slycat Documentation

Release 1.2.0

Slycat Team

Dec 05, 2018

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:

2.1 User Manual

2.1.1 Overview

Slycat™ is a web-based system for performing data analysis and visualization of potentially large quantities of remote, high-dimensional data. Slycat™ specializes in working with ensemble data. An ensemble is a group of related data sets, which typically consists of a set of simulation runs exploring the same problem space. An ensemble can be thought of as a set of samples within a multi-variate domain, where each sample is a vector whose value defines a point in high-dimensional space. To understand and describe the underlying problem being modeled in the simulations, ensemble analysis looks for shared behaviors and common features across the group of runs. Additionally, ensemble analysis tries to quantify differences found in any members that deviate from the rest of the group.

The Slycat™ system integrates data management, scalable analysis, and visualization. Results are viewed remotely on a user's desktop via commodity web clients using a multi-tiered hierarchy of computation and data storage, as shown in Figure 1. Our goal is to operate on data as close to the source as possible, thereby reducing time and storage costs associated with data movement. Consequently, we are working to develop parallel analysis capabilities that operate on High Performance Computing (HPC) platforms, to explore approaches for reducing data size, and to implement strategies for staging computation across the Slycat™ hierarchy.

Within Slycat™, data and visual analysis are organized around projects, which are shared by a project team. Project members are explicitly added, each with a designated set of permissions. Although users sign-in to access Slycat™, individual accounts are not maintained. Instead, authentication is used to determine project access. Within projects, Slycat™ models capture analysis results and enable data exploration through various visual representations. Although for scientists each simulation run is a model of real-world phenomena given certain conditions, we use the term model to refer to our modeling of the ensemble data, not the physics. Different model types often provide complementary perspectives on data features when analyzing the same data set. Each model visualizes data at several levels of abstraction, allowing the user to range from viewing the ensemble holistically to accessing numeric parameter values for a single run. Bookmarks provide a mechanism for sharing results, enabling interesting model states to be labeled and saved.

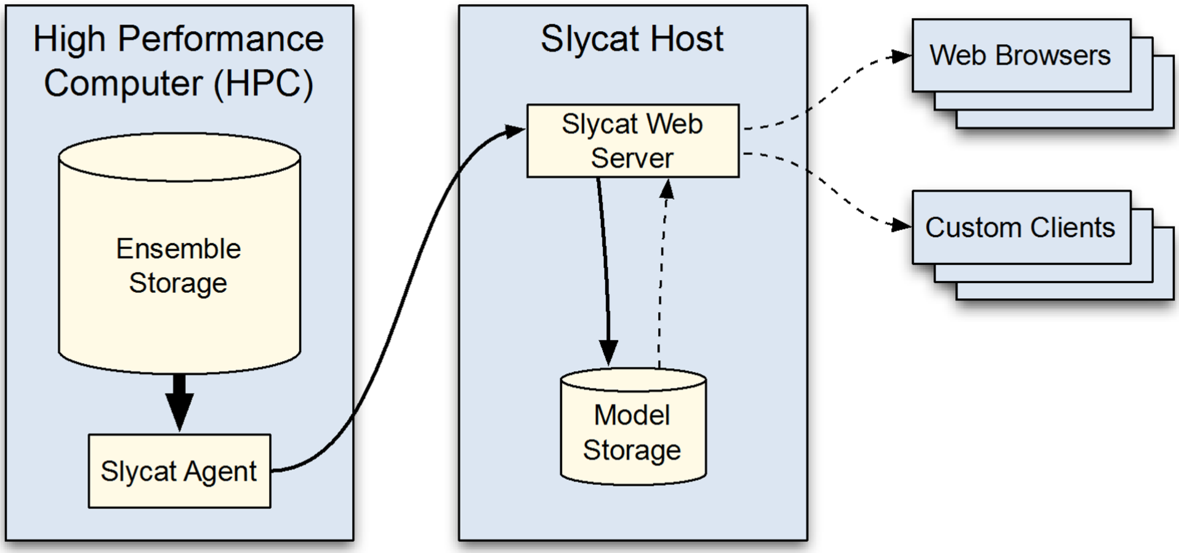


Fig. 1: **Figure 1: Slycat multi-tiered hierarchy, designed for large data analysis and exploration with minimal data movement.**

Getting Started

Slycat™ is accessed through a web browser from your desktop computer. Slycat™ currently supports Firefox, Chrome, and Safari browsers. We do not support Internet Explorer.

Since multiple Slycat™ servers are already in existence, you will need to obtain the URL for the Slycat™ server that you want to use. Enter this URL into the address bar of the browser. If the authentication mechanism for your institution relies on username and password, you will be taken to the Slycat™ login page, shown in Figure 2, where you will be prompted for your username and password. If your institution uses single sign-on, login will happen automatically and you will skip this step. Once your identity has been established, you will find yourself on the main Projects page.

Slycat™ pages exist at one of three levels: the main Projects page, an individual project page, and an individual model page. The main Projects page displays all projects which you are authorized to access. This list of projects is unique to you. Clicking on a project name will take you to that project page, which will contain a list of all models that have been generated within the project. Clicking on a model name will take you to that model page, which will display a visualization of its data. At any level, clicking on the Slycat™ logo will return you to the main *Projects* page.

The first time that you access the Slycat™ website, your projects list will probably be empty, unless someone else has already created a project and added you as a project member. Since models cannot exist outside of a project, you must first create a project (see *Project Creation*) before you can create a model. Project-specific information consists of the project name, a list of project members, a set of models, and a set of saved bookmarks for models within that project.

Slycat™ Navbar

At the top of every Slycat™ page is the Navbar. Working from left to right, we see the Slycat™ logo, a breadcrumb navigation path, and a set of colored buttons providing dropdown lists categorized by function. Depending on the type of page currently being viewed, the buttons and the contents of the dropdowns will vary. As shown in Figure 3, the Navbar for the main Projects page, the only button available is the green *Create* button for creating new projects. Since the main Projects page lies outside and above any projects, the breadcrumb navigation path points to the current page,

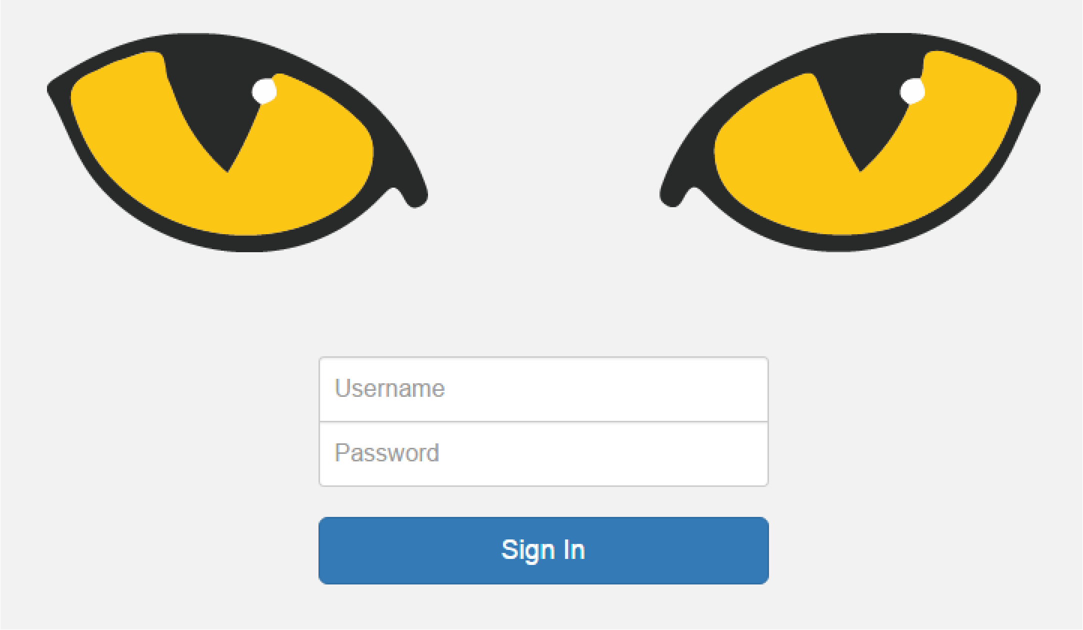


Fig. 2: **Figure 2: Slycat login page.**

which is simply labeled as *Slycat*. Figure 4 shows that for Navbars within a project or model page, there can be up to five buttons, including: *Create, Edit, Info, Bookmarks,* and *Delete*.

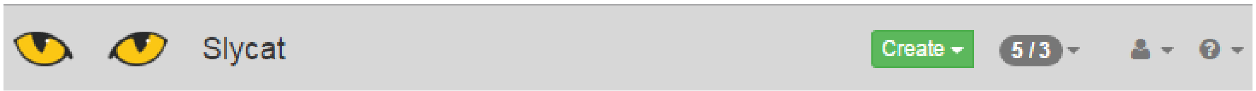


Fig. 3: **Figure 3: Slycat Navbar as seen on the main Projects page. At this level, the Navbar displays the title Slycat because we have yet to move to an individual project page.**

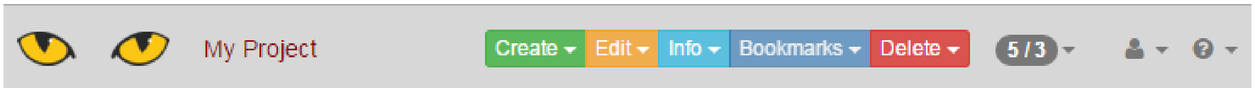
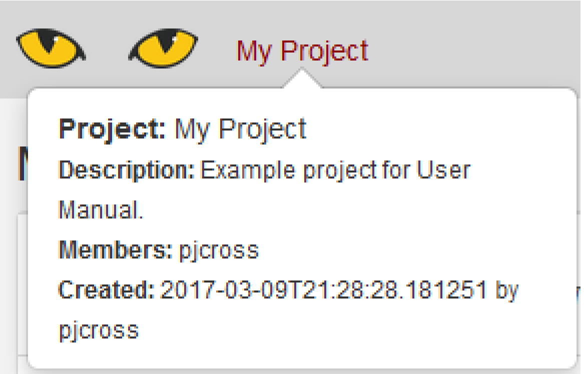


Fig. 4: **Figure 4: Navbar at the individual project page level. Here the name of the project is 'My Project'. Note that the Bookmarks button is hidden until at least one bookmark has been created.**

As you move between pages at various levels, the breadcrumb path in the Navbar will change to reflect your current location. The path has the format *Project Name / Model Name*. The path can be used to navigate within the hierarchy. Clicking on the *Project Name* will take you to that project's page with its list of associated models. Hovering over the *Project Name* will display the project description, the project members, the date of creation, and who created it (Figure 5). Similarly, hovering over the *Model Name* will display the model description, the date of creation, and who created it.

Projects

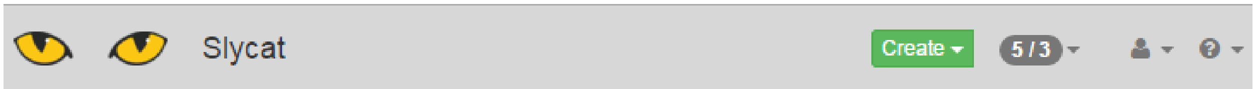
Project-specific information consists of the project name, a list of project members, a set of models, and a set of saved bookmarks for models within that project.



Project: My Project
Description: Example project for User Manual.
Members: pjcross
Created: 2017-03-09T21:28:28.181251 by pjcross

Fig. 5: **Figure 5: Hovering over the project name will display more detailed project information.**

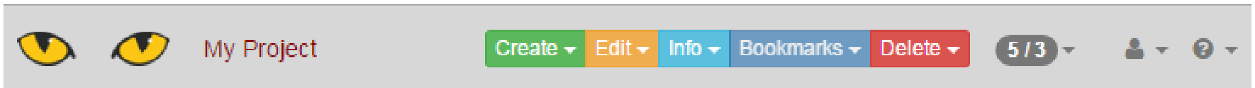
Project Creation



Projects are created by clicking the green *Create* button on the main Projects page and selecting *New Project* from the dropdown that appears. A dialog will pop up, providing editable regions for you to enter a *Name* and an optional *Description*. This brief text field allows you to provide more detailed comments or notes beyond the project name. Clicking the *Finish* button in the lower right corner creates the project and takes you to the newly created project page; clicking the *Cancel* button in the lower left corner aborts project creation.

Once the project is created, you will find yourself on the empty project page. As the project creator, you are automatically assigned the role of *Administrator* for that project (although there can be multiple project members with Administrator roles). A series of buttons appears to the right of the project name. Since you have an Administrator role, there will initially be four buttons: *Create, Edit, Info,* and *Delete*. Only project administrators can edit or delete the project. Otherwise, there will only be two buttons: *Create* and *Info*.

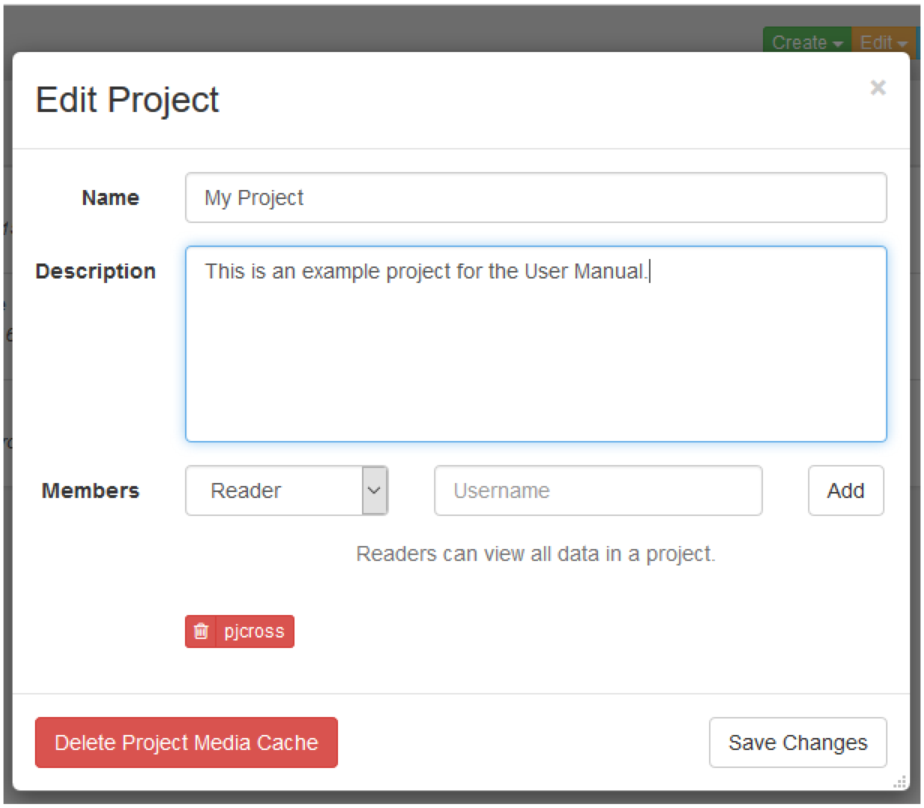
Editing Projects



Clicking on the yellow *Edit* button on a project page and selecting *Edit Project* from the dropdown list provides a means to change the project *Name, Description,* or project *Members*. An *Edit Project* popup will appear with the current project information (Figure 6). In a newly created project, the membership list consists solely of the project creator assigned the role of *Administrator*. The username of the creator will be shown within a red button (buttons are color-coded according to role and red is used for an *Administrator*) at the bottom.

There are three different roles that project members can have: *Reader, Writer,* and *Administrator*, whose buttons are color-coded blue, yellow, and red, respectively. *Readers* can view all data in a project, but they cannot create new models, modify existing models, or delete models. *Writers* can both view and modify the contents of a project, but they are unable to add new project members or edit the project name or description. *Administrators* have full access to all aspects of the project, including adding new project members or deleting the project itself.

To add a project member, select a role from the dropdown list to the right of *Members* and type in the person's username. Note that the username is checked against a list of legitimate usernames and will be rejected if it is not found. If the username is found, a popup will translate the username into the person's full name and verify both the



Edit Project

Name: My Project

Description: This is an example project for the User Manual.

Members: Reader

Readers can view all data in a project.

Fig. 6: **Figure 6:** *Edit Project* **dialog allows you to change the project name, add or change a description, and add, remove, or change the roles of project members.**

identity and the role selected. Click *OK* if both the person and role are as you intended, or *Cancel* if they are not. Now an additional button, color-coded by role and enclosing the newly added member's username, will appear in the member list below. Although the new member now appears to be in the project member list, this action has not been saved and will be discarded unless the *Save Changes* button is pressed.

To remove project members, click on the trashcan icon next to the name of the member to be removed. To change the role of a project member, add them as you would a new project member (you do not need to remove them first), but with the revised role. Note that as an *Administrator*, you have the power to delete yourself or reduce the level of your role (thereby losing your Administrator privileges), which is why we require you to first click the *Save Changes* button before we finalize any changes. If you find that you have accidently made a change that you do not want to execute, pressing the **X** button in the upper right corner of the *Edit Project* dialog cancels the edit and keeps the previous project state (*Name, Description, Members,* and *member roles*) intact.

Project Info

To see a non-editable version of a project's information, click on the cyan *Info* button on the project page and select *Project Details* from the dropdown. A popup will display the *Name, Description,* and project *Members* list. Click *Close* when you are finished viewing it. Note that this same information can be seen by hovering over the project name in the breadcrumb navigation path.

Deleting Projects

To remove a project, including **ALL ITS MODELS AND DATA**, click the red *Delete* button from within the project page of the project that you wish to delete. Select *Delete Project* from the dropdown. Note that only members with Administrator rights may delete the project. Project deletion is an irreversible operation, so deletion requires confirmation through a popup that asks if you really want to delete that project and all models within it. Press the red *Delete Project* button to confirm deletion, or the **X** button in the upper right corner of the dialog to cancel the operation and keep the project.

Models

In Slycat™, models combine analysis and visualization. Slycat™ provides three different types of models: Canonical Correlation Analysis (CCA), Parameter Space, and Time Series. The heart of every model is a data table. For each model type, there are predefined sets of linked views that provide different representations of the analysis results. Generally, the visualization for each model consists of three different representations, each showing the ensemble at a different level of abstraction. The highest-level view seeks to display the ensemble in a holistic manner. It seeks to show what high-level behaviors or trends can be seen across most, if not all, of the simulation runs. Slycat™ currently provides views showing correlations between inputs and outputs, or similarities between results. The intermediate-level view presents individually distinguishable runs in the context of the group, showing how well each member aligns with the high-level view of shared ensemble traits. The low-level view enables you to drill down to the raw data values, both to input parameters and to the results from individual runs.

Each model has a *Name*, a *Marking*, and an optional *Description*. Marking choices are defined as part of the server configuration, so they are specific to the institution that hosts the server. The intent is for Slycat™ to facilitate clear labeling of data sensitivity through explicit choice of marking. The marking appears as part of the model description on the Project page list, plus it is shown in both header and footer bars when visualizing the model.

Creating Models

Models are created by clicking the green *Create* button from within a project page and selecting one of the model types from the dropdown list. The information needed to create a model varies depending on which model you choose, so a popup dialog specific to the selected model will step you through entering the necessary information for that type (the details for each are covered below). Model creation can be aborted at any stage by clicking the *X* button in the upper right corner of the popup.

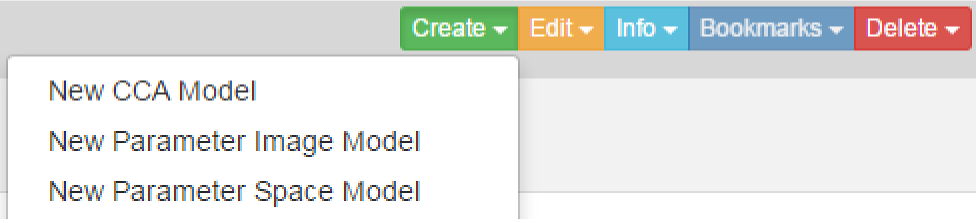


Fig. 7: **Figure 7:** ***Create*** **dropdown list of model choices, as seen from a project page.**

Editing Models

Clicking on the yellow *Edit* button on a model page and selecting Name Model from the dropdown list allows you to change the model *Name, Description,* and *Marking*. A *Name Model* dialog will popup with the current model information. Click *Save Changes* to modify the model description on the server, or click the *X* button in the upper right of the popup to abort the operation.

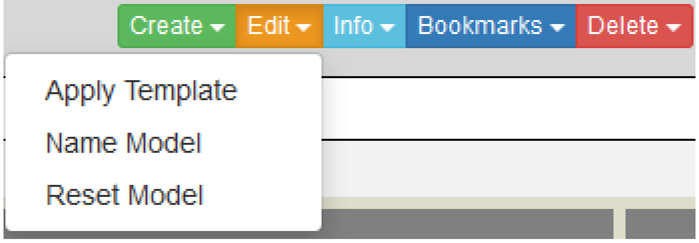


Fig. 8: **Figure 8:** ***Edit*** **dropdown list, as seen from a model page.**

Reset Model

As you interact with a model and change various aspects of the visualization, Slycat™ keeps track of the current model state. If you leave that model and return to it later, Slycat™ will resume with the model rendered according to the most recent configuration state. Note that this is only true if you are returning to a model on the same computer using the same browser that you previously used to view it. However, sometimes you might want to start over with the default settings to produce the initial visualization. Clicking on the yellow *Edit* button on a model page and selecting *Reset Model* from dropdown list will return the model state to its initial configuration.

Deleting Models

To remove a model and ALL ITS DATA, click the red *Delete* button from within the model page of the model that you wish to delete. Select *Delete Model* from the dropdown. Model deletion is an irreversible operation, so deletion requires confirmation through a popup asking if you really want to delete that model. Press the red Delete Model button to confirm deletion, or click the *X* button in the upper right of the popup to abort the operation and keep the model.

View Regions

The inner part of each model's visualization is subdivided into several views or regions, each separated from adjacent regions by a thick gray line. As you move the mouse over one of these region dividers, a double-headed arrow cursor perpendicular to the divider replaces the normal arrow cursor, the line extent (all but a darker gray center section) highlights in yellow, and the tooltip *Resize* pops up. If you click and drag the divider while this is enabled, the divider will move until you release it, resizing the regions on either side to reflect proportional changes created by the new divider location. The divider can be dragged to the very edge, effectively hiding the view.

Alternately, if you move the mouse over the darker center section of a divider, the center section highlights in yellow, the icon become a hand with a pointing finger, and the tooltip *Close* appears. Clicking the mouse button now will collapse one of the two adjacent regions. It collapses the region that is closer to the edge of the browser window. Clicking a second time on that same divider (now positioned along the edge of the model visualization) will restore the previous layout.

Download Data Table

Since data tables are at the core of each model type, all models provide a table download operation. The download can take one of several forms: download the entire table, download only selected items, or download only visible items. As will be described later (see *Selecting Points* and *Filtering*), selection and filtering can be used to divide the data into sets using two approaches, either through highlighting or through visibility. Highlighting and visibility are independently defined sets, so selected items are not necessarily visible.

This functionality can be used to download a table or a table subset to your desktop, which can then be used to generate a new model. For example, if you had an ensemble where some of the runs failed to terminate properly, you could filter those runs out and download the subset of runs that finished correctly. Then you could use that subset to generate new models where the failed runs are not biasing the analysis results. Or alternately, you could download the subset that failed and use that table to create a Parameter Space model to explore what the failures have in common.

Color Themes

Color is used extensively in Slycat™ to encode information of various types. In the table views that appear within each model, green columns are associated with input variables, lavender designates output variables, and unspecified variables are not colored (they are rendered using an off-white).

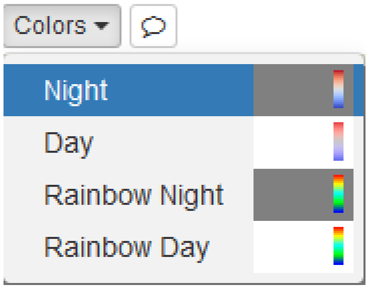


Fig. 9: **Figure 9: Dropdown list of color theme choices from the *Colors* button.**

Slycat™ provides a set of predefined color themes, which are individually assigned to each model. A color theme consists of a bundled scatterplot background color and color palette for mapping numeric values to color-coded objects in Slycat™ views. Below the Navbar on the model page, there is an additional row of model-specific buttons. To change the current color theme, click the *Colors* button. As shown in Figure 9, there are four color themes available in the dropdown: *Night, Day, Rainbow Night,* and *Rainbow Day*. *Night* is the default choice. *Night* has a gray background and uses a diverging palette that maps low values to blue and high values to red, transitioning through white for values in the middle of the range[1]. *Day* has a white background and a similar blue to red mapping, though the palette is slightly shifted to transition through gray instead of white to enable you to distinguish points in the middle of the range from the background. *Rainbow Night* has a gray background and a conventional rainbow palette. *Rainbow Day* has a white background and a conventional rainbow palette. Although we provide *Rainbow* themes, we discourage their use since color order in the middle of the range is not intuitive.

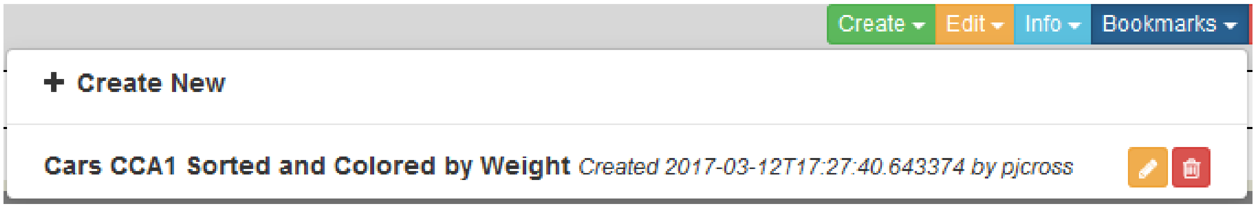
Bookmarks

Each time you interact with a model, changes in model state are preserved in a Bookmark and the URL in your browser's address bar is modified to incorporate the latest bookmark id. The id links to a description of the model state that is stored on the Slycat™ server. Although the contents of a bookmark are model dependent, all bookmarks capture the current visualization state so that it can be reproduced (though parameters such as view region sizes are not saved, since they are device dependent). Examples of the types of information stored in a bookmark include color-encoding, highlighted selections, filter values and limits, pinned media selections, and hidden points.

Bookmarking enables many useful functions. Dragging and dropping the URL from the address bar into an email, you can share a specific state of the visualization with other project members. If you save model pages as browser bookmarks, you can archive and recall interesting model states, though you will be limited to viewing them on the machine where you created them. The current bookmark id is stored locally in your browser's cache. This enables you to pick up where you left off when you begin new session with a previously viewed model.

Within Slycat™ there is the concept of a saved *Bookmark*. This *Bookmark* is a persistent link to a model state that you explicitly save within a project. Slycat™ saves the current bookmark id along with a label that you provide. This provides a convenient, machine-independent mechanism for saving exploratory results. *Bookmarks* can be used to remember visualizations that reveal interesting patterns, to share findings with other team members, or to create a flipbook-style narrative for a demonstration.

To create a *Bookmark*, click the blue *Bookmarks* button from within a model page and select *Create New* (Figure 10). A *Create Saved Bookmark* popup will appear (Figure 11). Type in a *Name* and click the *Save Bookmark* button on the right to save it, or click on the *X* button in the upper right corner of the dialog to abort the operation. The *Bookmarks* button dropdown will display a list of all the bookmarks associated with the project. If you are on a model page, *Bookmarks* associated with that model are listed at the top, while those for other models appear below, each labeled with their model type. Clicking on a *Bookmark* in the list takes you to the associated model and visualizes it according to the saved state. To modify the name of a bookmark, click on the yellow pencil icon. To delete a bookmark, click on its red trashcan.

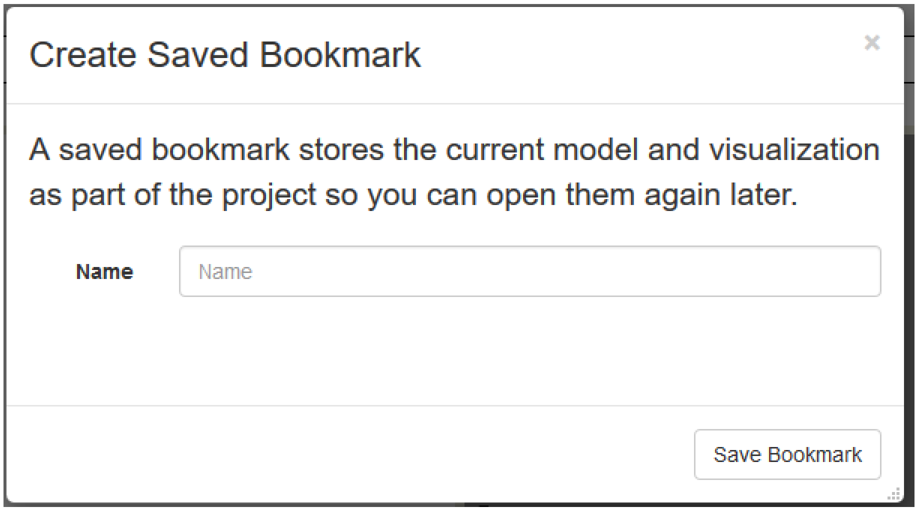


Cars CCA1 Sorted and Colored by Weight *Created 2017-03-12T17:27:40.643374 by pjcross*

Fig. 10: **Figure 10:** ***Bookmarks*** **dropdown, including one previously saved bookmark.**

Note that changes to the *View Regions* are not currently preserved in bookmarks. Consequently, when the layout has been modified prior to the model being bookmarked, visualizing the *Bookmark* will render the model using the default *View Regions* layout.

[1] Moreland, K., Diverging Color Maps for Scientific Visualization. Advances in Visual Computing, vol. 5876, pp. 92-103. Springer, Berlin (2009).



Create Saved Bookmark

A saved bookmark stores the current model and visualization as part of the project so you can open them again later.

Name: Name

Fig. 11: **Figure 11: Example of a *Create Saved Bookmark* dialog.**

Templates

Templates are essentially *Bookmarks*, but they lack an associated model. *Templates* provide a means for applying the same visualization state to another model of that same model type. Note that the similarity between the template's original data and the new data set will determine the similarity of the resulting visualization. For dissimilar data, some portions of the saved state may not be applicable (in which case those attributes are ignored), or the results may significantly differ from your expectations.

To create a *Template*, click the green *Create* button, then select *Template* from the dropdown list. A *Create Template* popup will appear. Type in a *Name* and click the *Save Template* button on the right to save it, or click the *X* button to abort the operation.

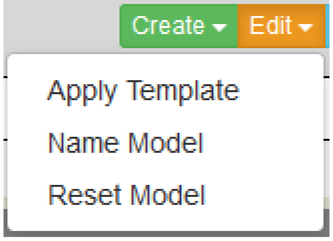
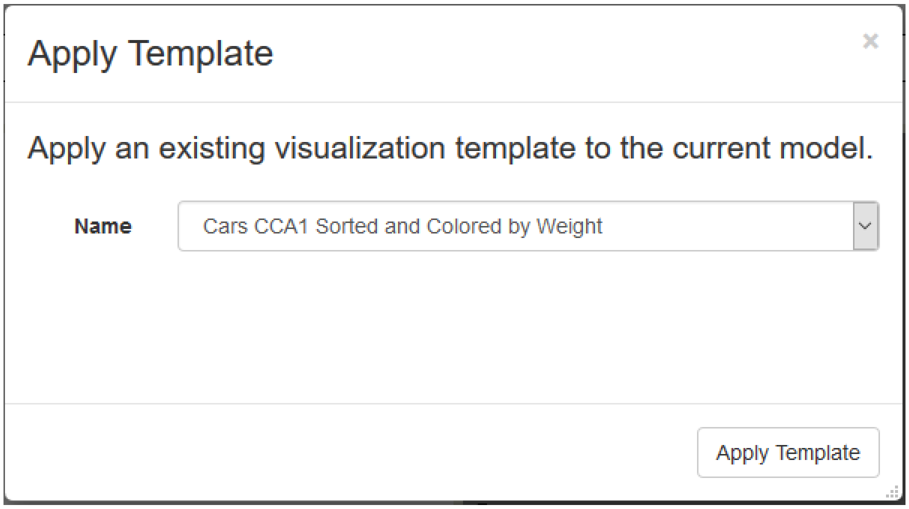


Fig. 12: **Figure 12: *Edit* dropdown for models, providing *Apply Template* functionality.**

To apply a *Template* to a model, from within that model's page click the yellow *Edit* button and select *Apply Template* from the dropdown list (Figure 12). The *Apply Template* popup will appear. Select the name of the template that you wish to use from the list and click the *Apply Template* button on the right to execute, or click the *X* button to abort the operation. Note that bookmarks can also be used as templates, so they are included in the list of available templates (Figure 13). However, they are not interchangeable. *Templates* will not appear in the *Bookmarks* dropdown list, since they cannot be rendered without an associated model.



Apply Template

Apply an existing visualization template to the current model.

Name: Cars CCA1 Sorted and Colored by Weight

Fig. 13: **Figure 13:** ***Apply Template*** **dialog. Note that a bookmark appears in the template list.**

2.1.2 Canonical Correlation Analysis Model

Canonical Correlation Analysis (CCA) was first proposed by Hotelling in 1936[1]. Because CCA finds correlations between two multivariate data sets, CCA data structures are a good fit for exploring relationships between the input and output variables found in ensemble data sets (such as those generated for sensitivity studies, uncertainty quantification, model tuning, or parameter studies). Slycat™ uses CCA to model the many-to-many relationships between multiple input parameters and multiple output metrics. CCA is a linear method and a direct generalization of several standard statistical techniques, including Principal Component Analysis (PCA), Multiple Linear Regression (MLR), and Partial Least Squares (PLS)[2][3].

CCA operates on a table of scalar data, where each column is a single input or output variable across all runs, and each row consists of the values for each of the variables in a single simulation. Slycat™ requires the number of rows (samples) to be greater than the minimum variable count of the inputs or the outputs. A more meaningful result will be obtained if the ratio of runs to variables is ten or more. Additionally, columns cannot contain the same value for all runs. Slycat™ will reject such columns from being included in the CCA analysis, since they contribute no differentiating information. CCA cannot handle rows with *missing data*, *Inf*, *-Inf*, *NAN*, or *NULL* values. Slycat™ will remove rows from the analysis if any of the values in either the input or output variable sets include such data. However, if the bad values are only in columns that are not analysis variables, the row will be used.

