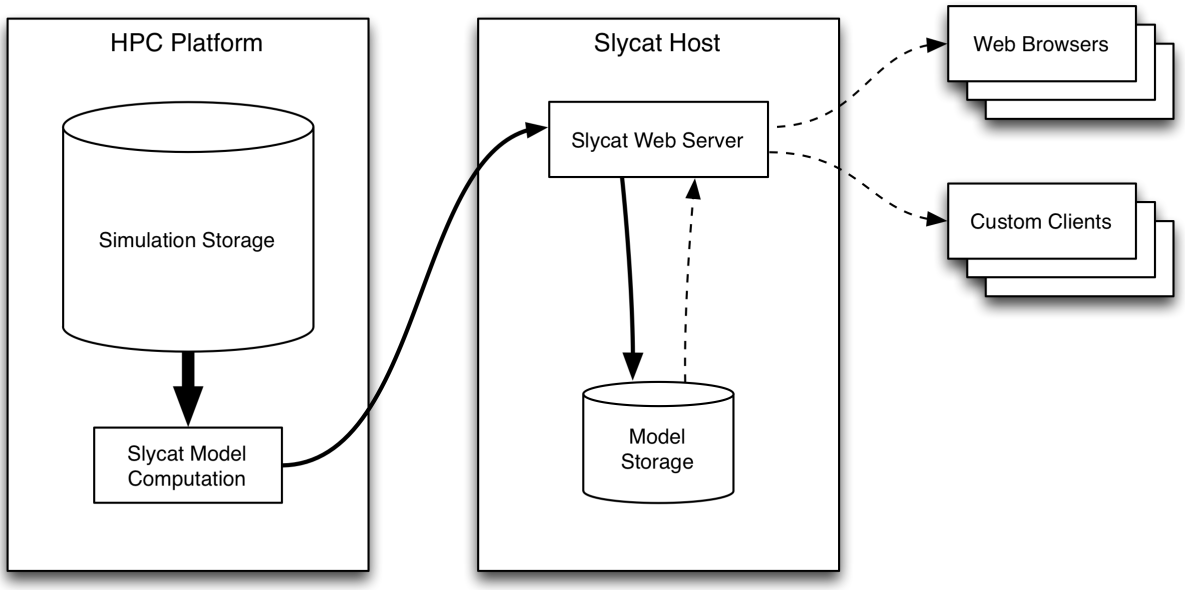
CCA Model

Parameter Space Model

Timeseries Model

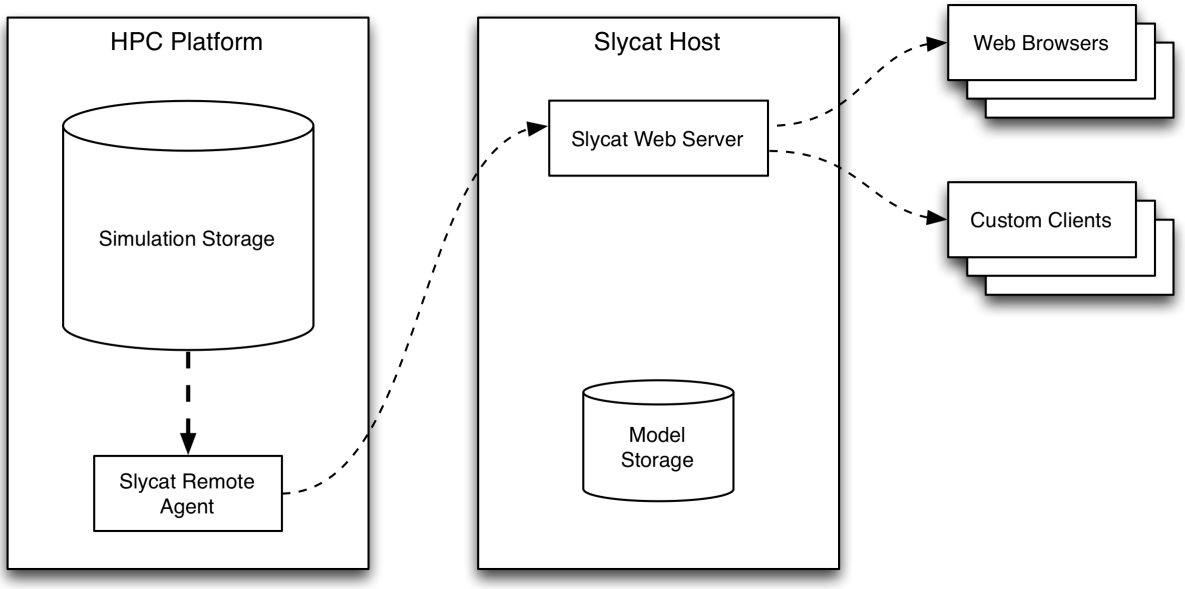
Design

Because Slycat™ is a system for analysis of data ensembles, and ensembles typically include orders of magnitude more data than individual simulation runs, managing data movement is an integral part of the Slycat™ design. Ideally, we want to perform one-time computation on the host where data lives so that only an analytical model – typically orders of magnitude smaller than the original data – is moved across the network to the Slycat™ host. This leads to the following Slycat™ architectural design:



In the above case, large data on an HPC platform is analyzed in-place to produce greatly reduced model artifacts that are stored by the Slycat™ web server. Later, these artifacts are delivered – incrementally and on-demand – to interactive clients.

However, it isn't always possible to reduce the analytical workflow to an ideal, reduced-size model. For example, users may wish to interactively browse through the raw outputs of an ensemble of simulations. For this case, Slycat™ provides a remote "agent" process that can access data on an HPC platform, packaging and compressing it on-demand for live delivery to interactive clients:



As an example, this mode of interaction is ideal for browsing through output image series on a remote server - in addition to delivering individual images, the agent can compress images on-the-fly into video streams for live playback.

Tutorial

Install Slycat

As a convenience, we provide a Docker image that has Slycat and all its dependencies preinstalled. Using the Slycat image, you can quickly begin exploring Slycat, try some tutorials, and run small analyses on your own data. Eventually you might want to *Setup Slycat Web Server* on your own hardware to perform large-scale analyses.

Install Docker

Installation

Because Docker uses Linux-specific kernel features, you will need to run Docker in a virtual machine (VM) on your Mac or Windows environment. Fortunately, Docker makes this relatively easy:

- Download the latest *docker* for your specific environment from https://www.docker.com/
- follow the instruction for installing docker on your machine

With docker installed and running and the DOCKER_* environment variables set, the rest of the install instructions are platform-independent.

Note: If you're using Docker behind a proxy, you'll need additional configuration so it can access the network to download the Slycat image:

- To configure proxy information, ssh into the Boot2Docker VM:

```
$ docker-machine ssh default
```

- Create / modify the */var/lib/boot2docker/profile* file to set proxy info:

```
$ sudo vi /var/lib/boot2docker/profile
```

- Add the proxy info using *protocol://host:port*, for example:

```
export HTTP_PROXY=http://your.proxy.name:80
export HTTPS_PROXY=http://your.proxy.name:80
```

- If your site uses SSL interception, you will need to get a copy of the interception certificate, and append it to /etc/ssl/cacerts.pem:

```
$ sudo vi /etc/ssl/cacert.pem
```

- Restart the Docker service and exit the Boot2Docker VM:

```
$ sudo /etc/init.d/docker restart
$ exit
```

Warning:

- If your site uses SSL interception, you must append the certificate to /etc/ssl/cacerts.pem and restart the Docker service before downloading images.

Download the Image and Create a Container

Now that you have the Docker daemon running and DOCKER_HOST set to connect to it, you're ready to download the Slycat image and create a container:

```
$ docker run -d -p 2222:22 -p 80:80 -p 443:443 --name slycat sandialabs/slycat-developer
```

Docker will begin downloading the *sandialabs/slycat* image, and will create a container with the name *slycat* (you will use this name as a convenient way to reference the container in subsequent commands). The Slycat server will begin running as soon as the download is complete. Leave the container running for the remainder of these tutorials.

Warning: A new image is currently being created so the image has to currently be built from scratch via build.py in the slycat github repi /open-source-docker/docker/open-source-build/build.py

Connect to Slycat with a Web Browser

Open a web browser and point it to the Slycat server at https://<docker host ip>

- If you're running the Slycat container on a Linux host, this will be *https://localhost*.
- If you're running the Slycat container using boot2docker on another platform, this will be the IP address returned by:

```
$ docker-machine ip

The VM's Host only interface IP address is: 192.168.99.100
```

- The browser will complain that the server certificate is untrusted. This is because we use a self-signed certificate for the Docker container. Follow your browser's procedures to temporarily trust the connection.
- When prompted for a username and password, enter *slycat* for both.
- The Slycat Projects page opens in the browser.

Next Steps

- That's it! Now that you're up-and-running, it's time to *Create a CCA Model*.

Create a CCA Model

In Slycat, we perform an analysis by ingesting data and creating a *model*. One type of Slycat model is *Canonical Correlation Analysis (CCA)*, used to model relationships between a set of input and output metrics. Before creating a CCA model however, we must create a *project*, which is used to organize and control access to models.

Create a Project

- With your web browser still pointed to the Slycat Projects page from the previous section, click the *Create* dropdown menu on the Slycat navbar, choose *New Project*, enter "MyProject" as the project name in the wizard that appears, and click *Finish*.
- The browser switches to a separate page for the new project.

Generate a CCA Model

- In the new model page, click the *Create* dropdown menu again, and choose *New Remote CCA Model*. Remote CCA is an analysis performed on a file retrieved from a host other than (remote to) the Slycat web server.
- In the wizard that appears, enter "MyCCA" as the model name and click *Next*.
- We are going to load a file that happens to be located on the same host as the Slycat server ("localhost"), but could be located on any host that's reachable from the Slycat server over ssh. In the wizard, choose *localhost* in the Hostname dropdown and enter username *slycat* and password *slycat*, and click *Next*.
- The remote file browser appears, displaying the filesystem of the host you chose in the previous step. Navigate to the */home/slycat/src/slycat/data* directory, then double-click *cars.csv*. This file contains data describing 406 different types of automobile in CSV format.
- A list of the variables (columns) from the uploaded file appears, along with two columns of checkboxes, allowing you to designate each variable as in input, an output, or neither. Use the checkboxes to select "Cylinders", "Displacement", "Weight", and "Year" as inputs, and "MPG", "Horsepower", and "Acceleration" as outputs. Uncheck "Origin".
- Leave the "Scale inputs to unit variance." checkbox checked, and click *Finish*.

Wait for Model Completion

The time to compute models can vary from seconds to hours, depending on the complexity of the model and the data. For this reason, Slycat computes models in the background, allowing you to:

- Continue interacting with existing projects and models.
- Create more than one model at a time.

This example is very small, so it should complete in a few seconds. You can jump to the new model by clicking the "You can check on it *here*" link in the final page of the wizard. Or, you can close the wizard, and you will see the new *MyCCA* model listed on the project page, where you can click on it to open it.

View a CCA Model

- The bottom half of the model page features a table containing the raw data used to compute the model. Input variables are color-coded green, output variables are color-coded purple, and unused variables are color-coded white.
- The upper-left corner of the page contains the CCA table, a high-level overview of the CCA results including statistical significance measures and bar-plots for each input and output variable over three CCA components.
- The upper-right corner of the page contains a scatterplot detailing how well each individual observation in the raw data fits the currently selected CCA component.

Interact with a CCA Model

- Click a component name ("CCA1", "CCA2", or "CCA3") in the CCA table to select that component, displaying its bar plot and updating the scatterplot.
- Click variable names in the CCA table or the raw data table to color code observations with that variable.
- Hover over columns in the CCA table and the raw data table to reveal sorting widgets.
- Click observations in the scatterplot to highlight the corresponding entry in the raw data table.
- Click and drag in the scatterplot to rubber-band-select multiple observations.

- Click rows or shift-click ranges of rows in the raw data table to highlight corresponding observations in the scatterplot.

Next Steps

Now that you've created your first CCA model it's time to *Create a Timeseries Model*.

Create a Timeseries Model

The Slycat *Timeseries Model* provides time series analysis based on clustering and comparative visualization of waveforms. However, unlike *Creating a CCA Model*, you can't upload data for a Timeseries model using your web browser. Instead, you'll use one Python script to synthesize some time series data in a format suitable for use with Slycat, and a second script to compute the model and push it to the Slycat Web Server. This will demonstrate how Slycat's *REST API* can be used to control Slycat programmatically, so you can transform and upload your data using any language that supports HTTP networking.

Generate Timeseries Data

- For this example we'll ssh into the Slycat Docker container, where the scripts to be run are already installed. Normally, you would run these scripts on the system where your data was located:

```
$ ssh slycat@<docker host ip> -p2222
```

In this case, substitute your docker host IP address. If you're running docker on a Linux host, this will be *localhost*. On systems using Boot2Docker, it will be the IP address returned by:

```
$ boot2docker ip
```

When prompted, enter password *slycat*.

- Switch to the Slycat source code directory containing the sample client scripts:

```
$ cd src/slycat/web-client
```

- The script for synthesizing data is designed to run in parallel, so start some parallel worker processes in the background:

```
$ ipcluster start --daemonize
```

- Synthesize some time series data, organized for use with Slycat:

```
$ python slycat-create-sample-timeseries-hdf5.py
```

- The script creates a *sample-timeseries* directory and populates it with a set of random input variables and ten output time series, each containing two variables (additional command line parameters are available to synthesize data of arbitrary size).

Compute a Timeseries Model

- Now that you have some sample data, run:

```
$ python slycat-create-timeseries-model-from-hdf5.py --no-verify sample-timeseries
```

and enter the password *slycat* when prompted (this script also runs in parallel, using the workers you started previously):

```
slycat password:
INFO - Storing clustering parameters.
INFO - Storing input table attribute 0
INFO - Storing input table attribute 1
INFO - Storing input table attribute 2
...
INFO - Your new model is located at https://localhost:8092/models/...
```

View a Timeseries Model

- Point your web browser to the Slycat home page at https://<docker host ip>, if it isn't already.
- In the Slycat navbar at the top of the page, you should see a gray status dropdown containing two numbers separated by a slash. Those numbers are the number of models being computed, and the number of recently completed models, respectively.
- Click on the status dropdown to see a menu containing an entry for all in-progress and recently completed models.
- Wait for the *sample-timeseries* model to be a completed (a green check appears to its left), if it hasn't already.
- Click the *sample-timeseries* entry in the status dropdown, and the browser opens the new model page.
- At the top of the page there is a list of output variables.
- At page left is a hierarchical clustering of the output variable timeseries, displayed as a dendrogram.
- At page right the raw output timeseries are plotted.
- At the bottom of the page is a table containing raw input data.

Interact with a Timeseries Model

- Click on an output variable name at the top of the page to select that output, updating the rest of the interface.
- Click variable names in the raw input table to color timeseries using that variable.
- Click individual raw input table rows or shift-click ranges of rows to highlight the corresponding timeseries.
- Click nodes in the dendrogram to display only those waveforms.
- Double-click nodes in the dendrogram to expand / collapse their children.

Next Steps

Next, let's move on to *Create a Parameter Image Model*.

Create a Parameter Image Model

The Slycat *Parameter Image Model* associates images with feature vectors, and would typically be used to explore the input parameters for an ensemble of image-generating simulations. For this type of model, you'll use one Python script to synthesize image and parameter data in a format suitable for use with Slycat, then import the data using a web browser user interface.

Generate Image Data

- If you haven't already ssh into the Slycat server:

```
$ ssh slycat@<docker ip address> -p2222
```

- Switch to the Slycat source code directory containing sample client scripts:

```
$ cd src/slycat/web-client
```

- Synthesize some parameter image data, organized for use with Slycat:

```
$ python slycat-create-sample-parameter-image-csv.py
```

- The script creates a *sample-parameter-images* directory containing a set of randomly-generated images, and a *sample-parameter-images.csv* file that contains links to the images, plus randomly-generated numeric, string, and categorical parameters (the script includes optional command line parameters to control how much data is generated). Now that you have some sample data, you're ready to pull it into Slycat.

Create a Project

- Point a web browser to the Slycat web server at https://<docker ip address>
- Use *Create* > *New Project* on the Slycat navbar, enter "My PI Project" as the project name in the wizard, and click *Finish*.
- The browser switches to a separate page for the new project.

Ingest a Parameter Image Model

- In the project page, choose *Create* > *New Remote Parameter Image Model*. This wizard is used to ingest a file from a machine other than the host running the web browser.
- In the wizard that opens, enter "MyPI" as the model name and click *Next*.
- In the login screen that follows, choose hostname "localhost", enter username "slycat" and password "slycat" and choose *Next*. Note that these credentials will be used to SSH to another machine to load the parameter image data (in this case, the "other" machine happens to be localhost, but the Slycat server can be configured to connect to any other host that's accessible via SSH).
- In the remote file browser that opens, navigate to the */home/slycat/src/slycat/web-client* directory, and double-click the *sample-parameter-images.csv* file that you generated in a previous step.
- A list of the variables (columns) from the file appears, along with five columns of checkboxes, allowing you to designate each variable as in input, output, rating, categorical, or image variable. Slycat trys to guess the types of the individual variables, but you will need to make some manual changes. Use the checkboxes to designate "category0" and "category1" as Category variables, and "rating0" and "rating1" as Rating variables. Change "output0", "output1", and "output2" to Output variables, and uncheck "unused0", "unused1", "unused2".
- Note that the "image0", "image1", and "image2" columns are already correctly identified as Image variables, so leave them alone, and click *Finish*.
- As before, you can navigate to the newly created model using the link in the last page of the wizard, the link on the underlying project page, or the link in the status dropdown in the navbar.

View a Parameter Image Model

- The bottom third of the model page features a table containing the raw data used to compute the model. Input variables are color-coded green, output variables are color-coded purple, and the remaining variables are color-coded white.
- The rest of the page contains a scatterplot with a point for each observation (row) in the data table.

Interact with a Parameter Image Model

- If you hover over any of the scatterplot points, you will be prompted for a username and password to retrieve the corresponding image - when this happens use *slycat* and *slycat* as you've done before.
- Use the "X Axis" and "Y Axis" dropdown menus at the top of the display to use any two numeric variables for the scatterplot axes.
- Click variable names in the raw data table or use the "Point Color" dropdown menu to color the scatterplot points using any numeric variable.
- Hover over columns in the raw data table to reveal sorting widgets.
- Click observations in the scatterplot to highlight the corresponding entry in the raw data table.
- Click and drag in the scatterplot to rubber-band-select multiple observations.
- Click rows or shift-click ranges of rows in the raw data table to highlight corresponding observations in the scatterplot.
- Choose an image variable using the "Image Set" dropdown at the top of the display, then hover the mouse over observations in the scatterplot to see the corresponding images.
- Click the "pin" icon in the upper-left-corner of an image to display it permanently.
- Click the "close" icon in the upper-left-corner of a pinned image to close it.
- Drag the "resize" icon in the lower-right-corner of a pinned image to resize it.
- Click-and-drag anywhere else within a pinned image to reposition it on the page.
- Click-and-drag the colorbar to reposition it on the page.

Next Steps

That's it for the tutorial ... now on to *Managing Docker*.

Managing Docker

Here are some tips on managing your Slycat Docker container:

Stopping Slycat

The processes in the Slycat container that you created with *docker run ...* will continue running until you stop it:

```
$ docker stop slycat
```

If you are using Boot2Docker to run your Slycat container in a VM on a non-Linux platform, you may want to shut the VM down too:

```
$ boot2docker stop
```

Starting Slycat

If you're using Boot2Docker to run your Slycat container on a non-Linux platform, you need to start the VM:

```
$ boot2docker start # If you aren't running on a Linux host.
```

To start the Slycat container:

```
$ docker start slycat
```

... and you're ready to use Slycat again!

Setup Slycat Clients

Note: If you're new to Slycat and are here give it a try, please see *Install Slycat* instead. The following outlines how to setup a host to use the client scripts included with Slycat to upload data to an existing Slycat web server. If you don't already have a web server, you probably want to start with *Setup Slycat Web Server*.

Slycat includes a Python package to simplify writing custom clients. Custom clients are often required to handle data ingestion, performing extraction and transformation of your specific data formats into a form usable by Slycat.

Prerequisites

You'll need to install the following with your system package manager:

- git
- python 2.7

Further, you'll need the following Python modules, installed using either your system package manager or pip:

- h5py
- ipython
- numpy
- pyzmq
- requests
- scipy

Installation

To use the functionality provided by the Slycat client scripts, you'll need to obtain a copy of the source code - typically by cloning the slycat repository from git:

```
$ cd
$ git clone git@github.com:sandialabs/slycat.git
```

Once you've cloned the repository, you need to tell Python where to find the Slycat package. The easiest way to do this is to add the slycat/packages directory to your PYTHONPATH environment variable:

```
$ export PYTHONPATH=~/slycat/packages:$PYTHONPATH
```

Now, you can run scripts that use the Slycat package.

See Also

- *REST API* - Details the underlying Slycat HTTP API, which can be used with any programming language.

Setup Slycat Web Server

Note: If you're new to Slycat and are here give it a try, please see *Install Slycat* instead. The following is a guide for users who are ready to setup their own Slycat Web Server for production.

Use the Docker Image

Many administrators should be able to use the Slycat Docker image in production directly, and we strongly urge you to try this approach first - after following the instructions at *Install Slycat*, you can simply ssh into the running Docker container:

```
$ ssh slycat@<docker ip address> -p2222
```

make a few configuration changes (assign real passwords to the root and slycat users, replace our self-signed server certificate with one of your own, configure a real password-check plugin, etc.) then continue using the image in production. Because the Slycat Docker image is a container rather than a VM, there is absolutely no performance penalty for using it in this configuration. You can even use Docker to automate this process, building your own site-specific Slycat image with our Slycat image as the base!

Installing Slycat from Scratch

If you insist on creating your own Slycat instance from scratch, we still prefer to point you to our *Dockerfiles* for information on installing Slycat and its dependencies, because these files are the actual scripts that we use to build the Slycat Docker image - thus they're an always-up-to-date and unambiguous specification of how to build a Slycat server. Even if you don't use Docker, the Dockerfiles are easy to understand and adapt to your own workflow and platform.