For a concise description of CCA, we need the following definitions. Given n samples (n rows in the table), the input variables (presumed to be independent) will be referred to as the set $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_\mathrm{n}\}$ and the output (dependent) variables as the set $Y = \{\mathbf{y}_1, \ldots, \mathbf{y}_\mathrm{n}\}$. Each vector $\mathbf{x}_\mathrm{i}$ has p_1 components and each vector $\mathbf{y}_\mathrm{j}$ has p_2 components. CCA attempts to find projections $\mathbf{a}$ and $\mathbf{b}$ such that $R^2 = \mathrm{corr}\,(\mathbf{a}^\mathrm{T}X, \mathbf{b}^\mathrm{T}Y)$ is maximized, where corr (•,•) denotes the standard Pearson correlation.

The vectors $\mathbf{a}^\mathrm{T}X$ and $\mathbf{b}^\mathrm{T}Y$ are known as the first pair of canonical variables. Further pairs of canonical variables are orthogonal and ordered by decreasing importance. In addition to the canonical variables, the R^2 value for each variable pair is obtained, and various statistics can be computed to determine the significance of the correlation. A common statistic used in this context is the p-value associated with Wilks' λ[4]. Slycat™ provides both R^2 and p-values for each

[1] Hotelling, H., Relations Between Two Sets of Variates. *Biometrika*, 28, 321-377 (1936).

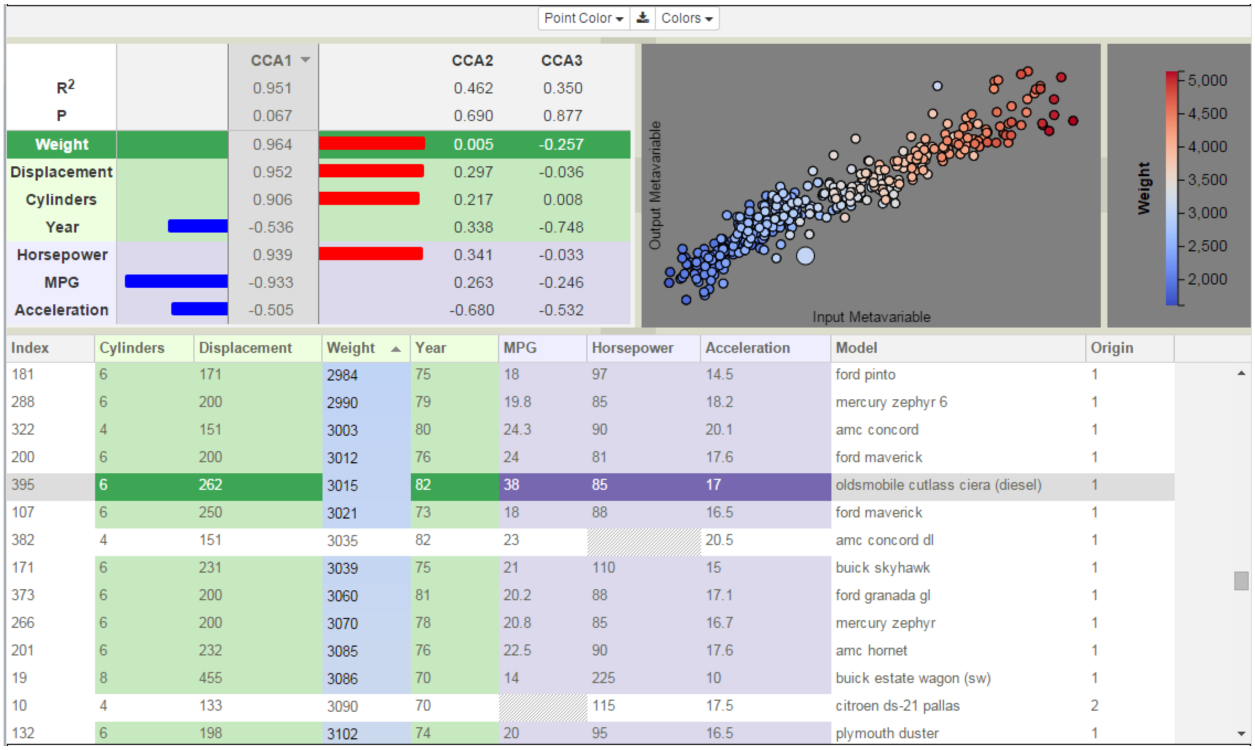
[2] Adams, B.M., Ebeida, M.S., Eldred, M.S., Jakeman, J.D., Swiler, L.P., Bohnhoff, W.J., Dalbey,K.R., Eddy, J.P., Hu, K.T., Vigil, D.M., Bauman, L.E., and Hough, P.D., *Dakota, a Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 5.3.1 User's Manual*. Tech. Rep. SAND2010-2183, Sandia National Laboratories (2013).

[3] Ayachit, U., Bauer, A., Geveci, B., O'Leary, P., Moreland, K., Fabian, N., and Mauldin, J., *ParaView Catalyst: Enabling In Situ Data Analysis and Visualization*, Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV2015), pp. 25-29, ACM, New York, NY (2015).

[4] Krzanowski, W. J., *Principles of Multivariate Analysis. A User's Perspective*. Oxford University Press, London (1988).

canonical component as part of the Correlation View (see the figure below). Note that these statistics assume that the data is normally distributed. If your data does not follow a normal distribution, be aware that these statistics will be suspect and adjust your interpretation of the results accordingly.

Once the canonical variables are determined, they can be used to understand how the variables in X are related to the variables in Y, although this should be done with some caution. The components of the vectors **a** and **b** can be used to determine the relative importance of the corresponding variables in X and Y. These components are known as canonical coefficients. However, the canonical coefficients are considered difficult to interpret and may hide certain redundancies in the data. For this reason, Slycat™ visualizes the canonical loadings, also known as the structure coefficients. The structure coefficients are generally preferred over the canonical coefficients because they are more closely related to the original variables. The structure coefficients are given by the correlations between the canonical variables and the original variables (e.g. corr $(\mathbf{a}^T X, X)$ and corr $(\mathbf{a}^T Y, Y)$). These are calculated using Pearson's correlation between each column of X or Y and the corresponding canonical variable.



	CCA1	CCA2	CCA3
R^2	0.951	0.462	0.350
P	0.067	0.690	0.877
Weight	0.964	0.005	-0.257
Displacement	0.952	0.297	-0.036
Cylinders	0.906	0.217	0.008
Year	-0.536	0.338	-0.748
Horsepower	0.939	0.341	-0.033
MPG	-0.933	0.263	-0.246
Acceleration	-0.505	-0.680	-0.532

Index	Cylinders	Displacement	Weight	Year	MPG	Horsepower	Acceleration	Model	Origin
181	6	171	2984	75	18	97	14.5	ford pinto	1
288	6	200	2990	79	19.8	85	18.2	mercury zephyr 6	1
322	4	151	3003	80	24.3	90	20.1	amc concord	1
200	6	200	3012	76	24	81	17.6	ford maverick	1
395	6	262	3015	82	38	85	17	oldsmobile cutlass ciera (diesel)	1
107	6	250	3021	73	18	88	16.5	ford maverick	1
382	4	151	3035	82	23	▨	20.5	amc concord dl	1
171	6	231	3039	75	21	110	15	buick skyhawk	1
373	6	200	3060	81	20.2	88	17.1	ford granada gl	1
266	6	200	3070	78	20.8	85	16.7	mercury zephyr	1
201	6	232	3085	76	22.5	90	17.6	amc hornet	1
19	8	455	3086	70	14	225	10	buick estate wagon (sw)	1
10	4	133	3090	70	▨	115	17.5	citroen ds-21 pallas	2
132	6	198	3102	74	20	95	16.5	plymouth duster	1

Fig. 14: **Canonical components are shown in the Correlation View in the upper left.**

Cars Example Data Set

In the following sections, we will use the *cars* data set[1] to illustrate model creation and CCA in general. *Cars* is not an ensemble of simulation data. Instead, it is a list of features for 406 automobiles built between 1970 and 1982. Selecting attributes which describe a car's physical system and labeling them as inputs, while grouping the performance-based variables as outputs, we can see the relationships between design choices and various performance metrics. Since CCA can only evaluate correlations between numeric variables, the analysis omits two columns, *Model* and *Origin*, which are string and categorical variables, respectively. Also note that *Acceleration* is a variable measuring the number of seconds required to accelerate from 0 to 60 mph, so lower values represent greater acceleration.

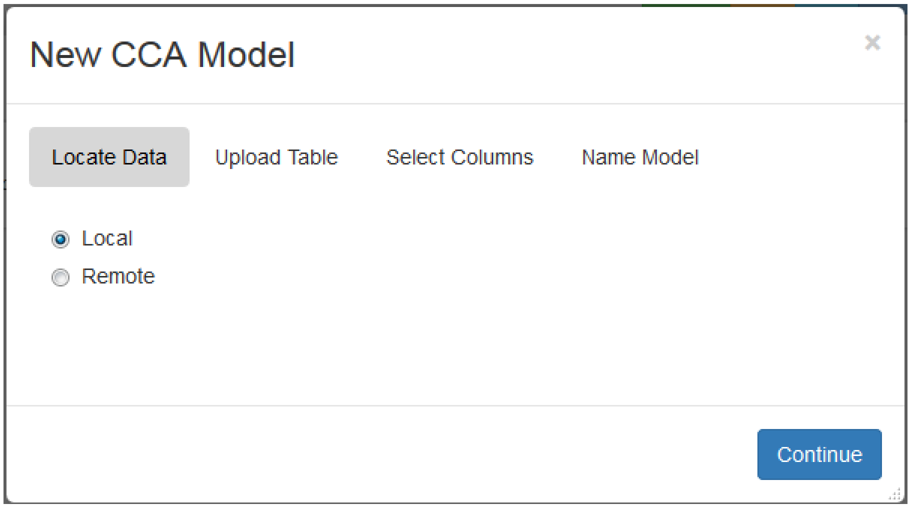
This data set provides an intuitive introduction to CCA because most people already have some idea of how a car's manufacturing and performance features are related. Increasing weight, displacement, and number of cylinders all

[1] Donoho, D. and Ramos, E., *PRIMDATA: Data Sets for Use With PRIM-H*, http://lib.stat.cmu.edu/datasets/cars.desc and http://lib.stat.cmu.edu/datasets/cars.data (1982)

represent larger engines, which are in turn correlated with greater horsepower, lower miles per gallon (MPG), and faster acceleration. Due to the Arab oil embargos during the model years in this data set, engine sizes decreased over time to facilitate increased MPG.

Creating a CCA Model

Slycat™ accepts two file formats for table data, either Comma Separated Value (CSV) files, or Dakota tabular files (generated by Dakota[1], software which is frequently used to generate ensemble data sets). If your data is not currently in one of these two formats, Excel can be used to create CSV files from most common table formats. Note that if output metrics have been created separately in a post-processing step, they will need to be integrated with the inputs into a single file prior to model creation. In a CSV file, we expect to see only a single row of header information consisting of the column names.



● Local
○ Remote

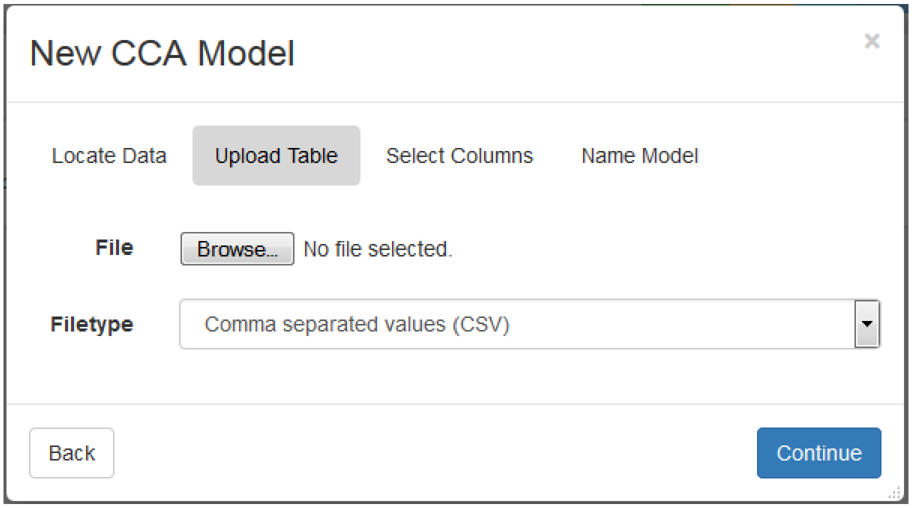
Fig. 15: **Figure 14: Popup dialog in the CCA model creation wizard.**

From your project page, click on the green *Create* button and select *New CCA Model* from the dropdown list. A dialog for walking you through the process will then pop up, as shown in Figure 14. The first page of the model creation wizard identifies whether the table is located on the local machine or whether the data is held on a remote machine. Select *Local* or *Remote*, followed by *Continue* to advance to the next page of the wizard.

Local Files

As shown in Figure 15, if you selected *Local*, the next page will display two fields, *File* and *Filetype*. Adjacent to *File*, is the button *Browse*. Clicking *Browse* brings up a local file browser, which you can use to navigate to the location of your data table. After selecting a file, the file browser closes and the name of your selected file appears to the right of the *Browse* button, as shown in Figure 16. Depending on the format of the selected file, select either *CSV* or *Dakota tabular* from the *Filetype* dropdown, followed by *Continue* to read the file. Note, you can change your mind and read the table from a *Remote* host by clicking the *Back* button to return to the previous page.

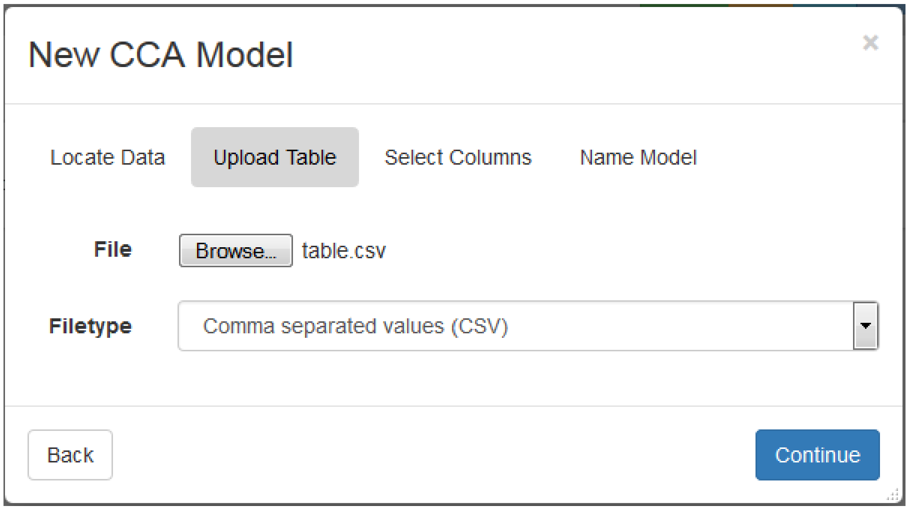
[1] Adams, B.M., Ebeida, M.S., Eldred, M.S., Jakeman, J.D., Swiler, L.P., Bohnhoff, W.J., Dalbey, K.R., Eddy, J.P., Hu, K.T., Vigil, D.M., Bauman, L.E., and Hough, P.D., *Dakota, a Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 5.3.1 User's Manual.* Tech. Rep. SAND2010-2183, Sandia National Laboratories (2013).



File: No file selected.

Filetype: Comma separated values (CSV)

Fig. 16: **Figure 15: Local file upload dialog (with no file selected) in CCA model creation wizard.**



File: table.csv

Filetype: Comma separated values (CSV)

Fig. 17: **Figure 16: Selected file, table.csv, shown in CCA model creation dialog.**

Remote Files

As shown in Figure 17, if you select *Remote*, the *Choose Host* page enables you to log into a remote machine through Slycat™. First select a machine from the dropdown list, which is revealed by clicking the triangle to the right of *Hostname*. If the machine you wish to access is not on the list, type the machine name into the field. The name will be remembered and used as the default host for the next time. *Username* defaults to the username that you provided when logging into Slycat™, but this field can be manually edited if desired. Finally, enter your *Password* and click the *Continue* button in the lower right to connect to the remote host.

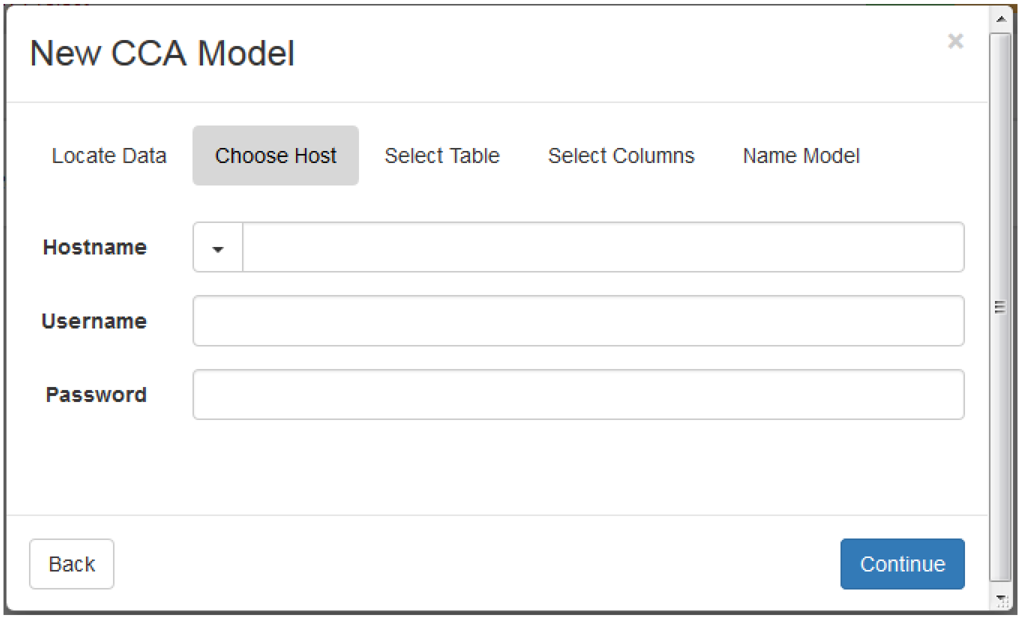


Fig. 18: **Figure 17: Remote system login for table ingestion in CCA model creation wizard.**

Once you are connected, the model creation wizard will display a remote file browser. If you have previously accessed this machine through Slycat™, the browser directory will be initialized to your last location. Otherwise, the browser default directory will be the machine's root directory. There are two methods for navigating the remote directory structure to find your data: (1) if you know the full directory path, type it directly into the field at the top of the page (shown in Figure 18) and click the *Go* button;

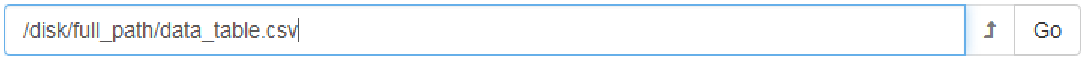

/disk/full_path/data_table.csv

Fig. 19: **Figure 18: File path field in remote file browser.**

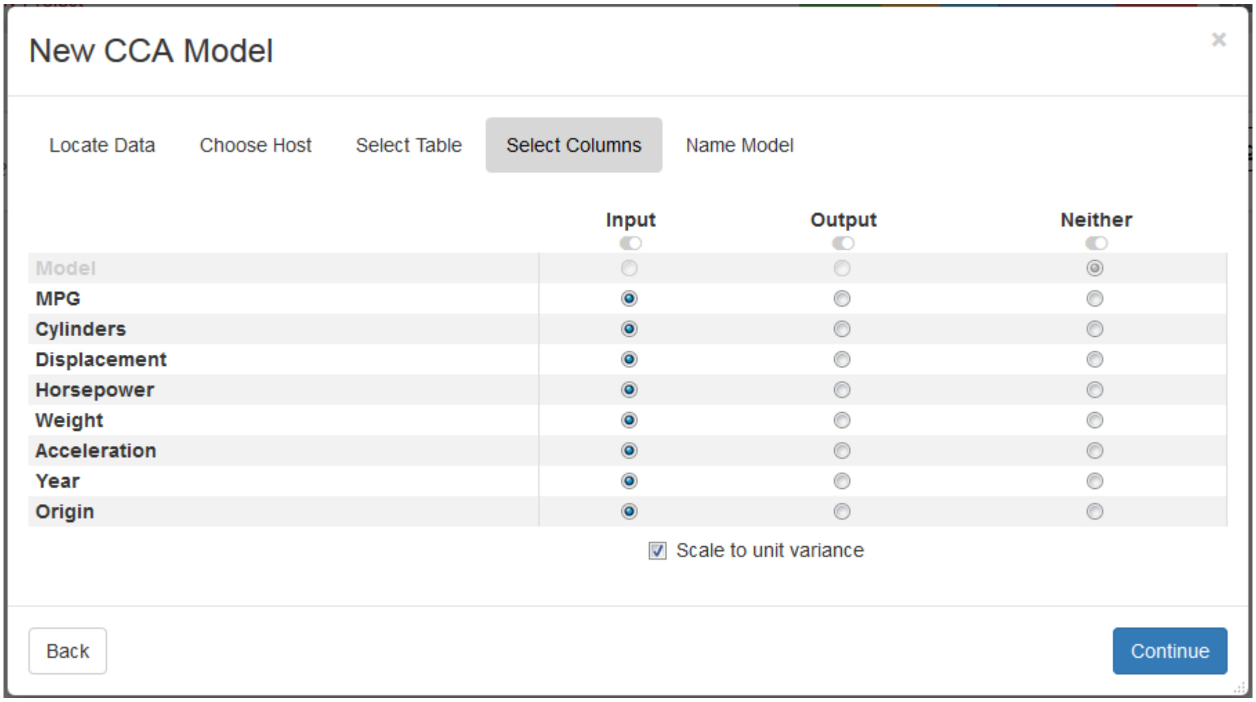
or (2) move up and down the directory hierarchy by clicking on folders in the list. Clicking on 📁 (the folder labeled '..' in the file list), or on the *Up Directory* button ↥ (to the right of the file path) moves you up a level in the hierarchy, while clicking on a named folder moves you down a level. Once you are in the directory that contains your table data, click on the file to select it. Ensure that the format shown in the *Filetype* dropdown matches the selected file's type, then click *Continue* to read the file. Note, you can change your mind and read the table from your *Local* host by clicking the *Back* button to return to the previous page.

Select Columns

Once the table has been read, either from a *Local* or a *Remote* source, the *Select Columns* page displays a list of the table's variable (column) names and asks you to categorize them as *Input*, *Output*, or *Neither* for the CCA analysis.

Variables marked as *Neither* are omitted from the analysis altogether. Since CCA requires numeric values, strings are automatically excluded from consideration.

Looking at the variables in our *Cars* example in Figure 19, the faded variable name at the top of the list, *Model*, is the name for each car model. Because its values are all strings, it has been automatically set to *Neither* and cannot be changed. Although *Origin* is a numeric variable, the numbers are encoding categorical labels whose value order has no meaning (US = 1, Europe = 2, Asia =3). Because the values have no ordinal interpretation, *Origin* should also be removed from the analysis.



	Input	Output	Neither
Model	○	○	●
MPG	●	○	○
Cylinders	●	○	○
Displacement	●	○	○
Horsepower	●	○	○
Weight	●	○	○
Acceleration	●	○	○
Year	●	○	○
Origin	●	○	○

☑ Scale to unit variance

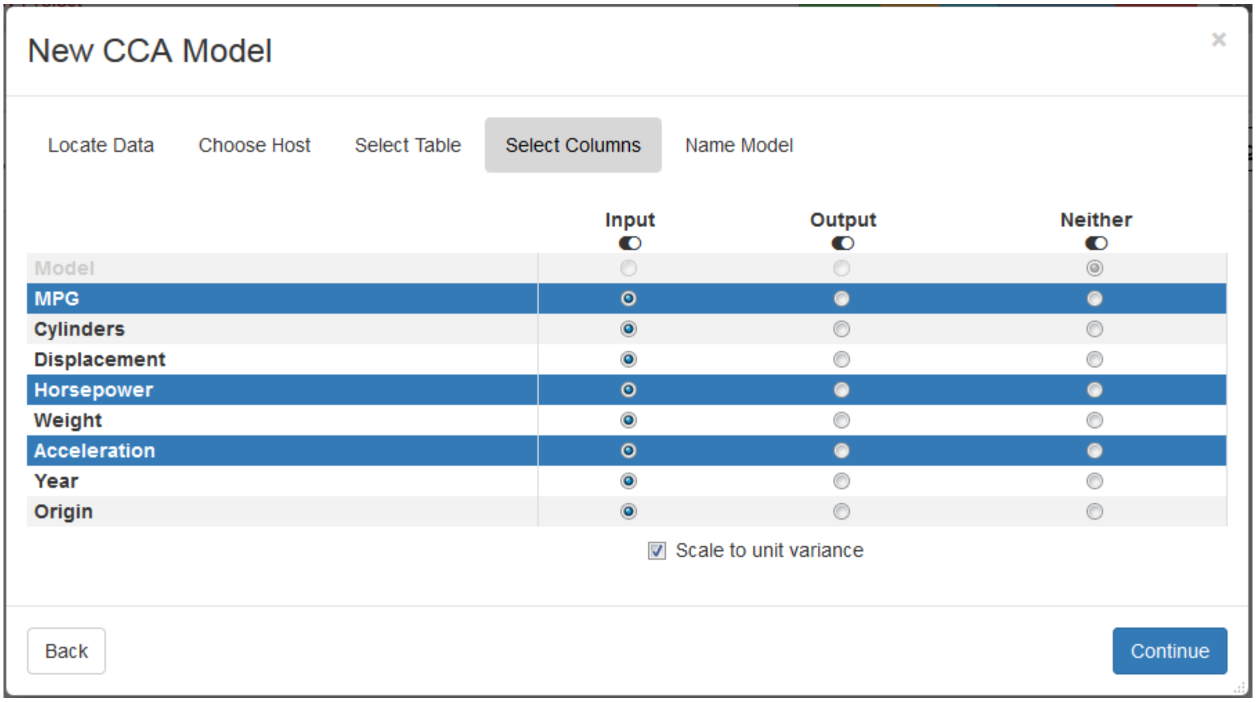
Fig. 20: **Figure 19: Initial configuration in the** Select Columns **dialog for the cars data set.**

Since the number of inputs typically exceeds the number of outputs, we initialize all numeric variables to be inputs, leaving you to identify just the output and excluded variables. If variables shown for this table don't correspond to the ones you wanted or expected, you can click the *Back* button to select a different table file.

Variables can be marked one at a time by clicking the radio buttons, or they can be marked in larger groups by using either shift-click to select a contiguous group of variables, or by using control-click to pick a scattered set of rows (as demonstrated in Figure 20). For group selections, you must click on the rows near the variable names instead of near the radio buttons. Once you have highlighted a set of lines for joint assignment, click on the ◐ icon under the desired category to set the radio buttons for the group, as shown in Figure 21. Since CCA can be performed on any subset of variables, you can also use it to calculate correlations between multiple inputs and a single output, or between any two individual variables.

Sometimes value ranges between variables differ by many orders of magnitude, which can bias the analysis. The checkbox, *Scale inputs to unit variance*, permits you to normalize the values prior to running CCA. This feature is enabled by default. If you wish to perform the analysis using the original unscaled values, click within the box to remove the checkmark.

Once you have finished defining the input/output variables for the CCA analysis and have determined whether you want the values to be scaled, click *Continue* to go to the final step where you provide a name for your model.



New CCA Model

×

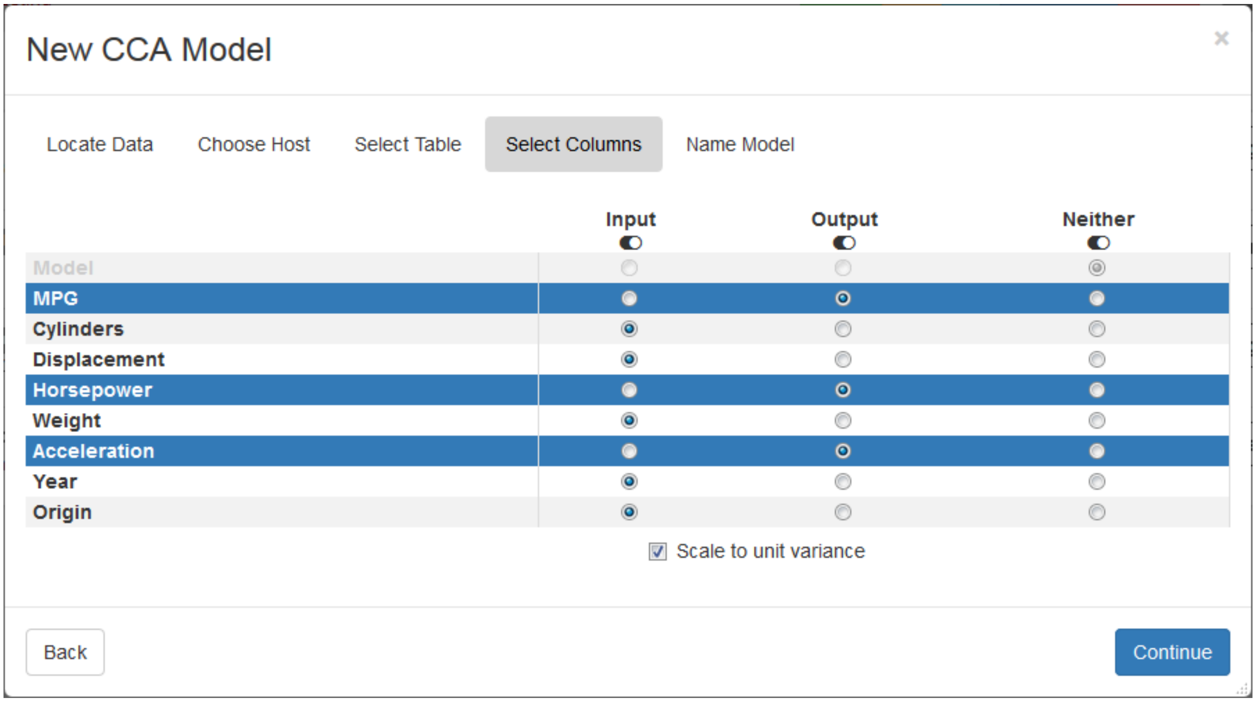
Locate Data | Choose Host | Select Table | Select Columns | Name Model

	Input ◐	Output ◐	Neither ◐
Model	○	○	●
MPG	●	○	○
Cylinders	●	○	○
Displacement	●	○	○
Horsepower	●	○	○
Weight	●	○	○
Acceleration	●	○	○
Year	●	○	○
Origin	●	○	○

☑ Scale to unit variance

Back | Continue

Fig. 21: **Figure 20: Click on the ◐ icon beneath Output to label the highlighted variables as outputs.**



New CCA Model

×

Locate Data | Choose Host | Select Table | Select Columns | Name Model

	Input ◐	Output ◐	Neither ◐
Model	○	○	●
MPG	○	●	○
Cylinders	●	○	○
Displacement	●	○	○
Horsepower	○	●	○
Weight	●	○	○
Acceleration	○	●	○
Year	●	○	○
Origin	●	○	○

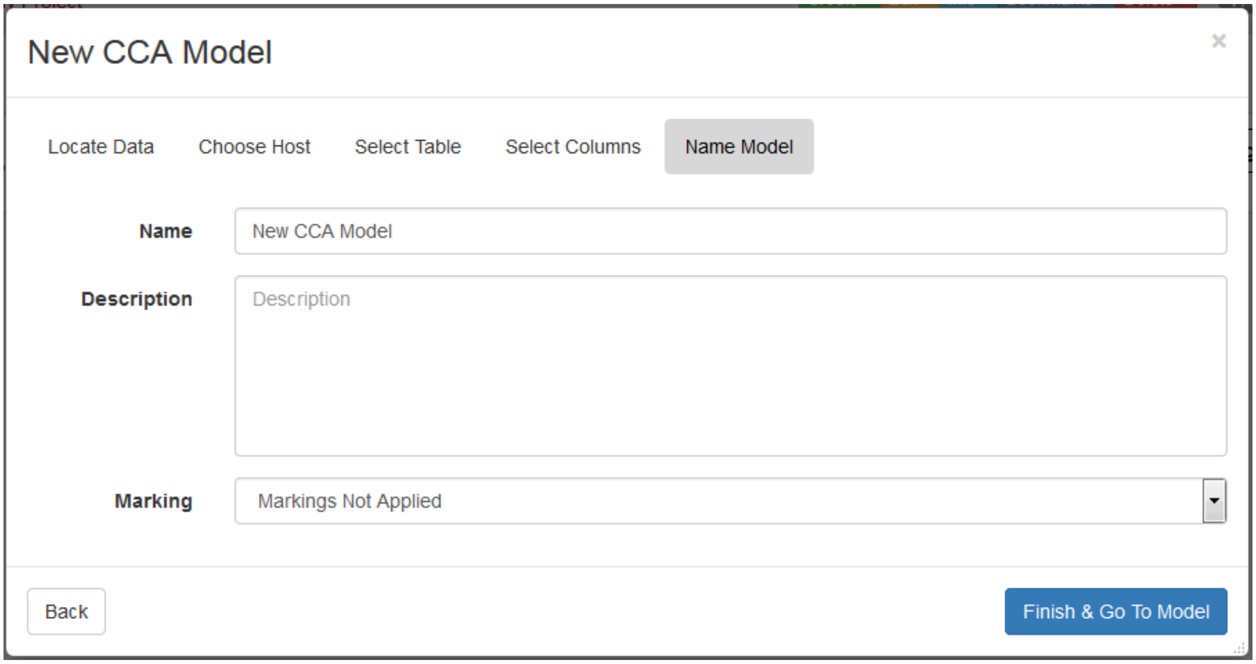
☑ Scale to unit variance

Back | Continue

Fig. 22: **Figure 21: Result of using shift-click and the group assignment icon to select** Output **variables.**

Name Model

The final page of the CCA model creation wizard, shown in Figure 22, provides editable fields to enter a *Name* and an optional *Description* for the model. A default name of *New CCA Model* is provided, but the model list on the project page will become uninterpretable if you use this for all your models. Additionally, you should select a *Marking* from the dropdown list of choices. These markings are specific to your institution and the Slycat™ server you are using. The selected marking identifies the sensitivity of the data that is being analyzed, both for your own benefit and for other team members. This marking is used to label the model, both on the project page and within banners at the top and bottom of the model visualization. Once this information has been entered, click the *Finish & Go To Model* button in the lower right. Slycat™ will then transfer you to the model visualization page. When the analysis has completed, the CCA model will be displayed. If processing is still ongoing, the message "The model is being computed. Patience!" will be shown.



New CCA Model

Name: New CCA Model

Description: Description

Marking: Markings Not Applied

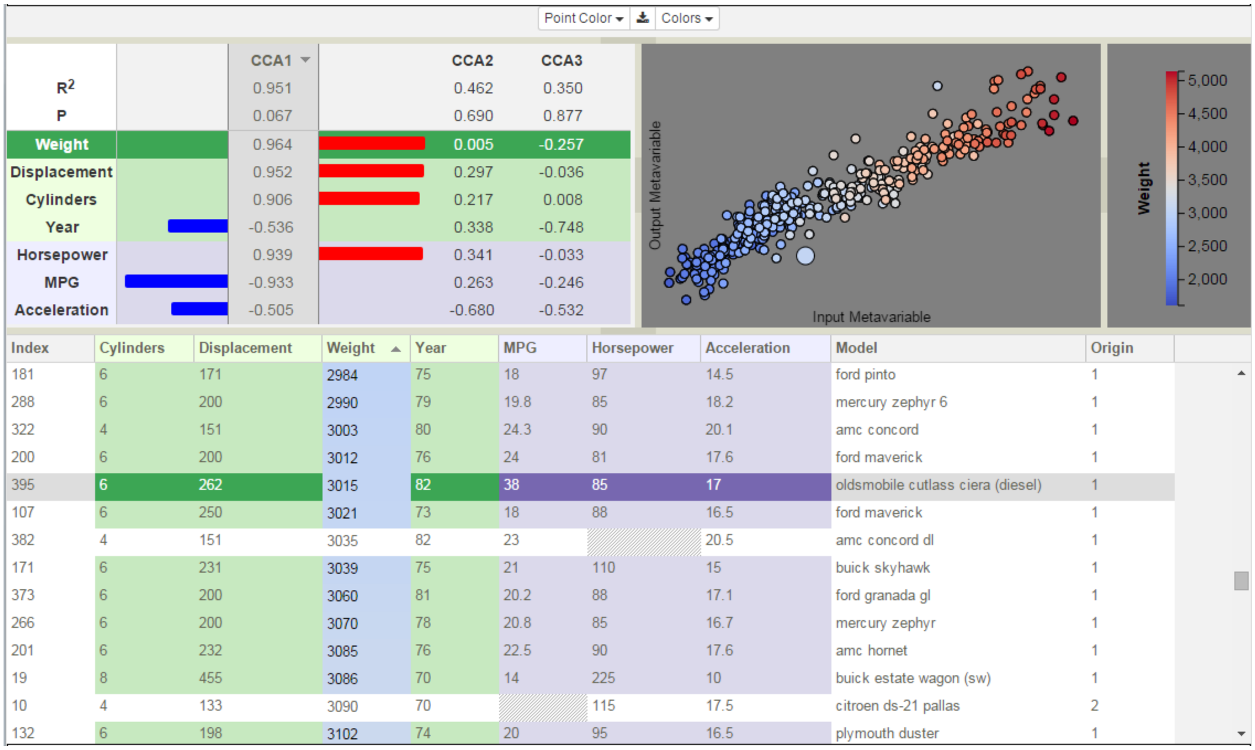
Fig. 23: **Figure 22: The final step in CCA model creation is to name the model and apply markings.**

CCA Model Visualization

As shown below, visualization of a CCA model consists of three linked views, each providing a different level of abstraction. The most abstract level is the *Correlation View*, where each column displays the structure coefficients for one of the canonical components. The scatterplot in the *Simulation View* shows how well each individual run is described by the correlations found in the ensemble overall. The least abstract view is the *Variable Table*, which provides the raw data values contained in the original table file. The views are all linked, so changing the selection in one view will modify the selection in one or more of the other views. As with most Slycat™ models, the views are arranged with the ensemble level view in the upper left, the midrange view in the upper right, and the lowest level view at the bottom.

Correlation View

The *Correlation View* displays the relationships found between variables when the ensemble is viewed holistically. Each column of the *Correlation View*'s bar chart represents a different canonical component. These orthogonal com-

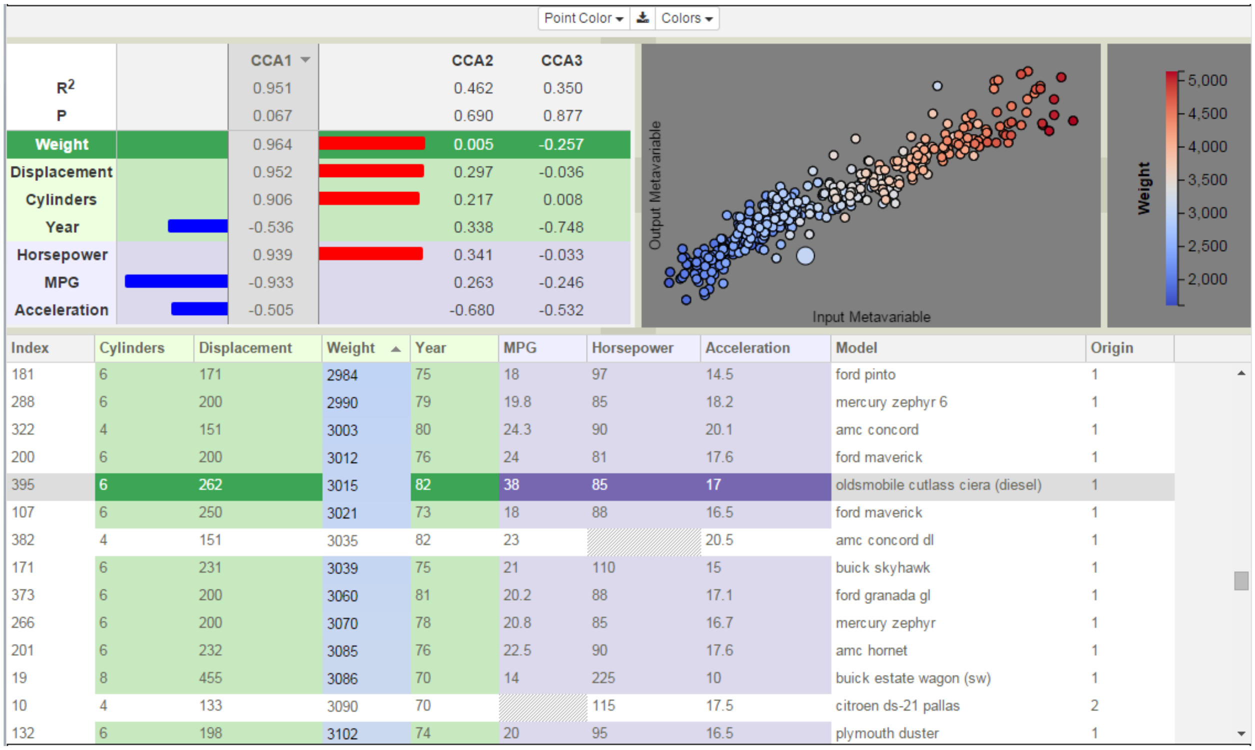


	CCA1	CCA2	CCA3
R^2	0.951	0.462	0.350
P	0.067	0.690	0.877
Weight	0.964	0.005	-0.257
Displacement	0.952	0.297	-0.036
Cylinders	0.906	0.217	0.008
Year	-0.536	0.338	-0.748
Horsepower	0.939	0.341	-0.033
MPG	-0.933	0.263	-0.246
Acceleration	-0.505	-0.680	-0.532

Index	Cylinders	Displacement	Weight	Year	MPG	Horsepower	Acceleration	Model	Origin
181	6	171	2984	75	18	97	14.5	ford pinto	1
288	6	200	2990	79	19.8	85	18.2	mercury zephyr 6	1
322	4	151	3003	80	24.3	90	20.1	amc concord	1
200	6	200	3012	76	24	81	17.6	ford maverick	1
395	6	262	3015	82	38	85	17	oldsmobile cutlass ciera (diesel)	1
107	6	250	3021	73	18	88	16.5	ford maverick	1
382	4	151	3035	82	23		20.5	amc concord dl	1
171	6	231	3039	75	21	110	15	buick skyhawk	1
373	6	200	3060	81	20.2	88	17.1	ford granada gl	1
266	6	200	3070	78	20.8	85	16.7	mercury zephyr	1
201	6	232	3085	76	22.5	90	17.6	amc hornet	1
19	8	455	3086	70	14	225	10	buick estate wagon (sw)	1
10	4	133	3090	70		115	17.5	citroen ds-21 pallas	2
132	6	198	3102	74	20	95	16.5	plymouth duster	1

Fig. 24: **CCA model of cars data set with three linked views, each providing a different level of abstraction. The Correlation View is in the upper left, the Simulation View and its associated Legend are in the upper right, and the Variable Table fills the bottom half.**

ponents are ordered from left to right in decreasing importance, as shown by the decreasing R^2 and increasing p-values in the first two rows of each component column. Variable names are shown along the left edge with rows for input variables colored in green, and rows for output variables colored in lavender. The number of components returned by CCA is equal to the minimum of the number of inputs versus the number of outputs. So, for the example in Figure 23, where there are four inputs and three outputs, CCA will return three canonical components.



		CCA1 ▾		CCA2	CCA3
R²		0.951		0.462	0.350
P		0.067		0.690	0.877
Weight		0.964	🟥	0.005	-0.257
Displacement		0.952	🟥	0.297	-0.036
Cylinders		0.906	🟥	0.217	0.008
Year	🟦	-0.536		0.338	-0.748
Horsepower		0.939	🟥	0.341	-0.033
MPG	🟦	-0.933		0.263	-0.246
Acceleration	🟦	-0.505		-0.680	-0.532

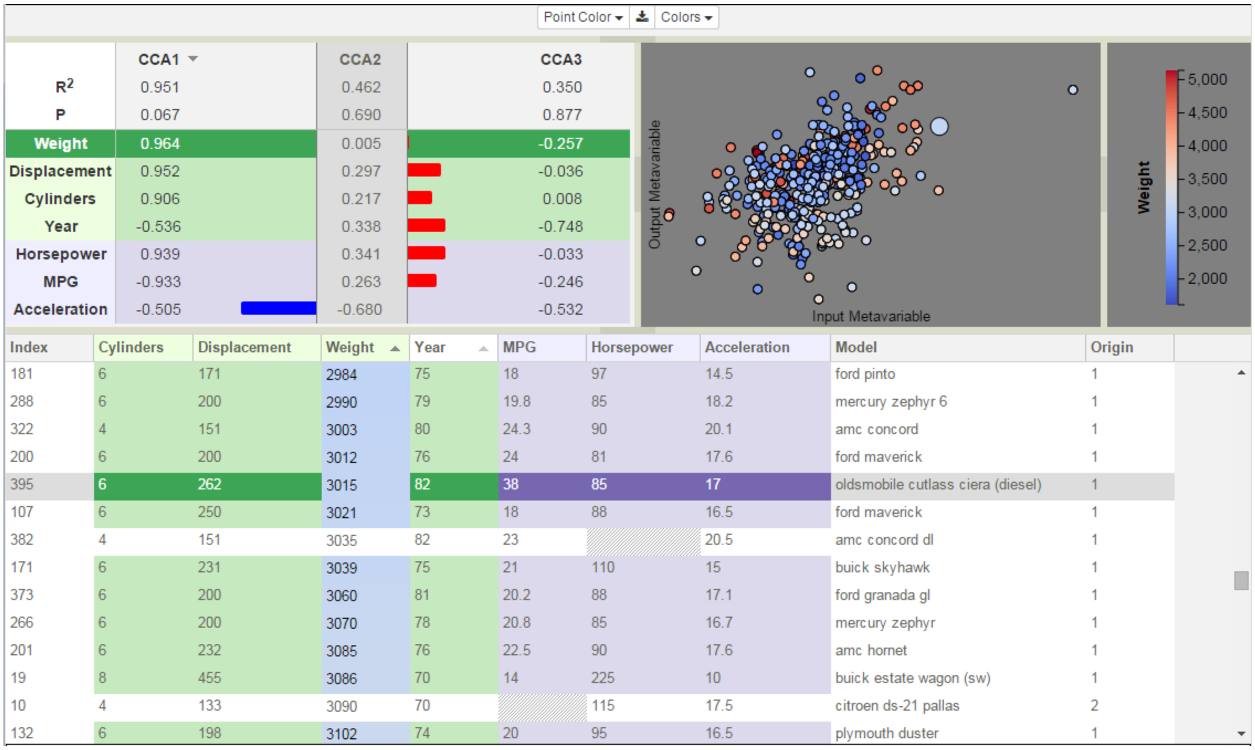
Index	Cylinders	Displacement	Weight ▲	Year	MPG	Horsepower	Acceleration	Model	Origin
181	6	171	2984	75	18	97	14.5	ford pinto	1
288	6	200	2990	79	19.8	85	18.2	mercury zephyr 6	1
322	4	151	3003	80	24.3	90	20.1	amc concord	1
200	6	200	3012	76	24	81	17.6	ford maverick	1
395	6	262	3015	82	38	85	17	oldsmobile cutlass ciera (diesel)	1
107	6	250	3021	73	18	88	16.5	ford maverick	1
382	4	151	3035	82	23	▨	20.5	amc concord dl	1
171	6	231	3039	75	21	110	15	buick skyhawk	1
373	6	200	3060	81	20.2	88	17.1	ford granada gl	1
266	6	200	3070	78	20.8	85	16.7	mercury zephyr	1
201	6	232	3085	76	22.5	90	17.6	amc hornet	1
19	8	455	3086	70	14	225	10	buick estate wagon (sw)	1
10	4	133	3090	70	▨	115	17.5	citroen ds-21 pallas	2
132	6	198	3102	74	20	95	16.5	plymouth duster	1

Fig. 25: **Figure 23: CCA model of cars data set displaying the first canonical component.**

In the bar chart, only one canonical component is expanded at a time (e.g. in Figure 23, *CCA1* is expanded, while in Figure 24, *CCA1* is collapsed and *CCA2* is expanded). Clicking on a CCA column name changes the selected canonical component. This collapses the bar chart from the previously selected component and expands that of the new component.

The horizontal bars in the expanded bar chart visually encode the relationships between variables, both in terms of the magnitude of the structure coefficients, and in terms of the correlation type (positive or negative). Numeric values for the coefficients are displayed in the center of each column. Positive values are drawn as red bars extending towards the right. Negative values are drawn in blue extending to the left. The orientation combined with the color-coding acts to visually reinforce the relationship information. At a glance, you can see correlative relationships between variables and their strength by comparing the color, direction, and length of the bars. Positively correlated variables will display the same color and bar orientation, while negatively correlated variables will be opposed.

The bar chart rows can be sorted by variable strength. To the right of the CCA column name is a small triangular icon. Clicking on this icon sorts the columns by the unsigned magnitudes of the structure coefficients in the expanded column, though all columns will reflect the order returned by this sorting operation (i.e. the rows are sorted using this column as the key). The initial sort is descending and the orientation of the triangle reflects this by rendering the triangle with the wide edge at the top and the point at the bottom (e.g. *CCA1* in Figure 23). The sorting order is reversed if you click the triangle again. Ascending sorts are signified by rendering the triangle with the point at the top. Inputs and outputs are sorted independently. For long lists of variables, the input and output variable sets are independently scrollable. Note that in Figure 24, although *CCA2* is selected, the decreasing sort order from *CCA1* is still maintained. Sorting column is independent of component selection. Hovering over any of the CCA column headers, the sorting icon for that column becomes visible and can be clicked without needing to expand the component.



	CCA1	CCA2	CCA3
R²	0.951	0.462	0.350
P	0.067	0.690	0.877
Weight	0.964	0.005	-0.257
Displacement	0.952	0.297	-0.036
Cylinders	0.906	0.217	0.008
Year	-0.536	0.338	-0.748
Horsepower	0.939	0.341	-0.033
MPG	-0.933	0.263	-0.246
Acceleration	-0.505	-0.680	-0.532

Index	Cylinders	Displacement	Weight	Year	MPG	Horsepower	Acceleration	Model	Origin
181	6	171	2984	75	18	97	14.5	ford pinto	1
288	6	200	2990	79	19.8	85	18.2	mercury zephyr 6	1
322	4	151	3003	80	24.3	90	20.1	amc concord	1
200	6	200	3012	76	24	81	17.6	ford maverick	1
395	6	262	3015	82	38	85	17	oldsmobile cutlass ciera (diesel)	1
107	6	250	3021	73	18	88	16.5	ford maverick	1
382	4	151	3035	82	23		20.5	amc concord dl	1
171	6	231	3039	75	21	110	15	buick skyhawk	1
373	6	200	3060	81	20.2	88	17.1	ford granada gl	1
266	6	200	3070	78	20.8	85	16.7	mercury zephyr	1
201	6	232	3085	76	22.5	90	17.6	amc hornet	1
19	8	455	3086	70	14	225	10	buick estate wagon (sw)	1
10	4	133	3090	70		115	17.5	citroen ds-21 pallas	2
132	6	198	3102	74	20	95	16.5	plymouth duster	1

Fig. 26: **Figure 24: CCA model of Cars data set displaying the second canonical component.**

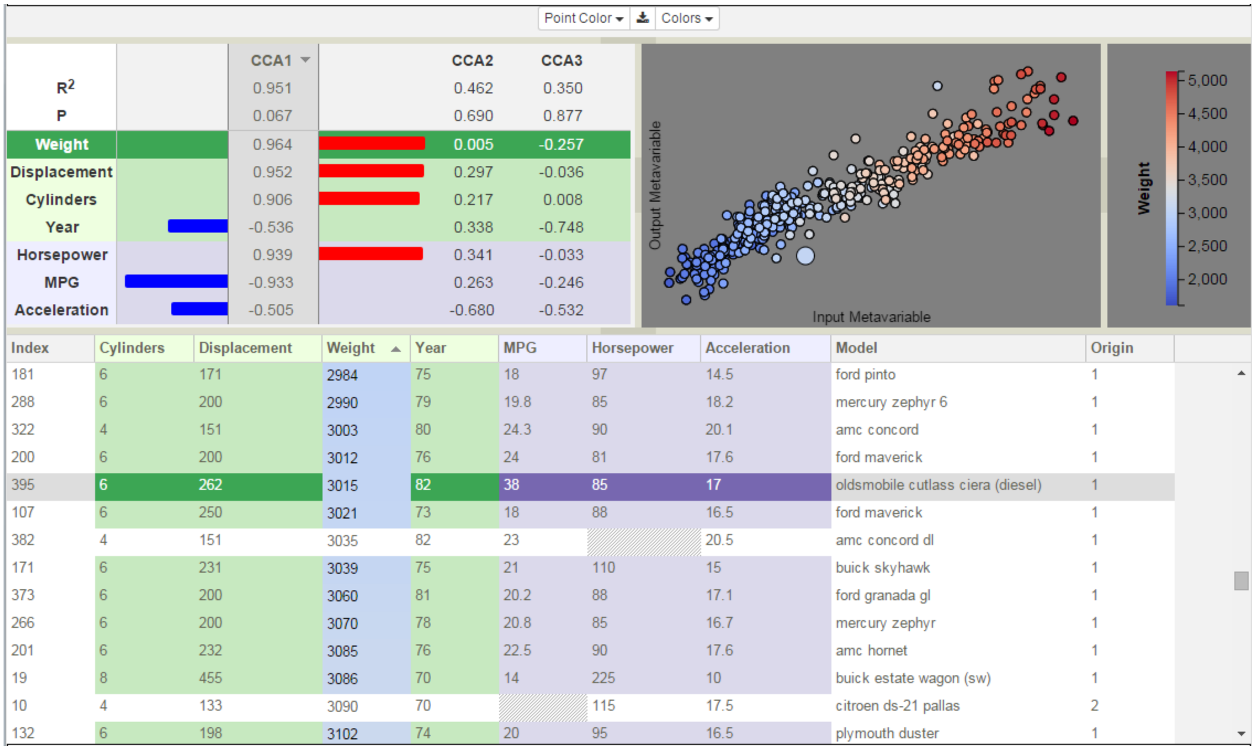
Clicking on a row in the bar chart selects that variable for color-coding the points in the *Simulation View* (i.e. each simulation point is color-coded according to its value for that variable). The variable row is highlighted by darkening the background color and changing the font color to white. The color palette, shown in the *Legend* alongside the value range for the color-coding variable, corresponds to the currently selected theme (see color-themes). This same color-coding is applied to the cell backgrounds of this column of the *Variable Table*, as is demonstrated by the *Weight* column in both Figure 23 and Figure 24.

Simulation View

The *Simulation View* is a scatterplot, in which each point represents an ensemble member. The axes of the scatterplot are the canonical variables, $\mathbf{a}^T X$ and $\mathbf{b}^T Y$, which are labeled as *Input Metavariable* and *Output Metavariable*. The x and y coordinates of each point are weighted sums of that point's input and output variable values, respectively. Because the values of the canonical variables differ for each canonical component, changing the selected component in the *Correlation View* changes the point coordinates, which are then re-rendered in the scatterplot. Comparing the scatterplots in Figure 23 and Figure 24, you can see how the point locations shift from a loose diagonal for *CCA1*, to a ball of points for *CCA2*. Given the low R^2 and high p-value for *CCA2*, the scatterplot point placement visually reveals the poor quality of the result (all points would be on the diagonal in an ideal result).

Legend

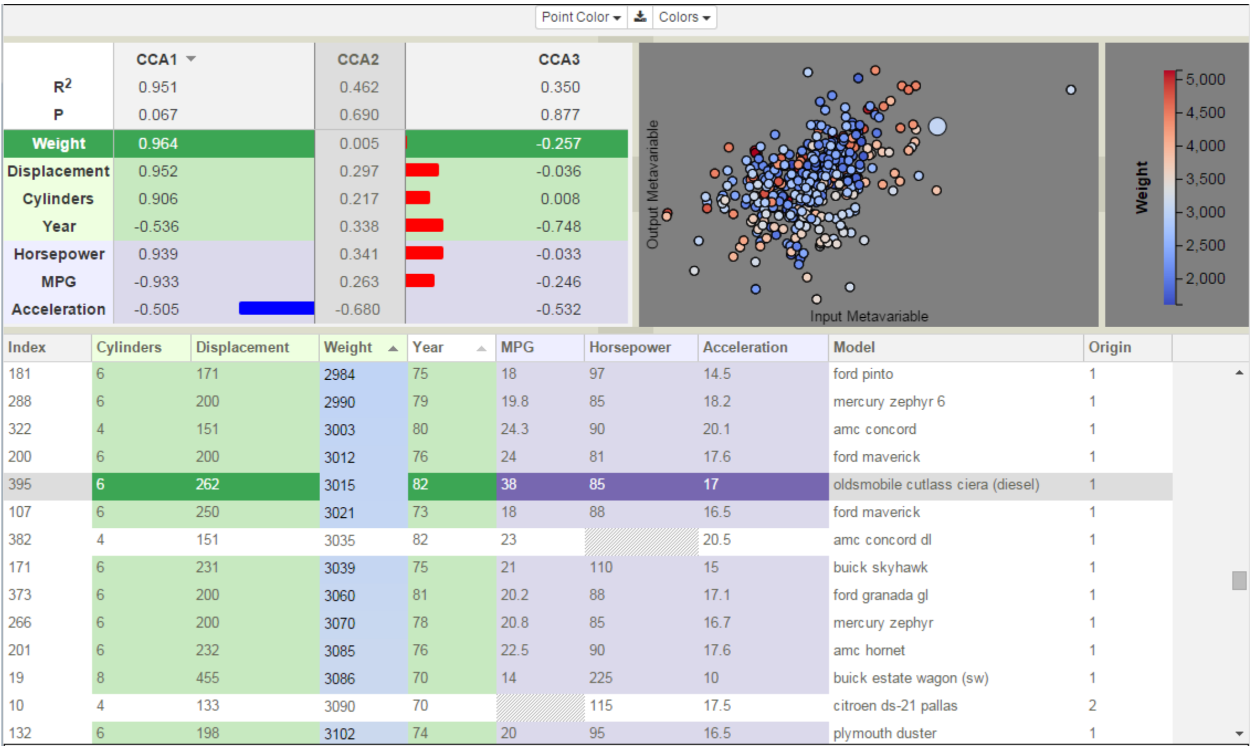
To the right of the scatterplot is the *Legend*. The *Legend* is in its own view, which can be resized or closed altogether. The *Legend* displays information about the current color-coding variable, including its name, range of values, and the mapping between values and colors. The color palette is defined by the current theme (see Color Themes).



	CCA1	CCA2	CCA3
R²	0.951	0.462	0.350
P	0.067	0.690	0.877
Weight	0.964	0.005	-0.257
Displacement	0.952	0.297	-0.036
Cylinders	0.906	0.217	0.008
Year	-0.536	0.338	-0.748
Horsepower	0.939	0.341	-0.033
MPG	-0.933	0.263	-0.246
Acceleration	-0.505	-0.680	-0.532

Index	Cylinders	Displacement	Weight	Year	MPG	Horsepower	Acceleration	Model	Origin
181	6	171	2984	75	18	97	14.5	ford pinto	1
288	6	200	2990	79	19.8	85	18.2	mercury zephyr 6	1
322	4	151	3003	80	24.3	90	20.1	amc concord	1
200	6	200	3012	76	24	81	17.6	ford maverick	1
395	6	262	3015	82	38	85	17	oldsmobile cutlass ciera (diesel)	1
107	6	250	3021	73	18	88	16.5	ford maverick	1
382	4	151	3035	82	23		20.5	amc concord dl	1
171	6	231	3039	75	21	110	15	buick skyhawk	1
373	6	200	3060	81	20.2	88	17.1	ford granada gl	1
266	6	200	3070	78	20.8	85	16.7	mercury zephyr	1
201	6	232	3085	76	22.5	90	17.6	amc hornet	1
19	8	455	3086	70	14	225	10	buick estate wagon (sw)	1
10	4	133	3090	70		115	17.5	citroen ds-21 pallas	2
132	6	198	3102	74	20	95	16.5	plymouth duster	1

Fig. 27: **Figure 23: CCA model of cars data set displaying the first canonical component.**



	CCA1	CCA2	CCA3
R²	0.951	0.462	0.350
P	0.067	0.690	0.877
Weight	0.964	0.005	-0.257
Displacement	0.952	0.297	-0.036
Cylinders	0.906	0.217	0.008
Year	-0.536	0.338	-0.748
Horsepower	0.939	0.341	-0.033
MPG	-0.933	0.263	-0.246
Acceleration	-0.505	-0.680	-0.532

Index	Cylinders	Displacement	Weight	Year	MPG	Horsepower	Acceleration	Model	Origin
181	6	171	2984	75	18	97	14.5	ford pinto	1
288	6	200	2990	79	19.8	85	18.2	mercury zephyr 6	1
322	4	151	3003	80	24.3	90	20.1	amc concord	1
200	6	200	3012	76	24	81	17.6	ford maverick	1
395	6	262	3015	82	38	85	17	oldsmobile cutlass ciera (diesel)	1
107	6	250	3021	73	18	88	16.5	ford maverick	1
382	4	151	3035	82	23		20.5	amc concord dl	1
171	6	231	3039	75	21	110	15	buick skyhawk	1
373	6	200	3060	81	20.2	88	17.1	ford granada gl	1
266	6	200	3070	78	20.8	85	16.7	mercury zephyr	1
201	6	232	3085	76	22.5	90	17.6	amc hornet	1
19	8	455	3086	70	14	225	10	buick estate wagon (sw)	1
10	4	133	3090	70		115	17.5	citroen ds-21 pallas	2
132	6	198	3102	74	20	95	16.5	plymouth duster	1

Fig. 28: **Figure 24: CCA model of Cars data set displaying the second canonical component.**

Color-Coding Points

The first time a model is rendered, the points are colored by their index number. There are three mechanisms for changing the variable that is used to do the color-coding: clicking on a variable in the *Correlation View*, clicking on the column header in the *Variable Table*, or selecting a variable from the *Point Color* dropdown list. Irrespective of the interface used, changing the variable selected for color mapping will lead to changes in all three views and the legend, including: highlighting the newly selected variable's row in the bar chart, recoloring the scatterplot using the new variable's values, coloring the cell backgrounds in its table column, returning the cell backgrounds of the previously selected variable's column to its default color (green, lavender, or white depending on its type), and relabeling and redefining the value range in the *Legend*. Note that the table may include variables that are not present in the *Correlation View*, columns that are neither inputs nor outputs. These columns are drawn on the right end of the table against white backgrounds. These provide additional color-coding options (numeric variables only), however, the bar chart will not be highlighted because it only includes variables passed to CCA.

Selecting Points

Points in the scatterplot may be selected through several mechanisms. The simplest is to place the mouse cursor over a point and click the left mouse button. The selected point is redrawn in the plot with a larger radius, while simultaneously in the *Variable Table*, the row corresponding to the selected point is darkened and scrolled to be visible.

For groups of adjacent points that lie within a rectangular region, rubber banding can be used to draw a rectangle around the desired point set. Position the mouse at one corner of the region. Press the left mouse button down while simultaneously moving the mouse towards the opposite corner of the region. A yellow rectangle will be drawn between the location of the initial button-press and the mouse's current position. Move the mouse until the rectangle encloses all the desired points, then release the mouse button to finish the selection.

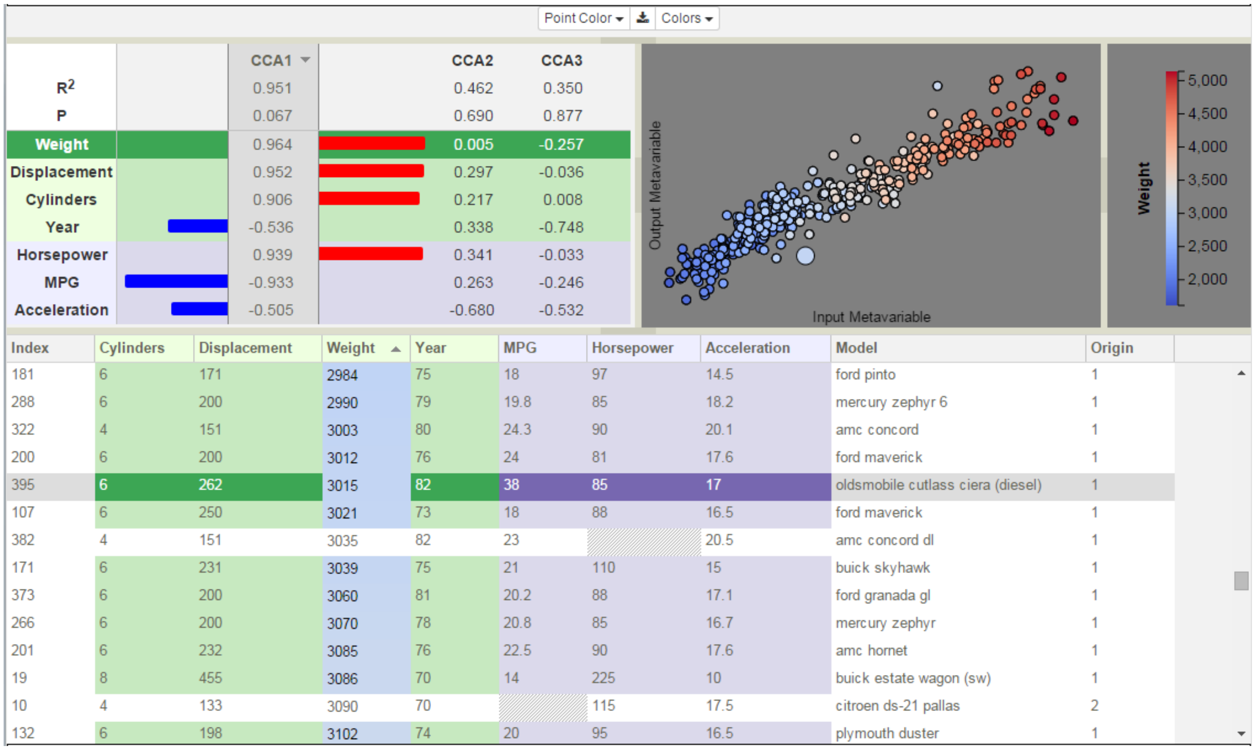
Holding the control-key while selecting new points, either through clicking or rubber banding, will add these additional points to the previously selected set. Alternately, scatterplot points can be selected by picking rows in the *Variable Table* (see Simulation Selection).

Clicking in the background (not on any point) deselects all previously selected points.

Variable Table

The *Variable Table* at the bottom of Figure 23 provides access to the original numeric variable values for each simulation run. It is essentially an interactive version of the original table data, where each column represents a single variable and each row contains the variable values for a single ensemble member. Within the table, the cell backgrounds take on one of four color-encodings: input variables are green, outputs are lavender, non-designated variables are white, and the elements of the selected variable are individually colored by their value using the current color map (see Color Themes). Coloring table elements by value highlights the selected color-coding variable, while concurrently providing color correspondence between rows and scatterplot points. The interactive capabilities of the table include: sorting within columns, column (variable) selection, and row (simulation) selection.