You will find our Dockerfiles in a set of directories located in the *docker* directory within the Slycat repo:

> https://github.com/sandialabs/slycat/tree/master/docker

There, you will find four subdirectories - *supervisord*, *sshd*, *slycat*, and *slycat-dev* - which are used to build four Docker images. Each image builds on the previous, adding new functionality:

- supervisord - Starts with a Fedora Core base system, and adds an instance of supervisord that will be used to startup the other processes.
- sshd - Installs an SSH server on top of the supervisord image, and configures supervisord to automatically start it when the container is run.
- slycat - Installs the Slycat servers and their dependencies atop the sshd image, and configures supervisord to automatically start them when the container is run.
- slycat-dev - Adds development tools to the base Slycat image, and configures the supervisorctl command so developers can easily start and stop servers themselves.

The main differences between platforms will be in how you install the various dependencies. One platform - such as Fedora Core in our Dockerfile - installs the Python h5py module and its compiled hdf5 library dependency using a single yum package, while another platform - such as Centos 6 - provides a yum package for hdf5, but no package for the Python h5py module, so you have to use pip to install it. Unfortunately, we can't enumerate all the possibilities here, so you'll have to begin with the packages listed in our Dockerfiles, and generalize to your platform.

Configuring Slycat Web Server

Whether you're setting-up an unmodified Slycat Web Server or developing new capabilities to suit your needs, you will need to know how to modify its configuration. When you start Slycat Web Server:

```
$ cd slycat/web-server
$ python slycat-web-server.py
```

... it automatically loads a file *config.ini* from the same directory as slycat-web-server.py. The sample *config.ini* that we provide with the source code is designed to start Slycat in a state that's useful for developers, so you'll likely want to copy it to some other filesystem location, modify it, and point Slycat to the modified *config.ini* instead. Once you've done that, you can specify the config file location at startup using the command-line:

```
$ python slycat-web-server.py --config=/etc/slycat/config.ini
```

The *config.ini* file is an INI file divided into sections using square braces. The *[slycat]* section is reserved for configuration specific to the functionality of the Slycat server, while the *[global]* section and any sections starting with a slash (for example: *[/style]*) are used to configure the CherryPy web server that Slycat is based upon.

The values for each setting in *config.ini* must be valid Python expressions. You should note that in the sample *config.ini* we provide, some values are simple scalars, such as *[global] server.socket_port*, while some values are nested data structures, such as *[slycat] remote-hosts*. This provides great flexibility to customize Slycat for your network. Here are some common settings you may wish to modify:

[global] Section

- engine.autoreload.on - Controls whether Slycat will automatically restart when the source code is modified. This is typically disabled in production.
- require.show_tracebacks - Controls whether exceptions during request handling will return debugging information to the client. This is typically disabled in production.
- server.socket_host - IP address of the interface to listen on for requests. Use "0.0.0.0" to listen on all interfaces. Use "127.0.0.1" to only accept requests from the local machine.
- server.socket_port - TCP port number to listen on for requests. Defaults to "8092" for development. Typically set to "443" in production with SSL enabled, or "80" with SSL disabled.
- server.ssl_certificate - Path to a certificate used for SSL encryption. Leave blank to disable SSL. Relative paths are relative to the slycat-web-server.py executable.
- server.ssl_private_key - Path to a private key used for SSL encryption. Leave blank to disable SSL. Relative paths are relative to the slycat-web-server.py executable.

[slycat] Section

- allowed-markings - List of marking types that may be assigned to models.
- plugins - List of filesystem plugin locations. You may specify individual .py files to be loaded, or directories. If you specify a directory, every .py file in the directory will be loaded, but directories are *not* searched recursively. Relative paths are relative to the slycat-web-server.py executable.

- remote-hosts - List containing an entry for each group of hosts that share a specific configuration. Each entry is a dict containing the following:
  - hostnames - Required list of hostnames that share a configuration.
  - agent - Optional dict configuring remote agent access to the entry hostnames. Some models require the Slycat Agent when accessing a remote host, and agents must be explicitly configured on a host to be used. The agent dict must contain the following:
    - command - Required string with the full remote command-line used to run the Slycat agent on the given host. Typically */full/path/to/python /full/path/to/slycat-agent.py*. Since an agent session can be initiated by any user able to login to the remote host via ssh, you should specify required environment variables as part of this command, too (for example, with *env*).
- server-admins - List of users allowed to administer the Slycat server. Server administrators have full read/write access to all projects, regardless of project ACLs.

Docker Development

One of the easiest ways to begin making changes or additions to Slycat is using our Docker image to quickly setup a development environment. Here are some guidelines to get you started:

Prerequisites

- We assume that you've already *Installed Slycat* and are familiar with how to manage the Slycat docker image.
- We provide a special developer's image that modifies the Slycat Docker image that you've been working with for easier development, so download and run it now:

```
$ docker run -p 2222:22 -p 80:80 -p 443:443 -p 5984:5984 -p 9001:9001 -d --name slycat-dev sandi
```

- You will need to note the IP address of the Docker host:
  - If you are running Docker on a Linux host, then the Docker host IP is "localhost" or "127.0.0.1"
  - If you are running Boot2Docker on a non-Linux host, then the Docker host IP is the address reported by the *boot2docker ip* command.
  - We will refer to the host address as <*docker host ip*> throughout the rest of this document.

Working Inside the Running Container

- The Slycat container includes an ssh server, so you can login to the container as user *slycat* with password *slycat*:

```
$ ssh slycat@<docker host ip> -p2222
```

- Once you're logged-in, you can pull the latest version of the source code (note that when we build the Docker container, we checkout a specific, known-good commit, so you have to switch to a branch before you pull):

```
$ cd src/slycat
$ git checkout master
$ git pull
```

- And you can edit the source code in-place:

```
$ vi packages/slycat/...
```

- The Slycat software stack includes four running servers: the couchdb database, the Slycat web server, the Slycat feed server, and an haproxy reverse proxy server. All four servers are automatically started by *supervisord* when you start the slycat-dev container. To check on their status, use the *supervisorctl* command:

```
$ supervisorctl status
couchdb                          RUNNING    pid 10, uptime 0:02:14
feed-server                      RUNNING    pid 11, uptime 0:02:14
proxy-server                     RUNNING    pid 13, uptime 0:02:14
sshd                             RUNNING    pid 9, uptime 0:02:14
web-server                       RUNNING    pid 12, uptime 0:02:14
```

  However, development is often much easier when you run one or more of the servers yourself - you can configure the server to restart automatically in response to code or configuration changes, see the server output in the console, and know immediately if a typo or syntax error causes the server to fail.

  You cannot simply kill a server process started by *supervisord*, because it will be automatically restarted. Use *supervisorctl* to stop it, then start your own copy for development:

  Running Your Own Web Server:

```
$ supervisorctl stop web-server
web-server: stopped
$ cd src/slycat/web-server
$ python slycat-web-server.py
```

  Running Your Own Feed Server:

```
$ supervisorctl stop feed-server
feed-server: stopped
$ cd src/slycat/feed-server
$ python slycat-feed-server.py
```

  Running Your Own Reverse Proxy:

```
$ supervisorctl stop proxy-server
proxy-server: stopped
$ cd src/slycat/proxy-server
$ sudo haproxy -f configuration.conf -d
```

  Typically, you would then use a separate ssh login for making code changes.

- To commit changes while logged-in to the container, you'll need to add your personal information to *~/.gitconfig*:

```
[user]
  name = Fred P. Smith
  email = fred@nowhere.com
```

- By default, the git repository in the container is configured to access the public Slycat repository using https://github.com/sandialabs/slycat repository. If you want to push your commits to the public repository, there are three alternatives:
  - Leave the repository URL unchanged, and push. You will be prompted for your github username and password.
  - Add your username to the repository URL. Then, you will only be prompted for your github password when you push:

```
$ git remote set-url origin https://username@github.com/sandialabs/slycat
```

  - Copy an existing github public key into the container, or generate a new github public key, and switch to communication over ssh:

```
$ git remote set-url origin git@github.com:sandialabs/slycat
```

Note: If you're working behind a proxy and using https:// for communication with github, you'll need to let git know about it:

```
$ export https_proxy=http://your.proxy.name:80
```

- If you need to install additional tools for development, use the *yum* and *pip* commands provided by the container to install them.

Note: If you're working behind a proxy, you'll also want to add it to /etc/yum.conf to yum can download packages:

```
proxy=http://your.proxy.name:80
```

And you'll need to specify the proxy when running pip:

```
pip install --proxy=http://your.proxy.name:80 mypackage
```

Working Outside the Running Container

Instead of working on the Slycat sources inside the running container, you may wish to edit them from the outside. One advantage of this approach is that you can edit the sources using more sophisticated graphical tools installed on your host system, instead of the minimalist command-line tools provided within the container. Another benefit is that the setup you perform (configuring your git credentials, setting-up proxy information) is part of your host system and will be retained even if you upgrade or replace the Slycat container.

One way to do this is to use *sshfs* to mount the source code inside the container to a directory on the host:

```
$ mkdir ~/src/slycat-container
$ sshfs -p 2222 slycat@<docker host ip>:/home/slycat/src/slycat ~/src/slycat-container -oauto_cache,
```

The main disadvantage to working this way is the increased latency caused by the sshfs filesystem ... some operations (such as building the documentation) will be noticably slower when run on an sshfs mount

Note that you'll still need to ssh into the container to run the Slycat server, but the server will still restart automatically whenever you save changes to the sshfs mount.

Testing

The following are required to run the Slycat test suite / view test coverage:

- behave - behavior-driven development (BDD) framework - http://pythonhosted.org/behave/
- coverage - code coverage module - http://nedbatchelder.com/code/coverage/

Setting Up Tests

The following set of instructions for the test setup assumes a new Ubuntu environment (desktop or server) with user *slycat*. It also assumes that Slycat's repository is cloned in the user's home directory. The next commands install the base packages needed for the test suite to run correctly:

```
$ cd
$ sudo apt-get update -qq
$ sudo apt-get install -y make build-essential python-software-properties ssh python-dev libldap2-dev
```

Slycat uses CouchDB as its database. Use the default installation settings for the database setup. See http://wiki.apache.org/couchdb/Installing_on_Ubuntu for troubleshooting:

```
$ sudo apt-get install -y couchdb
```

To install haproxy-1.5. Note that the Ubuntu 12.04's version is out-of-date:

```
$ sudo add-apt-repository -y ppa:vbernat/haproxy-1.5
$ sudo apt-get update -qq
$ sudo apt-get install haproxy
```

To install the virtual X server and Firefox:

```
$ sudo apt-get install xvfb firefox
```

To install FFmpeg for the agent testing:

```
$ wget http://johnvansickle.com/ffmpeg/releases/ffmpeg-release-64bit-static.tar.xz
$ mkdir ffmpeg
$ tar xf ffmpeg-release-64bit-static.tar.xz --strip-components 1 -C ffmpeg
$ export PATH=$HOME/ffmpeg:$PATH
```

To point Python to the Slycat packages:

```
$ export PYTHONPATH=$HOME/slycat/packages
```

To generate a private certificate authority:

```
$ openssl genrsa -out root-ca.key 2048
$ openssl req -x509 -new -nodes -key root-ca.key -days 365 -out root-ca.cert -subj "/C=US/ST=New Mexi
```

To generate a self-signed certificate:

```
$ openssl genrsa -out web-server.key 2048
$ openssl req -new -key web-server.key -out web-server.csr -subj "/C=US/ST=New Mexico/L=Albuquerque/C
$ openssl x509 -req -in web-server.csr -CA root-ca.cert -CAkey root-ca.key -CAcreateserial -out web-s
```

To point HAProxy to the server key and certificate:

```
$ cat web-server.key web-server.cert > ssl.pem
```

To create a directory to store HDF5 files:

```
$ mkdir slycat/data-store
```

To install and use Conda for the Python interpreter and dependencies:

```
$ wget http://repo.continuum.io/miniconda/Miniconda-latest-Linux-x86_64.sh -O miniconda.sh
$ chmod +x miniconda.sh
$ ./miniconda.sh -b
$ export PATH=$HOME/miniconda/bin:$PATH
$ conda update --yes conda
$ conda create --yes -n slycat coverage h5py mock nose paramiko Pillow pip pyparsing requests scipy
$ source activate slycat
$ pip install --no-use-wheel behave "cherrypy==3.2.6" couchdb coveralls python-ldap pystache routes t
```

Running Tests

Create a file *proxy-server-config.conf* in the */home/slycat* directory with the following content:

```
global
  daemon
  maxconn 256
  user slycat
  group slycat
  tune.ssl.default-dh-param 2048

defaults
  mode http
  option forwardfor
  timeout connect 5000ms
  timeout client 50000ms
  timeout server 50000ms
  timeout tunnel 1d

frontend http-in
  bind *:80
  redirect scheme https if !{ ssl_fc }

frontend https-in
  bind *:443 ssl crt /home/slycat/ssl.pem
  reqadd X-Forwarded-Proto:\ https
  redirect location /projects if { path / }
  use_backend slycat-feed-server if { path_beg /changes-feed }
  default_backend slycat-web-server

backend slycat-web-server
  server server1 127.0.0.1:8092

backend slycat-feed-server
  server server1 127.0.0.1:8093
```

To run the test suite, enter the following commands:

```
$ python slycat/web-server/slycat-couchdb-setup.py
$ sudo haproxy -f proxy-server-config.conf -db &
$ python slycat/feed-server/slycat-feed-server.py --config ../travis-ci/config.ini &
$ python slycat/web-server/slycat-web-server.py --config ../travis-ci/config.ini &
$ cd slycat
$ REQUESTS_CA_BUNDLE=/home/slycat/root-ca.cert coverage run --source agent,packages/slycat --omit="pa
```

Running Coverage

To run the coverage report:

```
$ coverage report
```

Modifying Tests

Behave feature and step definition files are located in the *slycat/features* and *slycat/features/steps* directories, respectively.

Coding Guidelines

- All new Python code in Slycat should follow the guidelines outlined in PEP 8 with one exception: we use two spaces for indentation instead of four. We have lots of code that predates those guidelines, but are actively updating it as we go along.

Plugins

The Slycat server includes a plugin system that streamlines the process of customizing it to suit your environment and adding new Slycat features.

Overview

A Slycat plugin is a Python module (.py file) that is loaded into the Slycat Web Server at startup. By default, Slycat ships with a set of plugins in the *web-server/plugins* directory. The set of plugins to be loaded is specified in the server's *config.ini* file. The *plugins* entry in *config.ini* is a Python list containing zero-or-more plugin locations, which may be individual .py files to be loaded, or directories. Every .py file in a directory will be loaded as a plugin, but directories are not searched recursively. Relative paths are relative to the slycat-web-server.py executable. Plugin developers can append their own paths to the list to deploy their plugins, by editing the *config.ini* file included with the Slycat source code, or by using a different config file altogether.

Once all plugin modules have been loaded, the server will call the *register_slycat_plugin* function in each module, if it exists. The function will be called with a *context* object as its sole argument. The plugin code uses the API provided by the *context* object to register new functionality with the server. This explicit registration process allows a single plugin module to register as many new capabilities as it wishes, and the registration API continues to expand as we add new categories of plugin functionality to the server.

Note: You are free to register as many plugins or as many *types* of plugins as you like within a plugin module - you are not obliged to split your code into one plugin per module, unless you want to. For example, if your organization created a new type of model and had three in-house marking types, you could put all four plugins in a single, organization-specific plugin module.

Warning: Plugin module names must be globally unique - that is, the filename of all plugin .py files loaded by the server must be unique, not just the filepaths. Thus, you should not use generic filenames like *plugin.py* for plugin modules. Instead, incorporate functionality- or organziation-specific strings into the filenames such as *bayesian-q-stat-model.py* or *acme-dynamite-division-authentication.py*. The prefix *slycat-* is reserved for plugin modules shipped with Slycat.

New Marking Types

Examples: plugins/slycat-no-marking.py, plugins/slycat-airmail-marking.py, plugins/slycat-faculty-only-marking.py

A plugin can register new *marking* types with the Slycat server. Markings are used to apply user-specific administrative or organizational labels to models such as "Draft" or "Human Resources Only".

A marking consists of the following:

1. A unique string identifier called the *marking type*.

2. A human-readable label that will become part of the user interface when prompting end-users to choose the marking for a model.
3. A block of HTML markup that provides a "badge" representation of the marking used in lists.
4. Optional block of HTML markup that will be inserted into the user interface before marked content.
5. Optional block of HTML markup that will be inserted into the user interface after marked content.

If the plugin doesn't provide 5), 4) will be displayed at the top and bottom of marked content. If 4) and 5) are omitted, 3) will be displayed at the top and bottom of marked content.

In practice, most marking plugins should include inline style information in their HTML markup to control the appearance of the marking. Note that models can currently have a single marking applied.

New Model Types

Examples: plugins/slycat-hello-world, plugins/slycat-linear-regression-demo, plugins/slycat-bookmark-demo

A plugin can add a new type of model to the Slycat server. In this context, a plugin model consists of the following:

- A unique string identifier called the *model type*.
- Code that will be executed on the server when a model is *finished* (i.e. one-time computation to perform after the model's input artifacts have been stored).
- A block of HTML code that will be used as the model's interactive user interface. This block of HTML will be inserted into a larger HTML frame that provides common functionality for manipulating models, and delivered to the end-user's client.

Here is a bare-minimum example of a do-nothing model plugin:

```
def register_slycat_plugin(context):

  def finish(database, model):
    import datetime
    import slycat.web.server.model
    slycat.web.server.model.update(database, model, state="finished", result="succeeded", finished=da

  def html(database, model):
    return "<h1>Hello, World!</h1>"

  context.register_model("my-model", finish, html)
```

Note that finish() simply marks the model as "finished" so clients will know that the model is ready to view, and the html() function returns a familiar message.

When the Slycat server starts, the plugin will be loaded into the server and register a new *my-model* model type. Of course, you'll need some way to actually create an instance of a *my-model* model. The easiest way is to use a script to create *my-model* model instances:

```
import slycat.web.client

parser = slycat.web.client.option_parser()
parser.add_argument("--marking", default="", help="Marking type.  Default: %(default)s")
parser.add_argument("--model-name", default="Hello World Model", help="New model name.  Default: %(de
parser.add_argument("--project-name", default="Hello World Project", help="New project name.  Default
arguments = parser.parse_args()

connection = slycat.web.client.connect(arguments)
```

```
pid = connection.find_or_create_project(arguments.project_name)

mid = connection.post_project_models(pid, "my-model", arguments.model_name, arguments.marking)

connection.post_model_finish(mid)
connection.join_model(mid)

slycat.web.client.log.info("Your new model is located at %s/models/%s" % (arguments.host, mid))
```

In this case the script provides a simple command line interface for specifying the name and marking for the model, along with the name of a new or existing project to contain the new model. Once the connection to the Slycat server has been made and a project identified or created, the new model is created and immediately finished (causing the finish() function to be called). When you view the new model in a web browser, it will display the content returned by the plugin's html() function.

Model Commands

Examples: plugins/slycat-matrix-demo-model

Typically, we assume that a Slycat model is created, artifacts are ingested, one-time server-side computation is performed (using a model plugin's *finish()* function), then a web browser provides interactive visualization of the results (using the output of a model plugin's *html()* function).

However, in some circumstances this may be insufficient - a model may need to provide additional server-side computation to be executed by the client. In this case, a model command plugin is used to register additional server-side *commands* that can be invoked by the client.

Password Check Plugins

Examples: plugins/slycat-identity-password-check.py, plugins/slycat-ldap-password-check.py

Password check plugins are callbacks that are executed whenever the server needs to verify a user's credentials. The password check plugin registers a callback that will be called with an authentication realm, username, and password, and returns a tuple containing *True* if the username and password can be authenticated, and a (possibly empty) list of groups of which the user is a member:

```
def register_slycat_plugin(context):
  def check_password(realm, username, password):
    """Allow any user, so long as their username and password are the same.
    Obviously, this is suitable only for testing."""
    groups = []
    return username and password and username == password, groups

  context.register_password_check("slycat-identity-password-check", check_password)
```

To use a password check plugin, you would have to add it to your server's *config.ini*:

```
[slycat]
password-check: {"plugin": "slycat-identity-password-check"}
```

In a more realistic authentication scenario, you might use the LDAP password check plugin that ships with Slycat to connect to an LDAP server. The following configuration enables the LDAP plugin and configures it to connect to a public test server:

```
[slycat]
password-check: {"plugin": "slycat-ldap-password-check", "kwargs":{"server":"ldaps://ldap.forumsys.co
```

Colophon

The following are needed to generate this documentation:

- Sphinx - documentation builder - http://sphinx-doc.org
- Sphinx readthedocs theme - https://github.com/snide/sphinx_rtd_theme
- napoleon - http://sphinxcontrib-napoleon.readthedocs.org/en/latest/
- httpdomain - http://pythonhosted.org/sphinxcontrib-httpdomain/

Writing the Documentation

The primary sources for this documentation are the docstrings embedded in the Slycat source code itself. When writing docstrings, strictly follow the guidelines at https://github.com/numpy/numpy/blob/master/doc/HOWTO_DOCUMENT.rst.txt

The remainder of the documentation is contained in **.rst* files in the *slycat/docs* directory.

Building the Documentation

To build the documentation, run:

```
$ cd slycat/docs
$ python setup.py
```

Once the documentation is built, you can view it by opening *slycat/docs/_build/html/index.html* in a web browser.

Deploying the Documentation

The slycat documentation is hosted at http://slycat.readthedocs.org and is automatically built and deployed whenever changes are pushed to the Slycat repository at github.com.

Models

From an end-user perspective, the main functions of the Slycat Web Server are creation and storage of models - where a *model* is a typed collection of *artifacts* and metadata. Slycat defines several specific model types, plus an interactive visual user interface for each type. The following documents each model type in detail:

Parameter Image Model

Overview

A Parameter Image model relates a set of images to a set of feature vectors, where we assume that each feature vector is a set of simulation inputs and outputs, and we assume that each image is a simulation output.

Currently, the preferred method to create a new Parameter Image model is to import a remote delimited text file (typically a CSV file) using a web browser. For low-level details on how the input file must be formatted, see `slycat.table.parse()`. In addition to the requirements documented there, the input delimited text file should contain the following:

- Zero to many "input" columns that contain simulation inputs, e.g: the parameters in a parameter study.
- Zero to many "output" columns that contain simulation outputs, e.g: features extracted from the simulations.
- Zero to many "rating" columns that end users will edit to designate regions in the parameter space that should be ignored / explored further in future studies.
- Zero to many "category" columns that contain categorical variables, such as the results of machine learning classification. Category variables may be numeric or string-based, and also may be edited by end users.
- Zero to many "image" columns that contain file URIs pointing to images on a remote host. Each file URI must be of the form *file://hostname/path/to/file* and files must be either PNG or JPEG images. Slycat uses the file URIs to retrieve images via SSH on-demand when end users hover over an observation in the scatterplot, so it is important that the files remain in-place and have appropriate file permissions.
- At least two numeric columns, regardless of type, so the visualization can generate a scatterplot.