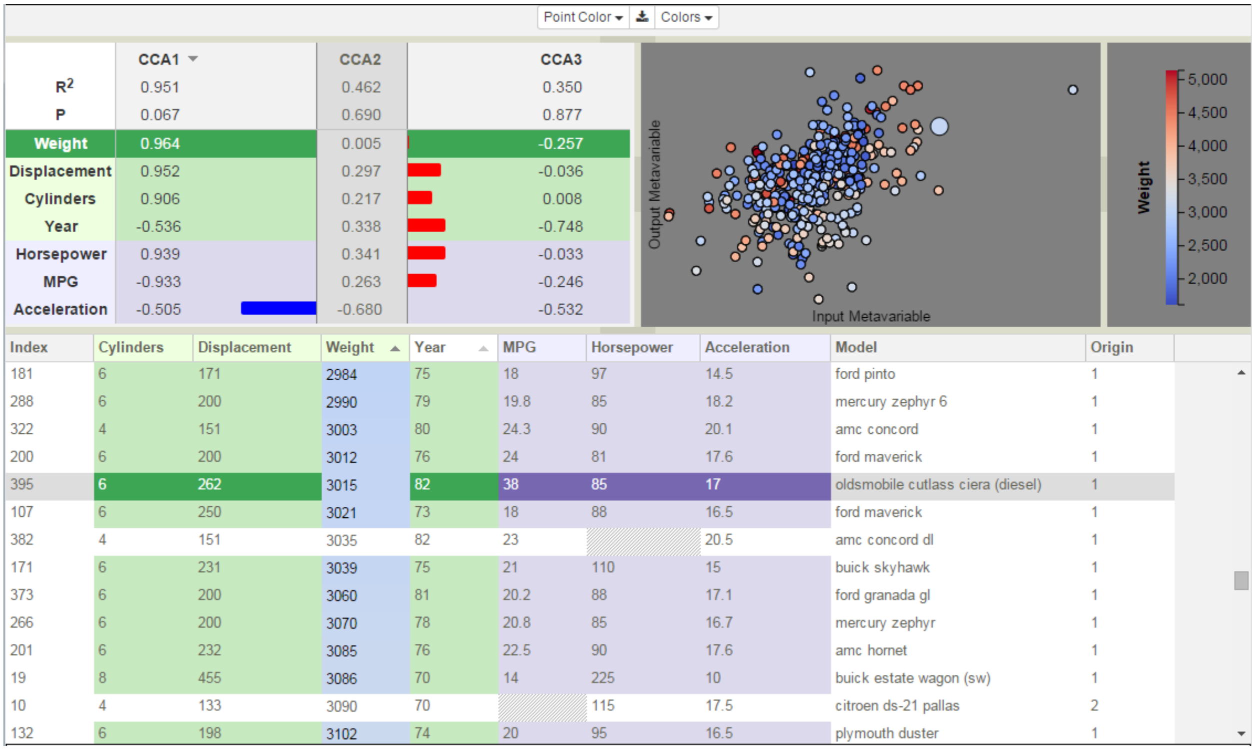
Rows that have not been used in the CCA analysis have white backgrounds. CCA requires that all rows have values for each of the columns that are included in the analysis. As shown in Figure 24, the cars in rows 382 and 10 are missing values for *Horsepower* and *MPG*, respectively. The missing data are shown with no numeric value and a hatched gray background in the table. Since the columns with missing values were declared as outputs during model creation, rows 382 and 10 have been entirely excluded from the analysis and are drawn in white. If CCA is later rerun, and if *Horsepower* and *MPG* are removed from the analysis (if the columns are marked as *Neither* when creating the new model), then these rows will be included in the calculation and will be color-coded.



	CCA1	CCA2	CCA3
R^2	0.951	0.462	0.350
P	0.067	0.690	0.877
Weight	0.964	0.005	-0.257
Displacement	0.952	0.297	-0.036
Cylinders	0.906	0.217	0.008
Year	-0.536	0.338	-0.748
Horsepower	0.939	0.341	-0.033
MPG	-0.933	0.263	-0.246
Acceleration	-0.505	-0.680	-0.532

Index	Cylinders	Displacement	Weight	Year	MPG	Horsepower	Acceleration	Model	Origin
181	6	171	2984	75	18	97	14.5	ford pinto	1
288	6	200	2990	79	19.8	85	18.2	mercury zephyr 6	1
322	4	151	3003	80	24.3	90	20.1	amc concord	1
200	6	200	3012	76	24	81	17.6	ford maverick	1
395	6	262	3015	82	38	85	17	oldsmobile cutlass ciera (diesel)	1
107	6	250	3021	73	18	88	16.5	ford maverick	1
382	4	151	3035	82	23		20.5	amc concord dl	1
171	6	231	3039	75	21	110	15	buick skyhawk	1
373	6	200	3060	81	20.2	88	17.1	ford granada gl	1
266	6	200	3070	78	20.8	85	16.7	mercury zephyr	1
201	6	232	3085	76	22.5	90	17.6	amc hornet	1
19	8	455	3086	70	14	225	10	buick estate wagon (sw)	1
10	4	133	3090	70		115	17.5	citroen ds-21 pallas	2
132	6	198	3102	74	20	95	16.5	plymouth duster	1

Fig. 29: **Figure 23: CCA model of cars data set displaying the first canonical component.**



	CCA1	CCA2	CCA3
R^2	0.951	0.462	0.350
P	0.067	0.690	0.877
Weight	0.964	0.005	-0.257
Displacement	0.952	0.297	-0.036
Cylinders	0.906	0.217	0.008
Year	-0.536	0.338	-0.748
Horsepower	0.939	0.341	-0.033
MPG	-0.933	0.263	-0.246
Acceleration	-0.505	-0.680	-0.532

Index	Cylinders	Displacement	Weight	Year	MPG	Horsepower	Acceleration	Model	Origin
181	6	171	2984	75	18	97	14.5	ford pinto	1
288	6	200	2990	79	19.8	85	18.2	mercury zephyr 6	1
322	4	151	3003	80	24.3	90	20.1	amc concord	1
200	6	200	3012	76	24	81	17.6	ford maverick	1
395	6	262	3015	82	38	85	17	oldsmobile cutlass ciera (diesel)	1
107	6	250	3021	73	18	88	16.5	ford maverick	1
382	4	151	3035	82	23		20.5	amc concord dl	1
171	6	231	3039	75	21	110	15	buick skyhawk	1
373	6	200	3060	81	20.2	88	17.1	ford granada gl	1
266	6	200	3070	78	20.8	85	16.7	mercury zephyr	1
201	6	232	3085	76	22.5	90	17.6	amc hornet	1
19	8	455	3086	70	14	225	10	buick estate wagon (sw)	1
10	4	133	3090	70		115	17.5	citroen ds-21 pallas	2
132	6	198	3102	74	20	95	16.5	plymouth duster	1

Fig. 30: **Figure 24: CCA model of Cars data set displaying the second canonical component.**

Sorting

Sorting allows rapid identification of simulations whose variable values are extrema. Additionally, sorting facilitates comparison between simulations whose values are similar within one variable, but whose values for other variables might differ significantly. For instance, in Figure 23 and Figure 24 the tables are sorted by weight. The highlighted row appears adjacent to cars having similar weights, yet the selected car is notable because it gets much better gas mileage.

Although the sorting order is defined by values within a single column, the full row moves in the reordering. To sort on a variable, left-click the small triangular icon in the column header. The initial sort is ascending and the orientation of the triangle reflects this by rendering the triangle with the tip at the top and the wide edge at the bottom. If you click the triangle again, the sorting order is reversed to be descending and the triangle is redrawn with the point at the bottom. The sorting column is independent of variable selection (see Color-Coding Points). Hovering over any of the column headers, the sorting icon for that column becomes visible and may be clicked to initiate sorting on that variable.

Variable Selection

Left-clicking the variable name in a column header selects that variable to color-code the scatterplot points in the *Simulation View*, to color the cell backgrounds in its associated table column, and to highlight that variable's row in the *Correlation View*.

Simulation Selection

Left-clicking within a table row selects that simulation, which is then highlighted in both the table and the scatterplot. Multiple row selection, as is commonly performed on lists, consists of clicking on a starting row, then using shift-click to select the ending row (either above or below the starting row). This selects both the starting and ending rows along with all rows in between. Individual rows may be added to an existing selection by clicking on them while pressing the control-key. Selected rows are highlighted in the table by increasing the saturation of cell backgrounds, except for cells in the color-coding column (see Variable Table). Selected scatterplot points are highlighted by being redrawn with an increased radius.

2.1.3 Parameter Space Model

Unlike the CCA and Time Series models, the Parameter Space model does not perform an analysis step. Instead, the Parameter Space model is an exploratory visual interface that combines a filterable scatterplot representation with remote access to images, videos, and other media-based ensemble data.

Taylor Anvil Impact Scenario (TAIS) Data Set

TAIS was generated using Sierra/SolidMechanics[1] (a Lagrangian, three-dimensional code for problems with large deformations and nonlinear material behaviors) in combination with ParaView/Catalyst[2]. The images were generated in situ (at the same time as the physics simulation) using Catalyst. The simulation is of an Oxygen Free High Conductivity (OFHC) copper cylinder, 2.54 cm long with a diameter of .762 cm and an initial velocity of 190 m/sec, impacting a rigid wall. The ensemble is a sensitivity study that evaluates the effects of changing four parameters of the Johnson-Cook inelastic constitutive law: *ajo*, *bjo*, *njo*, and *beta*. The height and radius of the cylinder after the impact

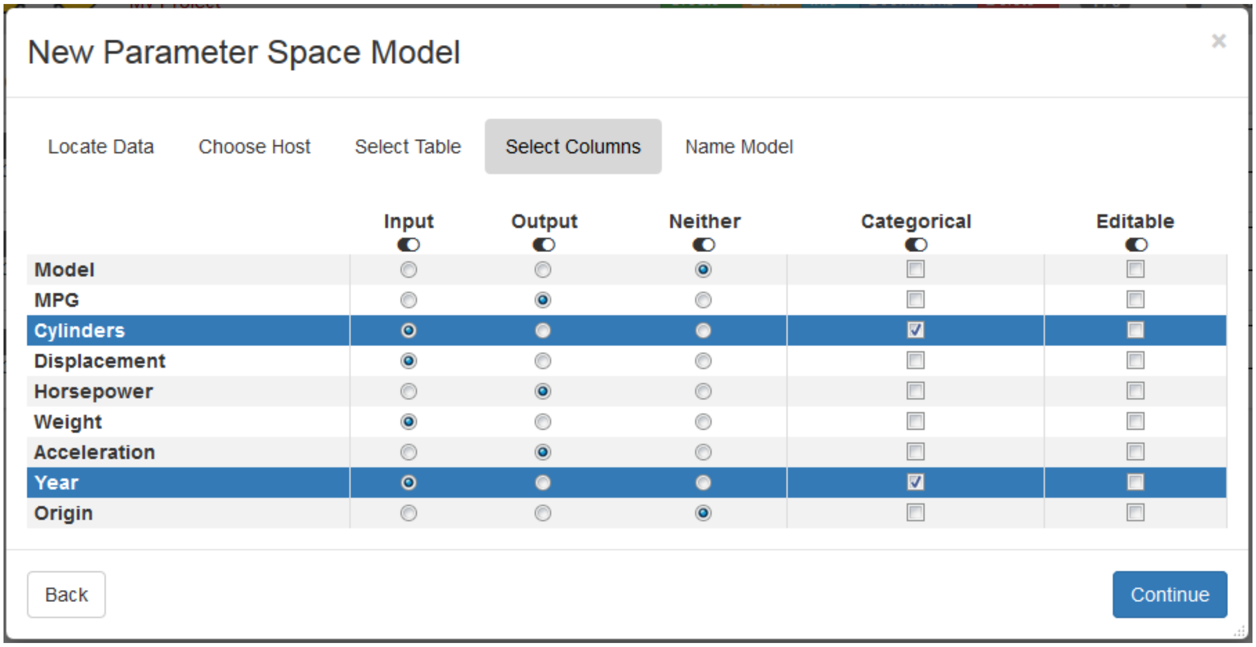
[1] Sierra Solid Mechanics Team. *Sierra/SolidMechanics 4.22 User's Guide*. Technical Report SAND2011-7597, Sandia National Laboratories (2011).

[2] Ayachit, U., Bauer, A., Geveci, B., O'Leary, P., Moreland, K., Fabian, N., and Mauldin, J., *ParaView Catalyst: Enabling In Situ Data Analysis and Visualization*, Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV2015), pp. 25-29, ACM, New York, NY (2015).

are compared to experimental photographic results. Two output metrics are calculated for each run, *ndrf_last* and *ndhf_last*. These variables are the normalized differences between the radius/height of the final cylinder state from the last timestep of the simulation and the final radius/height of the cylinder in the experiment, respectively. Since the differences would decrease in those cases where the simulation more closely matched the experimental results, the optimal case would be a simulation where the values of these two metrics were zero (i.e. there is no difference between the simulation and the experiment).

Creating a Parameter Space Model

Like the CCA model, the core of the Parameter Space model is table data. Up until the stage where inputs and outputs for the model are selected, the model creation steps are identical to CCA (see Creating a CCA Model). Instead of initializing all variables as *Input*, the variables default to being assigned as *Neither*. As with CCA, group selection operations using shift-click and/or control-click allow rapid assignment of variable types (see Select Columns). However, a central difference in the Parameter Space model is that variables can also be designated as being *Categorical* and/or *Editable*.



	Input	Output	Neither	Categorical	Editable
Model	○	○	●	□	□
MPG	○	●	○	□	□
Cylinders	●	○	○	☑	□
Displacement	●	○	○	□	□
Horsepower	○	●	○	□	□
Weight	●	○	○	□	□
Acceleration	○	●	○	□	□
Year	●	○	○	☑	□
Origin	○	○	●	□	□

Fig. 31: **Figure 25: Parameter Space model creation wizard dialog for designating variable attributes, including Categorical and Editable.**

Categorical variables are those with a limited number of discrete values. During scatterplot filtering, it is often advantageous to be able to turn on or off points that are associated with specific values or combinations of values. *Categorical* variables are filtered using labeled buttons, which can individually be set *on* or *off*. Continuous variables are filtered using a slider, which is limited to defining a single range of values to be included or excluded. By declaring *Year* and *Cylinders* as *Categorical* variables during model creation, the example in Figure 26 shows how we can filter the scatterplot display to be only those cars having 3, 5, or 8 cylinders that were manufactured in even-numbered years. This fine-grained filtering of the data would be impossible using a range slider.

An *Editable* variable is one that can be modified by the user. The type of values originally in the variable define the type of values that can be substituted (i.e. numeric variables cannot be changed into text strings). A variable can only be defined as *Editable* during model creation. The mechanism for changing variable values is through selection (see Selecting Points). Note that value modification actions in the *Selection Action* dropdown list are only enabled when an Editable variable has been declared.

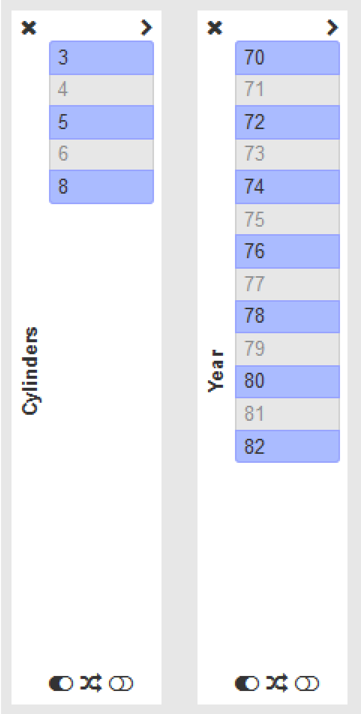
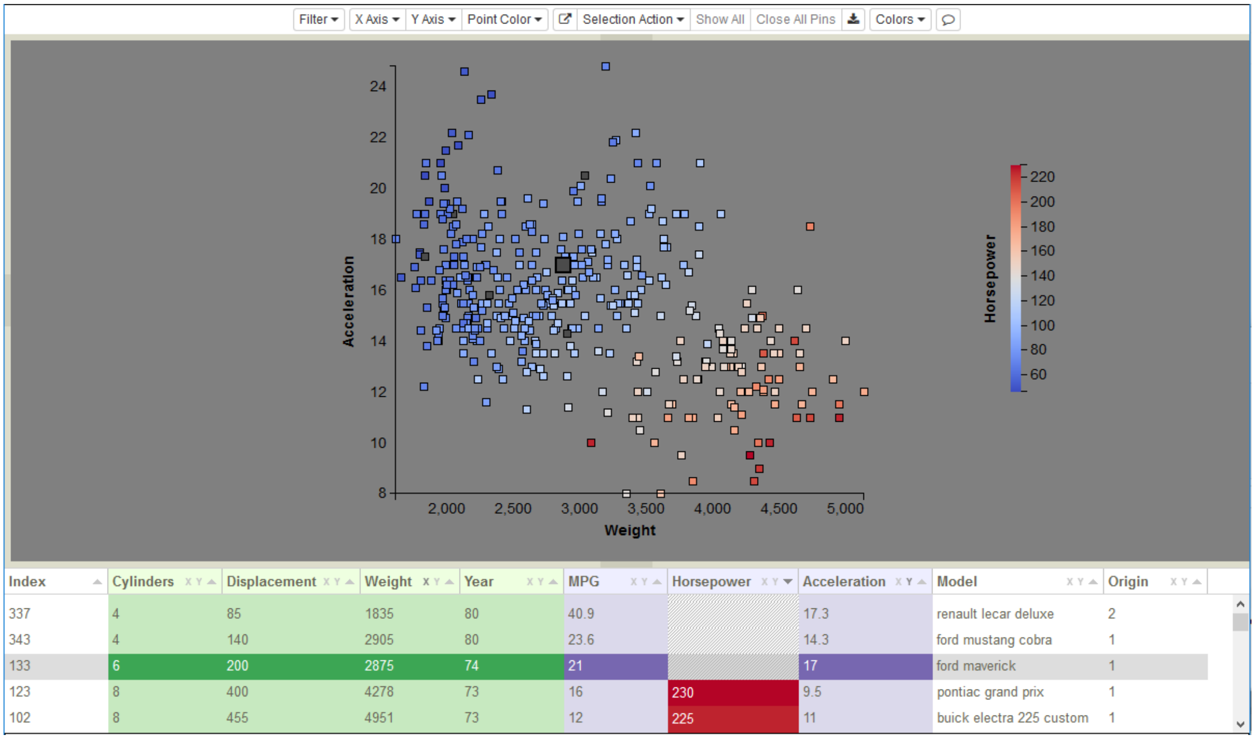


Fig. 32: **Figure 26: Filters for Categorical variables enumerate each discrete value as a labeled button, enabling filtering operations that would not be possible using range sliders. The values selected here (shown in dark blue) limit the scatterplot to display just those cars with 3, 5, or 8 cylinders that were manufactured in even-numbered years.**

Parameter Space Model Visualization

As shown below, the Parameter Space model page consists of linked scatterplot and data table views, combined with interactive filtering, data manipulation, and remote viewing of images and other media. The *Scatterplot View* provides an abstract representation of the ensemble members and their value distributions within the variables selected for the axes. The *Variable Table* provides the raw data values contained in the original table file. Changes made to the plot or the table will cause corresponding changes in the other.



Index	Cylinders	Displacement	Weight	Year	MPG	Horsepower	Acceleration	Model	Origin
337	4	85	1835	80	40.9	▨	17.3	renault lecar deluxe	2
343	4	140	2905	80	23.6	▨	14.3	ford mustang cobra	1
133	6	200	2875	74	21	▨	17	ford maverick	1
123	8	400	4278	73	16	230	9.5	pontiac grand prix	1
102	8	455	4951	73	12	225	11	buick electra 225 custom	1

Fig. 33: **Parameter Space Model of cars data set with Scatterplot and Variable Table views, each providing a different level of abstraction.**

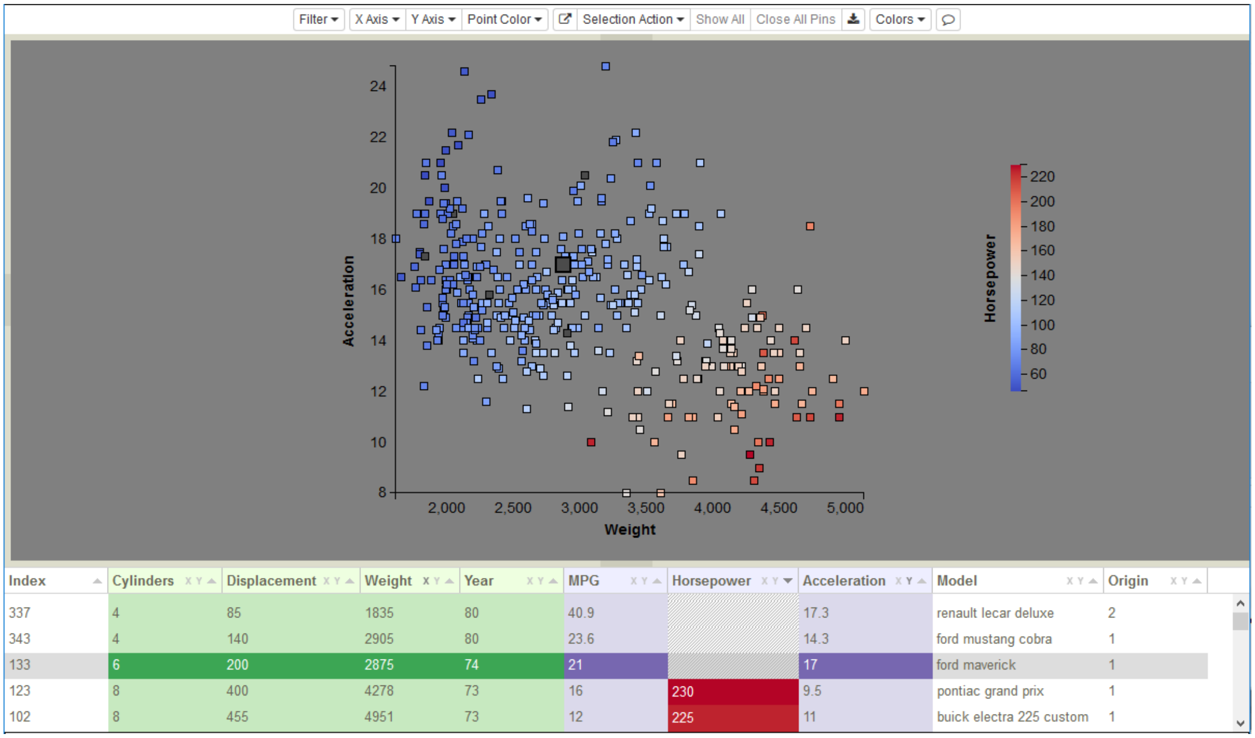
As with the CCA Model Visualization example, we will be using the Cars data set in the following sections to illustrate some of the Parameter Space Model features. We will also be using the Taylor Anvil Impact Scenario (TAIS) ensemble, which includes image data, to demonstrate some of the media-based functionality.

Scatterplot View

The *Scatterplot View* represents each ensemble member as a point in a two-dimensional plot, where the variables that are used for the x-axis, y-axis, and point color-coding are interactively selected. As with the CCA Model, when individual points or groups of points are selected within the plot (see Selecting Points), corresponding rows in the *Variable Table* are simultaneously selected, and vice versa.

However, the Parameter Space Model's *Scatterplot View* possesses additional capabilities that the CCA Model's *Simulation View* lacks. If the *Variable Table* contains media columns (URIs for images, videos, time series tables, or STLs), hovering over a point with the mouse can be used to retrieve media items from remote HPC systems. The media variable to be retrieved is selected through a *Media Set* dropdown list, which only appears in the controls (as shown in Figure 28) when media columns are present in the table.

Another key feature in the *Scatterplot View* is the ability to reduce the number of visible points through filtering or point hiding. This is important for interacting with large ensembles, since point overlap and occlusion increase with



Index	Cylinders	Displacement	Weight	Year	MPG	Horsepower	Acceleration	Model	Origin
337	4	85	1835	80	40.9		17.3	renault lecar deluxe	2
343	4	140	2905	80	23.6		14.3	ford mustang cobra	1
133	6	200	2875	74	21		17	ford maverick	1
123	8	400	4278	73	16	230	9.5	pontiac grand prix	1
102	8	455	4951	73	12	225	11	buick electra 225 custom	1

Fig. 34: **Figure 27: Parameter Space Model visualization of the Cars data set.**

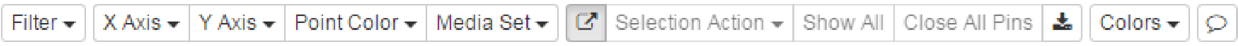


Fig. 35: **Figure 28: Full set of Parameter Space model-specific controls.**

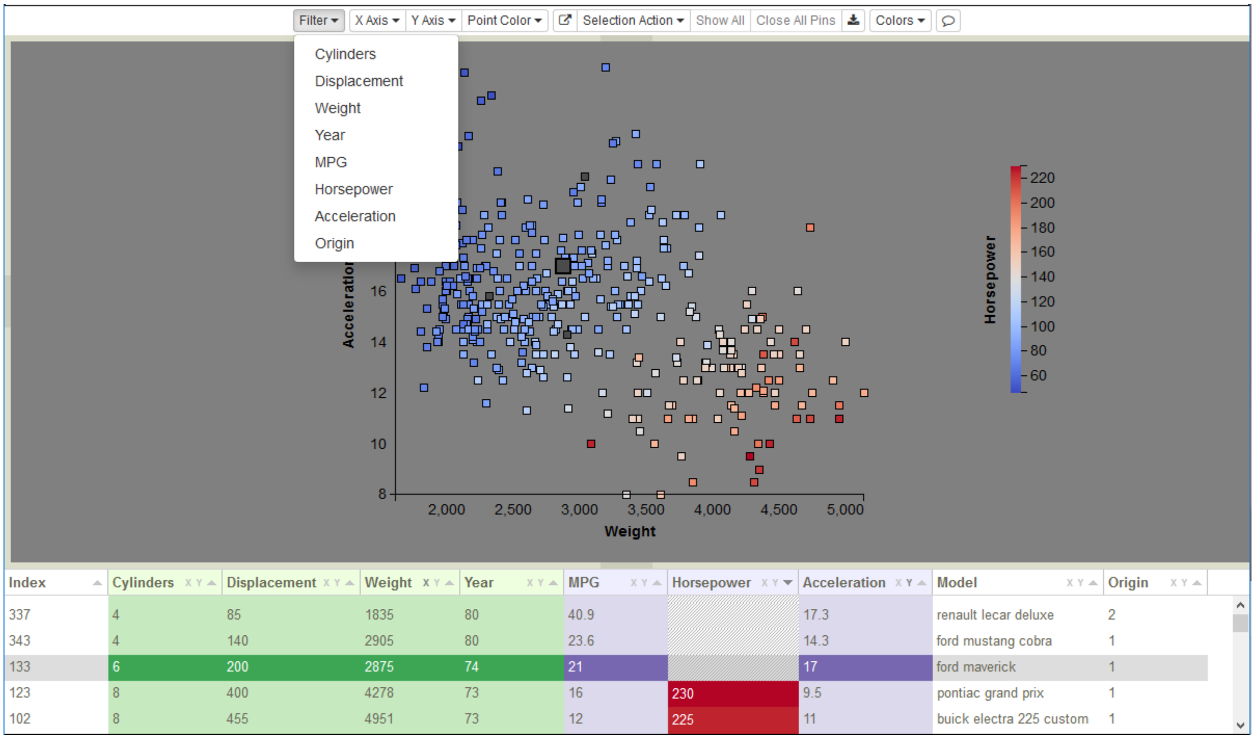
ensemble size. Additionally, the *Scatterplot View* enables the inclusion of points with missing data. Although a point still requires values to exist in both axis variables to define its coordinates, none of the other variables need to have a value for the point to be displayed. The points colored dark gray in Figure 27 are examples of rows with missing values for *Horsepower*. One of these is row 133, which is highlighted in both the scatterplot (enlarged point) and the table (darkened row).

Filtering

There are two mutually exclusive mechanisms in the Parameter Space scatterplot for removing points from the view, filtering and hiding selected points. Once a filter is present, point hiding operations in the *Selection Action* dropdown are disabled. If points had previously been hidden using point hiding, they immediately become visible again as soon as any filter is selected. This ensures that during filtering, visible points only reflect the filtering choices. We did this because we wanted screenshots of the model that included filters to be self-documenting as to which values were retained and which had been removed. This is useful for interpreting slides or figures outside of the Slycat™ interface.

However, sometimes manually removing points through selection is the only way to define a set of points matching criteria that are not achievable through filtering. Since multiple selections may be required to construct this set, we want to avoid discarding it unintentionally. Consequently, the last visibility state is saved when filters are enabled. Once all filters have been removed, the scatterplot returns to its previous state of visibility (i.e. any points that were previously hidden through the selection mechanism will return to being hidden). To explicitly return to a state of total point visibility, click the *Show All* button.

Similarly, filter states are saved, so if you remove a filter and then reinstate it, the filter will resume with its previous settings (button states or slider range). This way, if you accidentally remove a filter and don't remember the settings, you can instantly restore your state.

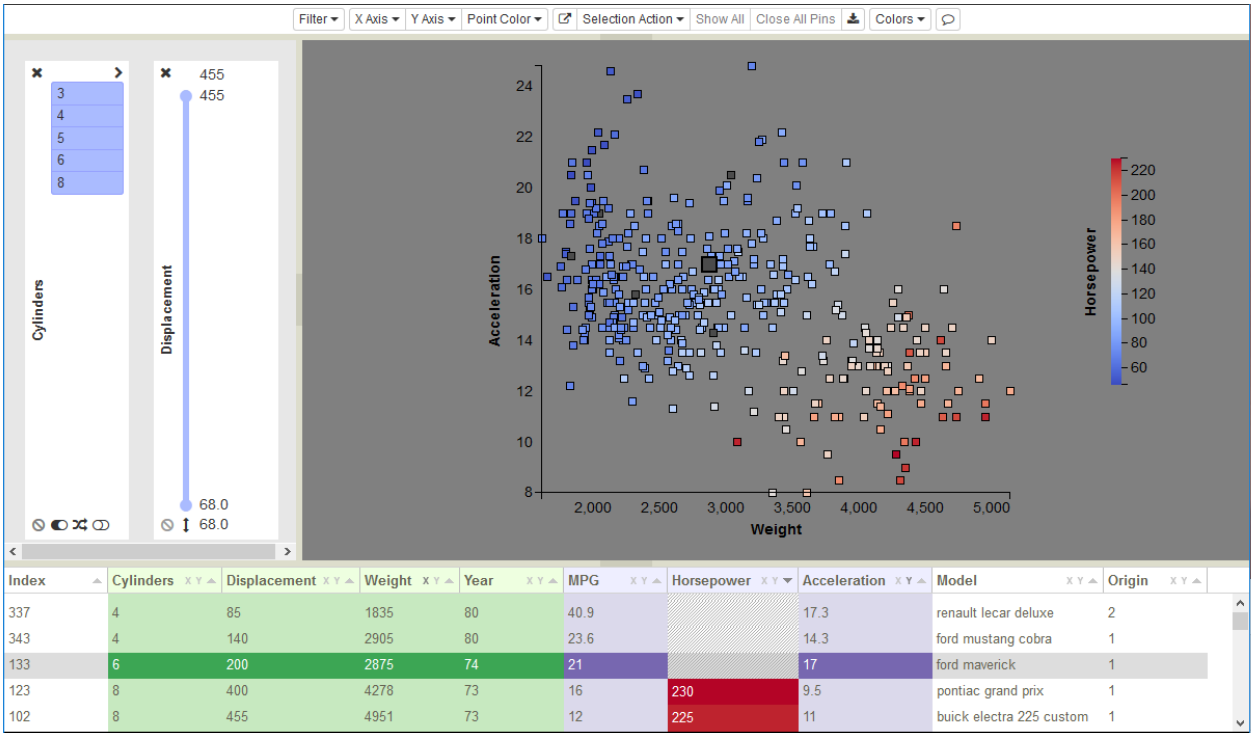


Index	Cylinders	Displacement	Weight	Year	MPG	Horsepower	Acceleration	Model	Origin
337	4	85	1835	80	40.9		17.3	renault lecar deluxe	2
343	4	140	2905	80	23.6		14.3	ford mustang cobra	1
133	6	200	2875	74	21		17	ford maverick	1
123	8	400	4278	73	16	230	9.5	pontiac grand prix	1
102	8	455	4951	73	12	225	11	buick electra 225 custom	1

Fig. 36: **Figure 29: Filter dropdown variable list for cars data.**

To add a filter, click on the *Filter* button to drop down a list of variables, such as those for the cars data shown in Figure 29. Selecting a variable from the list displays a filter consisting of either a set of buttons for categorical variables (see

Creating a Parameter Space Model for how to define a categorical variable), or a slider for continuous variables (Figure 30 shows examples of each). To remove a filter, click the ✖ icon in the upper left corner of the filter. In both types of filters, blue coloration of the buttons or the slider range indicates values that are visible, whereas gray is used for values that are not visible. As filters are changed to remove values, corresponding points are removed from the scatterplot and their rows in the table are grayed out. Conversely, as filters add values back into the visible set, corresponding points will reappear in the plot and their table rows will be re-colored.