Note that there are no constraints on variable names - end users will explicitly identify which columns are "input", "output", "rating", "category", "image", or "none of the above" when the data is imported.

Stored Artifacts

On the server side, a parameter image model includes the following artifacts that are accessible via the *REST API*:

- data-table - `darray` containing the input table data (a 1D darray with one attribute per table column).
- category-columns - JSON array containing a zero-based index for every column in *data-table* that contains categorical data.
- image-columns - JSON array containing a zero-based index for every column in *data-table* that contains images.
- input-columns - JSON array containing a zero-based index for every column in *data-table* that should be considered an input.
- output-columns - JSON array containing a zero-based index for every column in *data-table* that should be considered an output.
- rating-columns - JSON array containing a zero-based index for every column in *data-table* that contains ratings.

REST API

The Slycat server exposes a REST HTTP API that can be used with any programming language or library that supports HTTP requests.

Hyperchunks

To meet a wide variety of needs for incremental and interactive data ingestion and retrieval, Slycat has evolved a complex data storage hierarchy. At the top of the hierarchy are *projects*, which provide administrative and access controls, grouping together related analytical results. *Models* are owned by projects, and represent instances of specific analysis types. Models contain data *artifacts*, whose layout and structure are dictated by the model type. Each artifact in a model is identified by name, which can be an arbitrary string. There are three types of artifacts: *parameters* are JSON objects of arbitrary complexity, intended for storage of small quantities of metadata. *Files* are opaque binary objects that can store large quantities of data, along with an explicitly stored MIME type. The final and most widely used type of artifact is an *arrayset*, which is a one-dimensional array of *darrays*. A darray is a dense, multi-dimensional multi-attribute array, and an arrayset stores n darrays that can be accessed by integer indices in the range $[0, n)$. In-turn, each *attribute* in a darray can be accessed by its integer index, and the elements in each attribute can be identified using a *hyperslice*, which includes a *slice* of element indices for each dimension of the darray.

The bulk of the data in a Slycat model is stored in arraysets, and each time a client reads or writes data to an arrayset, it must specify all of the parameters mentioned above. To make this process simpler, while allowing for a wide variety of data access patterns, we group this information into *hyperchunks*, and have developed the *Hyperchunk Query Language* or *HQL* to serve as a compact specification for a set of hyperchunks. Using HQL, a client can read and write data that spans the arrays and attributes in an arrayset, including computed attributes and arbitrary expressions.

Basic HQL

To begin, the most basic building-block in HQL is a *slice* expression, which follows the same syntactic rules as slicing in the Python language: At its most general a slice takes the form "start:stop:skip", which specifies every $skip$-th element in the half-open range $[start, stop)$. If start is omitted, it defaults to zero. If stop is omitted, it defaults to the length of the available range. If skip is omitted it defaults to one. If start or stop are negative, they represent indices counted backwards from the end of the available range. Start, stop, and skip may be omitted or used in any combination desired:

- "10:20:2" - every other index in the range $[10, 20)$.
- "10:20" - every index in the range $[10, 20)$.
- "10:" - every index from 10 through the end of the available range.
- ":20" - every index in the range $[0, 20)$.
- "..." - every index in the available range.
- ":" - every index in the available range.
- "::" - every index in the available range.
- "::2" - every other index in the available range, starting with zero: $0, 2, 4, ...$
- "1::2" - every other index in the available range, starting with one: $1, 3, 5, ...$
- "10" - index 10.
- "-1" - last index in the available range.
- "-10:" - last ten indices in the available range.

Recall that a slice is a range of indices along a single dimension, while darrays are multi-dimensional. Thus, to retrieve data from a darray with more than one dimension, we need to specify *hyperslice* expressions. To do this, HQL uses slice expressions separated by commas. For example:

- "1" - index 1 of a vector.
- "1,2" - row 1, column 2 of a matrix.
- "3,..." - row 3 of a matrix.
- "...,4" - column 4 of a matrix.
- "50:60,7" - rows $[50, 60)$ from column 7 in a matrix.
- "50:60,7:10" - rows $[50, 60)$ from columns $[7, 10)$ in a matrix.

Additionally, HQL allows us to combine multiple hyperslice expressions, separated by vertical bars. This means we can specify irregular sets of data that can't be specified with the normal slice syntax alone:

- "1|3|4" - indices 1, 3, and 4 of a vector.
- "10:20|77" - indices $[10, 20)$ and 77 from a vector.
- "1,2|33,4" - cells 1,2 and 33,4 from a matrix.

With all this in mind, we can begin putting the pieces together into hyperchunks. A typical HQL expression includes three pieces of information, separated with forward slashes:

```
array expression / attribute expression / hyperslice expression
```

Since an arrayset is a one-dimensional set of darrays, an HQL array expression is a set of one-or-more one-dimensional hyperslice expressions. Similarly, array attributes are accessed by their one-dimensional attribute indices, so basic HQL attribute attribute expressions are also one-dimensional hyperslices. Finally, the subset of each attribute to retrieve is specified using one-or-more multi-dimensional hyperslices, which must match the dimensionality of the underlying array. Here are some simple examples:

- "1/2/10" - array 1, attribute 2, element 10
- "1/2/10:20" - array 1, attribute 2, elements $[10, 20)$.
- "1/2/..." - the entire contents of array 1, attribute 2
- "1/2:4/..." - the entire contents of array 1, attributes 2 and 3
- ".../2/..." - the entire contents of attribute 2 for every array in the arrayset.
- ".../.../..." - everything in the entire arrayset.

The preceding examples assume one-dimensional darrays. Here are some examples of working with matrices:

- "1/2/10:20,30:40" - a ten-by-ten subset of the matrix stored in array 1, attribute 2.
- "1/2/:,3" - column 3 of the matrix stored in array 1, attribute 2.
- "1/2/3,..." - row 3 of the matrix stored in array 1, attribute 2.

And here are examples using multiple hyperslices:

- "1|3|4/.../..." - the entire contents of arrays 1, 3, and 4.
- "1/3|7|8/..." - the entire contents of array 1, attributes 3, 7, and 8.
- "1/2/:,0|:,3|:10" - columns 0, 3, and 10 from the matrix stored in array 1, attribute 2.

Note that when you use HQL to specify the locations for reading and writing data, the data will contain the cartesian product of the specified arrays, attributes, and hyperslices, in array-attribute-hyperslice order. For example, retrieving the hyperchunk "0:2/4:6/10:20|30:40" will return, in-order:

- Array 0, attribute 4, elements 10:20
- Array 0, attribute 4, elements 30:40
- Array 0, attribute 5, elements 10:20
- Array 0, attribute 5, elements 30:40
- Array 1, attribute 4, elements 10:20
- Array 1, attribute 4, elements 30:40
- Array 1, attribute 5, elements 10:20
- Array 1, attribute 5, elements 30:40

All of the APIs that work with hyperchunks take a set of hyperchunks, rather than a single hyperchunk, as their parameter. You can combine multiple hyperchunks by separating them with semicolons:

- "1/2/...;3/4/..." - the entire contents of array 1 attribute 2 and array 3 attribute 4.

Advanced HQL

In addition to slices specifying attribute indices, HQL attribute expressions can include computed expressions that generate attribute data "on the fly". Attribute expressions currently include function execution and a full set of boolean expressions, including set operations:

- "0/1|index(0)/..." - The entire contents of array 0, attribute 1, plus coordinate indices along dimension 0.
- "0/1|rank(a1,"asc")/..." - The entire contents of array 0, attribute 1, plus the rank of each attribute 1 element in ascending order.
- "0/1|a1 > 5/..." - Return the entire contents of array 0, attribute 1, and whether each attribute 1 element is greater than five.
- "0/1|a1 > 5 and a1 < 13/..." - Return the entire contents of array 0, attribute 1, and whether each attribute 1 element is between five and thirteen.
- "0/1|a1 in ["red", "cinnamon"]/..." - Return the entire contents of array 0, attribute 1, and whether each attribute 1 element matches "red" or "cinnamon".

HQL provides a full set of boolean operators: <, >, <=, >=, ==, and *!*=, along with *in* and *not in* for testing set membership, plus *and* and *or* for logical comparisons. You may use parentheses to control the precedence of complex expressions. Of course, you can specify as many computed attribute expressions as you like, using vertical pipes as a separator.

HQL also allows an optional fourth type of expression, an "order" expression, used to sort the data to be returned. The order expression should return an integer rank for each element in the data to be returned and appears between the attribute expression and the hyperslices expression:

- 0/1/order:rank(a1,"asc")/... - The entire contents of array 0, attribute 1, sorted in ascending order.
- 0/1/order:rank(a2, "desc")/... - The entire contents of array 0, attribute 1, sorted in descending order of attribute 2
- 0/1/order:rank(a1,"asc")/0:10 - Array 0, attribute 1, first ten elements in ascending order.

Note that the hyperslice in the final example retrieves the first ten elements of the sorted data, rather than the first ten elements of the attribute.

HQL Context

Depending on the context, not all APIs allow every HQL feature. For example, APIs that write data don't allow computed attribute expressions; some APIs only allow array expressions; others allow only array and attribute expressions. For those situations, you may omit the other parts of the HQL. For example:

- "10:20;35" - arrays $[10, 20)$ plus array 35.
- "3/4;5/7" - array 3 attribute 4, plus array 5 attribute 7.

DELETE Logout

`DELETE /logout`
Deletes a session and its browser cookie.

Sample Request

```
DELETE /logout HTTP/1.1
Host: localhost:8093
Content-Length: 0
Authorization: Basic c2x5Y2F0OnNseWNhdA==
```

```
Accept-Encoding: gzip, deflate, compress
Accept: */*
User-Agent: python-requests/1.2.3 CPython/2.7.5 Linux/2.6.32-358.23.2.el6.x86_64
Cookie: slycatauth=dee8324c69d2424385246edc8d92e996; slycattimeout=timeout
```

Sample Response

```
HTTP/1.1 204 Model deleted.
Cache-Control: no-cache, no-store, must-revalidate
Content-Length: 0
Content-Type: text/html;charset=utf-8
Date: Wed, 16 Mar 2016 16:31:53 GMT
Expires: 0
Pragma: no-cache
Server: CherryPy/4.0.0
Set-Cookie: slycatauth=dee8324c69d2424385246edc8d92e996; expires=Wed, 16 Mar 2016 16:31:53 GMT
slycattimeout=timeout; expires=Wed, 16 Mar 2016 16:31:53 GMT
```

See Also

- *POST /login*

DELETE Model

DELETE /models/ (*mid*)

Deletes a model and all its artifacts.

Parameters

- **mid** (*string*) – Unique model identifier.

Status Codes

- 204 No Content – The model and its artifacts have been deleted.

Sample Request

```
DELETE /models/8b8122539570439cb3703c0f8806158e HTTP/1.1
Host: localhost:8093
Content-Length: 0
Authorization: Basic c2x5Y2F0OnNseWNhdA==
Accept-Encoding: gzip, deflate, compress
Accept: */*
User-Agent: python-requests/1.2.3 CPython/2.7.5 Linux/2.6.32-358.23.2.el6.x86_64
```

Sample Response

```
HTTP/1.1 204 Model deleted.
Date: Mon, 25 Nov 2013 20:36:04 GMT
Content-Type: text/html;charset=utf-8
Server: CherryPy/3.2.2
```

See Also

- *POST /projects/(pid)/models*
- *GET /models/(mid)*

- *PUT /models/(mid)*

DELETE Project

DELETE /projects/ (*pid*)

Deletes a project and all its models.

Parameters

- **pid** – Unique project identifier.

Status Codes

- 204 No Content – The project, its models, and artifacts have been deleted.

Sample Request

```
DELETE /projects/dbaf026f919620acbf2e961ad732433d HTTP/1.1
Host: localhost:8093
Content-Length: 0
Authorization: Basic c2x5Y2F0OnNseWNhdA==
Accept-Encoding: gzip, deflate, compress
Accept: */*
User-Agent: python-requests/1.2.3 CPython/2.7.5 Linux/2.6.32-358.23.2.el6.x86_64
```

Sample Response

```
HTTP/1.1 204 Project deleted.
Date: Mon, 25 Nov 2013 20:35:59 GMT
Content-Type: text/html;charset=utf-8
Server: CherryPy/3.2.2
```

See Also

- *GET /projects/(pid)*
- *PUT /projects/(pid)*

DELETE Project Cache Object

DELETE /projects/ (*pid*) **/cache/**

key Deletes an object from the project cache.

Parameters

- **pid** (*string*) – Unique project identifier.
- **key** (*string*) – Cache object identifier.

Status Codes

- 204 No Content – The cached object has been deleted.

Sample Request

```
DELETE /projects/dbaf026f919620acbf2e961ad732433d/cache/file://example.com/foo/bar/baz.jpg HTTP/
Host: localhost:8093
Content-Length: 0
Authorization: Basic c2x5Y2F0OnNseWNhdA==
```

```
Accept-Encoding: gzip, deflate, compress
Accept: */*
User-Agent: python-requests/1.2.3 CPython/2.7.5 Linux/2.6.32-358.23.2.el6.x86_64
```

Sample Response

```
HTTP/1.1 204 Object deleted.
Date: Mon, 25 Nov 2013 20:35:59 GMT
Content-Type: text/html;charset=utf-8
Server: CherryPy/3.2.2
```

See Also

- *GET /projects/(pid)/cache/(key)*

DELETE Remote

DELETE /remotes/(*sid*)

Deletes a remote session created with *POST /remotes*.

Parameters

- **sid** – Unique session identifier.

Status Codes

- 204 No Content – The remote session has been deleted.

Sample Request

```
DELETE /remotes/dbaf026f919620acbf2e961ad732433d HTTP/1.1
Host: localhost:8093
Content-Length: 0
Authorization: Basic c2x5Y2F0OnNseWNhdA==
Accept-Encoding: gzip, deflate, compress
Accept: */*
User-Agent: python-requests/1.2.3 CPython/2.7.5 Linux/2.6.32-358.23.2.el6.x86_64
```

Sample Response

```
HTTP/1.1 204 Session deleted.
Date: Mon, 25 Nov 2013 20:35:59 GMT
Content-Type: text/html;charset=utf-8
Server: CherryPy/3.2.2
```

See Also

- *POST /remotes*

DELETE Upload

DELETE /uploads/(*uid*)

Delete an upload session used to upload files for storage as model artifacts. This function must be called once the client no longer needs the session, whether the upload(s) have been completed successfully or the client is cancelling an incomplete session.

Parameters

- **uid** (*string*) – Unique upload session identifier.

Status Codes

- 204 No Content – The upload session and any temporary storage have been deleted.
- 409 Conflict – The upload session cannot be deleted, because parsing is in progress. Try again later.

See Also

- *POST /uploads/(uid)/finished*

GET Bookmark

GET /bookmarks/(*bid*)

Retrieves a bookmark - an arbitrary collection of client state.

Parameters

- **bid** (*string*) – Unique bookmark identifier.

Response Headers

- Content-Type – application/json

Sample Request

```
GET /bookmarks/da47466b64216fbb5f782bc2487ceed0 HTTP/1.1
Host: localhost:8092
Authorization: Basic c2x5Y2F0OnNseWNhdA==
Accept-Encoding: gzip, deflate, compress
Accept: */*
User-Agent: python-requests/1.2.0 CPython/2.7.3 Linux/2.6.32-358.6.1.el6.x86_64
```

Sample Response

```
HTTP/1.1 200 OK
Date: Thu, 25 Apr 2013 21:33:51 GMT
Content-Length: 40
Content-Type: application/json
Server: CherryPy/3.2.2

{"selected-column":34,"selected-row":13}
```

See Also

- *POST /projects/(pid)/bookmarks*

GET Home

GET /

Returns a redirect to /projects.

Status Codes

- 303 See Other – Redirect to *GET /projects*.

See Also

- *GET /projects*

GET Model Arrayset Data

GET /models/(*mid*)**/arraysets/***aid***/data** Retrieve data stored in arrayset darray attributes. The caller may request data stored using any combination of arrays, attributes, and hyperslices.

Parameters

- **mid** (*string*) – Unique model identifier.
- **aid** (*string*) – Arrayset artifact id.

Query Parameters

- **hyperchunks** – The request must contain a parameter *hyperchunks* that specifies the arrays, attributes, and hyperslices to be returned, in *Hyperchunks* format.
- **byteorder** – The request may optionally contain a parameter *byteorder* that specifies that the response should be binary data with the given endianness. The byteorder parameter must be either "little" or "big". Note that the byteorder parameter can only be used if every attribute in every hyperchunk is of numeric type. If the byteorder parameter is used, the request must accept application/octet-stream as the result content-type, and the response data will contain contiguous raw data bytes in the given byteorder, in the same order as the requested hyperchunks / hyperslices. For multi-dimension arrays, hyperslice array elements will be in "C" order (the last coordinate varies the fastest).

  If the byteorder parameter isn't specified, the response data will be a JSON-encoded array with length equal to the total number of hyperslices. Each element in this top level array will be an array containing the data for the corresponding hyperslice, in the same order as the requested hyperchunks / hyperslices. For multi-dimension arrays, data for the corresponding hyperslice will be nested further, in "C" order (the last coordinate varies the fastest).

Response Headers

- Content-Type – application/octet-stream or application/json

The following request will return all of the data for array 0, attribute 1 from an arrayset artifact with id "foo":

Sample Request

```
GET /models/6706e78890884845b6c709572a140681/arraysets/foo/data?hyperchunks=0/1/...&byteorder=li
Host: localhost:8093
Authorization: Basic c2x5Y2F0OnNseWNhdA==
Accept-Encoding: gzip, deflate, compress
accept: application/octet-stream
User-Agent: python-requests/1.2.3 CPython/2.7.5 Linux/2.6.32-358.23.2.el6.x86_64
```

Sample Response

```
HTTP/1.1 200 OK
Date: Tue, 26 Nov 2013 16:40:04 GMT
Content-Length: 80
Content-Type: application/octet-stream
```

```
    Server: CherryPy/3.2.2

    ..........................................................................
```

See Also

- *Hyperchunks*
- *GET /models/(mid)/arraysets/(aid)/metadata*
- *PUT /models/(mid)/arraysets/(aid)/data*

GET Model Arrayset Metadata

GET /models/(*mid*)**/arraysets/***aid***/metadata** Used to retrieve metadata and statistics for an arrayset artifact - a collection of dense, multidimensional darray objects. A darray is a dense, multi-dimensional, multi-attribute array, suitable for storage of arbitrarily-large data.

The metadata for a single darray includes the name, type, half-open range of coordinate values, and shape for each dimension in the array, plus the name and type of each attribute.

Statistics can be retrieved for individual darray attributes, and include minimum and maximum values, plus a count of unique values for an attribute. Although statistics are cached, retrieving them may be an extremely expensive operation, since they involve full scans through their respective attributes. Because of this, callers are encouraged to retrieve statistics only when needed.

GET Model Arrayset Metadata can be called in two ways: without any query string, it will return an array containing metadata for every array in the arrayset, without any statistics. Using the *arrays* argument, the caller can request metadata for an explicit list of arrays. The *statistics* argument is used to request statistics for an explicit list of array attributes. The *unique* argument is used to request unique values for an explicit list of array attributes. The three arguments can be combined to retrieve arbitrary combinations of array metadata and attribute statistics in a single request.

Parameters

- **mid** (*string*) – Unique model identifier.
- **aid** (*string*) – Arrayset artifact id.

Query Parameters

- **arrays** – Optional, retrieve array metadata for a set of arrays specified in *Hyperchunks* format. Note that only the array part of the hyperchunk is used in this case - attributes and hyperslices, if provided, are ignored.
- **statistics** – Optional, retrive statistics for a set of array attributes specified in *Hyperchunks* format. Note that only the array and attribute parts of the hyperchunk is used in this case - hyperslices, if provided, are ignored.
- **unique** – Optional, retrieve unique values for a set of array attributes specified in *Hyperchunks* format. Note that you must provide a full hyperchunk with array, attribute, and hyperslice(s), and that the hyperslice(s) refer to ranges of unique values, not ranges of attribute values. So a hyperchunk *0/1/:100* means "return the first 100 unique values in array 0, attribute 1".

Response Headers

- Content-Type – application/json

Simple Request

```
GET /models/e97077e27af141d6a06f17c9eed6c17a/arraysets/canonical-variables/metadata HTTP/1.1
Host: localhost:8092
Authorization: Basic c2x5Y2F0OnNseWNhdA==
Accept-Encoding: gzip, deflate, compress
Accept: application/json
User-Agent: python-requests/1.2.0 CPython/2.7.3 Linux/2.6.32-358.6.2.el6.x86_64
```

Simple Response

```
HTTP/1.1 200 OK
Date: Tue, 11 Jun 2013 19:00:50 GMT
Content-Length: 195
Content-Type: application/json
Server: CherryPy/3.2.2

[
  {
    "index": 0,
    "attributes":
    [
      {"type": "float64", "name": "correlation"}
    ],
    "dimensions":
    [
      {"end": 3, "begin": 0, "type": "int64", "name": "component"},
      {"end": 5, "begin": 0, "type": "int64", "name": "input"}
    ],
    "shape":
    [
      3, 5
    ],
  }
]
```

Complex Request

```
GET /models/e97077e27af141d6a06f17c9eed6c17a/arraysets/foo/metadata?arrays=0%3b1&statistics=0/0%
Host: localhost:8092
Authorization: Basic c2x5Y2F0OnNseWNhdA==
Accept-Encoding: gzip, deflate, compress
Accept: application/json
User-Agent: python-requests/1.2.0 CPython/2.7.3 Linux/2.6.32-358.6.2.el6.x86_64
```

Complex Response

```
HTTP/1.1 200 OK
Date: Tue, 11 Jun 2013 19:00:50 GMT
Content-Length: 195
Content-Type: application/json
Server: CherryPy/3.2.2

{
  "arrays":
  [
    {
      "index": 0,
      "attributes":
      [
```

```
            {"type": "float64", "name": "weight"}
            {"type": "string", "name": "animal"}
          ],
          "dimensions":
          [
            {"end": 10, "begin": 0, "type": "int64", "name": "i"},
          ],
          "shape":
          [
            10,
          ],
        },
        {
          "index": 1,
          "attributes":
          [
            {"type": "float64", "name": "c"}
            {"type": "float64", "name": "d"}
          ],
          "dimensions":
          [
            {"end": 10, "begin": 0, "type": "int64", "name": "i"},
          ],
          "shape":
          [
            10,
          ],
        }
      ],
      "statistics":
      [
        {
          "array": 0,
          "attribute": 0,
          "min": 0.1,
          "max": 1237.3,
          "unique": 3704,
        },
        {
          "array": 0,
          "attribute": 1,
          "min": "aardvark",
          "max": "zebra",
          "unique": 4,
        }
      ]
    }
```


See Also

- *Hyperchunks*
- *GET /models/(mid)/arraysets/(aid)/data*
- *PUT /models/(mid)/arraysets/(aid)/data*

GET Model Command

GET /models/ (*mid*) **/commands/**
type/*command* Execute a custom model command.