Index	Cylinders	Displacement	Weight	Year	MPG	Horsepower	Acceleration	Model	Origin
337	4	85	1835	80	40.9		17.3	renault lecar deluxe	2
343	4	140	2905	80	23.6		14.3	ford mustang cobra	1
133	6	200	2875	74	21		17	ford maverick	1
123	8	400	4278	73	16	230	9.5	pontiac grand prix	1
102	8	455	4951	73	12	225	11	buick electra 225 custom	1

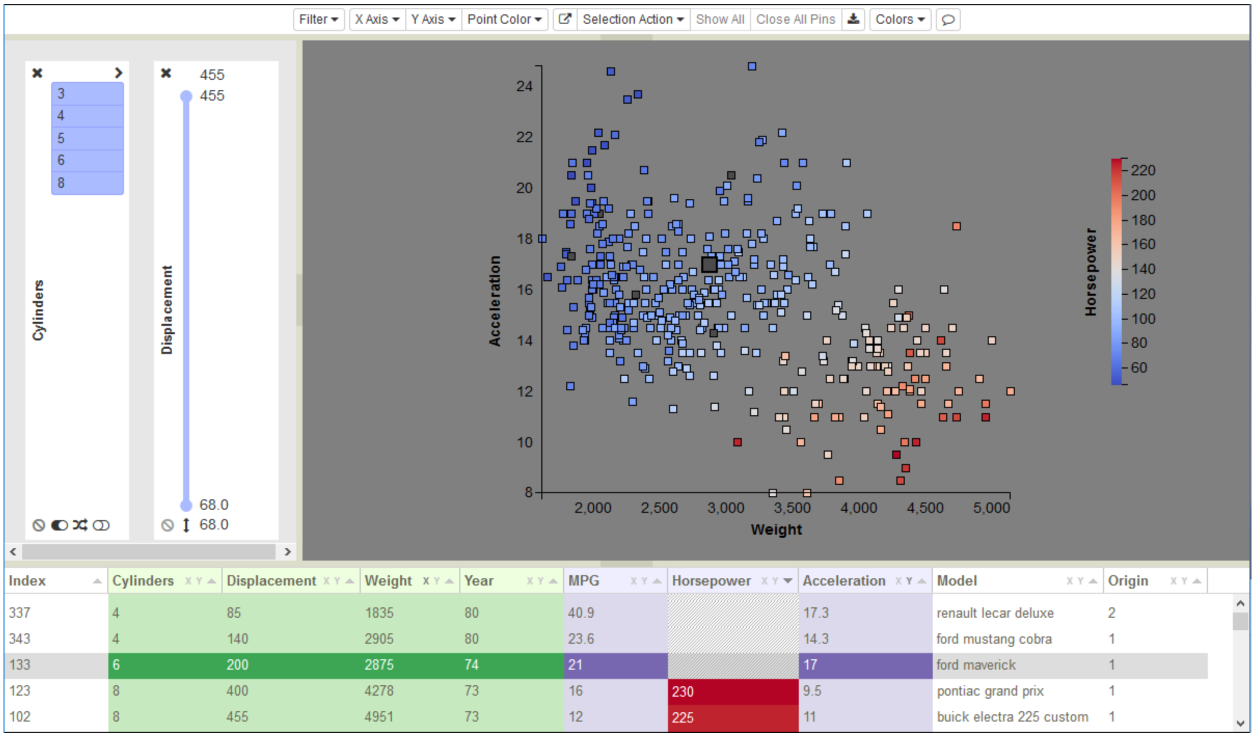
Fig. 37: **Figure 30: Examples of both a categorical and a continuous filter.**

Categorical Filters

Initially, all filter buttons are on (visible values are colored blue), as shown below. So all values are visible the first time that a filter is instantiated. Clicking a blue button sets its state to off, colors the button gray (hidden values are colored gray), hides points with that value in the scatterplot, and grays/fades the corresponding rows in the table. Buttons are toggles, so clicking the button again restores that value's point visibility in the plot and re-colors those rows in the table.

In addition to individual buttons, group operations are also available through the three icons ◐ 🔀 ○ at the bottom of the filter. Use ◐ to turn all buttons on, ○ to turn all buttons off, and 🔀 to flip the states of all buttons. As an example of using group operations, imagine that you wanted to see only cars with 5-cylinder engines. You could turn all buttons off with ○ , then click button 5 to show just those points. This would be faster than individually turning off the 3, 4, 6, and 8-cylinder buttons.

For categorical filters, sometimes the width of the button label exceeds the width of the button. To expand the filter width, click the › icon in the upper right corner of the filter. This will widen the filter and replace the › icon with the ‹ icon. To collapse the filter back to its original width, click the ‹ icon.



Index	Cylinders	Displacement	Weight	Year	MPG	Horsepower	Acceleration	Model	Origin
337	4	85	1835	80	40.9	▨	17.3	renault lecar deluxe	2
343	4	140	2905	80	23.6	▨	14.3	ford mustang cobra	1
133	6	200	2875	74	21	▨	17	ford maverick	1
123	8	400	4278	73	16	230	9.5	pontiac grand prix	1
102	8	455	4951	73	12	225	11	buick electra 225 custom	1

Fig. 38: **All filter values are initially visible the first time a filter is instantiated.**

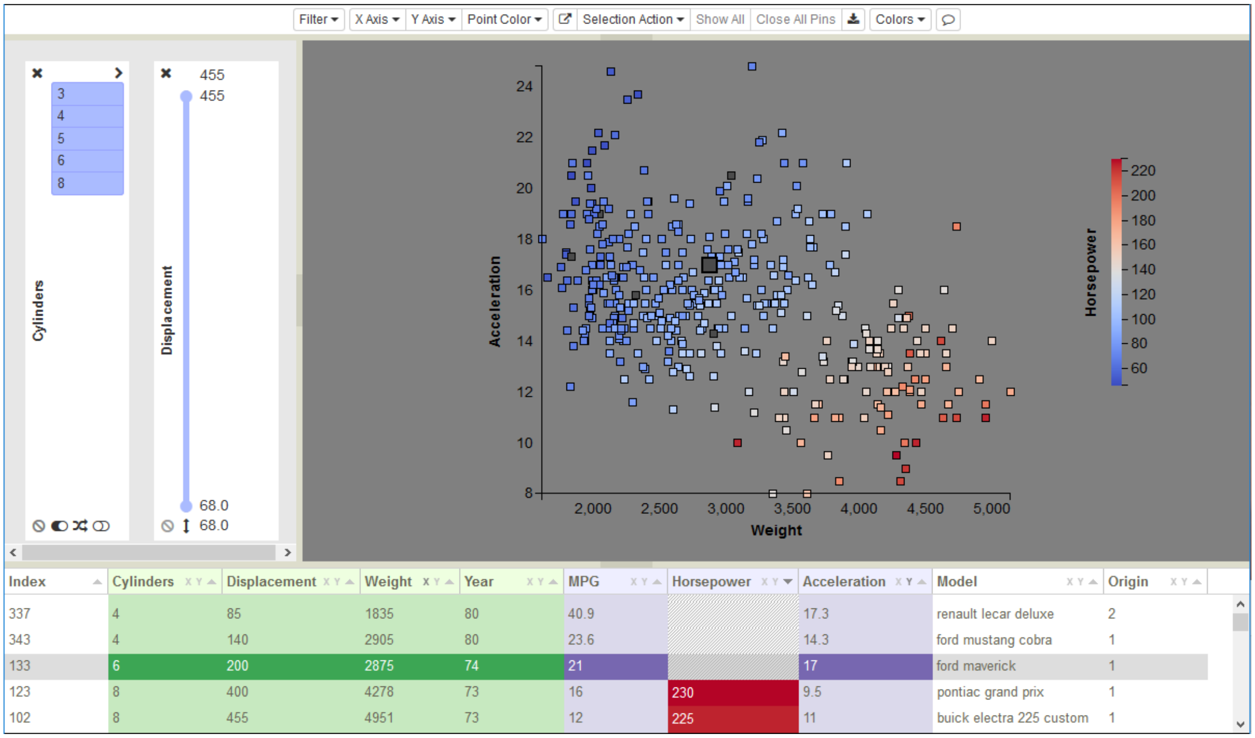
Continuous Filters

The first time a filter is used, the entire range of the variable is visible (drawn in blue), as in Figure 30. The maximum and minimum variable values are at the top and the bottom of the filter, while the max/min values for the visible range are shown next to the slider endpoints. These max/min values interactively track the slider endpoints, changing both the value and position. The excluded portions of the range appear in gray, as demonstrated in Figure 31, where the maximum value has been dragged down to 268 and the minimum value has been pulled up to 166. In addition to grabbing and dragging the endpoints, you can drag the visible range as a unit, creating a sliding window of fixed length. The scatterplot and table are only redrawn when you stop moving the mouse or release the button.

Sometimes the resolution of slider increments can be a problem. The values associated with the set of unique slider positions may not include the value that you want to use as your threshold. So, the minimum and maximum threshold values are editable. To edit the threshold extrema, hover over the value (which will then turn orange as in Figure 32), type in the new value, and hit enter.

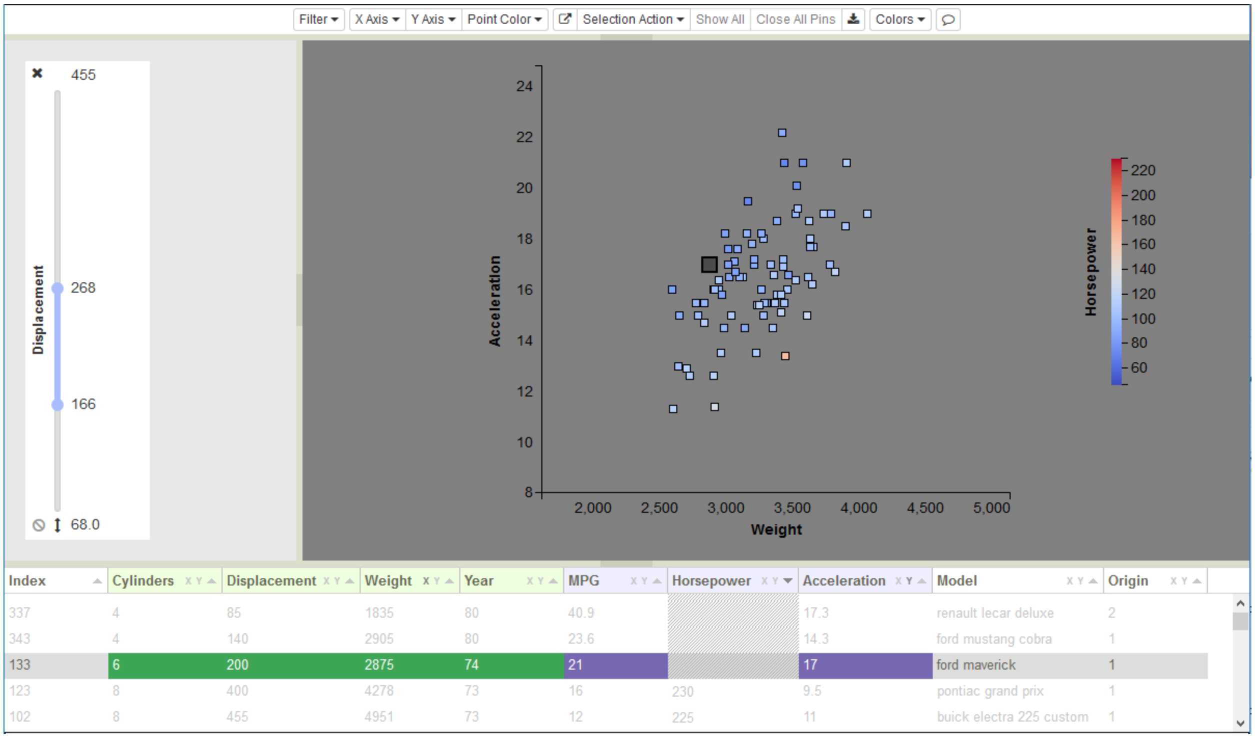
On startup, the slider defines a single region of values that are visible. The ↕ icon beneath the slider indicates that this is the current mode of operation. Clicking on that icon inverts the mode, so values in the middle region become hidden and values at the ends are visible. The icon changes to ⇕, the gray end regions become blue, and the central blue region is drawn in gray. Figure 33 shows the result of inverting the Displacement slider selection shown in Figure 31.

Another type of problem arises when the slider range itself is skewed by anomalous values that are far outside of the normal range for a variable (say points with values of 10,000, when the normal values are between 0 and 1). This forces most of the slider's value range to be empty, with all the points on the extreme ends. To eliminate this value bias, you can reset the overall slider range by editing the maximum and/or minimum range values at the top or bottom of the slider. Values outside of the revised range are hidden in the scatterplot and grayed out in the table. To edit the range extrema, hover over the value (which will then turn orange), type in the new value, and hit enter. In Figure 34,



Index	Cylinders	Displacement	Weight	Year	MPG	Horsepower	Acceleration	Model	Origin
337	4	85	1835	80	40.9		17.3	renault lecar deluxe	2
343	4	140	2905	80	23.6		14.3	ford mustang cobra	1
133	6	200	2875	74	21		17	ford maverick	1
123	8	400	4278	73	16	230	9.5	pontiac grand prix	1
102	8	455	4951	73	12	225	11	buick electra 225 custom	1

Fig. 39: **Figure 30: Entire variable range initially visible in slider.**



Index	Cylinders	Displacement	Weight	Year	MPG	Horsepower	Acceleration	Model	Origin
337	4	85	1835	80	40.9		17.3	renault lecar deluxe	2
343	4	140	2905	80	23.6		14.3	ford mustang cobra	1
133	6	200	2875	74	21		17	ford maverick	1
123	8	400	4278	73	16	230	9.5	pontiac grand prix	1
102	8	455	4951	73	12	225	11	buick electra 225 custom	1

Fig. 40: **Figure 31: Slider after dragging adjusting Displacement max/min values.**

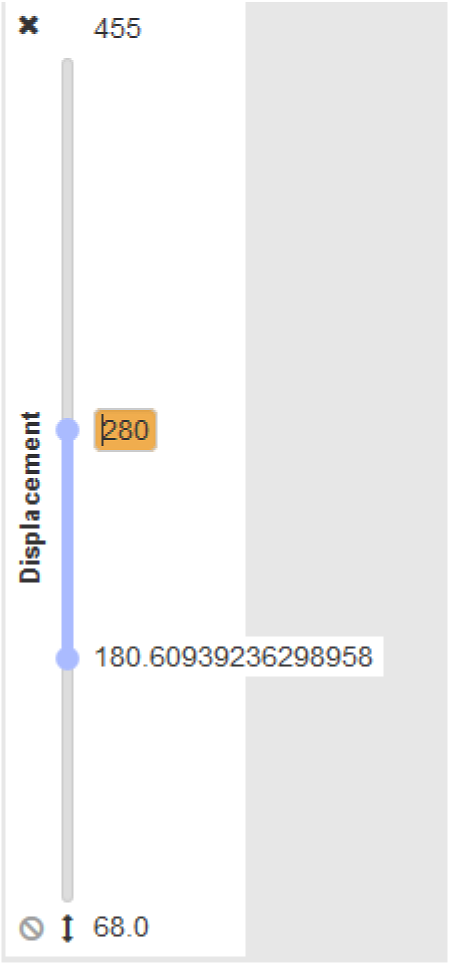
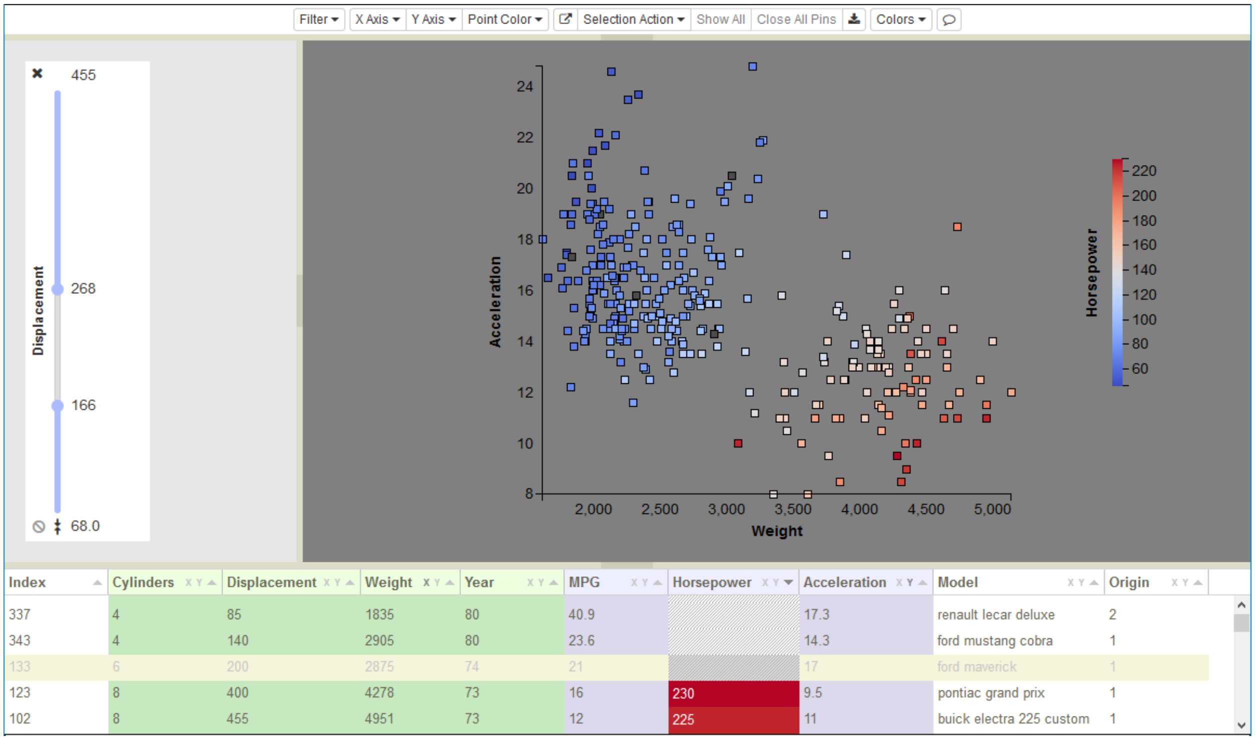


Fig. 41: **Figure 32: Editing a slider maximum threshold value.**



Index	Cylinders	Displacement	Weight	Year	MPG	Horsepower	Acceleration	Model	Origin
337	4	85	1835	80	40.9		17.3	renault lecar deluxe	2
343	4	140	2905	80	23.6		14.3	ford mustang cobra	1
133	6	200	2875	74	21		17	ford maverick	1
123	8	400	4278	73	16	230	9.5	pontiac grand prix	1
102	8	455	4951	73	12	225	11	buick electra 225 custom	1

Fig. 42: **Figure 33: Invert slider selection to display only values outside the selected range.**

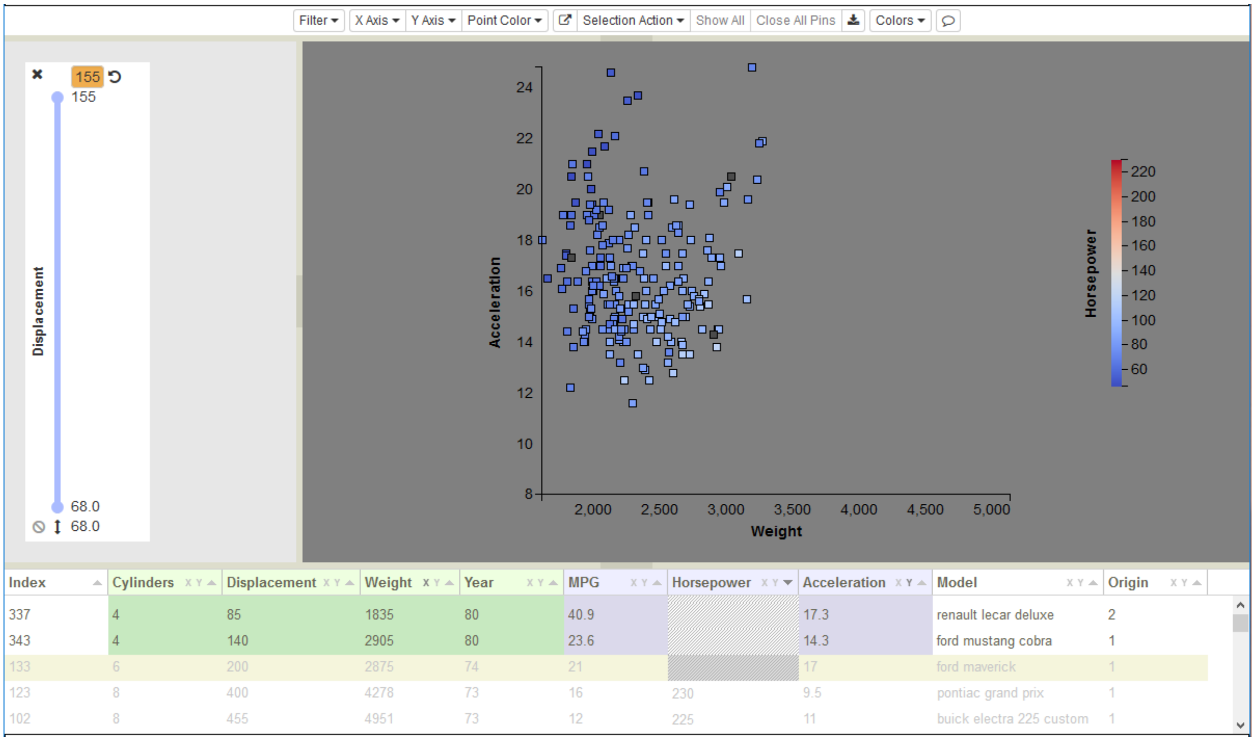
we have reset the filter maximum from 455 to 155. The maximum retains an orange background as a reminder that it has been modified. To reset the value back to the original max/min, click the ⟲ icon.

Missing Values

When filtering is performed on a variable where there are missing or non-defined data values (e.g. NULLs, NANs, Inf, -Inf), such as the *Horsepower* variable in Figure 35, those points corresponding to missing data are eliminated, as they are undefined and therefore not part of the range. This presents a problem if you want to compare points with missing data to points from a subset of that variable's value range. The ⊘ icon at the bottom of each filter enables the inclusion of points with missing data relative to that variable into the scatterplot. When a filter is first instantiated, the missing values are filtered out and the ⊘ icon is gray. Click ⊘ to make those points visible, as shown in Figure 36. The icon acts as a toggle, so clicking ⊘ will hide those points again.

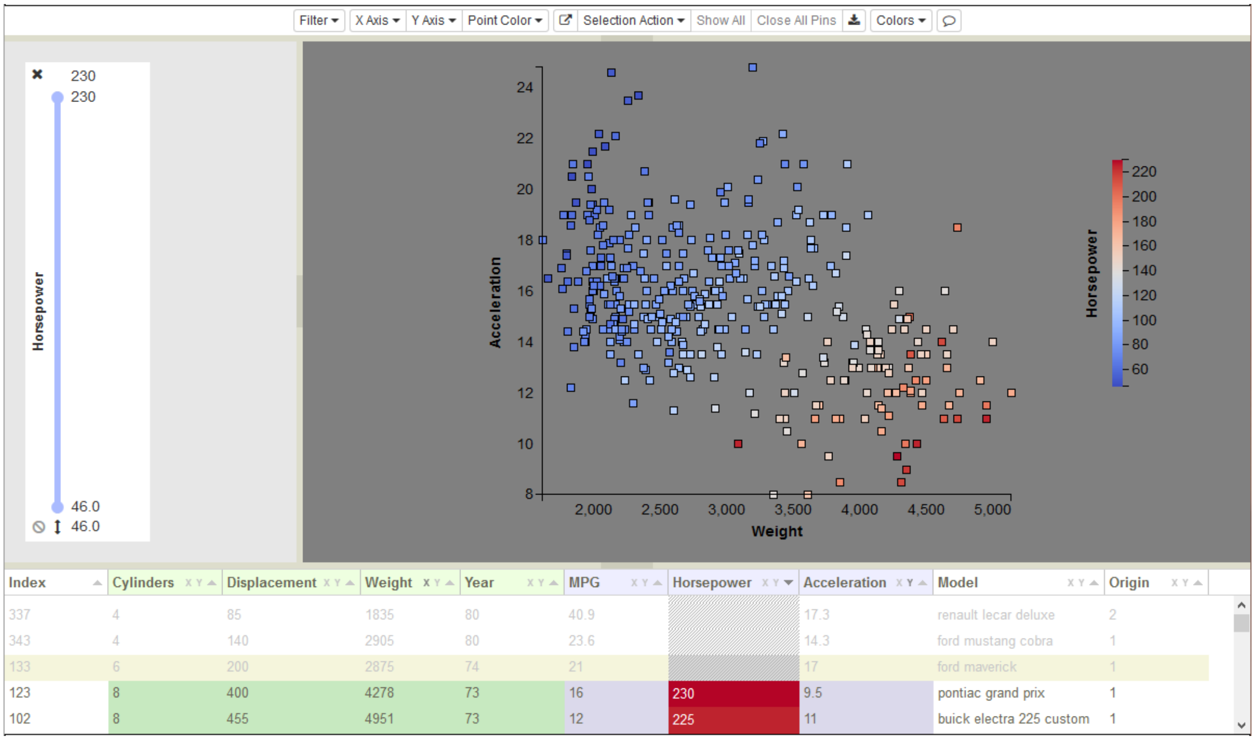
Axis Controls

There are two mechanisms for selecting variables for the X and Y axes. They can be selected using the *X Axis* and *Y Axis* dropdown lists. Or, they can be selected by clicking on the **X** or **Y** icons in the *Variable Table* column headers. The icons are found in each column to the right of the variable name. The two mechanisms are linked, so regardless of which one is used to make the choice, the selection is shown in both. For example, in Figure 37, Weight is highlighted in the dropdown list and the **X** icon is darkened in the *Weight* column of the table. Once the selection is changed, the corresponding axis is immediately redrawn displaying the name and value range for the new variable. Points are also re-rendered as each point's coordinates are changed to reflect the values of the new variable.



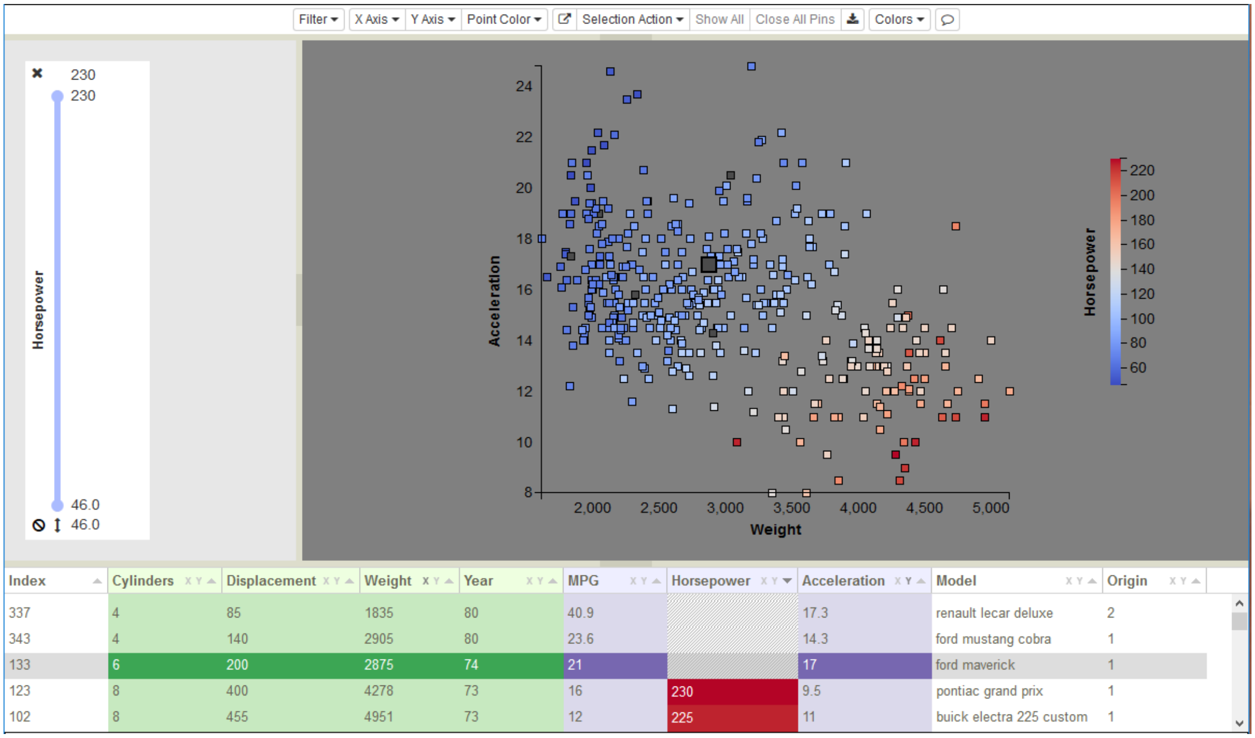
Index	Cylinders	Displacement	Weight	Year	MPG	Horsepower	Acceleration	Model	Origin
337	4	85	1835	80	40.9		17.3	renault lecar deluxe	2
343	4	140	2905	80	23.6		14.3	ford mustang cobra	1
133	6	200	2875	74	21		17	ford maverick	1
123	8	400	4278	73	16	230	9.5	pontiac grand prix	1
102	8	455	4951	73	12	225	11	buick electra 225 custom	1

Fig. 43: **Figure 34: Resetting slider maximum to increase resolution within slider range.**



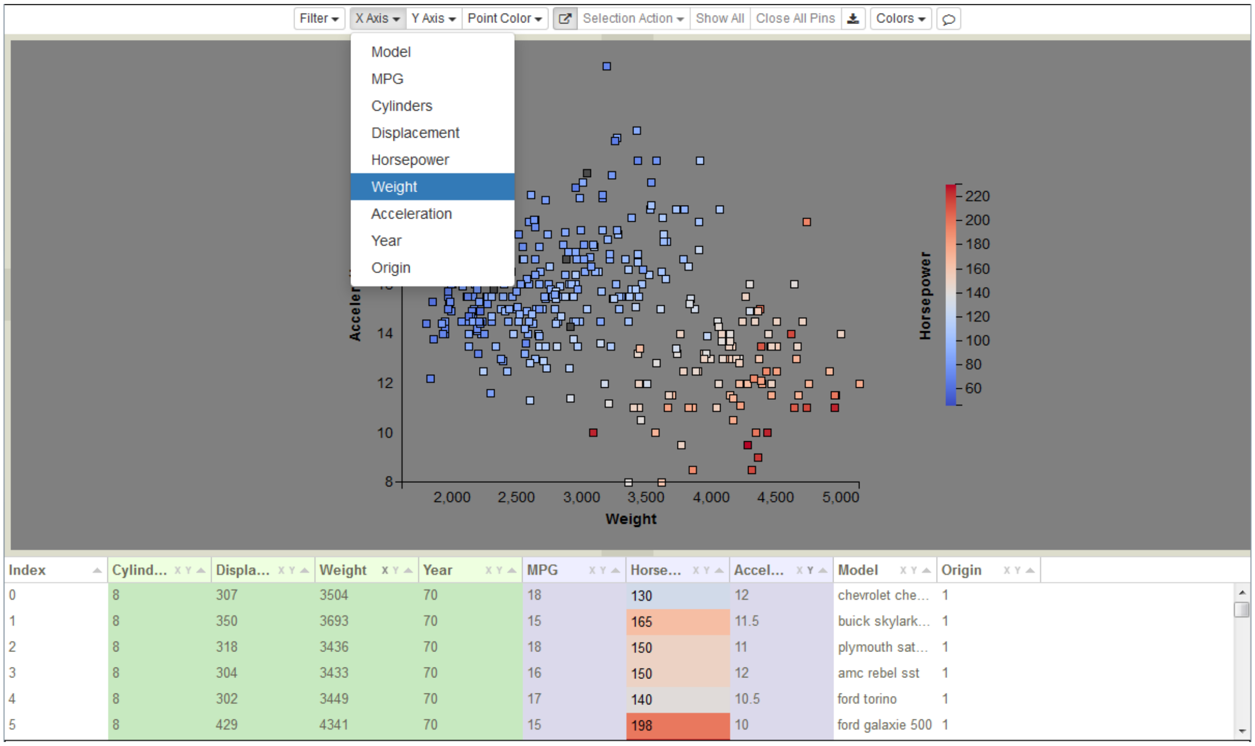
Index	Cylinders	Displacement	Weight	Year	MPG	Horsepower	Acceleration	Model	Origin
337	4	85	1835	80	40.9		17.3	renault lecar deluxe	2
343	4	140	2905	80	23.6		14.3	ford mustang cobra	1
133	6	200	2875	74	21		17	ford maverick	1
123	8	400	4278	73	16	230	9.5	pontiac grand prix	1
102	8	455	4951	73	12	225	11	buick electra 225 custom	1

Fig. 44: **Figure 35: Filtering normally excludes points with missing values.**



Index	Cylinders	Displacement	Weight	Year	MPG	Horsepower	Acceleration	Model	Origin
337	4	85	1835	80	40.9		17.3	renault lecar deluxe	2
343	4	140	2905	80	23.6		14.3	ford mustang cobra	1
133	6	200	2875	74	21		17	ford maverick	1
123	8	400	4278	73	16	230	9.5	pontiac grand prix	1
102	8	455	4951	73	12	225	11	buick electra 225 custom	1

Fig. 45: **Figure 36: Filtered values plus missing-valued points.**

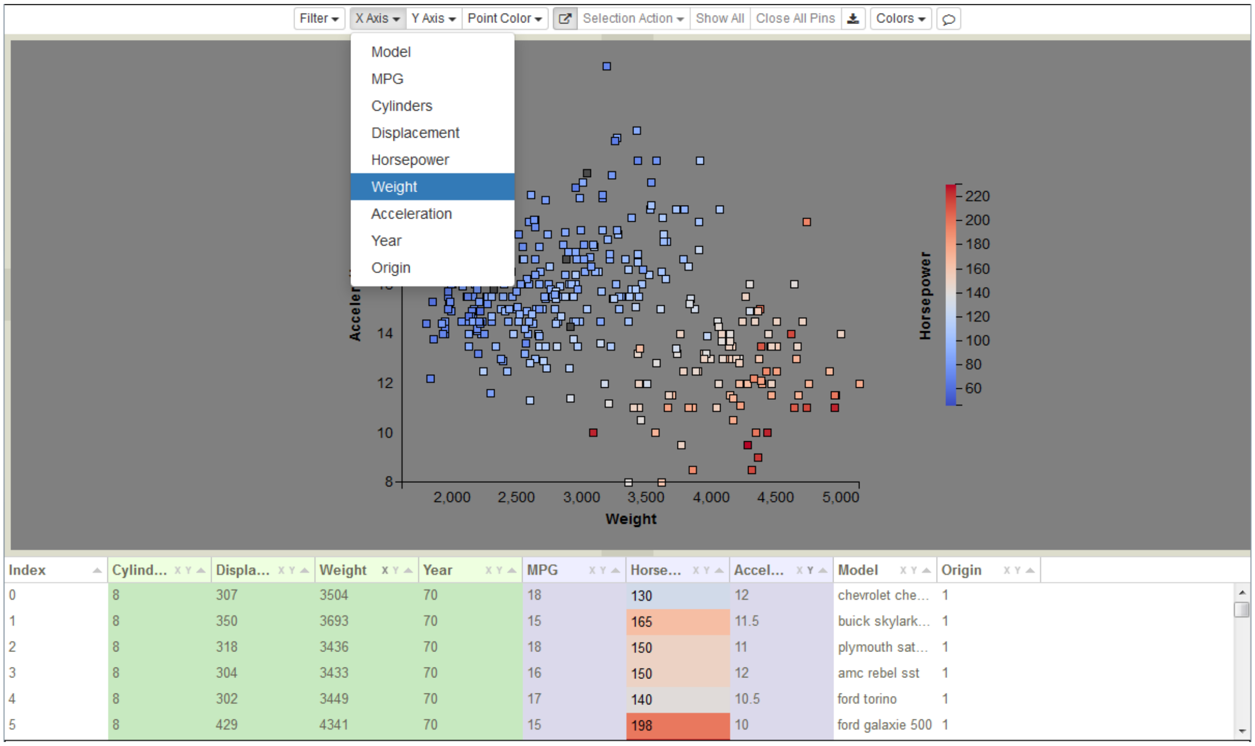


Index	Cylind...	Displa...	Weight	Year	MPG	Horse...	Accel...	Model	Origin
0	8	307	3504	70	18	130	12	chevrolet che...	1
1	8	350	3693	70	15	165	11.5	buick skylark...	1
2	8	318	3436	70	18	150	11	plymouth sat...	1
3	8	304	3433	70	16	150	12	amc rebel sst	1
4	8	302	3449	70	17	140	10.5	ford torino	1
5	8	429	4341	70	15	198	10	ford galaxie 500	1

Fig. 46: **Figure 37: Selecting the X Axis variable using the dropdown list. Horsepower is used to color-code the points.**

Point Color

Similarly, there are two mechanisms for selecting the variable for color-coding individual points. Either select the variable in the *Point Color* dropdown list, or click on the variable name in the *Variable Table* column header. The two mechanisms are linked, so regardless of which one is used to make the choice, the selection is shown in both. The variable name is highlighted in the dropdown, and the backgrounds of that column's elements in the table are color-coded to match the corresponding point's color in the scatterplot. Once the selection is changed, the points are re-rendered using the new encoding and the legend is changed to reflect the name and value range for the new variable. In the image below, the points are color-coded by each car's *Horsepower* value. Although the variable name is not fully visible in the table (we initially display each column at a uniform width, which tends to truncate many of the names), you can immediately tell which column has been selected for color-coding because the background coloring of the cells in that column makes it stand out.