Plugins may register custom commands to be executed on the server, using an existing model as context. Custom commands are used to perform computation on the server instead of the client, and would typically use model artifacts as inputs.

Parameters

- **mid** (*string*) – Unique model identifier.
- **type** (*string*) – Unique command category.
- **command** (*string*) – Custom command name.

Additional command-specific arguments may be passed using query strings.

Response Headers

- Content-Type – */*

Sample Request

```
GET /models/e32ef475e084432481655fe41348726b/commands/math-plugin/add HTTP/1.1
Host: localhost:8093
Authorization: Basic c2x5Y2F0OnNseWNhdA==
Accept-Encoding: gzip, deflate, compress
accept: application/json
User-Agent: python-requests/1.2.3 CPython/2.7.5 Linux/2.6.32-358.23.2.el6.x86_64
```

Sample Response

```
HTTP/1.1 200 OK
Date: Mon, 25 Nov 2013 20:36:01 GMT
Content-Length: 542
Content-Type: application/json
Server: CherryPy/3.2.2

{
  "result" : 5
}
```

See Also

- *POST /models/(mid)/commands/(type)/(command)*
- *PUT /models/(mid)/commands/(type)/(command)*

GET Model File

GET /models/ (*mid*) **/files/**
aid Retrieves a file artifact from a model. File artifacts are effectively binary blobs that may contain arbitrary data with an explicit content type.

Parameters

- **mid** (*string*) – Unique model identifier.
- **aid** (*string*) – File artifact id.

Response Headers

- Content-Type – The content type of the file artifact, which could be anything.

GET Model Parameter

GET /models/ (*mid*) **/parameters/**

aid Retrieves a model parameter (name / value pair) artifact. The result is a JSON expression and may be arbitrarily complex.

Parameters

- **mid** (*string*) – Unique model identifier.
- **aid** (*string*) – Parameter artifact id.

Response Headers

- Content-Type – application/json

Sample Request

```
GET /models/1385a75dd2eb4faba884cefdd0b94a56/parameters/baz HTTP/1.1
Host: localhost:8093
Content-Length: 0
Accept-Encoding: gzip, deflate, compress
Accept: */*
User-Agent: python-requests/1.2.3 CPython/2.7.5 Linux/2.6.32-358.23.2.el6.x86_64
Authorization: Basic c2x5Y2F0OnNseWNhdA==
```

Sample Response

```
HTTP/1.1 200 OK
Date: Mon, 25 Nov 2013 20:36:04 GMT
Content-Length: 20
Content-Type: application/json
Server: CherryPy/3.2.2

{
  value : [1, 2, 3],
  input : true
}
```

See Also

- *PUT /models/(mid)/parameters/(aid)*

GET Model Resource

GET /resources/models/ (*mtype*) **/**

resource Returns a custom model resource (stylesheet, font, javascript, etc).

Model plugins may register custom resources for use by the model's user interface. This API is used when the client needs to retrieve those resources.

Parameters

- **mtype** (*string*) – Unique model type code.

- **resource** (*string*) – Custom resource name.

Response Headers

- Content-Type – */*

Sample Request

```
GET /resources/models/calculator/ui.css HTTP/1.1
Host: localhost:8093
Authorization: Basic c2x5Y2F0OnNseWNhdA==
Accept-Encoding: gzip, deflate, compress
accept: application/json
User-Agent: python-requests/1.2.3 CPython/2.7.5 Linux/2.6.32-358.23.2.el6.x86_64
```

Sample Response

```
HTTP/1.1 200 OK
Date: Mon, 25 Nov 2013 20:36:01 GMT
Content-Length: 542
Content-Type: text/css
Server: CherryPy/3.2.2

...
```

See Also

- *GET /models/(mid)*
- *GET /models/(mid)/commands/(type)/(command)*

GET Model Table Chunk

GET /models/ (*mid*) **/tables/** *aid* **/arrays/** *array* **/chunk**

Warning: This request is deprecated. Use *GET /models/(mid)/arraysets/(ai* stead.

Used to retrieve a chunk (subset of rows and columns) from a 1D arrayset array artifact. Data is returned as a JSON array-of-arrays containing column-oriented data, one array for each column specified in the request. Both rows and columns may be specified using arbitrary combinations of half-open ranges and individual indices. The ordering of results (both rows and columns) always matches the order of rows and columns in the request. Out-of-range rows or columns are ignored, in which case the results will still contain in-range data. If the caller specifies a name using the optional "index" query parameter in the request, the response will be adjusted to include an additional index column with the given name and zero-based row indices. The optional "sort" query parameter can be used to return the results in sorted order.

Parameters

- **mid** (*string*) – Unique model identifier.
- **aid** (*string*) – Arrayset artifact name.
- **array** (*int*) – Array index.

Query Parameters

- **rows** – Chunk rows to retrieve.
- **columns** – Chunk columns to retrieve.

- **index** – Optional index column to append to the results.
- **sort** – Response sort order.

Response Headers

- Content-Type – application/json

Sample Request

```
GET /models/6b3c85df433e499e9680a135cabe3ab2/tables/test-array-set/arrays/0/chunk?rows=0,1,2,3,4
Host: localhost:8093
Authorization: Basic c2x5Y2F0OnNseWNhdA==
Accept-Encoding: gzip, deflate, compress
accept: application/json
User-Agent: python-requests/1.2.3 CPython/2.7.5 Linux/2.6.32-358.23.2.el6.x86_64
```

Sample Response

```
HTTP/1.1 200 OK
Date: Tue, 26 Nov 2013 16:40:16 GMT
Content-Length: 138
Content-Type: application/json
Server: CherryPy/3.2.2

{
  "sort": null,
  "column-names": ["int8"],
  "rows": [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
  "data": [ [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] ],
  "columns": [0]
}
```

Complex Request

The following request retrieves rows [0, 10), 15, 16, and 17 and columns [2, 5) and 8:

```
GET /models/(mid)/tables/(aid)/arrays/(array)chunk?rows=0-10,15,16,17&columns=2-5,8
```

See Also

- *GET /models/(mid)/tables/(aid)/arrays/(array)/metadata*
- *GET /models/(mid)/tables/(aid)/arrays/(array)/sorted-indices*
- *GET /models/(mid)/tables/(aid)/arrays/(array)/unsorted-indices*

GET Model Table Metadata

GET /models/(*mid*)**/tables/***aid***/arrays/***array***/metadata**

Warning: This request is deprecated. Use *GET /models/(mid)/arraysets/(* instead.

Used to retrieve metadata from a 1D arrayset array artifact, optimized for use as a table. The metadata for the table describes the number of rows and columns in the table, the name and datatype of each column, and the minimum and maximum values in each column. If the caller specifies a name using the optional "index" query parameter in the request, the response will be adjusted to include an additional index column with the given name and zero-based row indices.

Parameters

- **mid** (*string*) – Unique model identifier.
- **aid** (*string*) – Arrayset artifact id.
- **array** (*int*) – Array index.

Query Parameters

- **index** – Optional index column metadata to be appended to the results.

Response Headers

- Content-Type – application/json

Sample Request

```
GET /models/6b3c85df433e499e9680a135cabe3ab2/tables/test-array-set/arrays/0/metadata HTTP/1.1
Host: localhost:8093
Authorization: Basic c2x5Y2F0OnNseWNhdA==
Accept-Encoding: gzip, deflate, compress
accept: application/json
User-Agent: python-requests/1.2.3 CPython/2.7.5 Linux/2.6.32-358.23.2.el6.x86_64
```

Sample Response

```
HTTP/1.1 200 OK
Date: Tue, 26 Nov 2013 16:40:16 GMT
Content-Length: 395
Content-Type: application/json
Server: CherryPy/3.2.2

{
  "column-types": ["int8", "int16", "int32", "int64", "uint8", "uint16", "uint32", "uint64", "fl
  "column-min": [0, 0, 0, 0, 0, 0, 0, 0, 0.0, 0.0, "0"],
  "column-names": ["int8", "int16", "int32", "int64", "uint8", "uint16", "uint32", "uint64", "fl
  "row-count": 10,
  "column-count": 11,
  "column-max": [9, 9, 9, 9, 9, 9, 9, 9, 9.0, 9.0, "9"]
}
```

See Also

- *GET /models/(mid)/tables/(aid)/arrays/(array)/chunk*
- *GET /models/(mid)/tables/(aid)/arrays/(array)/sorted-indices*
- *GET /models/(mid)/tables/(aid)/arrays/(array)/unsorted-indices*

GET Model Table Sorted Indices

GET /models/(*mid*)**/tables/***aid*/**arrays/***array*/**sorted-indices**

Warning: This request is deprecated. Use *GET /models/(mid)/arra*
stead.

Given a collection of row indices and a specific sort order, return the corresponding sorted row indices.

Parameters

- **mid** (*string*) – Unique model identifier.
- **aid** (*string*) – Arrayset artifact id.
- **array** (*int*) – Array index.

Query Parameters

- **rows** – Row indices to be sorted.
- **index** – Optional index column that can be used for sorting.
- **sort** – Sort order.
- **byteorder** – Optionally return the results as binary data.

Response Headers

- Content-Type – application/json, application/octet-stream

See Also

- *GET /models/(mid)/tables/(aid)/arrays/(array)/chunk*
- *GET /models/(mid)/tables/(aid)/arrays/(array)/metadata*
- *GET /models/(mid)/tables/(aid)/arrays/(array)/unsorted-indices*

GET Model Table Unsorted Indices

GET /models/ (*mid*) **/tables/**

*aid***/arrays/***array***/unsorted-indices**

Warning: This request is deprecated. Use *GET /models/(mid)/ar* stead.

Given a collection of sorted row indices and a specific sort order, return the corresponding unsorted row indices.

Parameters

- **mid** (*string*) – Unique model identifier.
- **aid** (*string*) – Arrayset artifact id.
- **array** (*int*) – Array index.

Query Parameters

- **rows** – Row indices to be sorted.
- **index** – Optional index column that can be used for sorting.
- **sort** – Sort order.
- **byteorder** – Optionally return the results as binary data.

Response Headers

- Content-Type – application/json, application/octet-stream

See Also

- *GET /models/(mid)/tables/(aid)/arrays/(array)/chunk*
- *GET /models/(mid)/tables/(aid)/arrays/(array)/metadata*
- *GET /models/(mid)/tables/(aid)/arrays/(array)/sorted-indices*

GET Model

GET /models/ (*mid*)
Returns a model.

Parameters

- **mid** (*string*) – Unique model identifier.

Response Headers

- Content-Type – text/html, application/json

Sample Request

```
GET /models/e32ef475e084432481655fe41348726b HTTP/1.1
Host: localhost:8093
Authorization: Basic c2x5Y2F0OnNseWNhdA==
Accept-Encoding: gzip, deflate, compress
accept: application/json
User-Agent: python-requests/1.2.3 CPython/2.7.5 Linux/2.6.32-358.23.2.el6.x86_64
```

Sample Response

```
HTTP/1.1 200 OK
Date: Mon, 25 Nov 2013 20:36:01 GMT
Content-Length: 542
Content-Type: application/json
Server: CherryPy/3.2.2

{
  "description": "",
  "creator": "slycat",
  "artifact-types": {},
  "_rev": "2-80a35c0e45a33d6654fd13a90f17624a",
  "model-type": "generic",
  "finished": null,
  "result": null,
  "message": null,
  "marking": "",
  "name": "test-model",
  "created": "2013-11-25T20:36:01.064901",
  "input-artifacts": [],
  "uri": "http://localhost:8093/models/e32ef475e084432481655fe41348726b",
  "project": "dbaf026f919620acbf2e961ad7325359",
  "started": "2013-11-25T20:36:01.218447",
  "state": "running",
  "progress": 0.0,
  "_id": "e32ef475e084432481655fe41348726b",
  "type": "model"
}
```

See Also

- *POST /projects/(pid)/models*
- *PUT /models/(mid)*
- *DELETE /models/(mid)*

GET Project Cache Object

GET /projects/(*pid*)**/cache/**
key Retrieves an object from a project's cache. Cache objects are opaque binary blobs that may contain arbitrary data, plus an explicit content type.

Parameters

- **pid** – Unique project identifier.
- **key** (*string*) – Cache object identifier.

Status Codes

- 200 OK – The requested file is returned in the body of the response.
- 404 Not Found – The requested object isn't in the cache.

Response Headers

- Content-Type – The content type of the cached object, which could be any valid MIME type.

See Also

- *DELETE /projects/(pid)/cache/(key)*
- *GET /remotes/(sid)/file(path)*
- *GET /remotes/(sid)/image(path)*
- *GET /remotes/(sid)/videos/(vsid)*

GET Project Models

GET /projects/(*pid*)**/models**
Returns a list of project models.

Parameters

- **pid** (*string*) – Unique project identifier.

Response Headers

- Content-Type – application/json

GET Project

GET /projects/ (*pid*)
Returns a project.

Parameters

- **pid** (*string*) – Unique project identifier.

Response Headers

- Content-Type – text/html, application/json

Sample Request

```
GET /projects/dbaf026f919620acbf2e961ad73243c5 HTTP/1.1
Host: localhost:8093
Authorization: Basic c2x5Y2F0OnNseWNhdA==
Accept-Encoding: gzip, deflate, compress
accept: application/json
User-Agent: python-requests/1.2.3 CPython/2.7.5 Linux/2.6.32-358.23.2.el6.x86_64
```

Sample Response

```
HTTP/1.1 200 OK
Date: Mon, 25 Nov 2013 20:35:59 GMT
Content-Length: 308
Content-Type: application/json
Server: CherryPy/3.2.2

{
  "description": "My test project.",
  "created": "2013-11-25T20:35:59.555004",
  "_rev": "1-5af189cbba8ad4e0e200b2593f2594a2",
  "creator": "slycat",
  "acl": {"administrators": [{"user": "slycat"}], "writers": [], "readers": []},
  "_id": "dbaf026f919620acbf2e961ad73243c5",
  "type": "project",
  "name": "test-project"
}
```

See Also

- *PUT /projects/(pid)*
- *DELETE /projects/(pid)*

GET Projects

GET /projects
Returns the list of available projects. The HTML representation provides the main Slycat user interface.

Request Headers

- Accept – text/html or application/json

Sample Request

```
GET /projects HTTP/1.1
Host: localhost:8093
Authorization: Basic c2x5Y2F0OnNseWNhdA==
Accept-Encoding: gzip, deflate, compress
accept: application/json
User-Agent: python-requests/1.2.3 CPython/2.7.5 Linux/2.6.32-358.23.2.el6.x86_64
```

Sample Response

```
HTTP/1.1 200 OK
Date: Mon, 25 Nov 2013 20:35:59 GMT
Content-Length: 570
Content-Type: application/json
Server: CherryPy/3.2.2

[
  {
    "description": "",
    "created": "2013-11-25T20:35:58.955499",
    "_rev": "1-a4332c471d456db74398dd8ac20f8a61",
    "creator": "slycat",
    "acl": {"administrators": [{"user": "slycat"}], "writers": [], "readers": []},
    "_id": "dbaf026f919620acbf2e961ad732433d",
    "type": "project",
    "name": "bar"
  },
  {
    "description": "",
    "created": "2013-11-25T20:35:58.886682",
    "_rev": "1-99142f0b92a93266b9930914808fb286",
    "creator": "slycat",
    "acl": {"administrators": [{"user": "slycat"}], "writers": [], "readers": []},
    "_id": "dbaf026f919620acbf2e961ad7324011",
    "type": "project",
    "name": "foo"
  }
]
```

See Also

- *POST /projects*

GET Remote File

GET /remotes/(*sid*)**/file**

path Uses an existing remote session to retrieve a remote file. The remote session must have been created using *POST /remotes*. Use *POST /remotes/(sid)/browse(path)* to lookup remote file paths. The returned file may be optionally cached on the server and retrieved using *GET /projects/(pid)/cache/(key)*.

Parameters

- **sid** (*string*) – Unique session identifier returned from *POST /remotes*.
- **path** (*string*) – Remote filesystem path (must be absolute).

Query Parameters

- **cache** – Optional cache identifier. Set to *project* to store the retrieved file in a project cache.
- **project** – Project identifier. Required when *cache* is set to *project*.
- **key** – Cached object key. Must be specified when *cache* is set to *project*.

Status Codes

- 200 OK – The requested file is returned in the body of the response.
- 404 Not Found – The session doesn't exist or has timed-out.
- 400 Bad Request – "Can't read directory" The remote path is a directory instead of a file.
- 400 Bad Request – "File not found" The remote path doesn't exist.
- 400 Bad Request – "Access denied" The session user doesn't have permissions to access the file.

Response Headers

- Content-Type – The MIME type of the response is automatically determined using the requested filename.
- *X-Slycat-Message* – For errors, contains a human-readable description of the problem.
- *X-Slycat-Hint* – For errors, contains an optional description of how to fix the problem.

Sample Request

```
GET /remotes/505d0e463d5ed4a32bb6b0fe9a000d36/file/home/fred/checklist.txt
```

See Also

- *GET /remotes/(sid)/image(path)*
- *GET /remotes/(sid)/videos/(vsid)*

GET Remote Image

GET /remotes/(*sid*)**/image**

path Uses an existing remote session to retrieve a remote image. The remote session must have been created using *POST /remotes*, and the session must have a running agent. Use *POST /remotes/(sid)/browse(path)* to lookup remote file paths. The returned file may be optionally cached on the server and retrieved using *GET /projects/(pid)/cache/(key)*.

The caller *may* optionally choose to resize the image and / or convert it to another file type. Note that this can reduce performance significantly as the remote must then decompress, resample, and recompress the image before sending it to the client. Testing should be performed to verify that the bandwidth reduction of a smaller image is worth the increased latency.

Parameters

- **sid** (*string*) – Unique session identifier returned from *POST /remotes*.
- **path** (*string*) – Remote filesystem absolute path to be retrieved.

Query Parameters

- **content-type** (*string*) – Optional image content type to return. Currently limited to *image/jpeg* or *image/png*. If the requested content type doesn't match the content type of the remote image, it will be converted.

- **max-size** (*int*) – Optional maximum image size in pixels along either dimension. Images larger than this size will be resized to fit while maintaining their aspect ratio.
- **max-width** (*int*) – Optional maximum image width. Wider images will be resized to fit while maintaining their aspect ratio.
- **max-height** (*int*) – Optional maximum image height. Taller images will be resized to fit while maintaining their aspect ratio.
- **cache** – Optional cache identifier. Set to *project* to store the retrieved image in a project cache.
- **project** – Project identifier. Required when *cache* is set to *project*.
- **key** – Cached object key. Must be specified when *cache* is set to *project*.

Status Codes

- 200 OK – The requested file is returned in the body of the response.
- 400 Bad Request – "Access denied" The session user doesn't have permissions to access the file.
- 400 Bad Request – "Agent required" This call requires a remote agent, but the current session isn't running an agent.
- 400 Bad Request – "Can't read directory" The remote path is a directory instead of a file.
- 400 Bad Request – "File not found" The remote path doesn't exist.
- 404 Not Found – The session doesn't exist or has timed-out.

Response Headers

- Content-Type – image/jpeg or image/png, depending on the type of the remote file and optional conversion.
- *X-Slycat-Message* – For errors, contains a human-readable description of the problem.
- *X-Slycat-Hint* – For errors, contains an optional description of how to fix the problem.

Sample Request

```
GET /remotes/505d0e463d5ed4a32bb6b0fe9a000d36/image/home/fred/avatar.png?content-type=image/jpeg
```

See Also

- *GET /remotes/(sid)/file(path)*
- *GET /remotes/(sid)/videos/(vsid)*

GET Remote Video Status

GET /remotes/ (*sid*) **/videos/**
*vsid***/status** Uses an existing remote video session to retrieve the status of a video creation command. The remote session must have been created successfully using *POST /remotes* and video creation must have been started using *POST /remotes/(sid)/videos*.

Parameters

- **sid** (*string*) – Unique remote session identifier.
- **vsid** (*string*) – Unique video creation session identifier.

Status Codes

- 200 OK – The status is contained in the response body.
- 400 Bad Request – "Agent required" This call requires a remote agent, but the current session isn't running an agent.
- 404 Not Found – If the session doesn't exist or has timed out.

Response Headers

- Content-Type – application/json
- *X-Slycat-Message* – For errors, contains a human-readable description of the problem.
- *X-Slycat-Hint* – For errors, contains an optional description of how to fix the problem.

Response JSON Object

- **ok** (*boolean*) – Set to *true* if the video creation process is working, *false* if it has failed.
- **ready** (*boolean*) – Optional. Set to *true* if the video creation process has completed successfully and the video file is ready for retrieval.
- **message** (*string*) – Human-readable message describing the current video creation state or error.
- **returncode** (*number*) – Optional exit code from the video creation process. Note: this is for debugging only, could be removed in the future, and should not be displayed to end-users.
- **stderr** (*string*) – Optional capture of stderr from the video creation process. Note: this is for debugging only, could be removed in the future, and should not be displayed to end-users.

Sample Request

```
GET /remotes/505d0e463d5ed4a32bb6b0fe9a000d36/videos/431d0e463d5ed4a32bb6b0fe9a000a37/status
```

See Also

- *POST /remotes/(sid)/videos*
- *GET /remotes/(sid)/videos/(vsid)*

GET Remote Video

GET /remotes/(*sid*)**/videos/**

vsid Uses an existing remote session to retrieve a remote video. The session must have been created successfully using *POST /remotes* and video creation must have been started using *POST /remotes/(sid)/videos*. The caller should not attempt retrieving a video until a call to *GET /remotes/(sid)/videos/(vsid)/status* indicates that video creation is complete. The returned file may be optionally cached on the server and retrieved using *GET /projects/(pid)/cache/(key)*.

Parameters

- **sid** (*string*) – Unique remote session identifier.
- **vsid** (*string*) – Unique video creation session identifier.

Query Parameters

- **cache** – Optional cache identifier. Set to *project* to store the retrieved video in a project cache.
- **project** – Project identifier. Required when *cache* is set to *project*.
- **key** – Cached object key. Must be specified when *cache* is set to *project*.

Status Codes

- 200 OK – The video has been returned in the response body.
- 206 Partial Content – A portion of the video has been returned in the response body.
- 400 Bad Request – "Agent required" This call requires a remote agent, but the current session isn't running an agent.
- 404 Not Found – The session doesn't exist or has timed-out.

Response Headers

- Content-Type – video/mp4 or video/webm, depending on the original *POST /remotes/(sid)/videos* request.
- *X-Slycat-Message* – For errors, contains a human-readable description of the problem.
- *X-Slycat-Hint* – For errors, contains an optional description of how to fix the problem.

Sample Request

```
GET /remotes/505d0e463d5ed4a32bb6b0fe9a000d36/videos/431d0e463d5ed4a32bb6b0fe9a000a37
```

See Also

- *GET /remotes/(sid)/file(path)*
- *GET /remotes/(sid)/image(path)*

GET User

GET /users/ (*uid*)

Retrieve directory information for a given user.

Parameters

- **uid** (*string*) – User id to retrieve. As a special case, callers may pass - as the uid to request information about the currently-logged-in user.

Status Codes

- 200 OK – User metadata retrieved.
- 404 Not Found – Unknown user.

Response Headers

- Content-Type – application/json

Response JSON Object

- **uid** (*string*) – User id of the requested user.
- **email** (*string*) – Email address of the requested user.
- **name** (*string*) – Full name of the requested user.

Sample Response

```
{
  "uid": "frfreder",
  "email": "fred@example.com",
  "name": "Fred R. Frederickson",
}
```

POST Model Arrayset Data

GET /models/(*mid*)**/arraysets/**
*aid***/data** Retrieve data stored in arrayset darray attributes. The caller may request data stored using any combination of arrays, attributes, and hyperslices.

Parameters

- **mid** (*string*) – Unique model identifier.
- **aid** (*string*) – Arrayset artifact id.

Request Headers

- Content-Type – application/json

Request JSON Object

- **hyperchunks** (*string*) – The request must contain a parameter *hyperchunks* that specifies the arrays, attributes, and hyperslices to be returned, in *Hyperchunks* format.
- **byteorder** (*string*) – The request may optionally contain a parameter *byteorder* that specifies that the response should be binary data with the given endianness. The byteorder parameter must be either "little" or "big". Note that the byteorder parameter can only be used if every attribute in every hyperchunk is of numeric type. If the byteorder parameter is used, the request must accept application/octet-stream as the result content-type, and the response data will contain contiguous raw data bytes in the given byteorder, in the same order as the requested hyperchunks / hyperslices. For multi-dimension arrays, hyperslice array elements will be in "C" order (the last coordinate varies the fastest).

  If the byteorder parameter isn't specified, the response data will be a JSON-encoded array with length equal to the total number of hyperslices. Each element in this top level array will be an array containing the data for the corresponding hyperslice, in the same order as the requested hyperchunks / hyperslices. For multi-dimension arrays, data for the corresponding hyperslice will be nested further, in "C" order (the last

Response Headers

- Content-Type – application/json

The following request will return all of the data for array 0, attribute 1 from an arrayset artifact with id "foo":

Sample Request

```
POST /models/6706e78890884845b6c709572a140681/arraysets/foo/dataH TTP/1.1
Host: localhost:8093
Authorization: Basic c2x5Y2F0OnNseWNhdA==
Accept-Encoding: gzip, deflate, compress
accept: application/octet-stream
User-Agent: python-requests/1.2.3 CPython/2.7.5 Linux/2.6.32-358.23.2.el6.x86_64

{
    hyperchunks: "0/1/...,"
```

```
        byteorder: "little"
    }
```

Sample Response

```
    HTTP/1.1 200 OK
    Date: Tue, 26 Nov 2013 16:40:04 GMT
    Content-Length: 80
    Content-Type: application/octet-stream
    Server: CherryPy/3.2.2

    ................................................................................
```

See Also

- *Hyperchunks*
- *GET /models/(mid)/arraysets/(aid)/metadata*
- *PUT /models/(mid)/arraysets/(aid)/data*

POST Agent Function

POST /remotes/run-agent-function

Uses an existing remote sessions to submit a predefined Slycat function to a cluster running SLURM as a job. The session must have been created successfully using *POST /remotes*

Request JSON Object

- **sid** (*string*) – Unique remote session identifier.
- **wckey** (*string*) – Workload characterization key.
- **nnodes** (*int*) – Number of nodes requested for the job.
- **partition** (*string*) – Name of the partition where the job will be run.
- **ntasks_per_node** (*int*) – Number of tasks to run on a node.
- **ntasks** (*int*) – Number of tasks allocated for the job.
- **ncpu_per_task** (*int*) – Number of CPUs per task requested for the job.
- **time_hours** (*int*) – Number of hours requested for the job.
- **time_minutes** (*int*) – Number of minutes requested for the job.
- **time_seconds** (*int*) – Number of seconds requested for the job.
- **fn** (*string*) – Name of the Slycat predefined function.

Status Codes

- 200 OK – The response contains the command, its output and possible errors.
- 400 Bad Request – The request failed due to invalid parameters or a Slycat agent issue.
- 500 Internal Server Error – The request failed due to no Slycat agent present and configured on the remote system.

Response Headers

- Content-Type – application/json

- *X-Slycat-Message* – For errors, contains a human-readable description of the problem.

Response JSON Object

- **jid** (*int*) – Job ID.
- **errors** (*string*) – Error information, if any.

Sample Request

```
POST /remotes/run-agent-function

{
  sid: "505d0e463d5ed4a32bb6b0fe9a000d36",
  wckey: "user_00001",
  nnodes: 1,
  partition: "partition_name",
  ntasks_per_node: 1,
  ntasks: 1,
  ncpu_per_task: 4,
  time_hours: 0,
  time_minutes: 5,
  time_seconds: 0,
  fn: "slycat_predefined_function"
}
```

Sample Response

```
{
  "jid": 123456,
  "errors": ""
}
```

See Also

- *POST /remotes/cancel-job*
- *POST /remotes/checkjob*
- *POST /remotes/get-job-output*
- *POST /remotes/launch*
- *POST /remotes/submit-batch*

POST Cancel Job

POST /remotes/cancel-job

Uses an existing remote session to cancel a job submitted via the SLURM interface on a remote cluster. The session must have been created successfully using *POST /remotes*.

Request JSON Object

- **sid** (*string*) – Unique remote session identifier.
- **jid** (*string*) – Job ID.

Status Codes

- 200 OK – The response contains the command, its output and possible errors.
- 400 Bad Request – The request failed due to invalid parameters or a Slycat agent issue.

- 500 Internal Server Error – The request failed due to no Slycat agent present and configured on the remote system.

Response Headers

- Content-Type – application/json
- *X-Slycat-Message* – For errors, contains a human-readable description of the problem.

Response JSON Object

- **jid** (*int*) – Job ID.
- **output** (*string*) – Output information, if any.
- **errors** (*string*) – Error information, if any.

Sample Request

```
POST /remotes/checkjob

{
  sid: "505d0e463d5ed4a32bb6b0fe9a000d36",
  jid: 123456
}
```

Sample Response

```
{
  "jid": 123456,
  "output": "",
  "errors": ""
}
```

See Also

- *POST /remotes/checkjob*
- *POST /remotes/get-job-output*
- *POST /remotes/launch*
- *POST /remotes/run-agent-function*
- *POST /remotes/submit-batch*

POST Check Job

POST /remotes/checkjob

Uses an existing remote session to query the status of submitted SLURM job on a cluster. The session must have been created successfully using *POST /remotes*.

Request JSON Object

- **sid** (*string*) – Unique remote session identifier.
- **jid** (*string*) – Job ID.

Status Codes

- 200 OK – The response contains the command, its output and possible errors.
- 400 Bad Request – The request failed due to invalid parameters or a Slycat agent issue.

- 500 Internal Server Error – The request failed due to no Slycat agent present and configured on the remote system.

Response Headers

- Content-Type – application/json
- *X-Slycat-Message* – For errors, contains a human-readable description of the problem.

Response JSON Object

- **jid** (*int*) – Job ID.
- **status** (*string*) – Status for the queried job.
- **errors** (*string*) – Error information, if any.

The following status are reported: PENDING, RUNNING, COMPLETING, COMPLETED and CANCELLED.

Sample Request

```
POST /remotes/checkjob

{
  sid: "505d0e463d5ed4a32bb6b0fe9a000d36",
  jid: 123456
}
```

Sample Response

```
{
  "jid": 123456,
  "status": "PENDING",
  "errors": ""
}
```

See Also

- *POST /remotes/cancel-job*
- *POST /remotes/get-job-output*
- *POST /remotes/launch*
- *POST /remotes/run-agent-function*
- *POST /remotes/submit-batch*

POST Events

POST /events/ (*event*)

Insert a client-side event into the server log. Clients should use this API to record any user interaction events that may be of later interest for subsequent analytics. The structure of the request URI following the initial "/events/" is left to the client. Note that the request body is ignored.

Parameters

- **event** (*string*) – Path-like user interaction to be logged.

Sample Requests

The following is a hypothetical stream of events logged as a user interacts with a model. The structure and meaning of the events are completely client-driven.

```
POST /events/models/0bfb94cba9654faf904b6fe8b2aab603/select/component/3
POST /events/models/0bfb94cba9654faf904b6fe8b2aab603/select/variable/1
POST /events/models/0bfb94cba9654faf904b6fe8b2aab603/sort/variable/2
POST /events/models/0bfb94cba9654faf904b6fe8b2aab603/pan?dx=34&dy=2
POST /events/models/0bfb94cba9654faf904b6fe8b2aab603/zoom?factor=2.3
```

POST Get Job Output

POST /remotes/get-job-output

Uses an existing remote sessions to fetch the content of a SLURM output file on a cluster. The session must have been created successfully using *POST /remotes*

Request JSON Object

- **sid** (*string*) – Unique remote session identifier.
- **jid** (*string*) – Job ID.
- **path** (*string*) – Path of the SLURM output file, if different from the login node.

Status Codes

- 200 OK – The response contains the command, its output and possible errors.
- 400 Bad Request – The request failed due to invalid parameters or a Slycat agent issue.
- 500 Internal Server Error – The request failed due to no Slycat agent present and configured on the remote system.

Response Headers

- Content-Type – application/json
- *X-Slycat-Message* – For errors, contains a human-readable description of the problem.

Response JSON Object

- **jid** (*int*) – Job ID.
- **output** (*string*) – Content of the SLURM output file.
- **errors** (*string*) – Error information, if any.

Note that the *path* parameter is optional and the request will look for the output file within the home directory of a login node. Also, the content of the output file could potentially contain many lines of text.

Sample Request

```
POST /remotes/get-job-output

{
  sid: "505d0e463d5ed4a32bb6b0fe9a000d36",
  jid: 123456
}
```

Sample Response

```
{
  "jid": 123456,
  "output": "test",
  "errors": ""
}
```

See Also

- *POST /remotes/cancel-job*
- *POST /remotes/checkjob*
- *POST /remotes/launch*
- *POST /remotes/run-agent-function*
- *POST /remotes/submit-batch*

POST Login

POST /login

Creates a session and then returns the session cookie

Request Headers

- Content-Type – application/json

Request JSON Object

- **bit encoded string name** (*64*) – username
- **bit encoded string password** (*64*) – password
- **url** (*object*) – origin url from which you came

Response Headers

- Content-Type – application/json

Response JSON Object

- **success** (*boolean*) – boolean representing successful login
- **target** (*string*) – original url user tried to access (for a redirect after login)

Sample Request

```
POST /login HTTP/1.1
Host: localhost:8092
Content-Length: 45
Accept-Encoding: gzip, deflate, compress
Accept: */*
User-Agent: python-requests/1.2.0 CPython/2.7.3 Linux/2.6.32-358.2.1.el6.x86_64
content-type: application/json

{
"user_name":"64 bit encoded slycat(c2x5Y2F0)",
"password":"64 bit encoded slycat(c2x5Y2F0)",
"location":{
    "href":"https://192.168.99.100/login/slycat-login.html",
    "origin":"https://192.168.99.100",
```

```
        "protocol":"https:",
        "host":"192.168.99.100",
        "hostname":"192.168.99.100",
        "port":"",
        "pathname":"/login/slycat-login.html",
        "search":"",
        "hash":""
        }
    }
```

Sample Response

```
    HTTP/1.1 201 Project created.
    Date: Thu, 11 Apr 2013 21:30:16 GMT
    Content-Length: 42
    Content-Type: application/json
    Set-Cookie:"slycatauth=xyz;httponly;Max-Age=60000;Path=/;secure;slycattimeout=timeout;Max-Age=60
    Location: http://localhost:8092/projects/505d0e463d5ed4a32bb6b0fe9a000d36
    Server: CherryPy/3.2.2

    {"target": "https://192.168.99.100/projects","success":true}
```

See Also

- *DELETE /logout*

POST Model Command

POST /models/ (*mid*) **/commands/**

type/*command* Execute a custom model command.

Plugins may register custom commands to be executed on the server, using an existing model as context. Custom commands are used to perform computation on the server instead of the client, and would typically use model artifacts as inputs.

Parameters

- **mid** (*string*) – Unique model identifier.
- **type** (*string*) – Unique command category.
- **command** (*string*) – Custom command name.

Additional command-specific arguments may be passed using query strings.

Response Headers

- Content-Type – */*

Sample Request

```
    POST /models/e32ef475e084432481655fe41348726b/commands/math-plugin/add HTTP/1.1
    Host: localhost:8093
    Authorization: Basic c2x5Y2F0OnNseWNhdA==
    Accept-Encoding: gzip, deflate, compress
    accept: application/json
    User-Agent: python-requests/1.2.3 CPython/2.7.5 Linux/2.6.32-358.23.2.el6.x86_64
```

Sample Response

```
HTTP/1.1 200 OK
Date: Mon, 25 Nov 2013 20:36:01 GMT
Content-Length: 542
Content-Type: application/json
Server: CherryPy/3.2.2

{
  "result" : 5
}
```

See Also

- *GET /models/(mid)/commands/(type)/(command)*
- *PUT /models/(mid)/commands/(type)/(command)*

POST Model Finish

POST /models/(*mid*)**/finish**

Finish (internally compute) a waiting model. The model must be in the waiting state.

Parameters

- **mid** – Unique model identifier.

See Also

- *GET /models/(mid)*
- *PUT /models/(mid)*
- *DELETE /models/(mid)*

POST Project Bookmark

POST /projects/(*pid*)**/bookmarks**

Stores a bookmark - an arbitrary JSON object that captures client-side state - returning a unique identifier that can be used to retrieve that state.

Note that the bookmark contents are canonicalized and hashed to produce the returned identifier, so all bookmarks containing the same state automatically share the same id.

Typically, a client would store a bookmark anytime the client state changes as a user is interacting with a model, e.g. making selections, sorting, choosing color maps, etc. The client can then use the returned bookmark id to restore that state when the user returns to a given model. We strongly recommend that web browsers incorporate the returned bookmark id into the browser's URL, so the resulting visualization can be saved as a browser bookmark, emailed to a colleague, etc.

Parameters

- **pid** (*string*) – Unique project identifier.

Request Headers

- Content-Type – application/json

Response Headers

- Content-Type – application/json

Response JSON Object

- **id** (*string*) – Unique bookmark identifier.

Sample Request

```
POST /projects/957cb70e7a31529d266fb0c110000f27/bookmarks HTTP/1.1
Host: localhost:8092
Content-Length: 43
Accept-Encoding: gzip, deflate, compress
Accept: */*
User-Agent: python-requests/1.2.0 CPython/2.7.3 Linux/2.6.32-358.6.1.el6.x86_64
content-type: application/json
Authorization: Basic c2x5Y2F0OnNseWNhdA==

{"selected-row": 13, "selected-column": 34}
```

Sample Response

```
HTTP/1.1 201 Bookmark stored.
Date: Thu, 25 Apr 2013 21:33:44 GMT
Content-Length: 42
Content-Type: application/json
Location: http://localhost:8092/bookmarks/da47466b64216fbb5f782bc2487ceed0
Server: CherryPy/3.2.2

{"id": "da47466b64216fbb5f782bc2487ceed0"}
```

See Also

- *GET /bookmarks/(bid)*

POST Project Models

POST /projects/ (*pid*) **/models**

Adds a new, empty model to a project.

Parameters

- **pid** (*string*) – Unique project identifier.

Request Headers

- Content-Type – application/json

Request JSON Object

- **model-type** (*string*) – Model type identifier.
- **name** (*string*) – Model name.
- **description** (*string*) – Model description.
- **marking** (*string*) – Model marking identifier.

Response Headers

- Content-Type – application/json

Response JSON Object

- **id** (*string*) – Unique model identifier.

Sample Request

```
POST /projects/505d0e463d5ed4a32bb6b0fe9a000d36/models HTTP/1.1
Host: localhost:8092
Content-Length: 73
Accept-Encoding: gzip, deflate, compress
Accept: */*
User-Agent: python-requests/1.2.0 CPython/2.7.3 Linux/2.6.32-358.2.1.el6.x86_64
content-type: application/json
Authorization: Basic c2x5Y2F0OnNseWNhdA==

{"model-type": "generic", "description": "", "name": "Model", "marking": ""}
```

Sample Response

```
HTTP/1.1 202 Model scheduled for creation.
Date: Thu, 11 Apr 2013 21:30:16 GMT
Content-Length: 85
Content-Type: application/json
Server: CherryPy/3.2.2

{"id": "7f4b92c00af7465eb594a2ca77d0df91"}
```

See Also

- *GET /models/(mid)*
- *PUT /models/(mid)*
- *DELETE /models/(mid)*

POST Projects

POST /projects

Creates a new project. The caller *must* supply a human-readable project name. The caller *may* supply a human readable project description and/or access control list (ACL). The results will return the ID of the newly-created project.

If an ACL is not specified, the project will have a default ACL with the project administrator set to the user creating the project, and no project readers or writers.

Request Headers

- Content-Type – application/json

Request JSON Object

- **name** (*string*) – New project name.
- **description** (*string*) – New project description.
- **acl** (*object*) – New project access control list.

Response Headers

- Content-Type – application/json

Response JSON Object

- **id** (*string*) – Unique project identifier.

Sample Request

```
POST /projects HTTP/1.1
Host: localhost:8092
Content-Length: 45
Accept-Encoding: gzip, deflate, compress
Accept: */*
User-Agent: python-requests/1.2.0 CPython/2.7.3 Linux/2.6.32-358.2.1.el6.x86_64
content-type: application/json
Authorization: Basic c2x5Y2F0OnNseWNhdA==

{"name": "CCA Model Test", "description": ""}
```

Sample Response

```
HTTP/1.1 201 Project created.
Date: Thu, 11 Apr 2013 21:30:16 GMT
Content-Length: 42
Content-Type: application/json
Location: http://localhost:8092/projects/505d0e463d5ed4a32bb6b0fe9a000d36
Server: CherryPy/3.2.2

{"id": "505d0e463d5ed4a32bb6b0fe9a000d36"}
```

See Also

- *GET /projects*

POST Remote Browse

POST /remotes/(*sid*)**/browse**

path Uses an existing remote session to retrieve remote filesystem information. The session must have been created successfully using *POST /remotes*. The caller *may* supply additional parameters to filter directories and files in the results, based on regular expressions.

Parameters

- **sid** (*string*) – Unique remote session identifier.
- **path** (*string*) – Remote filesystem path (must be absolute).

Request Headers

- Content-Type – application/json

Request JSON Object

- **directory-reject** (*string*) – Optional regular expression for filtering directories.
- **directory-allow** (*string*) – Optional regular expression for retaining directories.
- **file-reject** (*string*) – Optional regular expression for filtering files.
- **file-allow** (*string*) – Optional regular expression for allowing files.

Status Codes

- 200 OK – The response contains the requested browsing information.

- 400 Bad Request – The browse request failed due to invalid parameters (e.g: the path doesn't exist).
- 404 Not Found – The remote session ID was invalid or expired.

Response Headers

- Content-Type – application/json
- *X-Slycat-Message* – For errors, contains a human-readable description of the problem.
- *X-Slycat-Hint* – For errors, contains an optional description of how to fix the problem.

Response JSON Object

- **path** (*string*) – Remote filesystem path.
- **names** (*array*) – Array of string filenames contained within the remote filesystem path.
- **sizes** (*array*) – Array of integer file sizes.
- **types** (*array*) – Array of string file types, "f" for regular files, "d" for directories.
- **mtimes** (*array*) – Array of string file modification times, in ISO-8601 format.
- **mime-types** (*array*) – Array of string MIME types.

The regular expression parameters are matched against full file / directory paths. If a file / directory matches a reject expression, it will not be included in the results, unless it also matches an allow expression. So, to remove JPEG files from the results:

```
file-reject: "[.]jpg$|[.]jpeg$"
```

but to only return CSV files:

```
file-reject: ".*",
file-allow: "[.]csv$"
```

Sample Request

```
POST /remotes/505d0e463d5ed4a32bb6b0fe9a000d36/browse/home/fred

{
  file-reject: "[.]jpg$"
}
```

Sample Response

```
{
  "path" : "/home/fred",
  "names" : ["a.txt", "b.png", "c.csv", "subdir"],
  "sizes" : [1264, 456730, 78005, 4096],
  "types' : ["f", "f", "f", "d"],
  "mtimes" : ["2015-03-03T16:52:34.599466", "2015-03-02T21:03:50", "2015-03-02T21:03:50", "2015-
  "mime-types" : ["text/plain", "image/png", "text/csv", null],
}
```

See Also

- *POST /remotes*
- *GET /remotes/(sid)/file(path)*
- *GET /remotes/(sid)/image(path)*

- *POST /remotes/(sid)/videos*

POST Remote Launch

POST /remotes/launch

Uses an existing remote session to submit a command. The session must have been created successfully using *POST /remotes*.

Request JSON Object

- **sid** (*string*) – Unique remote session identifier.
- **command** (*string*) – command to be ran on the remote system.

Status Codes

- 200 OK – The response contains the command, its output and possible errors.
- 400 Bad Request – The request failed due to invalid parameters or a Slycat agent issue.
- 500 Internal Server Error – The request failed due to a SSH exception.

Response Headers

- Content-Type – application/json
- *X-Slycat-Message* – For errors, contains a human-readable description of the problem.

Response JSON Object

- **command** (*string*) – Command issued to the remote system.
- **output** (*string*) – Output of the command.
- **errors** (*string*) – Error information, if any.