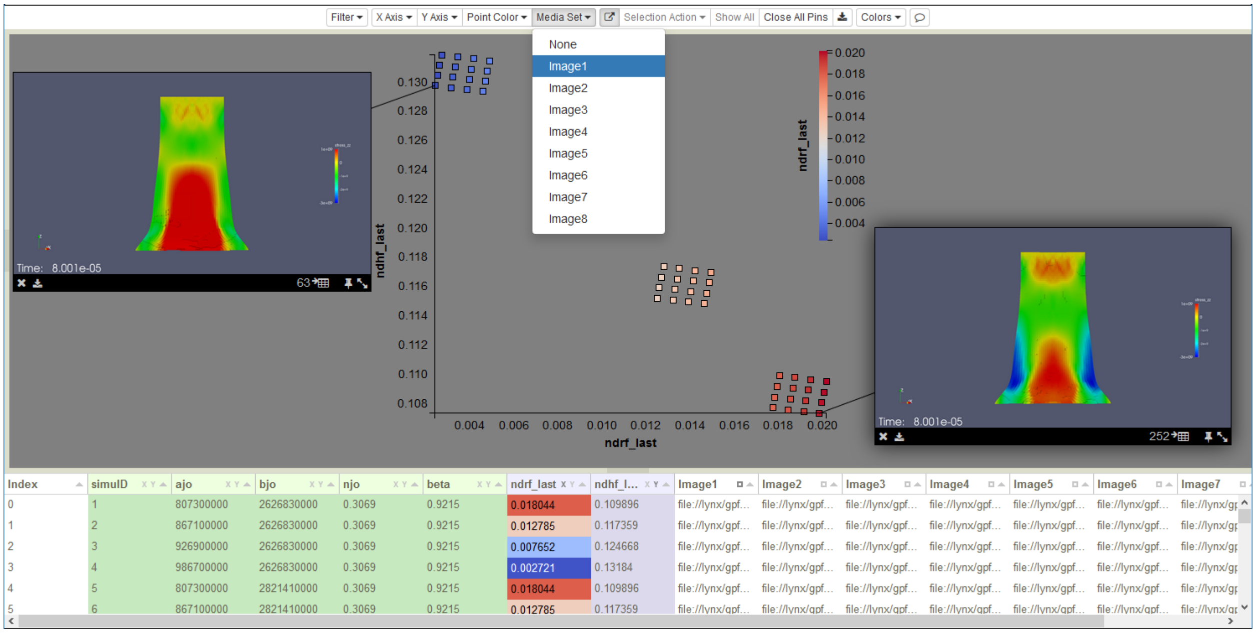
Index	Cylind...	Displa...	Weight	Year	MPG	Horse...	Accel...	Model	Origin
0	8	307	3504	70	18	130	12	chevrolet che...	1
1	8	350	3693	70	15	165	11.5	buick skylark...	1
2	8	318	3436	70	18	150	11	plymouth sat...	1
3	8	304	3433	70	16	150	12	amc rebel sst	1
4	8	302	3449	70	17	140	10.5	ford torino	1
5	8	429	4341	70	15	198	10	ford galaxie 500	1

Fig. 47: **Points color-coded by Horsepower values.**

Media Set

If media variables are present in any of the columns of the input data table, a *Media Set* button will appear to the right of the *Point Color* button. Media files are broadly defined. We currently support viewing various image formats, videos, and STLs (geometry files for a 3D printer). Each media variable in the input data table consists of a column with a shared file type, though blank entries are permitted within a column (i.e. not all ensemble members are required to have an image or video for media viewing to be used). Typically, the files in a column share a common theme, such as images showing the final state of a simulation, or an event-triggered animation, or the geometry of a specific part at a specific time. In all cases, a media file is described by a URI that provides a full path to a source file. Each URI must have the format: file://machine_name/absolute_directory_path/filename.ext.

Initially, media retrieval is disabled because the *Media Set* selection is set to *None*. Once a media variable has been selected, hovering over a scatterplot point will retrieve that point's remote media file as defined by the URI. For the first remote access, authentication will be required. Hovering provides a convenient means for rapidly examining and



Index	simulD	ajo	bjo	njo	beta	ndrf_last	ndhf_l...	Image1	Image2	Image3	Image4	Image5	Image6	Image7
0	1	807300000	2626830000	0.3069	0.9215	0.018044	0.109896	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gp
1	2	867100000	2626830000	0.3069	0.9215	0.012785	0.117359	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gp
2	3	926900000	2626830000	0.3069	0.9215	0.007652	0.124668	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gp
3	4	986700000	2626830000	0.3069	0.9215	0.002721	0.13184	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gp
4	5	807300000	2821410000	0.3069	0.9215	0.018044	0.109896	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gp
5	6	867100000	2821410000	0.3069	0.9215	0.012785	0.117359	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gp

Fig. 48: **Figure 38: Media Set Selection of Image1.**

comparing outputs generated by in situ visualization codes, such as Catalyst and SpyPlot. Viewers can be repositioned within the scatterplot by dragging. Each viewer is connected to its associated point in the scatterplot by a line, one end of which always tracks the viewer position (see Figure 38). For videos and STLs, the viewers include interaction controls (e.g. play buttons, or rotation axis selectors).

As with *Point Color*, there are two mechanisms for selecting which *Media Set* variable to display. Either select a variable in the *Media Set* dropdown list, or click on the small square to the right of the variable name in the *Variable Table* column header. The two mechanisms are linked, so regardless of which one is used, the selection is shown in both. As seen in Figure 38, the selected variable name (*Image1*) is highlighted in the dropdown, and the square to the right of the variable name is darkened in the column header. Once the selection is changed, hover will reference the new column's URIs when retrieving remote files, but currently visible images will not be affected. This allows you to compare different media variables from one or more simulations. Selecting *None* (the first choice in the dropdown list) disables the hover response.

The viewer in Figure 39 shows the set of standard viewer icons in the strip across the bottom of the image. From left to right, each viewer has a close icon ✖, a download icon ⤓, an index row icon ➔▦, a pinning icon 📌, and a resize icon ⤡. Viewers remain visible while the mouse remains within the viewer boundaries. Either clicking on the 📌 icon, resizing, playing a movie, or moving the viewer acts to *pin* the viewer in the scatterplot. A *pinned* viewer remains visible until your explicitly close it, either by clicking its ✖ icon, or by simultaneously closing all pinned views using the *Close All Pins* button. To simultaneously retrieve and pin media for a group of points, first select the points, then select pin in the *Selection Action* dropdown list. Using the 📌 icon within a viewer both *pins* and shrinks it to a standard thumbnail size. For arbitrary viewer sizing, press and hold the left mouse button on the ⤡ icon while dragging the corner. To download a copy of the media object to your local machine, click the ⤓ icon.

Note that Slycat™ video functionality is not available for all movie formats. In fact, due to technical issues related to our web-based delivery, videos must be created in a very specific way to be viewable in Slycat™.

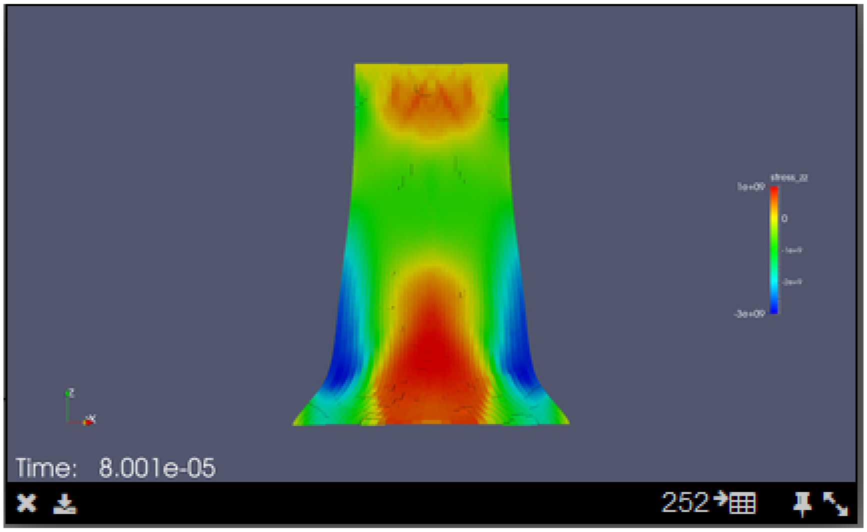


Fig. 49: **Figure 39: Image viewer with close, download, index row, pinning, and resizing icons.**

Video Source Files

There is no standardized support for videos between browsers. We have found that the h264 codec in combination with an mp4 container format is compatible with Firefox, Chrome, and Safari on both Windows and Mac platforms. In our testing, we have found that initial key frames are frequently lost, rendering the following compressed frames useless. This requires explicit key frame forcing during movie creation. We use the ffmpeg utility to convert images into videos. Make sure that your version of ffmpeg was built with the h264 library, since some versions of ffmpeg don't include this codec by default.

If you are within Sandia, we provide this custom version of the library on the cluster machines. You can generate Slycat™ compatible movies as follows:

```
> module load slycat
```

If your images are PNGs, they must be first converted to JPG format (ffmpeg won't complain about the input images being PNG, but the movie that it generates won't play). If you already have JPG images, skip this step:

```
> mogrify -format jpg myImageName.0*
```

This last step generates the mp4. Don't forget to enclose the image path in single quotes:

```
>  ffmpeg -pattern_type glob -i '/someDisk/someUser/someDirectoryPath/myImageName.0*.
↪jpg' -force_key_frames 0.0,0.04,0.08 myMovieName.mp4
```

Automatic Scaling

The auto-scale button ↗ (to the right of *Media Set*) enables/disables automatic scaling of the scatterplot axes when points are hidden or filtered out (these two mechanisms are mutually exclusive, see *Selection Action* and *Filtering*). As points are removed, the locations of the remaining points shift to span the available space and the axes are redrawn to reflect the reduced value range. The mapping between point colors and the values shown in the legend is also modified to reflect the reduced value range (i.e. the new maximum value is now mapped to the brightest red and the new minimum value maps to the deepest blue, maximizing the color distinctions between the remaining points). Compare the scatterplots in Figure 40 and Figure 41 to see the effects of using auto-scaling.

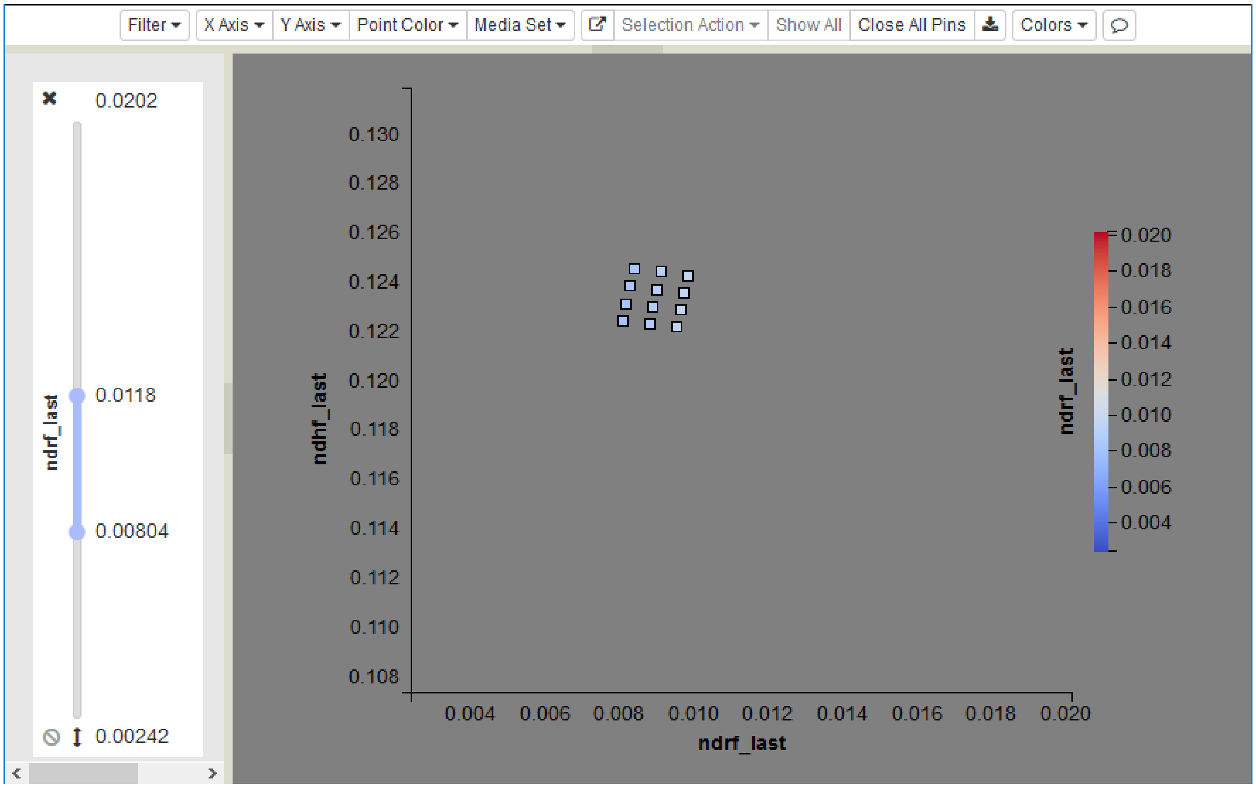


Fig. 50: **Figure 40: Filtered scatterplot with auto-scaling off. Note that most of the scatterplot is empty.**

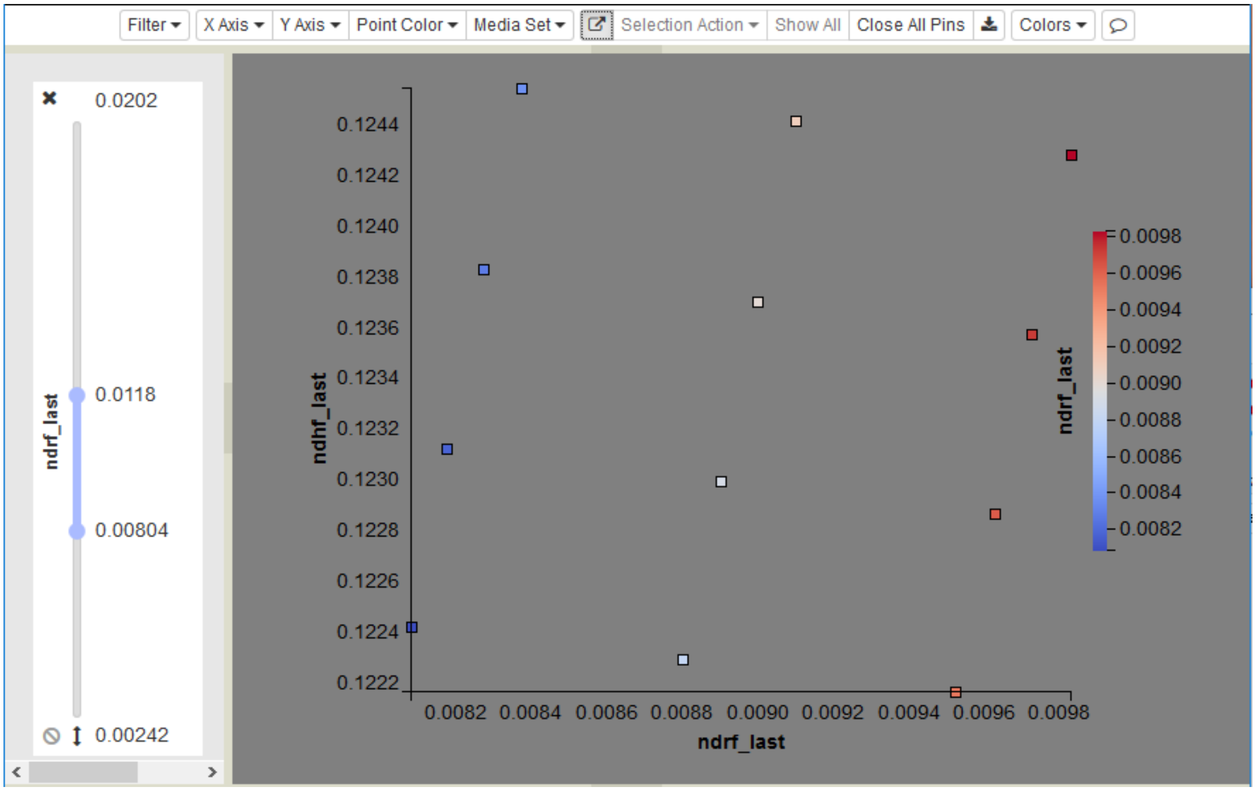


Fig. 51: **Figure 41: The same filtered scatterplot with auto-scaling enabled. The color range is scaled as well.**

This feature is especially useful if a variable's value range is skewed by a few anomalous points. For example, imagine a data set where most points have x-coordinates in the range from 0 to 1, but a few points have extreme values of 10,000. Including the extreme points means that most of the plot consists of empty space, while the majority of points are drawn on top of one another near the origin. By filtering and rescaling, you can remove the extreme points and have the remaining points fill the plot. Note that automatic scaling is enabled as the default, indicated by the darkened state of the button (Figure 41).

Selection Action

Selected points (see Selecting Points) create a set that is used as the designated input for any of the group operations listed in the *Selection Action* dropdown, as shown in Figure 42. Since filtering and hide/show selections are mutually exclusive within the Slycat™ interface, if there are any filters enabled (visible), all of these actions, except *Pin*, will be disabled.

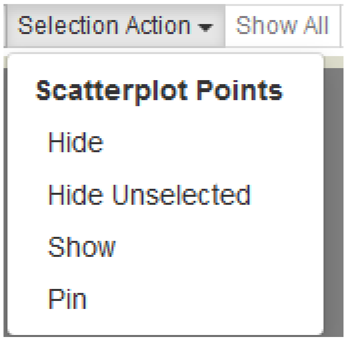


Fig. 52: **Figure 42: Selection Action dropdown list of operations on selected groups of points.**

Hide removes the selected points from the scatterplot and yellows out the associated rows from the table. The points are still selected, but they are no longer visible, which is why their rows are colored yellow instead of white. If automatic scaling is enabled (see Automatic Scaling), the remaining visible points in the scatterplot will be shifted and recolored to reflect any changes in the value range.

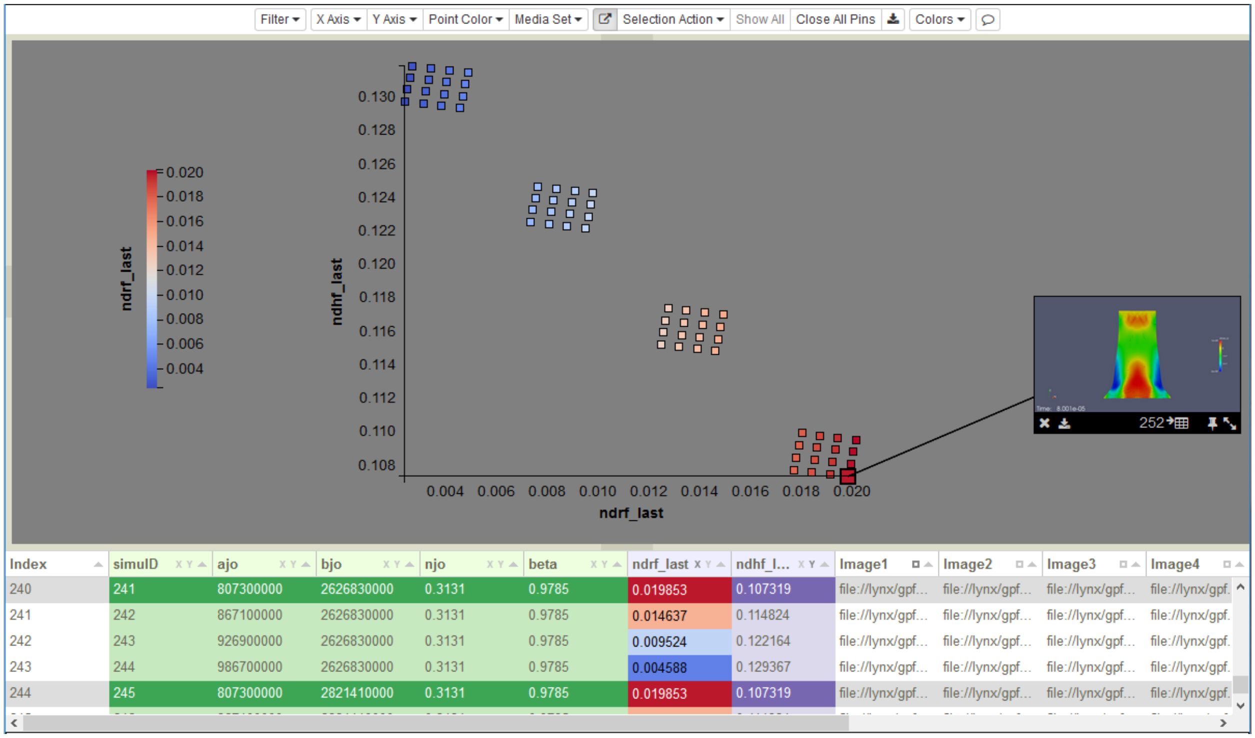
Hide Unselected performs the same operation on the set of points that are not selected, thereby reducing the visible points to just the selected set.

Show restores the last hidden selection to visibility, both in the scatterplot and in the table. If the selection has been lost by clicking elsewhere within the scatterplot, *Show* will not be able to restore the previously hidden set, since the selected set is now empty. For the same reason, there is not a function to restore points hidden using *Hide Unselected*. However, the *Show All* button (to the right of the *Selection Action* dropdown) can be used to make all hidden points visible again.

Using the currently selected media variable, *Pin* retrieves and pins items for each of the points in the selection set (only if visible), performing the equivalent of a group hover and pin operation. This is much more efficient than doing individual retrieval when there are large numbers of runs to compare. Note that the images may be stacked on top of one another if the associated points are coincident, as is the case in Figure 43. However, you can separate the images by dragging them apart, as shown in Figure 44.

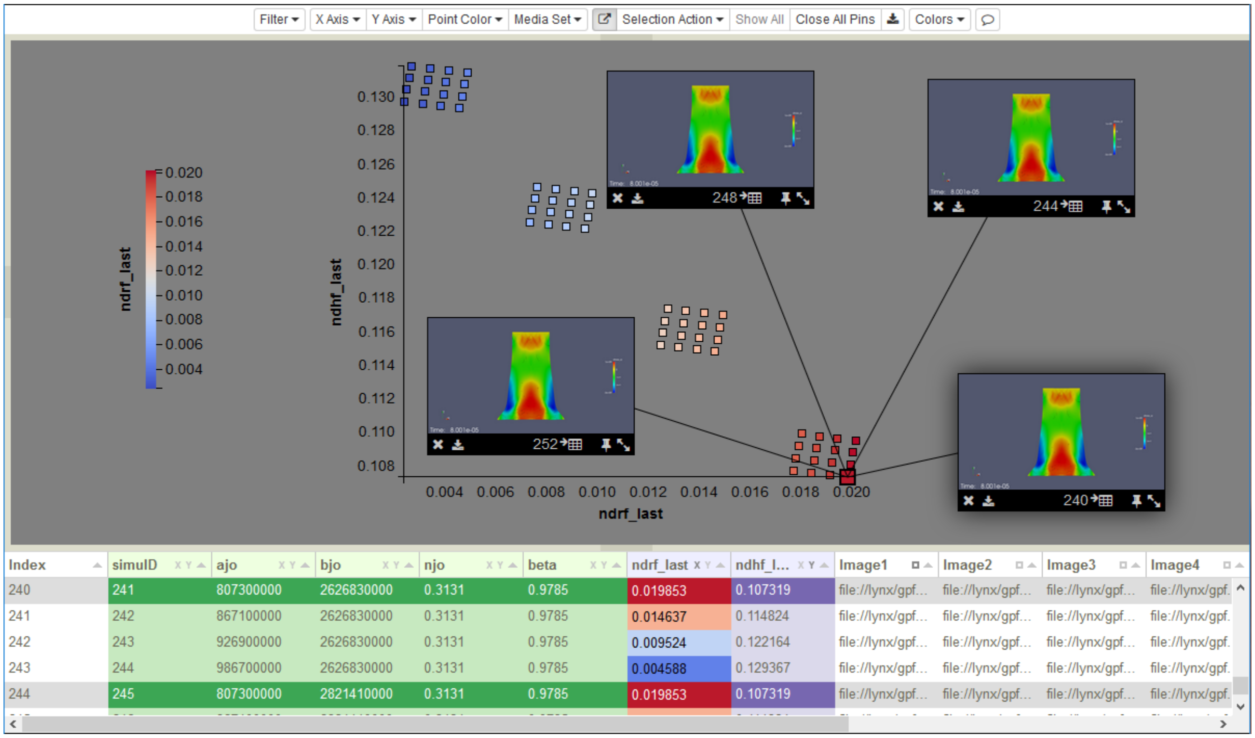
Show All

The *Show All* button restores all points that were hidden using Selection Action to visibility. *Show All* is disabled whenever a filter is present, so it cannot be used to make filtered points visible again.



Index	simulD	ajo	bjo	njo	beta	ndrf_last	ndhf_l...	Image1	Image2	Image3	Image4
240	241	807300000	2626830000	0.3131	0.9785	0.019853	0.107319	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf.
241	242	867100000	2626830000	0.3131	0.9785	0.014637	0.114824	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf.
242	243	926900000	2626830000	0.3131	0.9785	0.009524	0.122164	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf.
243	244	986700000	2626830000	0.3131	0.9785	0.004588	0.129367	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf.
244	245	807300000	2821410000	0.3131	0.9785	0.019853	0.107319	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf.

Fig. 53: **Figure 43: Four selected runs are coincident in the scatterplot, so the associated pinned images are stacked.**



Index	simulD	ajo	bjo	njo	beta	ndrf_last	ndhf_l...	Image1	Image2	Image3	Image4
240	241	807300000	2626830000	0.3131	0.9785	0.019853	0.107319	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf.
241	242	867100000	2626830000	0.3131	0.9785	0.014637	0.114824	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf.
242	243	926900000	2626830000	0.3131	0.9785	0.009524	0.122164	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf.
243	244	986700000	2626830000	0.3131	0.9785	0.004588	0.129367	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf.
244	245	807300000	2821410000	0.3131	0.9785	0.019853	0.107319	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf...	file://lynx/gpf.

Fig. 54: **Figure 44: Separating pinned images by dragging. Note that the legend can also be repositioned by dragging.**

Close All Pins

The *Close All Pins* button closes all pinned media in the view. It provides a quick method to clear a Parameter Space model of all viewers.

Video Synchronization 📹

Beyond the video functionality described earlier (see the Media Set section), Slycat™ provides additional fine-grained and group-based video controls. Once the first video is pinned, the interface shown in Figure 45 will appear in the model-specific controls to right of the Download Data Table ⤓ icon. These video controls will remain visible until all videos are closed. For a single video, these global controls provide single step accuracy in advancing or rewinding the animation, which is not possible using the limited controls of an individual viewer. However, the larger goal of this interface is to enable synchronized playback of multiple videos from a single set of controls.

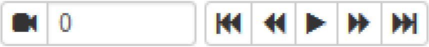


Fig. 55: **Figure 45: Video controls: enable video synch, current video location (seconds from start), go to first frame, step back a frame, play, step forward a frame, and go to last frame, respectively.**

From left to right the controls are as follows: the video synchronization button 📹, a numeric field providing the current video location in seconds from the video start, a button to go to the start of the video ⏮, a button to step backward by one frame ⏪ , play ▶ /pause ⏸ buttons (the icon changes to pause once play is pressed), a button to step forward by one frame ⏩, and a button to go to the end of the video ⏭.
The video synch button is a toggle that enables/disables shared control of multiple videos. The background color of the icon shows the state of the synchronization. The background is gray when it is enabled 📹, and white when it is disabled 📹. When video is synched, the playback buttons operate on all pinned videos. When synch is disabled, the playback operates only on the *current* video. The current video is highlighted by drawing a shadow behind it, making it appear to float above the other videos, such as the middle video in Figure 46. At all times, the video location field shows the current value of the video's elapsed playback time (note that this is not the same as the simulation time stamp for a particular frame). You can directly edit this field to align all videos to the frame that is closest to a specific time of interest in the playback.

Adding Text 💬

To add text, click the note icon 💬 on the right end of the controls. A text window will popup. The intent was to provide two types of annotation: notes and titles. The icon in the upper left toggles between T↕ and 📄 to switch between text sizes, as shown in Figures 47A and 47B. The T↕ icon increases the text size to generate a title. The 📄 icon decreases the text size back to a font more suitable for notes. The text window can be closed and resized using the ☒ and ◢ icons, respectively. The icons are only visible when the mouse is in the window. Figure 48 provides an example showing both types of annotation. Notes in combination with bookmarks can be used to draw the attention of project members to discoveries or generate explanatory visualizations that can be self-contained.