Sample Request

```
POST /remotes/launch

{
  sid: "505d0e463d5ed4a32bb6b0fe9a000d36",
  command: "echo test"
}
```

Sample Response

```
{
  "command": "echo test",
  "output": "test",
  "errors": ""
}
```

See Also

- *POST /remotes/cancel-job*
- *POST /remotes/checkjob*
- *POST /remotes/get-job-output*
- *POST /remotes/run-agent-function*
- *POST /remotes/submit-batch*

POST Remote Videos

POST /remotes/(*sid*)**/videos**

Uses an existing remote session to create a video from a sequence of images. The session must have been created successfully using *POST /remotes* and the session must have a running agent. The caller *must* supply the session id, the desired video content type, and the paths of source images on the remote filesystem. Because video compression may be time-consuming, a unique video session ID is returned. The client must supply the video session ID along with the remote session ID in subsequent *GET /remotes/(sid)/videos/(vsid)/status* and *GET /remotes/(sid)/videos/(vsid)* requests.

Parameters

- **sid** (*string*) – Unique remote session identifier.

Request Headers

- Content-Type – application/json

Request JSON Object

- **content-type** (*string*) – Content type for the final video. Currently limited to "video/mp4" or "video/webm".
- **images** (*array*) – List of absolute paths pointing to static images.

Status Codes

- 202 Accepted – Video creation has started.
- 400 Bad Request – "Agent required" This call requires a remote agent, but the current session isn't running an agent.
- 400 Bad Request – Couldn't start video creation with the current request parameters.
- 404 Not Found – The session doesn't exist or has timed-out.

Response Headers

- Content-Type – application/json
- *X-Slycat-Message* – For errors, contains a human-readable description of the problem.
- *X-Slycat-Hint* – For errors, contains an optional description of how to fix the problem.

Response JSON Object

- **sid** (*string*) – Unique video-creation session identifier.

Sample Request

```
POST /remotes/505d0e463d5ed4a32bb6b0fe9a000d36/videos

{
  content-type: "video/mp4",
  images: ["/home/fred/1.jpg", "/home/fred/2.jpg", "/home/fred/3.jpg", ...],
}
```

Sample Response

```
{
  "sid" : 431d0e463d5ed4a32bb6b0fe9a000a37
}
```

See Also

- *GET /remotes/(sid)/videos/(vsid)*
- *GET /remotes/(sid)/videos/(vsid)/status*

POST Remotes

POST /remotes

Creates a new remote connection from the Slycat server to another host. The caller *must* supply a remote hostname, username, and password.

If the connection is created successfully, a unique session ID is returned. The client must use the session ID in subsequent requests.

Request Headers

- Content-Type – application/json

Request JSON Object

- **hostname** (*string*) – Remote hostname.
- **username** (*string*) – Remote host username.
- **password** (*string*) – Remote host password.
- **agent** (*boolean*) – (optional) Create an agent when the connection is established. By default, agents are created automatically if the hostname has an agent configuration. Use this parameter to explicitly require / prevent agent creation.

Status Codes

- 200 OK – The connection was created successfully.
- 400 Bad Request – "Missing agent configuration" The server isn't configured to start an agent on the given hostname.
- 403 Forbidden – "Remote authentication failed" Authentication of the provided username and password failed.
- 500 Internal Server Error – "Missing agent configuration" The server isn't properly configured to start an agent on the given hostname.
- 500 Internal Server Error – "Agent startup failed" The server couldn't start an agent on the given hostname.
- 500 Internal Server Error – "Remote connection failed" Unknown failure making the remote connection.

Response Headers

- Content-Type – application/json

Response JSON Object

- **sid** (*string*) – Unique remote session identifier.

Sample Request

```
POST /remotes HTTP/1.1
Host: localhost:8092
Content-Length: 45
Accept-Encoding: gzip, deflate, compress
```

```
Accept: */*
User-Remote: python-requests/1.2.0 CPython/2.7.3 Linux/2.6.32-358.2.1.el6.x86_64
content-type: application/json
Authorization: Basic c2x5Y2F0OnNseWNhdA==

{"hostname":"example.com", "username":"fred", "password":"foobar"}
```

Sample Response

```
HTTP/1.1 200 OK.
Date: Thu, 11 Apr 2013 21:30:16 GMT
Content-Length: 42
Content-Type: application/json
Location: http://localhost:8092/projects/505d0e463d5ed4a32bb6b0fe9a000d36
Server: CherryPy/3.2.2

{"sid": "505d0e463d5ed4a32bb6b0fe9a000d36"}
```

See Also

- *DELETE /remotes/(sid)*
- *POST /remotes/(sid)/browse(path)*

POST Submit Batch

POST /remotes/submit-batch

Uses an existing remote sessions to submit a batch file to start a job on a cluster running SLURM. The session must have been created successfully using *POST /remotes*.

Request JSON Object

- **sid** (*string*) – Unique remote session identifier.
- **filename** (*string*) – Name for the batch file.

Status Codes

- 200 OK – The response contains the command, its output and possible errors.
- 400 Bad Request – The request failed due to invalid parameters or a Slycat agent issue.
- 500 Internal Server Error – The request failed due to no Slycat agent present and configured on the remote system.

Response Headers

- Content-Type – application/json
- *X-Slycat-Message* – For errors, contains a human-readable description of the problem.

Response JSON Object

- **filename** (*string*) – Name of the file submitted in the request.
- **jid** (*int*) – Job ID.
- **errors** (*string*) – Error information, if any.

Sample Request

```
POST /remotes/submit-batch

{
  sid: "505d0e463d5ed4a32bb6b0fe9a000d36",
  filename: "/home/jdoe/batch.test.bash"
}
```

Sample Response

```
{
  "filename": "/home/jdoe/batch.test.bash",
  "jid": 123456,
  "errors": ""
}
```

See Also

- *POST /remotes/cancel-job*
- *POST /remotes/checkjob*
- *POST /remotes/get-job-output*
- *POST /remotes/launch*
- *POST /remotes/run-agent-function*

POST Uploads

POST /uploads

Create an upload session used to upload files for storage as model artifacts. Once an upload session has been created, use *PUT /uploads/(uid)/files/(fid)/parts/(pid)* to upload files directly from the client to the server or from a remote host to the server using a remote session.

In either case this call must include the id of the model to receive new artifacts, a boolean "input" parameter to specify whether the created artifacts are input artifacts, the name of a parsing plugin in "parser", and one or more artifact ids using "aids". Any additional parameters will be passed unchanged to the parsing plugin for use as plugin-specific parsing parameters.

The set of parsing plugins will vary based on server configuration, and parsing plugins have wide latitude in how they map parsed file data to model artifacts. For example, the slycat-blob-parser plugin will store N files as unparsed model file artifacts, and thus requires N corresponding artifact ids to use for storage. Similarly, the slycat-csv-parser plugin stores N parsed files as arrayset artifacts, and also requires N artifact ids. However, more sophisticated parsing plugins could split one file into multiple artifacts, combine multiple files into one artifact, or store any other combination of M files into N artifacts.

Request Headers

- Content-Type – application/json

Request JSON Object

- **mid** (*string*) – Unique model identifier.
- **input** (*string*) – Set to "true" to store results as input artifacts.
- **parser** (*string*) – Parsing plugin name.
- **aids** (*array*) – Artifact ids for storage.

Status Codes

- 200 OK – The new upload session was created, and the response contains the new session id.
- 400 Bad Request – An upload session couldn't be created due to invalid parameters (e.g: unknown model, unknown parser, invalid parser parameters).
- 403 Forbidden – Client doesn't have write access to the given model

Response Headers

- Content-Type – application/json

Response JSON Object

- **id** (*string*) – New upload session id.

See Also

- *PUT /uploads/(uid)/files/(fid)/parts/(pid)*

POST Upload Finished

POST /uploads/(*uid*)**/finished**

Notify the server that all files have been uploaded for the given upload session, and processing can begin. The request must include the *uploaded* parameter, which specifies the number of files that were uploaded, and the number of parts in each file. The server uses this information to validate that it received every part of every file that the client sent.

Parameters

- **uid** (*string*) – Unique upload session identifier.

Request Headers

- Content-Type – application/json

Request JSON Object

- **uploaded** (*array*) – array containing the number of parts M for every uploaded file N.

Status Codes

- 202 Accepted – The server has validated all of the uploaded data, and will begin the parsing process.
- 400 Bad Request – "Upload incomplete" The server did not receive all of the file parts specified in the *uploaded* parameter. Parsing will not begin until the missing parts have been uploaded and *POST /uploads/(uid)/finished* is called again.
- 400 Bad Request – "Client confused" The server received more file parts than those specified in the *uploaded* parameter. Parsing will not begin unless *POST /uploads/(uid)/finished* is called again with the correct part counts in *uploaded*.

Response Headers

- Content-Type – application/json

Response JSON Object

- **missing** (*array*) – array containing a [fid, pid] tuple for every file part that wasn't uploaded successfully.

See Also

- *PUT /uploads/(uid)/files/(fid)/parts/(pid)*
- *DELETE /uploads/(uid)*

PUT Model Arrayset Array

PUT /models/ (*mid*) **/arraysets/**
*aid***/arrays/***array* Adds an array to an arrayset, ready to upload data. The arrayset must already have been initialized with *PUT /models/(mid)/arraysets/(aid)*.

Parameters

- **mid** (*string*) – Unique model identifier.
- **aid** (*string*) – Unique artifact id.
- **array** (*int*) – Unique array index.

Request Headers

- Content-Type – application/json

Request JSON Object

- **attributes** (*object*) – New array attributes.
- **dimensions** (*object*) – New array dimensions.

Sample Request

```
PUT /models/6f48db3de2b6416091d31e93814a22ae/arraysets/test-array-set/arrays/0 HTTP/1.1
Host: localhost:8093
Content-Length: 203
Accept-Encoding: gzip, deflate, compress
Accept: */*
User-Agent: python-requests/1.2.3 CPython/2.7.5 Linux/2.6.32-358.23.2.el6.x86_64
content-type: application/json
Authorization: Basic c2x5Y2F0OnNseWNhdA==

{
  "attributes": [
    {"type": "int64", "name": "integer"},
    {"type": "float64", "name": "float"},
    {"type": "string", "name": "string"}],
 "dimensions": [
    {"end": 10, "begin": 0, "type": "int64", "name": "row"}]
}
```

Sample Response

```
HTTP/1.1 200 OK
Date: Mon, 25 Nov 2013 20:36:07 GMT
Content-Length: 4
Content-Type: application/json
Server: CherryPy/3.2.2

null
```

See Also

- *PUT /models/(mid)/arraysets/(aid)*
- *PUT /models/(mid)/arraysets/(aid)/data*

PUT Model Arrayset Data

PUT /models/ (*mid*) **/arraysets/**
*aid***/data** Upload data to be stored in arrayset array attributes. The request may contain data to be stored in any combinations of arrays, attributes, and hyperslices. The destination array(s) must have already been initialized with *PUT /models/(mid)/arraysets/(aid)/arrays/(array)*.

Parameters

- **mid** (*string*) – Unique model identifier.
- **aid** (*string*) – Unique artifact id.

Request Headers

- Content-Type – multipart/form-data

Form Parameters

- **hyperchunks** – (Required) The arrays, attributes, and hyperslices to be overwritten, in *Hyperchunks* format.
- **byteorder** – (Optional) Specifies that the request contains binary data with the given endianness.

  The byteorder parameter must be either "little" or "big". Note that the byteorder parameter can only be used if every attribute in every hyperchunk is of numeric type.
- **data** – (Required) The data to be stored.

  If the byteorder is specified, the request data must contain contiguous raw data bytes in the given byteorder, in the same order as the hyperchunks / hyperslices. For multi-dimension arrays, hyperslice array elements must be in "C" order.

  If the byteorder parameter isn't specified, the request data must contain a JSON-encoded array with length equal to the total number of hyperslices. Each element in this top level array must be an array containing the data for the corresponding hyperslice, in the same order as the hyperchunks / hyperslices. For multi-dimension arrays, data for the corresponding hyperslice will be nested further.

Sample Request

The following request would write data in binary format to the following locations:

•Element number 5 in vector array 0, attribute 1

•A half-open range of elements [10-20) in vector array 2, attribute 3

•A 4x4 subset of elements in matrix array 4, attribute 5

•Elements [0-10) and [20-30) in vector array 6, attribute 7

```
PUT /models/25f1cdb62c34465286cecbaeccc1460d/arraysets/test-array-set/data HTTP/1.1
Host: localhost:8093
Content-Length: 470
Accept-Encoding: gzip, deflate, compress
Accept: */*
```

```
User-Agent: python-requests/1.2.3 CPython/2.7.5 Linux/2.6.32-358.23.2.el6.x86_64
Content-Type: multipart/form-data; boundary=573af150d64b4d70b35689f41c136ed3
Authorization: Basic c2x5Y2F0OnNseWNhdA==

--573af150d64b4d70b35689f41c136ed3
Content-Disposition: form-data; name="byteorder"

little
--573af150d64b4d70b35689f41c136ed3
Content-Disposition: form-data; name="hyperchunks"

0/1/5;2/3/10:20;4/5/0:4,0:4;6/7/0:10|20:30
--573af150d64b4d70b35689f41c136ed3
Content-Disposition: form-data; name="data"; filename="data"
Content-Type: application/octet-stream

....................................
--573af150d64b4d70b35689f41c136ed3--
```

Sample Response

```
HTTP/1.1 200 OK
Date: Tue, 26 Nov 2013 16:40:05 GMT
Content-Length: 4
Content-Type: application/json
Server: CherryPy/3.2.2

null
```

See Also

- *Hyperchunks*
- *PUT /models/(mid)/arraysets/(aid)*
- *PUT /models/(mid)/arraysets/(aid)/arrays/(array)*

PUT Model Arrayset

PUT /models/(*mid*)**/arraysets/**

aid Initialize an arrayset, a collection of zero-to-many arrays.

Parameters

- **mid** (*string*) – Unique model identifier.
- **aid** (*string*) – Unique artifact id.

Request Headers

- Content-Type – application/json

Request JSON Object

- **input** (*bool*) – Set to true if this arrayset is a model input.

Sample Request

```
PUT /models/6f48db3de2b6416091d31e93814a22ae/arraysets/test-array-set HTTP/1.1
Host: localhost:8093
Content-Length: 2
Accept-Encoding: gzip, deflate, compress
Accept: */*
User-Agent: python-requests/1.2.3 CPython/2.7.5 Linux/2.6.32-358.23.2.el6.x86_64
content-type: application/json
Authorization: Basic c2x5Y2F0OnNseWNhdA==

{ input : true }
```

Sample Response

```
HTTP/1.1 200 OK
Date: Mon, 25 Nov 2013 20:36:07 GMT
Content-Length: 0
Content-Type: text/html;charset=utf-8
Server: CherryPy/3.2.2
```

See Also

- *PUT /models/(mid)/arraysets/(aid)/arrays/(array)*
- *PUT /models/(mid)/arraysets/(aid)/data*

PUT Model Command

PUT /models/ (*mid*) **/commands/**
type/*command* Execute a custom model command.

Plugins may register custom commands to be executed on the server, using an existing model as context. Custom commands are used to perform computation on the server instead of the client, and would typically use model artifacts as inputs.

Parameters

- **mid** (*string*) – Unique model identifier.
- **type** (*string*) – Unique command category.
- **command** (*string*) – Custom command name.

Additional command-specific arguments may be passed using query strings.

Response Headers

- Content-Type – */*

Sample Request

```
PUT /models/e32ef475e084432481655fe41348726b/commands/math-plugin/add HTTP/1.1
Host: localhost:8093
Authorization: Basic c2x5Y2F0OnNseWNhdA==
Accept-Encoding: gzip, deflate, compress
accept: application/json
User-Agent: python-requests/1.2.3 CPython/2.7.5 Linux/2.6.32-358.23.2.el6.x86_64
```

Sample Response

```
HTTP/1.1 200 OK
Date: Mon, 25 Nov 2013 20:36:01 GMT
Content-Length: 542
Content-Type: application/json
Server: CherryPy/3.2.2

{
  "result" : 5
}
```

See Also

- *GET /models/(mid)/commands/(type)/(command)*
- *POST /models/(mid)/commands/(type)/(command)*

PUT Model Inputs

PUT /models/ (*mid*) **/inputs**

Copies the input artifacts from one model to another. Both models must be part of the same project. By default, array artifacts are copied by reference instead of value for efficiency.

Parameters

- **mid** (*string*) – Unique model identifier.

Request Headers

- Content-Type – application/json

Request JSON Object

- **sid** (*string*) – Unique identifier of the source model.
- **deep-copy** (*bool*) – Optional, make deep copies of input data if "true".

PUT Model Parameter

PUT /models/ (*mid*) **/parameters/**

aid Stores a model parameter (name / value pair) artifact. The value is a JSON expression and may be arbitrarily complex.

Parameters

- **mid** (*string*) – Unique model identifier.
- **aid** (*string*) – Unique artifact id (parameter name).

Request Headers

- Content-Type – application/json

Request JSON Object

- **value** (*object*) – New parameter value.
- **input** (*bool*) – Set to true if the parameter is a model input.

Sample Request

```
PUT /models/1385a75dd2eb4faba884cefdd0b94a56/parameters/baz HTTP/1.1
Host: localhost:8093
Content-Length: 20
Accept-Encoding: gzip, deflate, compress
Accept: */*
User-Agent: python-requests/1.2.3 CPython/2.7.5 Linux/2.6.32-358.23.2.el6.x86_64
content-type: application/json
Authorization: Basic c2x5Y2F0OnNseWNhdA==

{
  value : [1, 2, 3],
  input : true
}
```

Sample Response

```
HTTP/1.1 200 OK
Date: Mon, 25 Nov 2013 20:36:04 GMT
Content-Length: 0
Content-Type: text/html;charset=utf-8
Server: CherryPy/3.2.2
```

See Also

- *GET /models/(mid)/parameters/(aid)*

PUT Model

PUT /models/ (*mid*)

Modifies a model. Callers may change the model name, description, state, result status, progress, and message.

Parameters

- **mid** (*string*) – Unique model identifier.

Request Headers

- Content-Type – application/json

Request JSON Object

- **name** (*string*) – optional, New model name.
- **description** (*string*) – optional, New model description.
- **state** (*string*) – optional, New model state.
- **progress** (*float*) – optional, New model progress percent.
- **message** (*string*) – optional, New model status message.

See Also

- *GET /models/(mid)*
- *POST /models/(mid)/finish*
- *DELETE /models/(mid)*

PUT Project

PUT /projects/ (*pid*)

Modifies a project. Callers may use PUT to specify a new name, description, or access control list (ACL) for the project.

Parameters

- **pid** (*string*) – Unique project identifier.

Request JSON Object

- **name** (*string*) – optional, New project name.
- **description** (*string*) – optional, New project description.
- **acl** (*object*) – optional, New project access control list.

Sample Request

```
PUT /projects/dbaf026f919620acbf2e961ad73243c5 HTTP/1.1
Host: localhost:8093
Content-Length: 176
Accept-Encoding: gzip, deflate, compress
Accept: */*
User-Agent: python-requests/1.2.3 CPython/2.7.5 Linux/2.6.32-358.23.2.el6.x86_64
content-type: application/json
Authorization: Basic c2x5Y2F0OnNseWNhdA==

{
  "acl": {"administrators": [{"user": "slycat"}], "writers": [{"user": "foo"}], "readers": [{"us
  "name": "modified-project",
  "description": "My modified project."
}
```

Sample Response

```
HTTP/1.1 200 OK
Date: Mon, 25 Nov 2013 20:35:59 GMT
Content-Length: 4
Content-Type: application/json
Server: CherryPy/3.2.2

null
```

See Also

- *GET /projects/(pid)*
- *DELETE /projects/(pid)*

PUT Upload File Part

PUT /uploads/ (*uid*) **/files/** *fid* **/parts/** *pid*

Upload a file (or part of a file) as part of an upload session created with *POST /uploads*.

Use the "pid" and "fid" parameters to specify that the data being uploaded is for part M of file N. To upload a file from the client, specify the "file" parameter. To upload a remote file, specify the "sid" and "path" parameters with a session id and remote filepath for the file to upload.

Parameters

- **uid** (*string*) – Unique upload session identifier.
- **fid** (*integer*) – Zero-based file index of the data to be uploaded.
- **pid** (*integer*) – Zero-based part index of the data to be uploaded.

Request Headers

- Content-Type – form/multipart

Form Parameters

- **file** – Local file for upload.
- **path** – Remote host absolute filesystem path.
- **sid** – Remote session id.

Status Codes

- 200 OK – The data was uploaded successfully.

See Also

- *POST /uploads*
- *POST /uploads/(uid)/finished*

Javascript API

For the convenience of Javascript clients and Slycat plugin code, we provide a set of custom AMD modules containing useful components, along with wrappers around the *REST API*.

slycat-login-controls

The slycat-login-controls AMD module registers a Knockout component of the same name. The slycat-login-controls component provides a standard GUI widget for selecting a username and password to complete a login.

Note: you don't need to import the slycat-login-controls module using `require()` or `define()` - it registers the knockout component automatically at startup.

To use slycat-login-controls, create `ko.observable()` objects for each of the login parameters, including the username and password, and bind them to the page DOM:

```
var page =
{
  username: ko.observable("fred"),
  password: ko.observable(""),
};

ko.applyBindings(page);
```

Then, embed the slycat-login-controls component in your markup and bind your observables to the component parameters:

```
<p>Login to orbiting brain lasers:</p>
<slycat-login-controls params="
  username: username,
  password: password,
  ">
</slycat-login-controls>
```

Now, changes to any of the input parameters automatically update the login controls, and user interaction with the login controls will update the *username* and *password* observables.

The full set of parameters supported by slycat-login-controls are as follows:

- username, `ko.observable()`: String username to be entered by the user. If this parameter is null or empty, it will default to the last-used username.
- password, `ko.observable()`: String password to be entered by the user.
- status, `ko.observable()`: Optional string status message to be displayed under the controls.
- status_type, `ko.observable()`: Optional string status type that controls the appearance of the status message. Must be one of "success", "info", "warning", or "danger".
- enable, `ko.observable()`: Optional boolean value to enable / disable the controls.
- focus, `ko.observable()`: Optional, used to focus the controls. Set to *"username"* to focus the username control, *"password"* to focus the password control, or *true* to automatically choose which control to focus. Because the caller may wish to focus the same control more than once in a row (for example: to refocus the password control after a failed login attempt), it is useful to configure the focus observable to always notify subscribers, even if its value doesn't change, using *focus.extend({notify: "always"})*.
- activate, function: Optional callback function that will be invoked if the user presses the "enter" key while using the login controls.

See Also

- *slycat-remote-controls* - if you also need to prompt users for a hostname.
- *slycat-remotes* - for a higher-level API that provides a modal login dialog, and can manage a pool of remote connections.

slycat-range-slider

The slycat-range-slider AMD module registers a Knockout component of the same name. The slycat-range-slider component provides a standard GUI widget for selecting a closed range of values from a continuous domain.

Note: you don't need to import the slycat-range-slider module using `require()` or `define()` - it registers the slider component automatically at startup.

To use slycat-range-slider, create `ko.observable()` objects for each of the range slider parameters, including the output range values, and bind them to the page DOM:

```
var page =
{
  slider_length: 500,
  minimum_price: ko.observable(150),
  low_price: ko.observable(1000),
  high_price: ko.observable(5000),
  maximum_price: ko.observable(20000),
};
```

```
ko.applyBindings(page);
```

Then, embed the slycat-range-slider component in your markup and bind your observables to the component parameters:

```
<p>Filter results by price:</p>
<slycat-range-slider params="
  length: slider_length,
  min: minimum_price,
  low: low_price,
  high: high_price,
  domain: maximum_price,
  ">
</slycat-range-slider>
```

Now, changes to any of the input parameters automatically update the slider, and user interaction with the slider will update the *low* and *high* observables.

The full set of parameters supported by slycat-range-slider are as follows:

- axis, string: "vertical" or "horizontal" to create a slider with the given orientation. Default: "vertical".
- reverse, bool: If true, the orientation of the slider is reversed so that high and low values are swapped. Default: false.
- length, `ko.observable()`: Length of the slider in pixels. Default: 500 pixels.
- thumb_length, `ko.observable()`: Length of the slider thumb buttons in pixels. Default: 12 pixels.
- dragging, `ko.observable()`: Set to true while the user is dragging a thumb button.
- min, `ko.observable()`: Minimum allowed value. Default: 0.
- low, `ko.observable()`: Currently-selected range low value. Default: 0.33.
- high, `ko.observable()`: Currently-selected range high value. Default: 0.66.
- max, `ko.observable()`: Maximum allowed value. Default: 1.

slycat-remote-controls

The slycat-remote-controls AMD module registers a Knockout component of the same name. The slycat-remote-controls component provides a standard GUI widget for selecting a hostname, username, and password to complete a login.

Note: you don't need to import the slycat-remote-controls module using `require()` or `define()` - it registers the knockout component automatically at startup.

To use slycat-remote-controls, create `ko.observable()` objects for each of the login parameters, including the hostname, username and password, and bind them to the page DOM:

```
var page =
{
  hostname: ko.observable("localhost"),
  username: ko.observable("fred"),
  password: ko.observable(""),
};

ko.applyBindings(page);
```

Then, embed the slycat-remote-controls component in your markup and bind your observables to the component parameters:

```
<p>Login to mutant cybergoat server:</p>
<slycat-remote-controls params="
  hostname: hostname,
  username: username,
  password: password,
  ">
</slycat-remote-controls>
```

Now, changes to any of the input parameters automatically update the login controls, and user interaction with the login controls will update the *username* and *password* observables.

The full set of parameters supported by slycat-remote-controls are as follows:

- hostname, `ko.observable()`: String hostname to be entered by the user. If this parameter is null or empty, it will default to the last-used hostname.
- username, `ko.observable()`: String username to be entered by the user. If this parameter is null or empty, it will default to the last-used username.
- password, `ko.observable()`: String password to be entered by the user.
- status, `ko.observable()`: Optional string status message to be displayed under the controls.
- status_type, `ko.observable()`: Optional string status type that controls the appearance of the status message. Must be one of "success", "info", "warning", or "danger".
- enable, `ko.observable()`: Optional boolean value to enable / disable the controls.
- focus, `ko.observable()`: Optional, used to focus the controls. Set to *"hostname"* to focus the hostname control, *"username"* to focus the username control, *"password"* to focus the password control, or *true* to automatically choose which control to focus. Because the caller may wish to focus the same control more than once in a row (for example: to refocus the password control after a failed login attempt), it is useful to configure the focus observable to always notify subscribers, even if its value doesn't change, using *focus.extend({notify: "always"})*.
- activate, function: Optional callback function that will be invoked if the user presses the "enter" key while using the login controls.

See Also

- *slycat-login-controls* - if you don't need to prompt users for a hostname.
- *slycat-remotes* - for a higher-level API that provides a modal login dialog, and can manage a pool of remote connections.

slycat-remotes

The slycat-remotes AMD module provides a high-level API for making a remote connection to another host, when the hostname is known in advance, and maintaining a pool of remote connections.

For example, once the module has been imported into the current namespace:

```
require(["slycat-remotes"], function(remotes)
{
  // Use the module here
});
```

A remote session can be created as follows (the user will be prompted for their username and password with a modal dialog):

```
remotes.login(
{
  hostname: "localhost",
  success: function(sid)
  {
    // Do something with the remote session id
  },
});
```

slycat-remotes.**login**(*params*)

Prompt the user for a username and password, and create a remote session:

Arguments

- **params** (*object*) – a set of key/value login parameters:
  - hostname (string) - Required, remote hostname.
  - title (string) - Optional title for the login dialog.
  - message (string) - Optional message for the login dialog.
  - success (function) - Optional, called with the remote session ID when the remote connection is made.
  - cancel (function) - Optional, called if the user cancels making a connection.

The user will be prompted for their login information until they are successful, or cancel the operation.

slycat-remotes.**create_pool**()

Create and return an object that manages a collection of remote sessions.

Returns an instance of slycat-remote.pool that manages a collection of remote sessions, organized by hostname.

slycat-remotes.pool.**get_remote**(*params*)

Retrieve an existing remote session ID for a given host, or prompt the user to create a new session.

Arguments

- **params** (*object*) – a set of key/value parameters:
  - hostname (string) - Required remote hostname.
  - title (string) - Optional title for the login dialog, if the remote session doesn't already exist.
  - message (string) - Optional message for the login dialog, if the remote session doesn't already exist.
  - success (function) - Optional, called with the remote session ID if it already exists, or the user successfully creates a new session.
  - cancel (function) - Optional, called if the host connection doesn't already exist, and the user cancels session creation.

slycat-remotes.pool.**delete_remote**(*hostname*)

Shut-down and remove the remote session (if any) for the given host.

Arguments

- **hostname** (*string*) – the host whose session should be closed. Calls with unknown hostnames will be quietly ignored.

Note that this method could cause a harmless failed AJAX request, if the given session has already expired.

See Also

- *slycat-login-controls* - for a lower-level set of login controls.
- *slycat-remote-controls* - for a lower-level set of hostname + login controls.

slycat-server-root

Like any web service, the Slycat server could be deployed behind a reverse proxy, altering the URLs used by a client to access the *REST API*. For example, if an organization deployed an instance of Slycat at *http://example.com/services/slycat*, clients would retrieve the list of available projects at */services/slycat/projects* instead of the usual */projects*.

To facilitate this, the slycat-server-root AMD module returns a single constant string - the server root - which *must* be prepended to all URLs used by clients. For example, clients should never use hard-coded URLs:

```
jquery.ajax("/projects"); // NEVER DO THIS
```

Instead, the server root must be imported into the current namespace:

```
require(["slycat-server-root"], function(server_root)
{
  // Use the server_root string here
});
```

And used to construct URLs dynamically at runtime:

```
jquery.ajax(server_root + "projects");
```

Note that clients should rarely need to construct URLs in the first place - instead, they should use the *slycat-web-client* module, which provides simplified access to the *REST API* and uses the server root for you.

slycat-web-client

The slycat-web-client AMD module provides convenient Javascript bindings for the *REST API*, in a style similar to `jquery.ajax()`.

For example, once the module has been imported into the current namespace:

```
require(["slycat-web-client"], function(client)
{
  // Use the module here
});
```

A model can be retrieved using:

```
client.get_model(
{
  mid: model_id, // Unique model identifier
  success: function(model)
  {
    // Do something with the model
  },
});
```

slycat-web-client.**delete_model**(*params*)
Delete an existing model.

Arguments

- **params** (*object*) – a set of key/value pairs that configure the request:
  - mid (string) - required, unique model identifier.
  - success (function) - optional, called when the request completes successfully.
  - error (function) - optional, called if the request fails.

    param request

    param status

    param reason_phrase

slycat-web-client.**delete_project**(*params*)
Delete an existing project.

Arguments

- **params** (*object*) – a set of key/value pairs that configure the request:
  - pid (string) - required, unique project identifier.
  - success (function) - optional, called when the request completes successfully.
  - error (function) - optional, called if the request fails.

Python API

The Slycat server and plugins used to enhance it are implemented in Python. In addition, we provide wrappers around the *REST API* for writing Python clients, typically used for custom data ingestion.

slycat.cca

slycat.cca.**cca**(*X*, *Y*, *scale_inputs=True*, *force_positive=None*, *significant_digits=None*)
Compute Canonical Correlation Analysis (CCA).

Parameters

- **X** (*numpy.ndarray*) – $M \times I$ matrix containing M observations and I input features.
- **Y** (*numpy.ndarray*) – $M \times O$ matrix containing M observations and O output features.
- **scale_inputs** (*bool, optional*) – Scale input and output features to unit variance.
- **force_positive** (*integer, optional*) – If specified, flip signs in the *x*, *y*, *x_loadings*, and *y_loadings* output values so that the values in row n of *y_loadings* are all positive.
- **significant_digits** (*integer, optional*) – Optionally specify the number of significant digits used to compute the *X* and *Y* ranks.

Returns

- **x** (*numpy.ndarray*) – $M \times C$ matrix containing input metavariable values for M observations and C CCA components.
- **y** (*numpy.ndarray*) – $M \times C$ matrix containing output metavariable values for M observations and C CCA components.
- **x_loadings** (*numpy.ndarray*) – $I \times C$ matrix containing weights for I input variables and C CCA components.
- **y_loadings** (*numpy.ndarray*) – $O \times C$ matrix containing weights for O output variables and C CCA components.
- **r2** (*numpy.ndarray*) – length-C vector containing r^2 values for C CCA components.
- **wilks** (*numpy.ndarray*) – length-C vector containing the likelihood-ratio for C CCA components.

slycat.darray

Slycat makes extensive use of *darray* objects - dense, multi-dimension, multi-attribute arrays - as its fundamental unit of storage and organization. In the abstract, a darray can be modeled as follows:

- A set of dimensions. Each dimension has a name, index type, and a half-open range of valid index values. Currently, the only supported index type is "int64", and indices are all zero-based (i.e. the range always begins at zero), but these may change in the future. Collectively, the dimensions define the size and shape of the array.
- A set of attributes, each with a name and type. Allowed attribute types include a full complement of signed and unsigned fixed-width integer types, plus floating-point and string types. Collectively, attributes define *what* will be stored in the array.
- The array data. Because darrays are dense, the data will include one value per attribute, for every location in the array.

This definition allows darrays to be flexible and efficient - for example, a "table" data structure with heterogenous column types can be stored as a 1D darray with multiple attributes, while a "matrix" would be stored as a 2D darray with a single floating-point attribute.

Note that darrays are an abstract concept with multiple concrete representations. This module defines an abstract interface for manipulating Python darrays, and a concrete implementation with in-memory storage. The *slycat.hdf5* module defines functionality for manipulating darrays stored in HDF5 files on disk, and the *REST API* defines functionality for working with darrays using HTTP.

Note that it is rare to manipulate entire darrays in memory at once, due to their size - most applications will work with *slices* of a darray to keep memory use manageable.

class slycat.darray.**MemArray**(*dimensions*, *attributes*, *data*)
Bases: *slycat.darray.Stub*

darray implementation that holds the full array contents in memory.

get_data(*attribute=0*)
Return a data slice from one attribute.

get_statistics(*attribute=0*)
Return statistics describing one attribute.

set_data(*attribute*, *slice*, *data*)
Write a data slice to one attribute.

class slycat.darray.**Prototype**
Bases: object

Abstract interface for all darray implementations.

attributes
Return a description of the array attributes.

dimensions
Return a description of the array dimensions.

get_data(*attribute=0*)
Return data from one attribute.

get_statistics(*attribute=0*)
Return statistics describing one attribute.

ndim
Return the number of dimensions in the array.

set_data(*attribute*, *slice*, *data*)
Write data to one attribute.

shape
Return the shape (size along each dimension) of the array.

size
Return the size (total number of elements) of the array.

class slycat.darray.**Stub**(*dimensions*, *attributes*)
Bases: *slycat.darray.Prototype*

darray implementation that only stores array metadata (dimensions and attributes).

attributes
Return a description of the array attributes.

dimensions
Return a description of the array dimensions.

ndim
Return the number of dimensions in the array.

shape
Return the shape (size along each dimension) of the array.

size
Return the size (total number of elements) of the array.

slycat.hdf5

class slycat.hdf5.**ArraySet**(*file*)
Bases: object

Wraps an instance of h5py.File to implement a Slycat arrayset.

array_count()
Note: this assumes that array indices are contiguous, which we don't explicitly enforce.

keys()

start_array(*array_index*, *dimensions*, *attributes*)
Add an uninitialized darray to the arrayset.

An existing array with the same index will be overwritten.

Parameters

- **array_index** (*integer, required.*) – Zero-based index of the array to create.
- **dimensions** (*list of dicts, required.*) – Description of the new array dimensions.
- **attributes** (*list of dicts, required.*) – Description of the new array attributes.

Returns array

Return type *slycat.hdf5.DArray*

store_array(*array_index*, *array*)

Store a *slycat.darray.Prototype* in the arrayset.

An existing array with the same index will be overwritten.

Parameters

- **array_index** (*integer, required.*) – The index of the array to be created / overwritten.
- **array** (*slycat.darray.Prototype*, required.) – Existing darray to be stored.

Returns array

Return type *slycat.hdf5.DArray*

class slycat.hdf5.**DArray**(*storage*)

Bases: *slycat.darray.Prototype*

Slycat darray implementation that stores data in an HDF5 file.

attributes

Return metadata describing the darray attributes.

Returns attributes

Return type list of dicts

dimensions

Return metadata describing the darray dimensions.

Returns dimensions

Return type list of dicts

get_data(*attribute*)

Return a reference to the data storage for a darray attribute.

Parameters **attribute** (*integer, optional*) – The integer index of the attribute data to retrieve.

Returns **data** – An object implementing a subset of the `numpy.ndarray` interface that contains the attribute data. Note that the returned object only *references* the underlying data - data is not retrieved from the file until you access it using the *[]* operator.

Return type reference to a numpy-array-like object.

get_statistics(*attribute*)

get_unique(*attribute*, *hyperslice*)

ndim

Return the number of dimensions in the darray.

Returns **ndim** – The number of dimensions in the darray.

Return type integer

set_data(*attribute*, *hyperslice*, *data*)
Overwrite the contents of a darray attribute.

Parameters

- **attribute** (*integer*) – The zero-based integer index of the attribute to be overwritten.
- **hyperslice** (integer, `slice`, `Ellipsis`, or tuple containing one or more integer, `slice`, and `Ellipsis` instances.) – Defines the attribute region to be overwritten.
- **data** (*numpy.ndarray*) – Data to be written to the attribute.

shape
Return the darray shape (its size along each dimension).

Returns **shape** – The size of the darray along each dimension.

Return type tuple of integers

size
Return the darray size (total number of elements stored in the darray).

Returns **size** – The total number of elements stored in the darray.

Return type integer

slycat.hdf5.**dtype**(*type*)
Convert a string attribute type into a dtype suitable for use with h5py.

slycat.hdf5.**path**(*array*, *directory*)

slycat.hdf5.**start_arrayset**(*file*)
Create a new array set using an open hdf5 file.

Parameters **file** (`h5py.File`, required.) – An hdf5 file open for writing.

Returns **arrayset**

Return type *slycat.hdf5.ArraySet*

slycat.hyperchunks

Functionality for working with hyperchunk specifications (collections of array/attribute/slice information).

slycat.hyperchunks.**arrays**(*hyperchunks*, *array_count*)
Iterate over the arrays in a set of hyperchunks.

slycat.hyperchunks.**parse**(*string*)
Parse a string hyperchunks representation.

Parameters **string** (*string representation of a hyperchunk.*) –

Returns **hyperchunks**

Return type parsed representation of a hyperchunk.

slycat.hyperchunks.**tostring**(*value*)
Convert hyperchunks to their string representation.

slycat.table

slycat.timeseries

slycat.timeseries.segmentation

slycat.uri

Provides server-side functionality for creating, parsing, and editing Uniform Resource Identifiers (URIs) using an API that is based on the excellent URI.js library (which is available for Slycat clients).

class slycat.uri.**URI**(*value=''*)
Bases: object

Encapsulates URI creation and editing with a URI.js compatible interface.

hostname(*value=None*)
Return / assign the URI hostname.

href(*value=None*)
Return / assign the string representation of a URI.

password(*value=None*)
Return / assign the URI password.

port(*value=None*)
Return / assign the URI port.

protocol(*value=None*)
Return / assign the URI protocol.

removeQuery(*keys*, *value=None*)
Alias for URI.removeSearch().

removeSearch(*keys*, *value=None*)
Remove values from the URI search section.

scheme()
Alias for URI.protocol()

toString()
Return the string representation of the URI.

username(*value=None*)
Return / assign the URI username.

valueOf()
Return the string representation of the URI.

slycat.web.client

slycat.web.server

slycat.web.server.**checkjob**(*sid*, *jid*)
Submits a command to the slycat-agent to check the status of a submitted job to a cluster running SLURM.

Parameters

- **sid** (*int*) – Session identifier

- **jid** (*int*) – Job identifier

Returns **response** – A dictionary with the following keys: jid, status, errors

Return type dict

slycat.web.server.**create_session**(*hostname*, *username*, *password*)
Create a cached remote session for the given host.

Parameters

- **hostname** (*string*) – Name of the remote host to connect via SSH.
- **username** (*string*) – Username for SSH authentication.
- **password** (*string*) – Password for SSH authentication

Returns **sid** – A unique session identifier.

Return type string

slycat.web.server.**evaluate**(*hdf5_array*, *expression*, *expression_type*, *expression_level=0*)
Evaluate a hyperchunk expression.

slycat.web.server.**get_model_file**(*database*, *model*, *aid*)

slycat.web.server.**get_model_parameter**(*database*, *model*, *aid*)

slycat.web.server.**get_remote_file**(*sid*, *path*)
Returns the content of a file from a remote system.

Parameters

- **sid** (*int*) – Session identifier
- **path** (*string*) – Path for the requested file

Returns **content** – Content of the requested file

Return type string

slycat.web.server.**mix**(*a*, *b*, *amount*)
Linear interpolation between two numbers. Useful for computing model progress.

slycat.web.server.**post_model_file**(*mid*, *input=None*, *sid=None*, *path=None*, *aid=None*, *parser=None*, ***kwargs*)

slycat.web.server.**put_model_array**(*database*, *model*, *aid*, *array_index*, *attributes*, *dimensions*)
store array for model

Parameters

- **database** – database of model
- **model** – model as an object
- **aid** – artifact id (eg data-table)
- **array_index** – index of the array
- **attributes** – name and type in column
- **dimensions** – number of data rows

Returns

slycat.web.server.**put_model_arrayset**(*database*, *model*, *aid*, *input=False*)
Start a new model array set artifact. :param database: the database with our model :param model: the model :param aid: artifact id :param input: :return:

slycat.web.server.**put_model_arrayset_data**(*database*, *model*, *aid*, *hyperchunks*, *data*)
Write data to an arrayset artifact.

Parameters

- **database** (*database object, required*) –
- **model** (*model object, required*) –
- **aid** (*string, required*) – Unique (to the model) arrayset artifact id.
- **hyperchunks** (*string or hyperchunks parse tree, required*) – Specifies where the data will be stored, in *Hyperchunks* format.
- **data** (*iterable, required)*) – A collection of numpy.ndarray data chunks to be stored. The number of data chunks must match the number implied by the *hyperchunks* parameter.

See also:

PUT /models/(mid)/arraysets/(aid)/data

slycat.web.server.**put_model_file**(*database*, *model*, *aid*, *value*, *content_type*, *input=False*)

slycat.web.server.**put_model_inputs**(*database*, *model*, *source*, *deep_copy=False*)

slycat.web.server.**put_model_parameter**(*database*, *model*, *aid*, *value*, *input=False*)

slycat.web.server.**update_model**(*database*, *model*, ***kwargs*)
Update the model, and signal any waiting threads that it's changed. will only update model base on "state", "result", "started", "finished", "progress", "message"

slycat.web.server.authentication

slycat.web.server.authentication.**is_project_administrator**(*project*)
Return True if the current request is from a project administrator.

slycat.web.server.authentication.**is_project_reader**(*project*)
Return True if the current request is from a project reader.

slycat.web.server.authentication.**is_project_writer**(*project*)
Return True if the current request is from a project writer.

slycat.web.server.authentication.**is_server_administrator**()
Return True if the current request is from a server administrator.

slycat.web.server.authentication.**project_acl**(*project*)
Extract ACL information from a project.

slycat.web.server.authentication.**require_project_administrator**(*project*)
Raise an exception if the current request doesn't have project administrator privileges.

slycat.web.server.authentication.**require_project_reader**(*project*)
Raise an exception if the current request doesn't have project read privileges.

slycat.web.server.authentication.**require_project_writer**(*project*)
Raise an exception if the current request doesn't have project write privileges.

slycat.web.server.authentication.**require_server_administrator**()
Raise an exception if the current request doesn't have server administrator privileges.

slycat.web.server.authentication.**test_project_administrator**(*project*)
Return True if the current request has project administrator privileges.

slycat.web.server.authentication.**test_project_reader**(*project*)
Return True if the current request has project read privileges.

slycat.web.server.authentication.**test_project_writer**(*project*)
Return True if the current request has project write privileges.

slycat.web.server.authentication.**test_server_administrator**()
Return True if the current request has server administrator privileges.

slycat.web.server.database.couchdb

Slycat uses CouchDB as its primary storage for projects, models, bookmarks, metadata, and small model artifacts. For large model artifacts such as *darrays*, the CouchDB database stores links to HDF5 files stored on disk.

class slycat.web.server.database.couchdb.**Database**(*database*)
Wraps a couchdb.client.Database to convert CouchDB exceptions into CherryPy exceptions.

changes(**arguments*, ***keywords*)

delete(**arguments*, ***keywords*)

get(*type*, *id*)

get_attachment(**arguments*, ***keywords*)

put_attachment(**arguments*, ***keywords*)

save(**arguments*, ***keywords*)

scan(*path*, ***keywords*)

view(**arguments*, ***keywords*)

write_file(*document*, *content*, *content_type*)

slycat.web.server.database.couchdb.**connect**()
Connect to a CouchDB database.