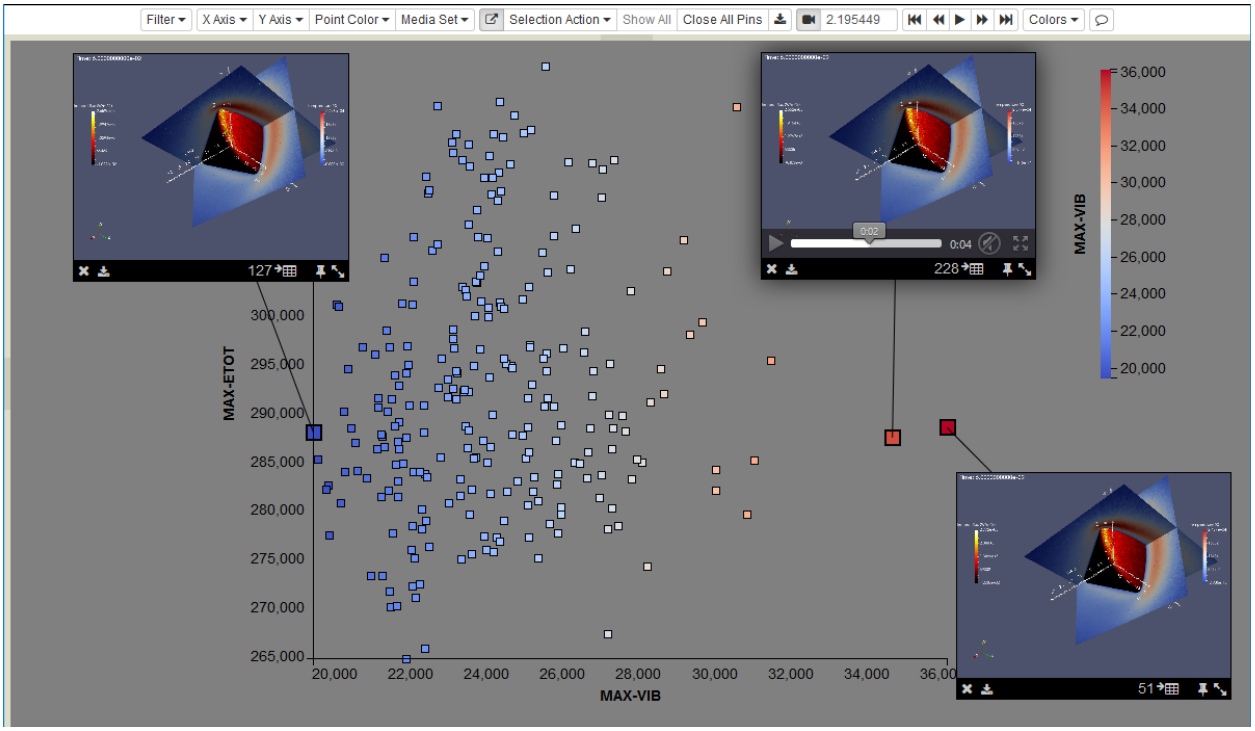


Fig. 56: **Figure 46: Three synched videos, where the middle video is the current video.**

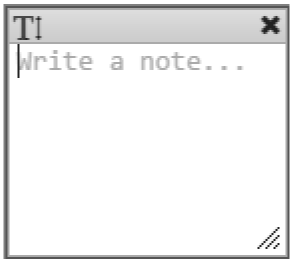


Fig. 57: **Figure 47A: Text window for adding notes.**

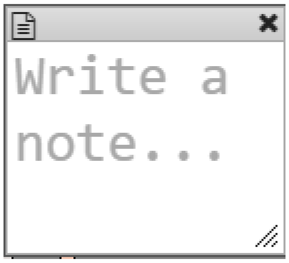
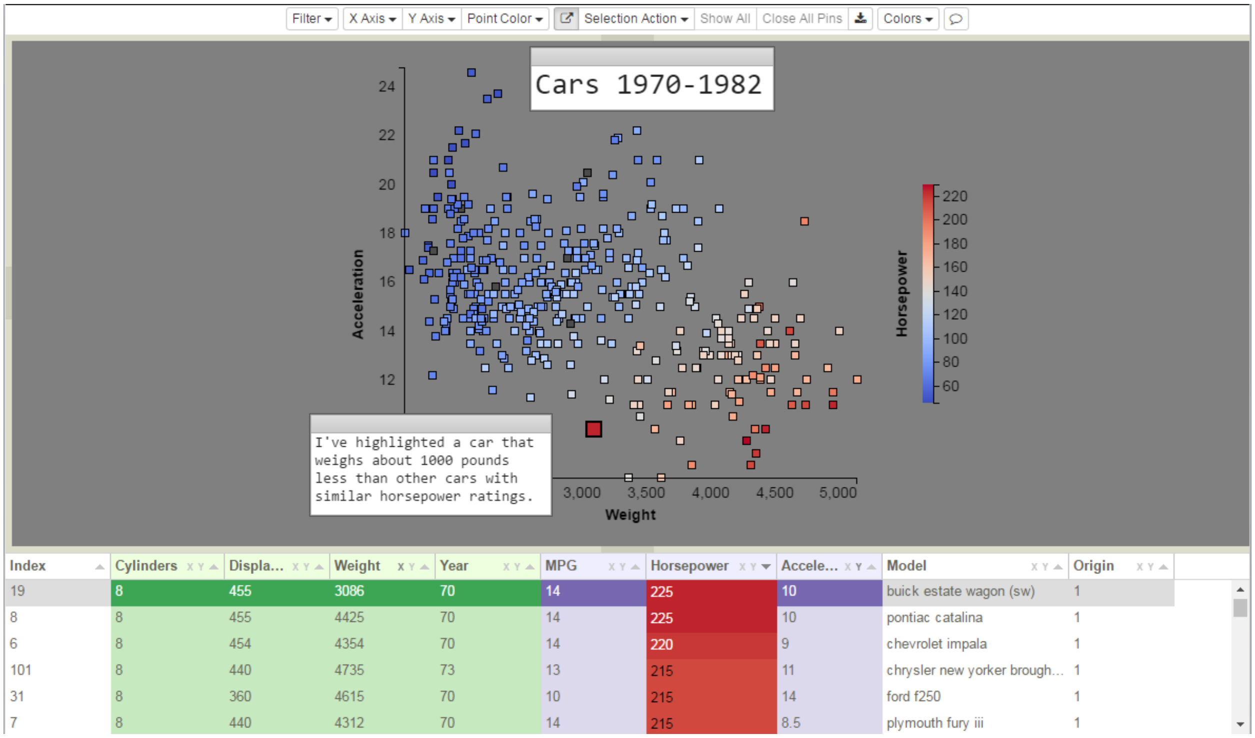


Fig. 58: **Figure 47B: Text window for adding titles.**



Cars 1970-1982

I've highlighted a car that weighs about 1000 pounds less than other cars with similar horsepower ratings.

Index	Cylinders	Displa...	Weight	Year	MPG	Horsepower	Accele...	Model	Origin
19	8	455	3086	70	14	225	10	buick estate wagon (sw)	1
8	8	455	4425	70	14	225	10	pontiac catalina	1
6	8	454	4354	70	14	220	9	chevrolet impala	1
101	8	440	4735	73	13	215	11	chrysler new yorker brough...	1
31	8	360	4615	70	10	215	14	ford f250	1
7	8	440	4312	70	14	215	8.5	plymouth fury iii	1

Fig. 59: **Figure 48: Example of adding text as both a title and a note.**

Variable Table

The *Variable Table* is the same as the table in the CCA model (see Variable Table) with one small difference. **'X'** and **'Y'** icons to the right of variable names in the column headers provide an alternate method to the dropdown lists for selecting the scatterplot axes.

2.1.4 Timeseries Model

The Time Series model was originally developed to evaluate similarities between waveforms generated by electrical circuit simulations. However, this model can be used more generally to compare any set of time series data, so long as the starting and ending times for each sequence of values are the same. Although the time series inputs do not need to be sampled identically, our initial step is to resample since the analysis ultimately requires corresponding samples for comparison. Points in each sequence are binned, with the bin size calculated by dividing the time range into equal intervals. The values of the points within each bin are then averaged. The underlying assumption is that there are enough samples in the sequence to have at least one sample per bin. The number of bins is a user-supplied parameter to the analysis.

Slycat™ calculates a table of the distances between each pair of resampled time series by summing the differences between corresponding points. The distance table is used to create an agglomerative, hierarchical model of similarities between the sequences. The resulting model is a tree, in which the set of time series contained within each subtree are more and more similar as successive subtrees get closer to the leaves. Because our distance metric was developed for electrical simulation data, both amplitude (y value) and timing (x value) must match for a pair of sequences to be regarded as similar (i.e. identical shapes that are shifted in time are not seen as similar).

Time Series Data

Slycat™ accepts two different time series data formats, which we will call *Xyce* and *CSV*. Each input format consists of two parts, a table file describing the entire ensemble, and a time series data file for each simulation in the ensemble. Like the CCA and Parameter Space models, the table is at the heart of the model. For each simulation (for each row in the data table), there must be a file with time series data. Within each of these time series files are sequences of values, sampling one or more output variables over the course of the simulation. It is not necessary that each simulation write the same number of samples into their time series files, but it is required that each simulation have a corresponding data file with matching output variables that cover the same time range.

Xyce Format File Structure

The *Xyce* format consists of Xyce-generated time series files stored within a fixed directory hierarchy. The hierarchy is rooted within a single high-level directory where there must be a *dakota_tabular.dat* file (providing the data table). It is not that the file must be named *dakota_tabular.dat*, but rather the file format must correspond to the *dakota_tabular.dat* files generated by Dakota. Additionally, a set of subdirectories (one per run) must be located in the same directory as the *dakota_tabular* file. These subdirectories should all be named using a template like *workdir.n*, where *n* is the simulation number. Within each subdirectory, there must be a time series file generated by Xyce that is formatted as a *.prn* file. The time series files must all be named identically (the subdirectory defines which simulation generated them), and each file must contain a shared set of time series variables (columns with matching headers within each of the *.prn* files).

CSV Format File Structure

The *CSV* format (such as *heartbeat.dat* files produced by Sierra, or *.csv* outputs from Catalyst), is less structured than the *Xyce* format. The individual time series files need not be stored in the same directory hierarchy as the data table, nor does the directory structure need to follow any structure or naming conventions. Instead, the data table is a *CSV* file, which contains a column of URIs providing full paths to each of the time series files, which must also be *CSV* files (no *.prn* files). Each URI must have the format: file://machine/absolute_directory_path/timeseries_filename.csv.

Time Series Files

Whether we are using *.prn* files or *CSV* files, both formats are essentially tables in which each column is a separate variable and each row is a set of concurrent samples for each of the variable columns. The first line of a time series file contains headers, which provide the names of the time series output variables. Note that in a *CSV* file, we expect to see only a single row of header information consisting of the column names (some physics codes output two rows of header information, with the variable names in the first row and the units in the second row – this is not a legal *CSV* format). At least one column must be a time value (typically the first column).

If your data is not currently in one of these two formats, Excel can be used to create *CSV* files from most common table formats. Note that if output metrics have been created separately in a post-processing step, they will need to be integrated with the inputs to form a single file prior to model creation.

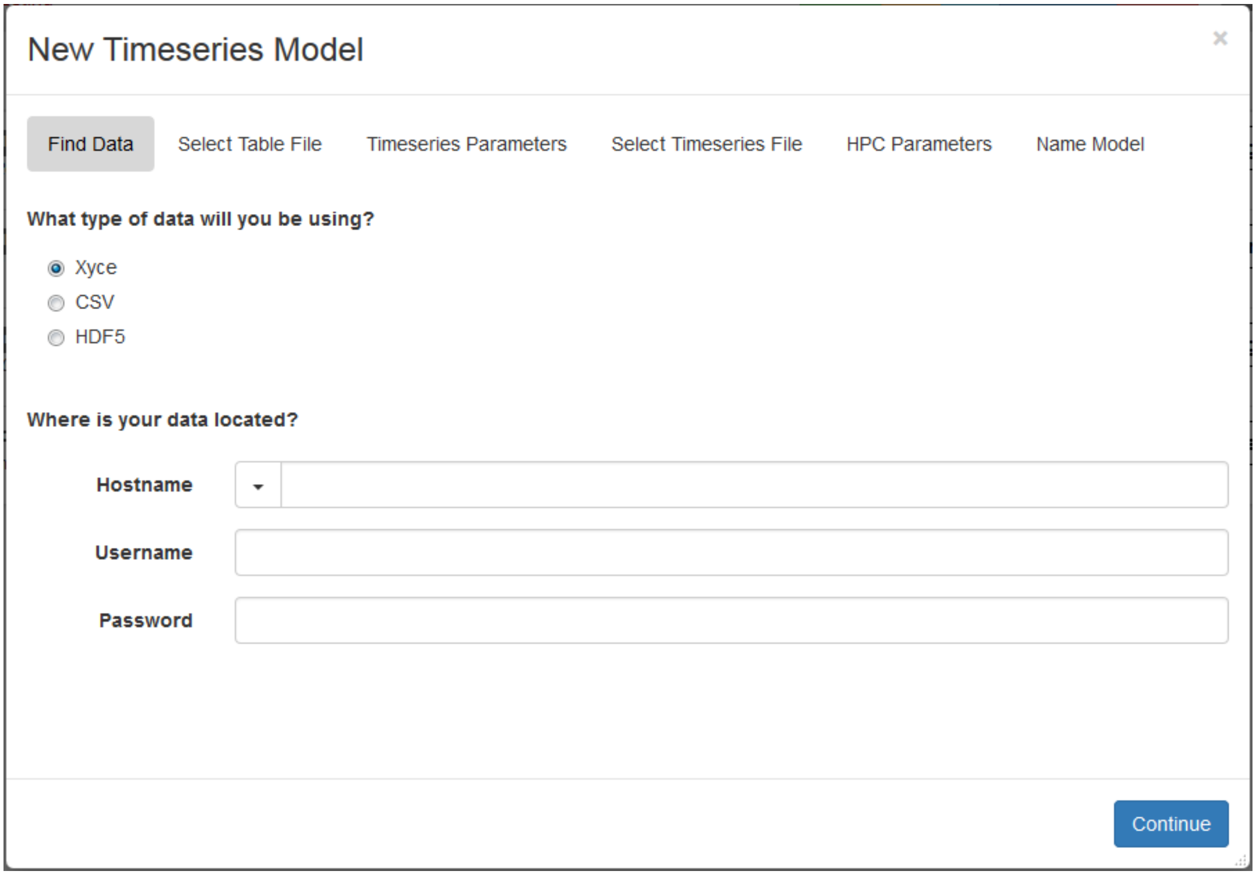
HDF5 Intermediary Format

In the time series creation wizard, both formats are rewritten as *HDF5* files in a temporary Slycat™ directory (we have found that this significantly speeds up our processing compared to working with the originally-formatted files). If you opt to keep these *HDF5* files, they constitute a third data format that the wizard will accept, though be aware that *HDF5* files created through other means are not interchangeable since their internal structures will be different.

Creating a Time Series Model

Creating a Time Series model is more complicated than the models that we have described in previous sections. This is due to the size and structure of the data, combined with the computationally intensive nature of the analysis. The data is stored in multiple files, typically in multiple directories. This data complexity and scale compels the use of parallel processing to reduce the model creation time. Unfortunately, our cluster's batch environment increases complexity through the need for additional High Performance Computing (HPC) parameter choices, uncertain wait times in the job queue, and potentially long processing times. All these factors are at odds with an interactive interface. Consequently, our time series wizard is designed to collect all the necessary information, then autonomously launch the analysis and finish the model creation. You are free to do other things while it completes, although we do provide a means to remotely check on the status of your job through the Slycat™ interface.

To access the wizard, go to your project page, click on the green *Create* button and select *New Timeseries Model* from the dropdown list. A dialog for walking you through the process will then pop up, as shown below. The first page identifies the format of the time series data (see Time Series Data above) and the location of the ensemble's table file. The assumption is that time series data is large and difficult to move, so it will be located on the same remote HPC machine where it was generated. Consequently, we do not provide a Local option, as we do for other model types.



New Timeseries Model

What type of data will you be using?

● Xyce
○ CSV
○ HDF5

Where is your data located?

Hostname

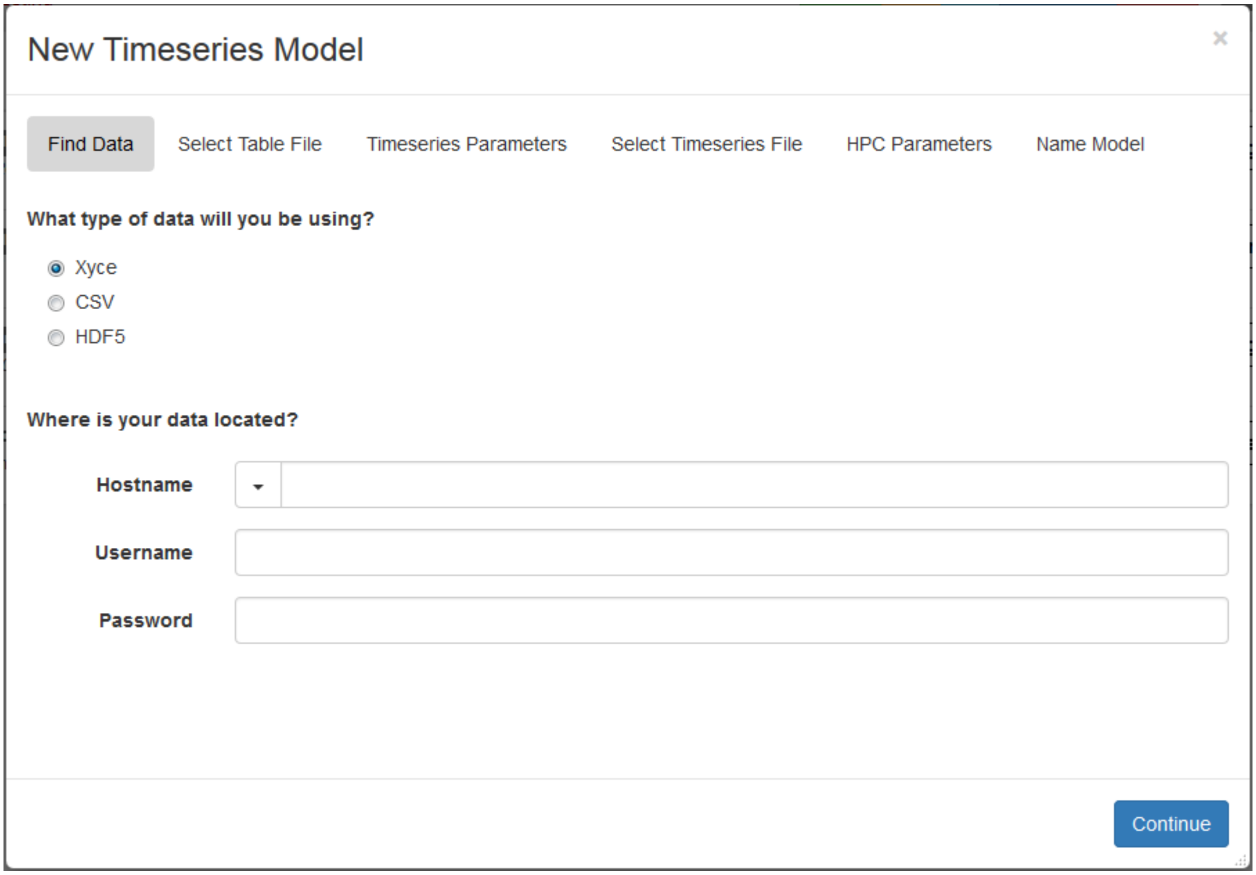
Username

Password

Fig. 60: **Initial dialog screen in Timeseries model creation wizard.**

Find Data Dialog

Using the radio buttons, select the data input type. Next, select a remote host from the *Hostname* dropdown. Unlike CCA or Parameter Space, only hosts included in the dropdown list may be used. This is because time series analysis requires parallel job launching functionality found in the Slycat™ agent, which must be running on the remote machine. Although the Slycat™ agent assists you in remotely submitting the analysis job to the cluster queue, you will be running the job under your own user credentials. Consequently, the host must be a machine on which you already have a user account. Enter your username and password into the associated fields. Hitting the Enter button after typing in your password will both log you into the remote machine and take you to the next screen in the wizard. If you have recently accessed the selected host through Slycat™, the Username and Password fields will not be shown. This is because Slycat™ maintains remote sessions for a fixed period of time after your initial login to reduce the number of login requests. In this case, click the Continue button to advance the wizard.



New Timeseries Model

What type of data will you be using?

● Xyce
○ CSV
○ HDF5

Where is your data located?

Hostname
Username
Password

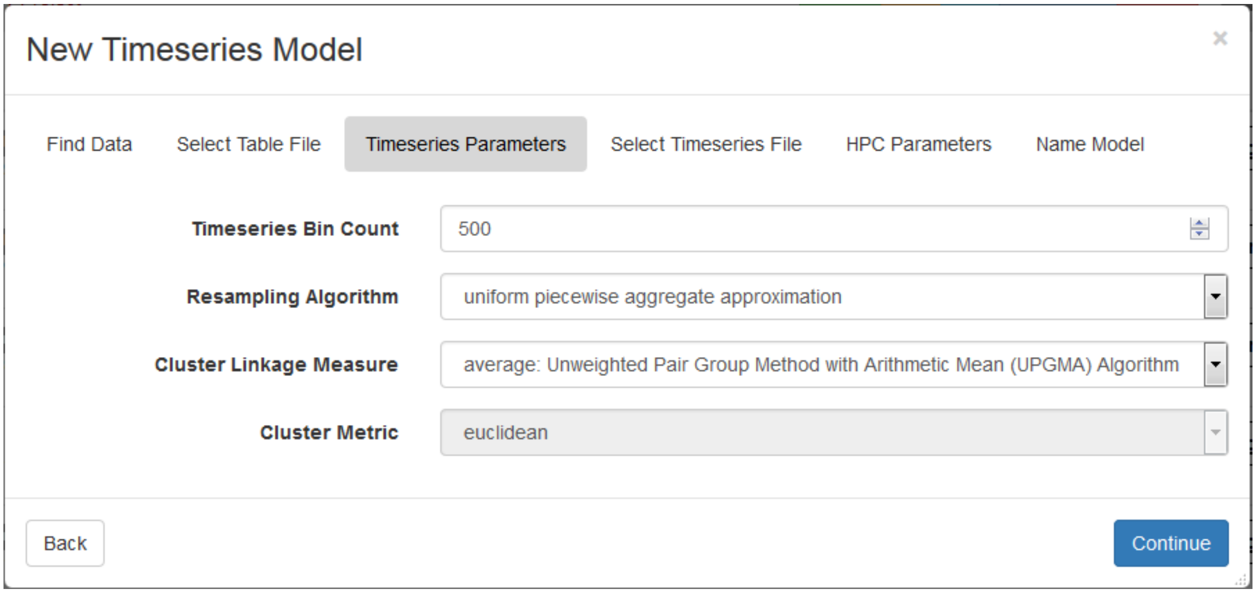
Fig. 61: **Figure 49: Initial dialog to identify data set type and where it is located.**

Time Series Parameters Dialog

The next screen of the wizard will depend on which data format you selected. If you selected *Xyce* or *CSV*, the next screen will be a file browser on the remote host. Navigate to the location of the ensemble data table (a *dakota_tabular* file within the directory hierarchy described above for *Xyce* inputs, or a *CSV* file containing full paths to each time series file for *CSV* inputs). Navigation is identical to that described in the earlier section on Remote Files. Click on the data table file in the remote file browser to select it, then click *Continue*. If you selected *HDF5*, the wizard skips this step since there is no need to select a table file.

For all three input types, the next step is setting the parameters to be used for binning and clustering the time series. *Xyce*, *CSV*, and *HDF5* have slightly different interfaces for this step, which are shown in Figure 50, Figure 51, and Figure 52, respectively.

The *CSV* screen includes two additional fields that are not needed by the other formats, *Table File Delimiter* and *Timeseries Column Name*. *Table File Delimiter* allows you to use other delimiters besides commas in the data table,



New Timeseries Model

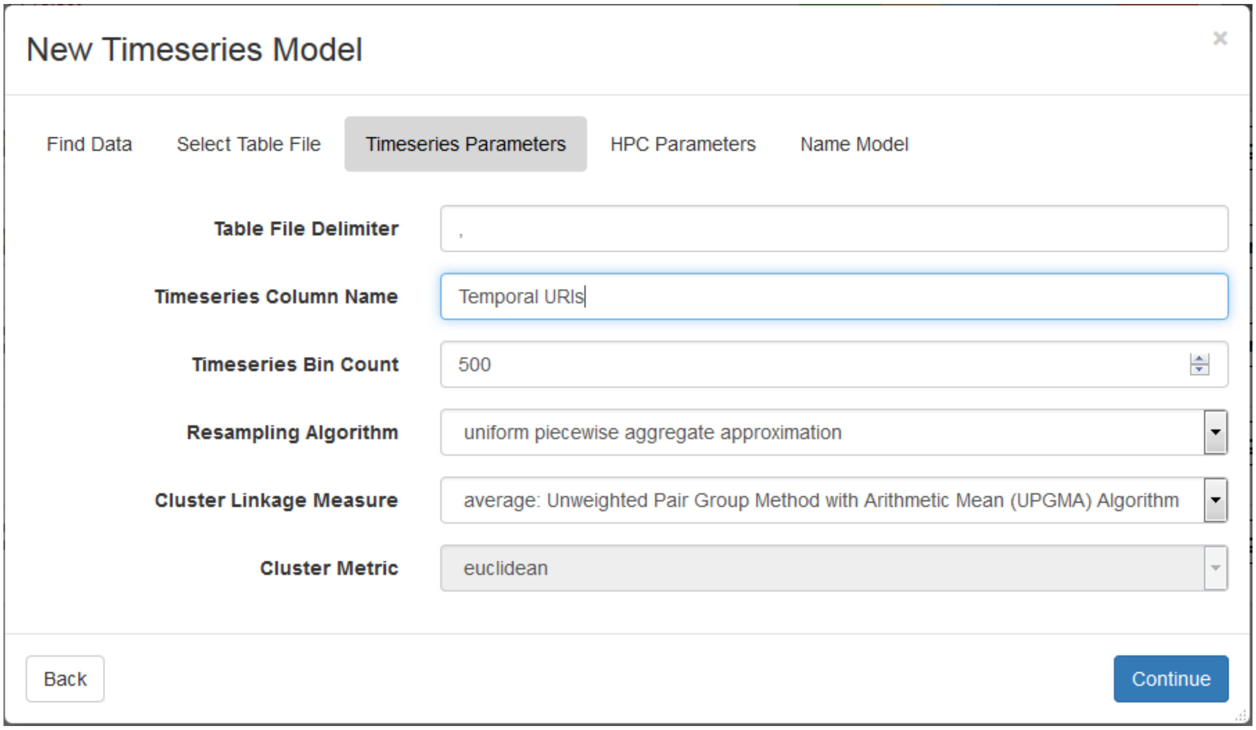
Timeseries Bin Count: 500

Resampling Algorithm: uniform piecewise aggregate approximation

Cluster Linkage Measure: average: Unweighted Pair Group Method with Arithmetic Mean (UPGMA) Algorithm

Cluster Metric: euclidean

Fig. 62: **Figure 50: Timeseries Parameters for Xyce data sets.**



New Timeseries Model

Table File Delimiter: ,

Timeseries Column Name: Temporal URIs

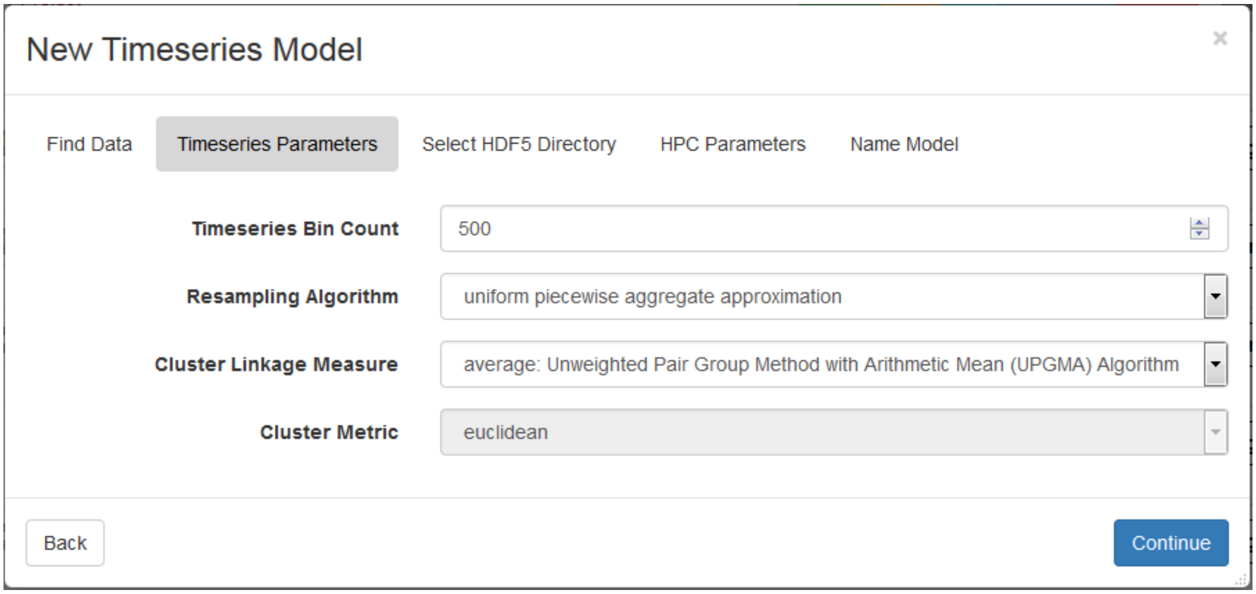
Timeseries Bin Count: 500

Resampling Algorithm: uniform piecewise aggregate approximation

Cluster Linkage Measure: average: Unweighted Pair Group Method with Arithmetic Mean (UPGMA) Algorithm

Cluster Metric: euclidean

Fig. 63: **Figure 51: Timeseries Parameters for CSV data sets.**



New Timeseries Model

Timeseries Bin Count: 500

Resampling Algorithm: uniform piecewise aggregate approximation

Cluster Linkage Measure: average: Unweighted Pair Group Method with Arithmetic Mean (UPGMA) Algorithm

Cluster Metric: euclidean

Fig. 64: **Figure 52: Timeseries Parameters for HDF5 data sets.**

such as *tabs* or *spaces*. *Tabs* are difficult to specify because the web interface uses *tabs* to move between fields, but if you cut-and-paste a *tab* into the field, enclosing it with single quotes, Slycat™ will accept a tab-delimited table. To designate a *space* as a delimiter, enclose it with single quotes, since otherwise the field is interpreted as being empty. *Commas* do not require quotes.

Since *CSV* data tables can have multiple columns of time series data (e.g. if you sampled a set of variables over time at various locations within the simulation), the *Timeseries Column Name* identifies which time series data set to analyze. Type in the column name, taking care to exactly match the header as it appears in the table.

The remaining parameters are shared by all three input types. *Timeseries Bin Count* controls how finely the time series is sampled. The resulting binned sequences are used for calculating similarities and the reduced representations are drawn in the model visualization. Generally, bin counts between 500 and 1000 produce a reasonable tradeoff between speed and accuracy. Although increasing the number of bins increases both the analysis and rendering times, a greater bin count also helps preserve spikes or other localized features that could be lost when using a smaller number.

The *Resampling Algorithm* dropdown has two options. Both algorithms use a uniform set of bins, with the choice between using *uniform piecewise linear approximation* or *uniform piecewise aggregate approximation* as the resampling method. *Uniform piecewise aggregate approximation* is the default.

The *Cluster Linkage Measure* dropdown selects the metric used when evaluating distance between groups of elements. There are four choices:

- *single: Nearest Point Algorithm*
- *complete: Farthest Point Algorithm*
- *average: Unweighted Pair Group Method with Arithmetic Mean (UPGMA) Algorithm*
- *weighted: Weighted Pair Group Method with Arithmetic Mean (WPGMA) Algorithm*

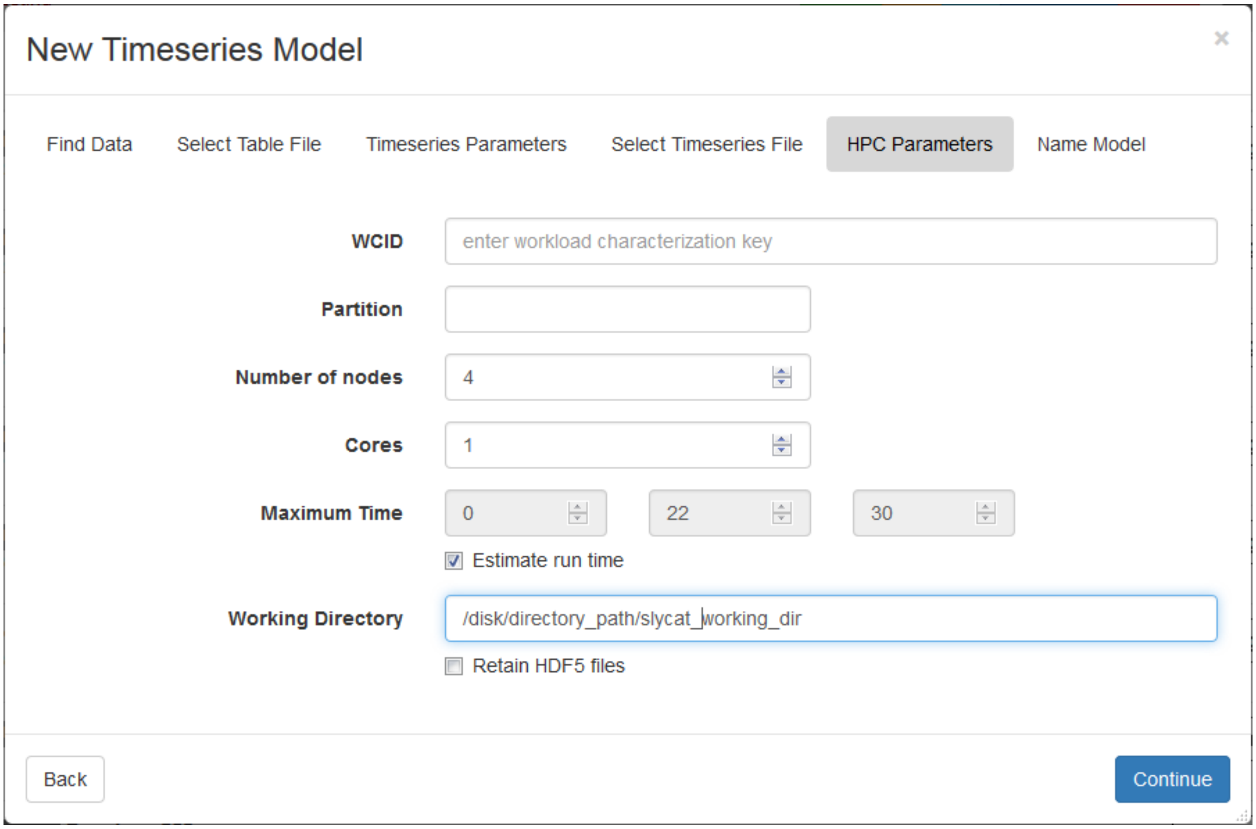
Single evaluates the distance using the closest elements/minimum linkage; *complete* uses the farthest elements/maximum linkage; *average* uses the distance between the group averages; and *weighted* uses the values from the distance matrix. *Average* is the default. We are using SciPy to perform the clustering, so a more complete description of the linkage choices can be found at https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.linkage.html.

The *Cluster Metric* currently only has a single choice, *Euclidean*, so it cannot be changed (hence the field is grayed out). This field is provided to inform you that we are using Euclidean distances in our algorithms.

Once you are satisfied with these parameter choices, click *Continue* to go to the next screen.

High Performance Computing Parameters Dialog

The *HPC Parameters* screen is specific to your institution. Figure 53 shows the parameters for Sandia's cluster systems. Other than differences in the list of steps shown along the top, the same screen is used for all three data formats (*Xyce*, *CSV*, and *HDF5*).