Returns database

Return type *slycat.web.server.database.couchdb.Database*

slycat.web.server.engine

class slycat.web.server.engine.**SessionIdFilter**
Bases: logging.Filter

Python log filter to keep session ids out of logfiles.

filter(*record*)

slycat.web.server.engine.**start**(*root_path*, *config_file*)

slycat.web.server.handlers

slycat.web.server.handlers.**css_bundle**()

slycat.web.server.handlers.**delete_job**(*hostname*, *jid*)

slycat.web.server.handlers.**delete_model**(*mid*)

slycat.web.server.handlers.**delete_project**(*pid*)

slycat.web.server.handlers.**delete_project_cache**(*pid*)
clears all the cached images and videos for a project given a project ID :param pid: Project ID :return: status

slycat.web.server.handlers.**delete_project_cache_object**(*pid*, *key*)

slycat.web.server.handlers.**delete_reference**(*rid*)

slycat.web.server.handlers.**delete_remote**(*sid*)

slycat.web.server.handlers.**delete_upload**(*uid*)

slycat.web.server.handlers.**get_bookmark**(*bid*)

slycat.web.server.handlers.**get_checkjob**(*hostname*, *jid*)

slycat.web.server.handlers.**get_configuration_markings**()

slycat.web.server.handlers.**get_configuration_parsers**()

slycat.web.server.handlers.**get_configuration_remote_hosts**()

slycat.web.server.handlers.**get_configuration_support_email**()

slycat.web.server.handlers.**get_configuration_version**()

slycat.web.server.handlers.**get_configuration_wizards**()

slycat.web.server.handlers.**get_global_resource**(*resource*)

slycat.web.server.handlers.**get_job_output**(*hostname*, *jid*, *path*)

slycat.web.server.handlers.**get_model**(*mid*, ***kwargs*)

slycat.web.server.handlers.**get_model_array_attribute_chunk**(*mid*, *aid*, *array*, *attribute*, ***arguments*)

slycat.web.server.handlers.**get_model_arrayset_data**(*mid*, *aid*, *hyperchunks*, *byteorder=None*)

slycat.web.server.handlers.**get_model_arrayset_metadata**(*mid*, *aid*, ***kwargs*)

slycat.web.server.handlers.**get_model_file**(*mid*, *aid*)

slycat.web.server.handlers.**get_model_parameter**(*mid*, *aid*)

slycat.web.server.handlers.**get_model_statistics**(*mid*)
returns statistics on the model :param mid: model ID :return json: {

"mid":mid, "hdf5_file_size":hdf5_file_size, "total_server_data_size": total_server_data_size, "hdf5_store_size":total_hdf5_server_size, "model":model, "delta_creation_time":delta_creation_time, "couchdb_doc_size": sys.getsizeof(model)

}

slycat.web.server.handlers.**get_model_table_chunk**(*mid*, *aid*, *array*, *rows=None*, *columns=None*, *index=None*, *sort=None*)

slycat.web.server.handlers.**get_model_table_metadata**(*mid*, *aid*, *array*, *index=None*)

slycat.web.server.handlers.**get_model_table_sorted_indices**(*mid*, *aid*, *array*, *rows=None*, *index=None*, *sort=None*, *byteorder=None*)

slycat.web.server.handlers.**get_model_table_unsorted_indices**(*mid*, *aid*, *array*, *rows=None*, *index=None*, *sort=None*, *byteorder=None*)

slycat.web.server.handlers.**get_page**(*ptype*)

slycat.web.server.handlers.**get_page_resource**(*ptype*, *resource*)

slycat.web.server.handlers.**get_project**(*pid*)
returns a project based on "content-type" header :param pid: project ID :return: Either html landing page of given project or the json representation of the project

slycat.web.server.handlers.**get_project_cache_object**(*pid*, *key*)

slycat.web.server.handlers.**get_project_models**(*pid*)

slycat.web.server.handlers.**get_project_references**(*pid*)

slycat.web.server.handlers.**get_projects**(*_=None*)

slycat.web.server.handlers.**get_remote_file**(*hostname*, *path*, ***kwargs*)
Given a hostname and file path returns the file given by the path :param hostname: connection host name :param path: path to file :param kwargs: :return: file

slycat.web.server.handlers.**get_remote_host_dict**()

slycat.web.server.handlers.**get_remote_image**(*hostname*, *path*, ***kwargs*)
Given a hostname and image path returns the image given by the path :param hostname: connection host name :param path: path to image :param kwargs: :return: image

slycat.web.server.handlers.**get_remote_video**(*hostname*, *vsid*)
Given a hostname and vsid returns the video given by the vsid :param hostname: connection host name :param vsid: video uuid :return: video

slycat.web.server.handlers.**get_remote_video_status**(*hostname*, *vsid*)
Given a hostname and vsid returns the video status given by the vsid :param hostname: connection host name :param vsid: video uuid :return: json

slycat.web.server.handlers.**get_remotes**(*hostname*)
Returns {status: True} if the hostname was found in the user's session :param hostname: connection host name :return: {"status":status, "msg":msg}

slycat.web.server.handlers.**get_session_status**(*hostname*)

slycat.web.server.handlers.**get_sid**(*hostname*)
Takes a hostname address and returns the established sid value base on what is found in the users session raises 400 and 404 :param hostname: name of the host we are trying to connect to :return: sid : uuid for the session name

slycat.web.server.handlers.**get_table_metadata**(*file*, *array_index*, *index*)
Return table-oriented metadata for a 1D array, plus an optional index column.

slycat.web.server.handlers.**get_table_sort_index**(*file*, *metadata*, *array_index*, *sort*, *index*)

slycat.web.server.handlers.**get_time_series_names**(*hostname*, *path*, ***kwargs*)
Parse a time series csv for all column names :param hostname: connection host name :param path: path to csv file :param kwargs: :return: json object of column names

slycat.web.server.handlers.**get_user**(*uid*)

slycat.web.server.handlers.**get_user_config**(*hostname*)

slycat.web.server.handlers.**get_wizard_resource**(*wtype*, *resource*)

slycat.web.server.handlers.**job_time**(*nodes*, *tasks*, *size*)
gives the time in seconds recommended given job meta data :param nodes: number of hpc nodes for job :param tasks: number of tasks per node for job :param size: size of data file used in the job :return: json time in seconds as an integer {'time-seconds': 1800}

slycat.web.server.handlers.**js_bundle**()

slycat.web.server.handlers.**login**()
Takes the post object under cherrypy.request.json with the users name and password and determins with the user can be authenticated with slycat :return: authentication status

slycat.web.server.handlers.**logout**()
See if the client has a valid session. If so delete it :return: the status of the request

slycat.web.server.handlers.**model_command**(*mid*, *type*, *command*, ***kwargs*)

slycat.web.server.handlers.**model_sensitive_command**(*mid*, *type*, *command*)

slycat.web.server.handlers.**post_events**(*event*)

slycat.web.server.handlers.**post_model_arrayset_data**(*mid*, *aid*)
get the arrayset data based on aid, mid, byteorder, and hyperchunks

requires hyperchunks to be included in the json payload

Parameters

- **mid** – model id
- **aid** – artifact id

Returns stream of data

slycat.web.server.handlers.**post_model_files**(*mid*, *input=None*, *files=None*, *sids=None*, *paths=None*, *aids=None*, *parser=None*, ***kwargs*)

slycat.web.server.handlers.**post_model_finish**(*mid*)

slycat.web.server.handlers.**post_project_bookmarks**(*pid*)

slycat.web.server.handlers.**post_project_models**(*pid*)
When a pid along with json "model-type", "marking", "name" is sent with POST creates a model and saves it to the database :param pid: project ID for created model :return: json {"id" : mid}

slycat.web.server.handlers.**post_project_references**(*pid*)

slycat.web.server.handlers.**post_projects**()

slycat.web.server.handlers.**post_remote_browse**(*hostname*, *path*)

slycat.web.server.handlers.**post_remote_launch**(*hostname*)

slycat.web.server.handlers.**post_remote_videos**(*sid*)

slycat.web.server.handlers.**post_remotes**()
Given username, hostname, password as a json payload establishes a session with the remote host and attaches it to the users session :return: {"sid":sid, "status":boolean, msg:""}

slycat.web.server.handlers.**post_submit_batch**(*hostname*)

slycat.web.server.handlers.**post_upload_finished**(*uid*)
ask the server to finish the upload :param uid: upload session ID :return: status of upload

slycat.web.server.handlers.**post_uploads**()
creates a session for uploading a file to :return: Upload ID

slycat.web.server.handlers.**put_model**(*mid*)

slycat.web.server.handlers.**put_model_arrayset**(*mid*, *aid*)

slycat.web.server.handlers.**put_model_arrayset_array**(*mid*, *aid*, *array*)

slycat.web.server.handlers.**put_model_arrayset_data**(*mid*, *aid*, *hyperchunks*, *data*, *byte-order=None*)

slycat.web.server.handlers.**put_model_inputs**(*mid*)

slycat.web.server.handlers.**put_model_parameter**(*mid*, *aid*)

slycat.web.server.handlers.**put_project**(*pid*)

slycat.web.server.handlers.**put_reference**(*rid*)

slycat.web.server.handlers.**put_upload_file_part**(*uid*, *fid*, *pid*, *file=None*, *host-name=None*, *path=None*)

slycat.web.server.handlers.**require_array_json_parameter**(*name*)

slycat.web.server.handlers.**require_boolean_json_parameter**(*name*)

slycat.web.server.handlers.**require_integer_array_json_parameter**(*name*)

slycat.web.server.handlers.**require_integer_parameter**(*value*, *name*)

slycat.web.server.handlers.**require_json_parameter**(*name*)
checks to see if the parameter is in the cherrypy.request.json and errors gracefully if it is not there :param name: name of json param :return: value of the json param

slycat.web.server.handlers.**run_agent_function**(*hostname*)

slycat.web.server.handlers.**set_user_config**(*hostname*)

slycat.web.server.handlers.**tests_request**(**arguments*, ***keywords*)

slycat.web.server.handlers.**validate_table_byteorder**(*byteorder*)

slycat.web.server.handlers.**validate_table_columns**(*columns*)

slycat.web.server.handlers.**validate_table_rows**(*rows*)

slycat.web.server.handlers.**validate_table_sort**(*sort*)

slycat.web.server.hdf5

slycat.web.server.hdf5.**create**(*array*)
Create a new array in the data store, ready for writing.

slycat.web.server.hdf5.**delete**(*array*)
Remove an array from the data store.

class slycat.web.server.hdf5.**null_lock**
Bases: object

Do-nothing replacement for a thread lock, useful for debugging threading problems with h5py.

slycat.web.server.hdf5.**open**(*array*, *mode='r'*)
Open an array from the data store for reading.

slycat.web.server.hdf5.**path**(*array*)
Convert an array identifier to a data store filesystem path.

slycat.web.server.plugin

class slycat.web.server.plugin.**Manager**
Bases: object

Manages server plugin modules.

load(*plugin_path*)
Load plugin modules from a filesystem.

If the the given path is a directory, loads all .py files in the directory (non-recursive). Otherwise, assumes the path is a module and loads it.

register_directory(*type*, *init*, *user*)
Register a new directory type.

Parameters

- **type** (*string, required*) – A unique identifier for the new directory type.
- **init** (*callable, required*) – Called with parameters specified by an administrator in the server config.ini to initialize the directory.
- **user** (*callable, required*) – Called with a username to retrieve information about a user. Must return a dictionary containing user metadata.

register_marking(*type*, *label*, *badge*, *page_before=None*, *page_after=None*)
Register a new marking type.

Parameters

- **type** (*string, required*) – A unique identifier for the new marking type.
- **label** (*string, required*) – Human-readable string used to represent the marking in the user interface.
- **badge** (*string, required*) – HTML representation used to display the marking as a "badge". The HTML must contain everything needed to properly format the marking, including inline CSS styles.
- **page_before** (*string, optional*) – HTML representation used to display the marking at the top of an HTML page. If left unspecified, the badge representation will be used instead.
- **page_after** (*string, optional*) – HTML representation used to display the marking at the bottom of an HTML page. If left unspecified, the badge representation will be used instead.
- **that the page_before and page_after markup need not be self-contained, i.e. they** (*Note*) –
- **be used together to define a "container" that encloses the page markup.** (*may*) –

register_model(*type*, *finish*, *ptype=None*)
Register a new model type.

Parameters

- **type** (*string, required*) – A unique identifier for the new model type.

- **finish** (*callable, required*) – Called to finish (perform computation on) a new instance of the model.
- **ptype** (*string, optional*) – A unique page type identifier to be used as the default interface when viewing the model. Defaults to the same string as the model type.

register_model_command(*verb*, *type*, *command*, *handler*)
Register a custom request handler.

Parameters

- **verb** (*string, required*) – The HTTP verb for the command, "GET", "POST", or "PUT".
- **type** (*string, required*) – Unique category for the command. Typically, this would be a model, parser, or wizard type.
- **command** (*string, required*) – Unique command name.
- **handler** (*callable, required*) – Called with the database, model, verb, type, command, and optional keyword parameters to handle a matching client request.

register_page(*type*, *html*)
Register a new page type.

Parameters

- **type** (*string, required*) – A unique identifier for the new page type.
- **html** (*callable, required*) – Called to generate an HTML representation of the page.

register_page_bundle(*type*, *content_type*, *paths*)

register_page_resource(*type*, *resource*, *path*)
Register a custom resource associated with a page type.

Parameters

- **type** (*string, required*) – Unique identifier of an already-registered page type.
- **resource** (*string, required*) – Server endpoint to retrieve the resource.
- **path** (*string, required*) – Absolute filesystem path of the resource to be retrieved. The resource may be a single file, or a directory.

register_parser(*type*, *label*, *categories*, *parse*)
Register a new parser type.

Parameters

- **type** (*string, required*) – A unique identifier for the new parser type.
- **label** (*string, required*) – Human readable label describing the parser.
- **categories** (*list, required*) – List of string categories describing the type of data this parser produces, for example "table".
- **parse** (*callable, required*) – Called with a database, model, input flag, list of file objects, list of artifact names, and optional keyword arguments. Must parse the file and insert its data into the model as artifacts, returning True if successful, otherwise False.

register_password_check(*type*, *check*)
Register a new password check function.

Parameters

- **type** (*string, required*) – A unique identifier for the new check type.
- **check** (*callable, required*) – Called with a realm, username, and password plus optional keyword arguments. Must return a (success, groups) tuple, where success is True if authentication succeeded, and groups is a (possibly empty) list of groups to which the user belongs.

register_plugins()
Called to register plugins after all plugin modules have been loaded.

register_tool(*name*, *hook_point*, *callable*)
Register a new cherrypy tool.

Parameters

- **name** (*string, required*) – A unique identifier for the new tool.
- **hook_point** (*string, required*) – CherryPy hook point where the tool will be installed.
- **callable** (*callable object, required*) – Called for every client request.

register_wizard(*type*, *label*, *require*)
Register a wizard for creating new entities.

Parameters

- **type** (*string, required*) – A unique identifier for the wizard.
- **label** (*string, required*) – Human-readable name for the wizard, displayed in the UI.
- **require** (*dict, required*) – Requirements in order to use the wizard. Supported requirements include:
  - "action": "create" - the wizard will be used to create new objects.
  - "action": "edit" - the wizard will be used to edit existing objects.
  - "action": "delete" - the wizard will be used to delete existing objects.
  - "context": "global" - the wizard does not require any resources to run.
  - "context": "project" - the wizard requires a project to run.
  - "context": "model" - the wizard requires a model to run.
  - "model-type":[list of model types] - a model matching one of the given types is required to run the wizard.

register_wizard_resource(*type*, *resource*, *path*)
Register a custom resource associated with a wizard.

Parameters

- **type** (*string, required*) – Unique identifier of an already-registered wizard.
- **resource** (*string, required*) – Server endpoint to retrieve the resource.
- **path** (*string, required*) – Absolute filesystem path of the resource to be retrieved.

slycat.web.server.remote

Functions for managing cached remote ssh sessions.

Slycat makes extensive use of ssh and the *Slycat Agent* to access remote resources located on the high performance computing platforms used to generate ensembles. This module provides functionality to create cached remote ssh / agent sessions that can be used to retrieve data from remote hosts. This functionality is used in a variety of ways:

- Web clients can browse the filesystem of a remote host.
- Web clients can create a Slycat model using data stored on a remote host.
- Web clients can retrieve images on a remote host (an essential part of the *Parameter Image Model*).
- Web clients can retrieve video compressed from still images on a remote host.

When a remote session is created, a connection to the remote host over ssh is created, an agent is started (only if the required configuration is present), and a unique session identifier is returned. Callers use the session id to retrieve the cached session and communicate with the remote host / agent. A "last access" time for each session is maintained and updated whenever the cached session is accessed. If a session times-out (a threshold amount of time has elapsed since the last access) it is automatically deleted, and subsequent use of the expired session id will fail.

Each session is bound to the IP address of the client that created it - only the same client IP address is allowed to access the session.

class `slycat.web.server.remote.`**Session**(*client*, *username*, *hostname*, *ssh*, *sftp*, *agent=None*)
Bases: `object`

Encapsulates an open session connected to a remote host.

Examples

Calling threads must serialize access to the Session object. To facilitate this, a Session is a context manager - callers should always use a *with statement* when accessing a session:

```
>>> with slycat.web.server.remote.get_session(sid) as session:
...   print session.username
```

accessed
Return the time the session was last accessed.

browse(*path*, *file_reject*, *file_allow*, *directory_reject*, *directory_allow*)

cancel_job(*jid*)
Submits a command to the slycat-agent to cancel a running job on a cluster running SLURM.

Parameters **jid** (*int*) – Job ID

Returns **response** – A dictionary with the following keys: jid, output, errors

Return type dict

checkjob(*jid*)
Submits a command to the slycat-agent to check the status of a submitted job to a cluster running SLURM.

Parameters **jid** (*int*) – Job ID

Returns **response** – A dictionary with the following keys: jid, status, errors

Return type dict

client
Return the IP address of the client that created the session.

close()

get_file(*path*, ***kwargs*)

get_image(*path*, ***kwargs*)

get_job_output(*jid*, *path*)
Submits a command to the slycat-agent to fetch the content of the a job's output file from a cluster running SLURM.

Note that the expected format for the output file is slurm-[jid].out.

Parameters **jid** (*int*) – Job ID

Returns **response** – A dictionary with the following keys: jid, output, errors

Return type dict

get_user_config()
Submits a command to the slycat-agent to fetch the content of a user's .slycatrc file in their home directory.

Returns **response** – A dictionary with the configuration values

Return type dict

get_video(*vsid*)

get_video_status(*vsid*)

hostname
Return the remote hostname accessed by the session.

launch(*command*)
Submits a single command to a remote location via the slycat-agent or SSH.

Parameters **command** (*string*) – Command

Returns **response** – A dictionary with the following keys: command, output, errors

Return type dict

post_video(*content_type*, *images*)

run_agent_function(*wckey*, *nnodes*, *partition*, *ntasks_per_node*, *time_hours*, *time_minutes*, *time_seconds*, *fn*, *fn_params*, *uid*)
Submits a command to the slycat-agent to run a predefined function on a cluster running SLURM.

Parameters

- **wckey** (*string*) – Workload characterization key
- **nnodes** (*int*) – Number of nodes requested for the job
- **partition** (*string*) – Name of the partition where the job will be run
- **ntasks_per_node** (*int*) – Number of tasks to run on a node
- **ntasks** (*int*) – Number of tasks allocated for the job
- **ncpu_per_task** (*int*) – Number of CPUs per task requested for the job
- **time_hours** (*int*) – Number of hours requested for the job
- **time_minutes** (*int*) – Number of minutes requested for the job
- **time_seconds** (*int*) – Number of seconds requested for the job
- **fn** (*string*) – Name for the Slycat agent function
- **fn_params** (*dict*) – Additional params for the agent function

Returns **response** – A dictionary with the following keys: jid, errors

Return type dict

set_user_config(*config*)
Submits a command to the slycat-agent to set the content of a user's .slycatrc file in their home directory.

Returns **response**

Return type dict

sftp

submit_batch(*filename*)
Submits a command to the slycat-agent to start an input batch file on a cluster running SLURM.

Parameters **filename** (*string*) – Name of the batch file

Returns **response** – A dictionary with the following keys: filename, jid, errors

Return type dict

username
Return the username used to create the session.

slycat.web.server.remote.**cache_object**(*pid*, *key*, *content_type*, *content*)

slycat.web.server.remote.**check_session**(*sid*)
Return a true if session is active

If the session has timed-out or doesn't exist, returns false

Parameters **sid** (*string*) – Unique session identifier returned by *slycat.web.server.remote.create_session()*.

Returns

Return type boolean

slycat.web.server.remote.**create_session**(*hostname*, *username*, *password*, *agent*)
Create a cached remote session for the given host.

Parameters

- **hostname** (*string*) – Name of the remote host to connect via ssh.
- **username** (*string*) – Username for ssh authentication.
- **password** (*string*) – Password for ssh authentication.
- **agent** (*bool*) – Used to require / prevent agent startup.

Returns **sid** – A unique session identifier.

Return type string

slycat.web.server.remote.**delete_session**(*sid*)
Delete a cached remote session.

Parameters **sid** (*string, required*) – Unique session identifier returned by *slycat.web.server.remote.create_session()*.

slycat.web.server.remote.**get_session**(*sid*)
Return a cached remote session.

If the session has timed-out or doesn't exist, raises a 404 exception.

Parameters **sid** (*string*) – Unique session identifier returned by *slycat.web.server.remote.create_session()*.

Returns **session** – Session object that encapsulates the connection to a remote host.

Return type *slycat.web.server.remote.Session*

slycat.web.server.template

slycat.web.server.template.**render**(*path*, *context*)
Render an HTML template using Mustache syntax.

Support

For Slycat questions, comments, or suggestions, get in touch with the team at:

- https://gitter.im/sandialabs/slycat

Visit our GitHub repository for access to source code, issue tracker, and the wiki:

- http://github.com/sandialabs/slycat

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

/uploads

/users

S

A

B

C

D

E

F

G

H

I

S

T

U

V

W