WCID: enter workload characterization key

Partition:

Number of nodes: 4

Cores: 1

Maximum Time: 0 22 30

☑ Estimate run time

Working Directory: /disk/directory_path/slycat_working_dir

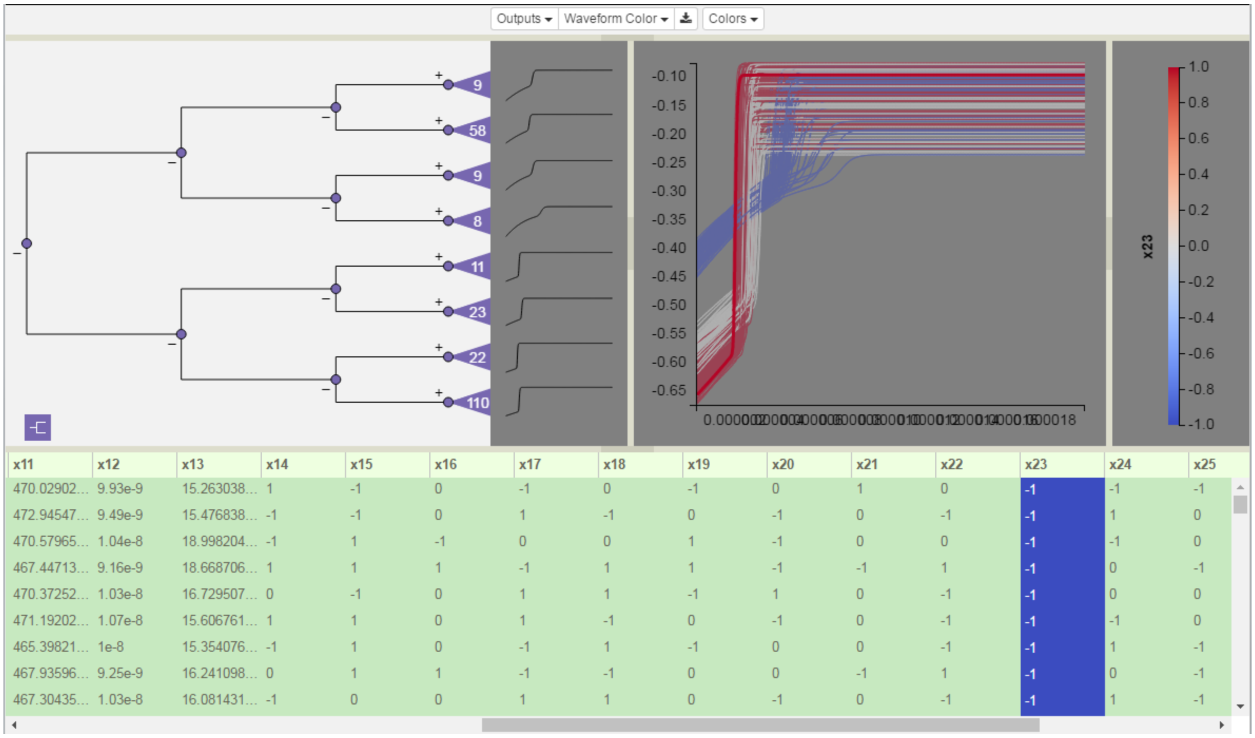
☐ Retain HDF5 files

Fig. 65: **Figure 53: HPC Parameters for Sandia clusters.**

WCID stands for Workload Characterization ID, which is required for job submission on the clusters. Because Slycat™ uses parallel processing to speed up Time Series model creation, users are required to have obtained a *WCID* prior to creating a Time Series model. Your *WCID* is associated with an Strategic Management Unit/Program Management Unit (SMU/PMU), which is used to specify the *Partition* or queue-name. At Sandia, the choices are *nw*, *ec*, *dsa*, *ihns*, *ldrd*, *cee*, *viz*, or *viz batch*).

Time Series Model Visualization

Our examples in this section come from an ensemble of 250 electrical circuit simulations, where our time series outputs are current and voltage variables sampled over time. A Time Series model of this ensemble is shown in Figure 54. The model provides three linked views, each providing a different level of abstraction. The *Dendrogram View* in the upper left provides a high-level view that groups waveforms by similarity. In the upper right, the *Simulation View* displays a line plot for each ensemble member. The plots are superimposed in a shared coordinate space to facilitate comparisons. The lowest level view is the *Variable Table*, which provides the raw data values from the original table file. It is drawn across the bottom of the display and may be scrolled both vertically and horizontally if the number of rows or columns exceeds the available space. Although the *Variable Table* may contain columns with scalar output metrics, or columns of file URIs that would be categorized as being neither input nor output, the ingestion wizard assumes all table columns are inputs and colors them green.



x11	x12	x13	x14	x15	x16	x17	x18	x19	x20	x21	x22	x23	x24	x25
470.02902...	9.93e-9	15.263038...	1	-1	0	-1	0	-1	0	1	0	-1	-1	-1
472.94547...	9.49e-9	15.476838...	-1	-1	0	1	-1	0	-1	0	-1	-1	1	0
470.57965...	1.04e-8	18.998204...	-1	1	-1	0	0	1	-1	0	0	-1	-1	0
467.44713...	9.16e-9	18.668706...	1	1	1	-1	1	1	-1	-1	1	-1	0	-1
470.37252...	1.03e-8	16.729507...	0	-1	0	1	1	-1	1	0	-1	-1	0	0
471.19202...	1.07e-8	15.606761...	1	1	0	1	-1	0	-1	0	-1	-1	-1	0
465.39821...	1e-8	15.354076...	-1	1	0	-1	1	-1	0	0	-1	-1	1	-1
467.93596...	9.25e-9	16.241098...	0	1	1	-1	-1	0	0	-1	1	-1	0	-1
467.30435...	1.03e-8	16.081431...	-1	0	0	1	1	0	-1	0	-1	-1	1	-1

Fig. 66: **Figure 54: Time Series model with Dendrogram View in upper left, Simulation View in upper right, and Variable Table below.**

The *Dendrogram View* controls visibility of ensemble members in both the *Simulation View* and the *Variable Table*. Selections can be made in any one of the views, after which they are propagated to the other two. Color-encoding is shared by all views.

The model specific controls for a Time Series model are shown in Figure 55. Within each time series file, multiple output variables may be present. Each variable is input as a column of values, and each row provides the values for all variables at a single instant in time. Each time series variable generates a distinctly different dendrogram and has a set of unique plots. The *Outputs* dropdown provides a list of the time series variables so you can select which one is currently being displayed in the *Dendrogram View* and the *Simulation View*.

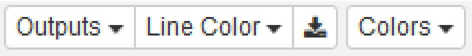


Fig. 67: **Figure 55: Time Series model specific controls. The Outputs dropdown switches the views between different time series variables. Line Color selects from the scalar input variables for color-coding the lines. The Download Table icon enables download of the entire Variable Table or subsets thereof. Colors selects the color theme. The latter two functions are shared by all model types.**

To facilitate discovering relationships between input parameters and groups of output plots, it is often useful to color-code the lines by input variable values. Select a color-coding variable either through the *Line Color* dropdown or by clicking on a table column header (see Color-Coding Lines below).

Dendrogram View

The *Dendrogram View* displays a tree that clusters line plots from a single temporal variable by similarity. The analysis begins by calculating distances between each pair of time series vectors, an O(n^2) calculation. Then the distance matrix is used to build the dendrogram using agglomerative clustering. Each time series output variable generates a different tree.

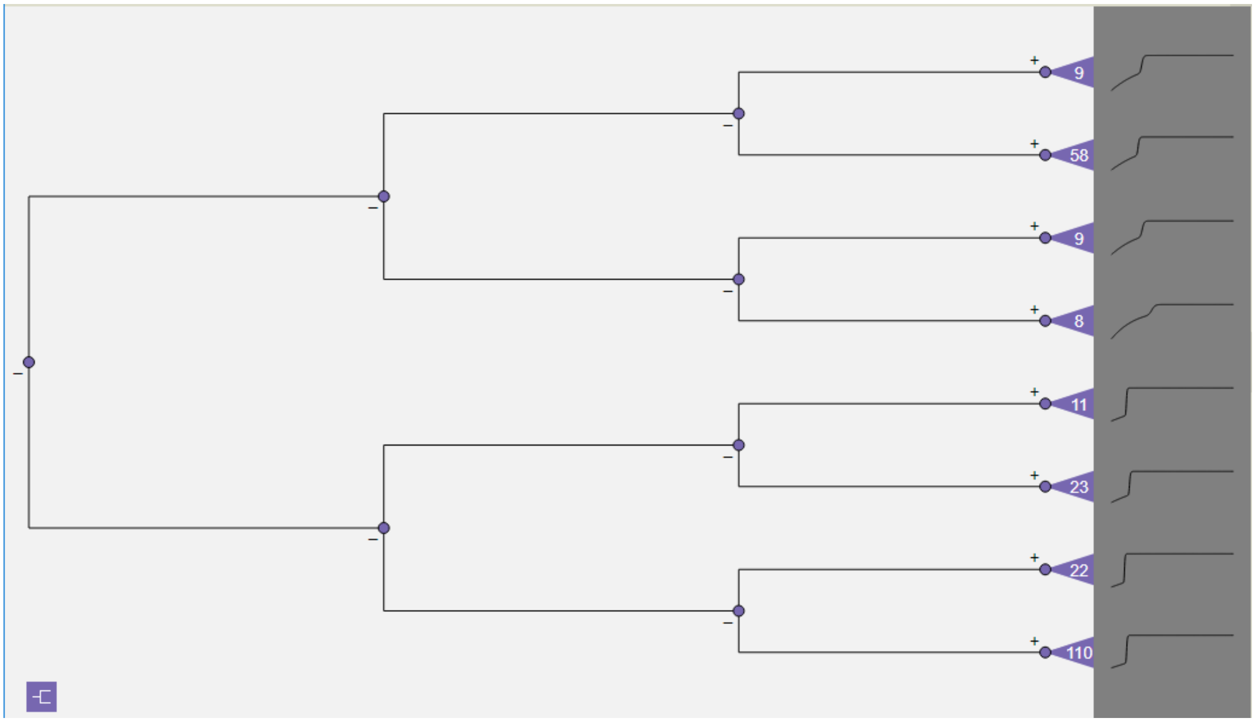


Fig. 68: **Figure 56: Dendrogram tree level compression.**

The dendrogram is drawn with the root on the left and the leaves on the right. To reduce visual clutter, the tree is not drawn at full resolution at every level down to the leaves. Instead, only the first four levels of the tree are initially rendered (as shown in Figure 56), with the last level on the right consisting of collapsed subtrees for the remaining sections of the tree down to the leaves. The subtrees are represented by purple triangular icons, each labeled with the number of nodes in its subtree. Non-collapsed nodes in the tree are drawn as purple dots.

Dendrogram Expansion/Contraction

The expansion or contraction of each node is individually controlled through the '+' or '-' icons that appear to the left of each node, respectively above or below the line connecting the node to its parent. In Figure 57, the subtree with 58 nodes in Figure 56 has been expanded by clicking the '+'. Each expansion adds two levels of the tree to the dendrogram. Note that leaf-level nodes are drawn as dots.

To expand a subtree all the way down to the leaves in a single action, click on the subtree triangle itself. In Figure 58, the subtree with 9 nodes at the top of the dendrogram in Figure 56 has been expanded to the leaf level with this one-click operation. Caution should be exercised when doing a full subtree expansion for subtrees over 20 or so nodes, since the dendrogram can become cluttered and largely unintelligible. Figure 59 demonstrates the results of clicking on the subtree with 110 nodes at the bottom of the dendrogram in Figure 56.

The operation of collapsing nodes reduces all of the nodes below the designated node (the node whose '-' icon is clicked) into a single subtree, regardless of the number of levels that are currently visible. In Figure 60, the bottom

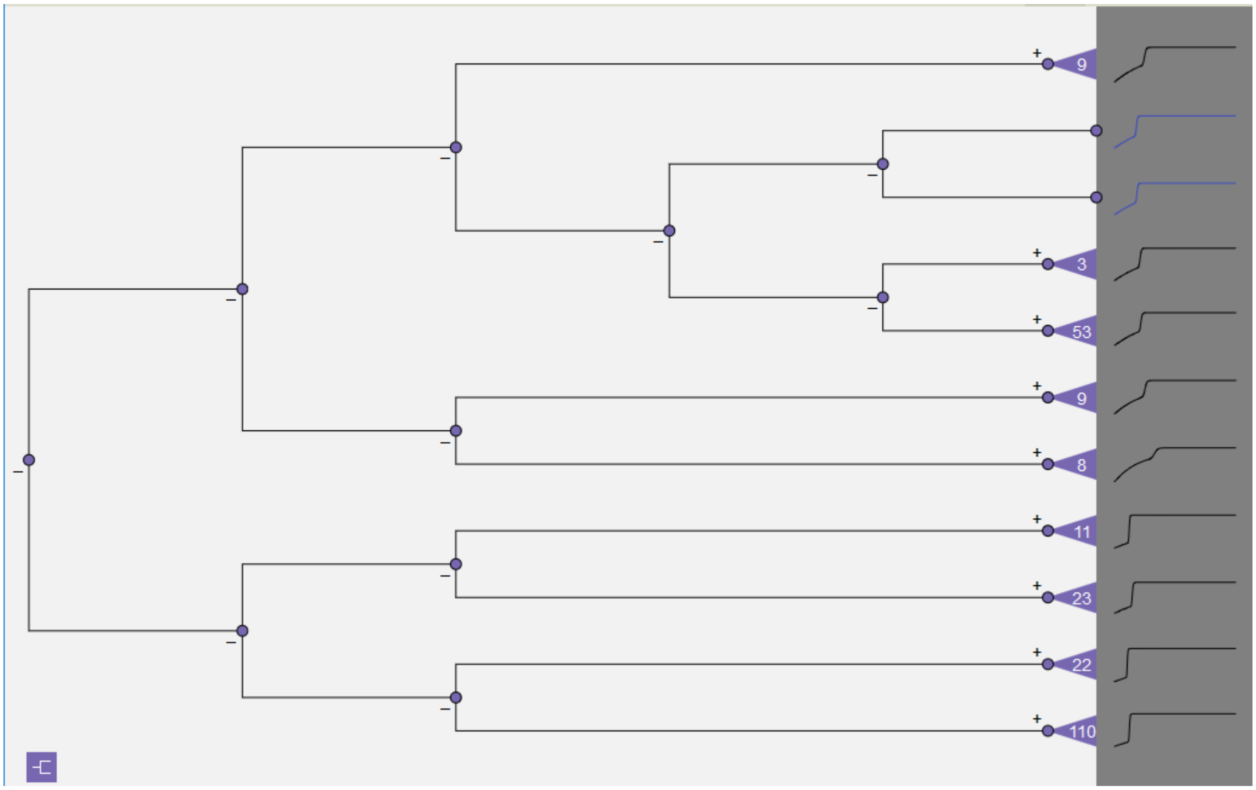


Fig. 69: **Figure 57: Expansion of second subtree from the top (previously shown as the subtree with 58 nodes in Figure 56). Each expansion adds two additional levels.**

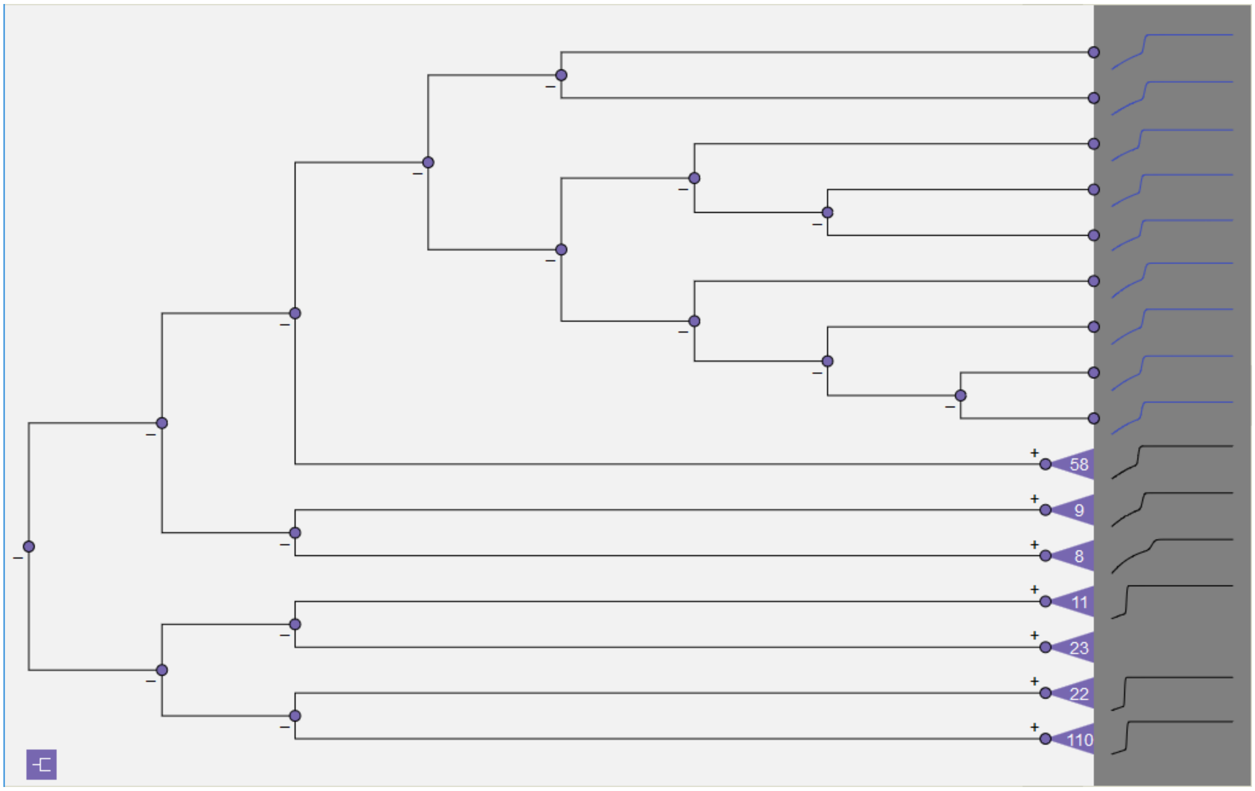


Fig. 70: **Figure 58: Subtree with 9 nodes expanded to leaf level by clicking triangular subtree icon.**

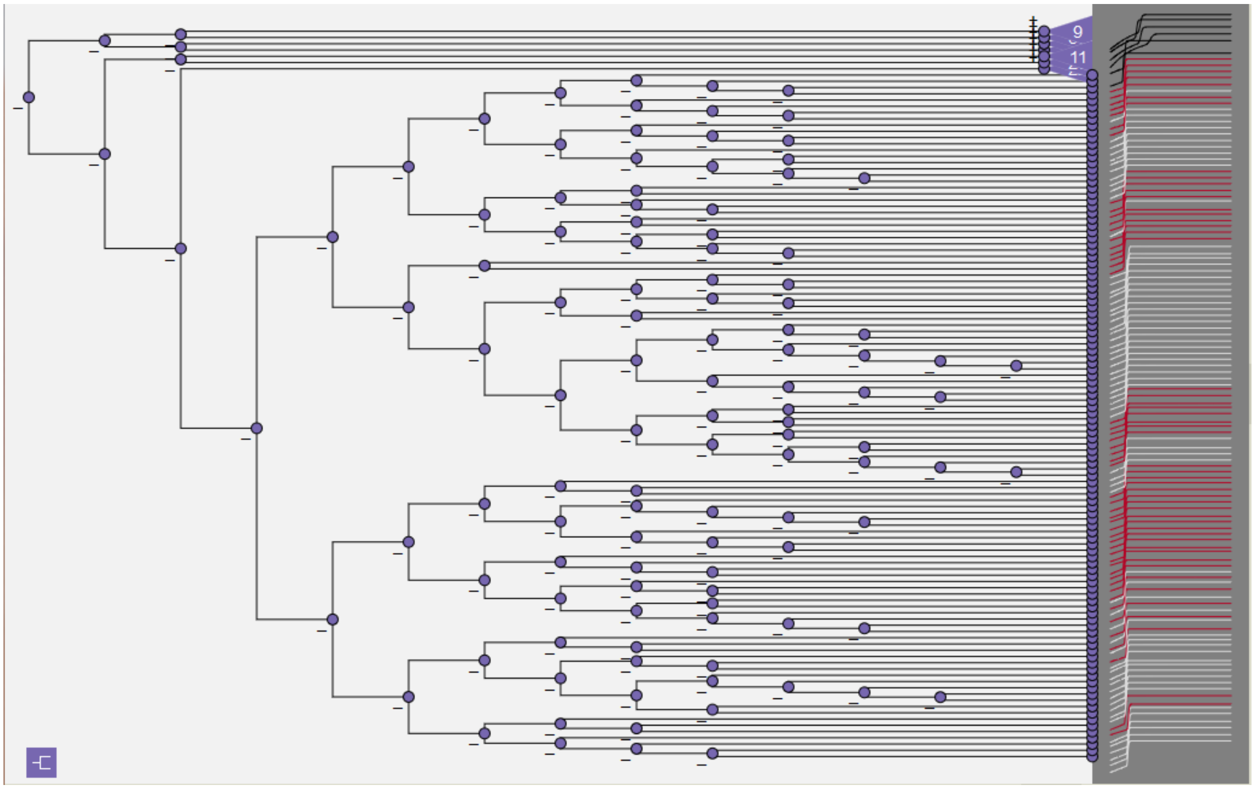


Fig. 71: **Figure 59: One-click expansion of subtree with 110 nodes.**

half of the dendrogram in Figure 57 has been collapsed into a single subtree with 166 nodes. This ability to expand and contract sections of the dendrogram controls the level of detail, maximizing the rendered portion of the tree around areas of interest.

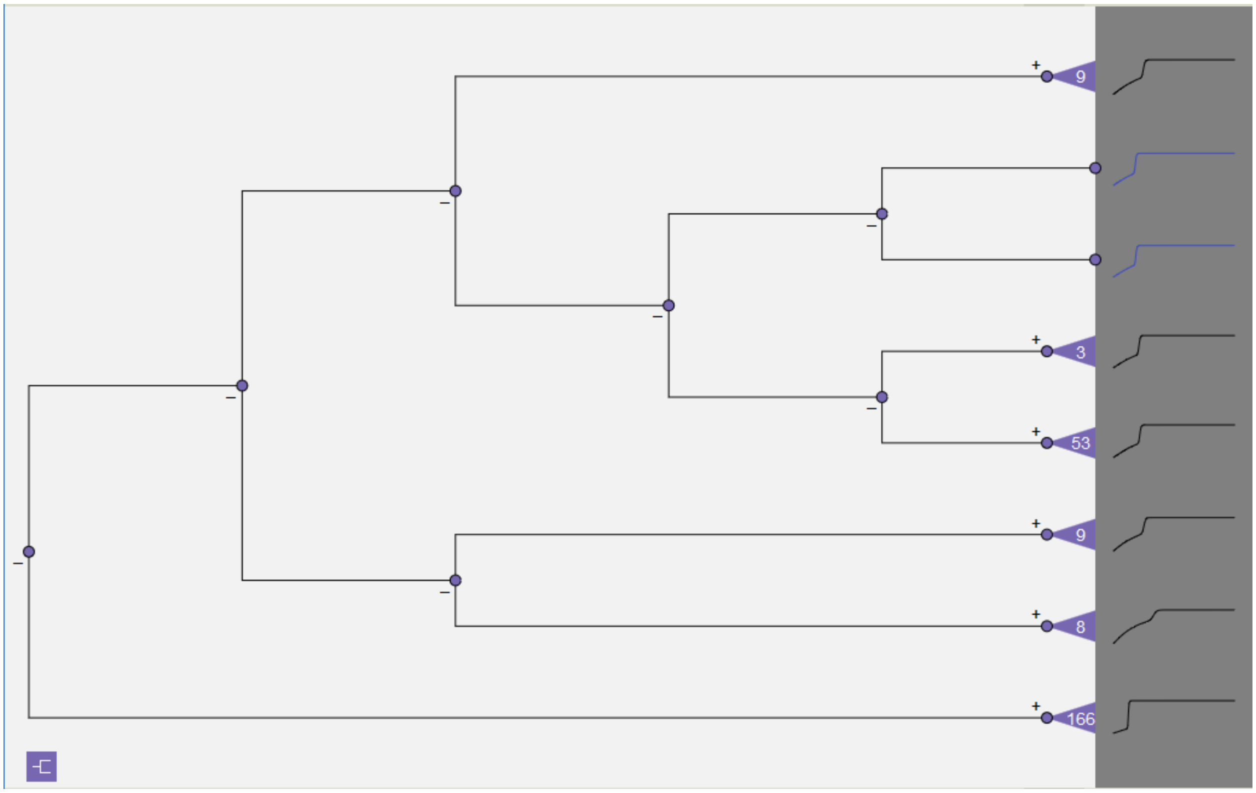
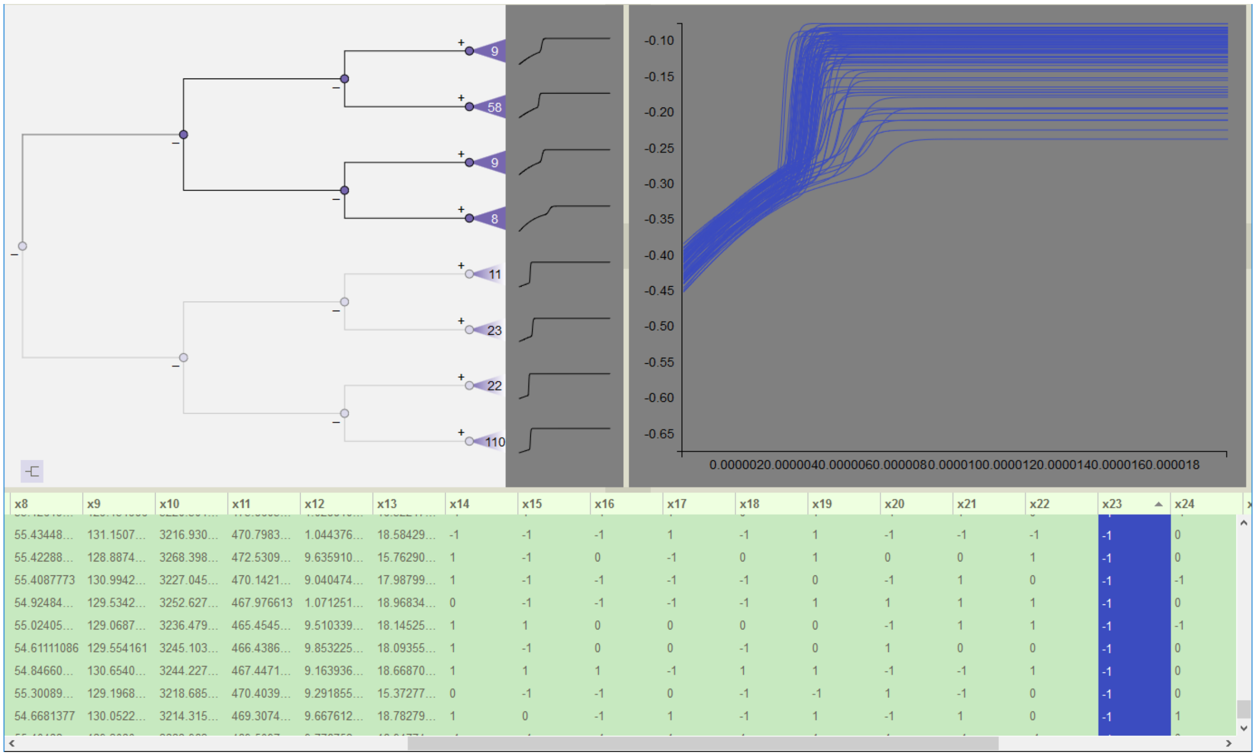


Fig. 72: **Figure 60: Collapsed subtree that combines the 4 subtrees from the lower half of the dendrogram in Figure 57.**

Dendrogram Visibility Filtering

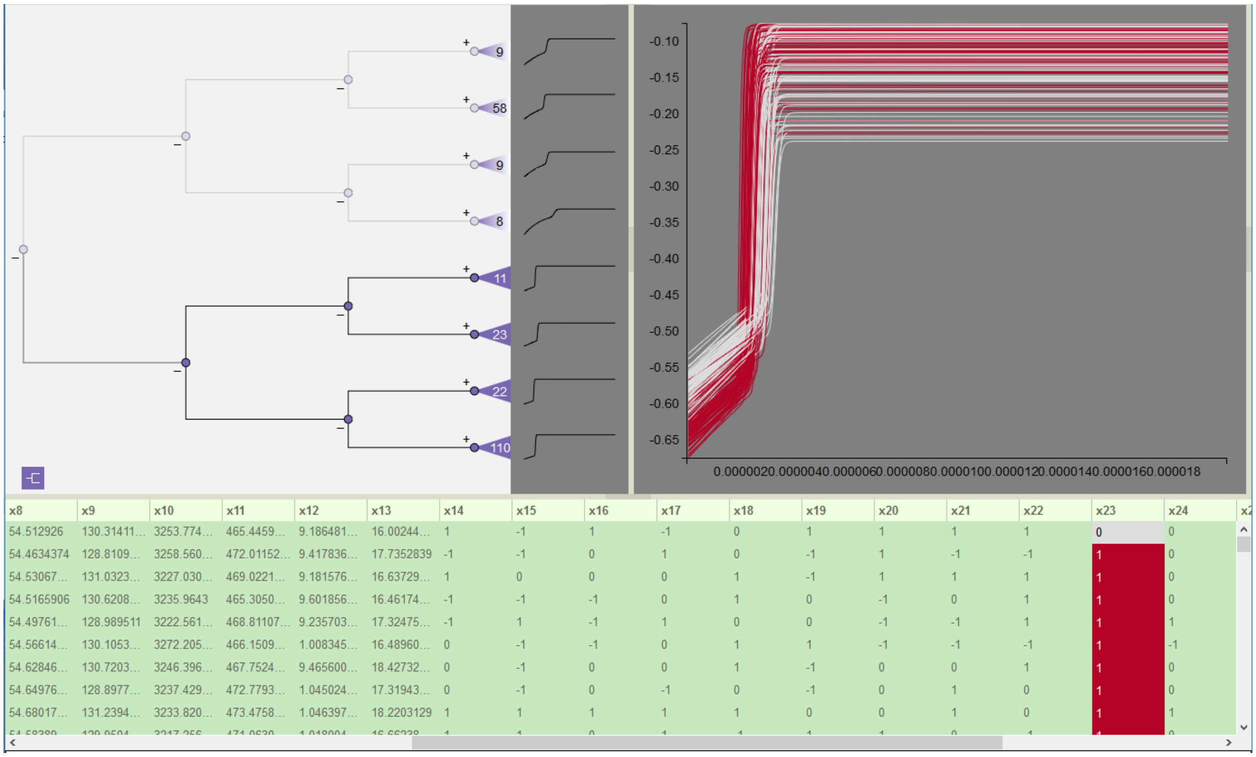
The dendrogram also acts as a visibility filter to select which lines are shown in the Simulation View and which rows appear in the Variable Table. Click on the purple dot representing any node (or the dot at the tip of a subtree triangle) to restrict visibility to the leaf nodes associated with that subset of the tree. Non-visible nodes are grayed out. Figure 61 and Figure 62 show the results of limiting visibility to the subtrees of the upper and lower halves of the dendrogram, respectively. These examples clearly demonstrate that the upper and lower subtrees are grouping the results generated by input differences in the variable *x23* into two categories. Inputs of -1 (color-coded blue) start higher on the y-axis and escalate slowly over a longer time period before peaking, while 0 and 1 (white and red, respectively) start lower on the y-axis and peak more rapidly. These two groups have distinctly different characteristics, which the analysis has captured.

As shown in Figure 63, more complex visibility selections can be constructed through a combination of dendrogram expansion operations and using control-click to add individual nodes to the visible set. Control-click functions as a toggle, so it flips the visibility state for any node with each click. The paths from the root to all visible nodes are darkly drawn, while the remaining paths, though still visible, are grayed out.



x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19	x20	x21	x22	x23	x24
55.43448...	131.1507...	3216.930...	470.7983...	1.044376...	18.58429...	-1	-1	-1	1	-1	1	-1	-1	-1	-1	0
55.42288...	128.8874...	3268.398...	472.5309...	9.635910...	15.76290...	1	-1	0	-1	0	1	0	0	1	-1	0
55.4087773	130.9942...	3227.045...	470.1421...	9.040474...	17.98799...	1	-1	-1	-1	-1	0	-1	1	0	-1	-1
54.92484...	129.5342...	3252.627...	467.976613	1.071251...	18.96834...	0	-1	-1	-1	-1	1	1	1	1	-1	0
55.02405...	129.0687...	3236.479...	465.4545...	9.510339...	18.14525...	1	1	0	0	0	0	-1	1	1	-1	-1
54.61111086	129.554161	3245.103...	466.4386...	9.853225...	18.09355...	1	-1	0	0	-1	0	1	0	0	-1	0
54.84660...	130.6540...	3244.227...	467.4471...	9.163936...	18.66870...	1	1	1	-1	1	1	-1	-1	1	-1	0
55.30089...	129.1968...	3218.685...	470.4039...	9.291855...	15.37277...	0	-1	-1	0	-1	-1	1	-1	0	-1	0
54.6681377	130.0522...	3214.315...	469.3074...	9.667612...	18.78279...	1	0	-1	1	-1	1	-1	1	0	-1	1

Fig. 73: **Figure 61: Visibility is limited to nodes in top half of dendrogram by clicking the upper second-level node. Table order is sorted by the values of variable x23 (not in dendrogram leaf order, shown by lavender graph icon in dendrogram lower left).**



x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19	x20	x21	x22	x23	x24
54.512926	130.31411...	3253.774...	465.4459...	9.186481...	16.00244...	1	-1	1	-1	0	1	1	1	1	0	0
54.4634374	128.8109...	3258.560...	472.01152...	9.417836...	17.7352839	-1	-1	0	1	0	-1	1	-1	-1	1	0
54.53067...	131.0323...	3227.030...	469.0221...	9.181576...	16.63729...	1	0	0	0	1	-1	1	1	1	1	0
54.5165906	130.6208...	3235.9643	465.3050...	9.601856...	16.46174...	-1	-1	-1	0	1	0	-1	0	1	1	0
54.49761...	128.989511	3222.561...	468.81107...	9.235703...	17.32475...	-1	1	-1	1	0	0	-1	-1	1	1	1
54.56614...	130.1053...	3272.205...	466.1509...	1.008345...	16.48960...	0	-1	-1	0	1	1	-1	-1	-1	1	-1
54.62846...	130.7203...	3246.396...	467.7524...	9.465600...	18.42732...	0	-1	0	0	1	-1	0	0	1	1	0
54.64976...	128.8977...	3237.429...	472.7793...	1.045024...	17.31943...	0	-1	0	-1	0	-1	0	1	0	1	0
54.68017...	131.2394...	3233.820...	473.4758...	1.046397...	18.2203129	1	1	1	1	1	0	0	1	0	1	1

Fig. 74: **Figure 62: Visibility is limited to nodes in bottom half of dendrogram by clicking the lower second-level node. Table order is in default dendrogram leaf order (shown by purple graph icon in lower left of dendrogram).**

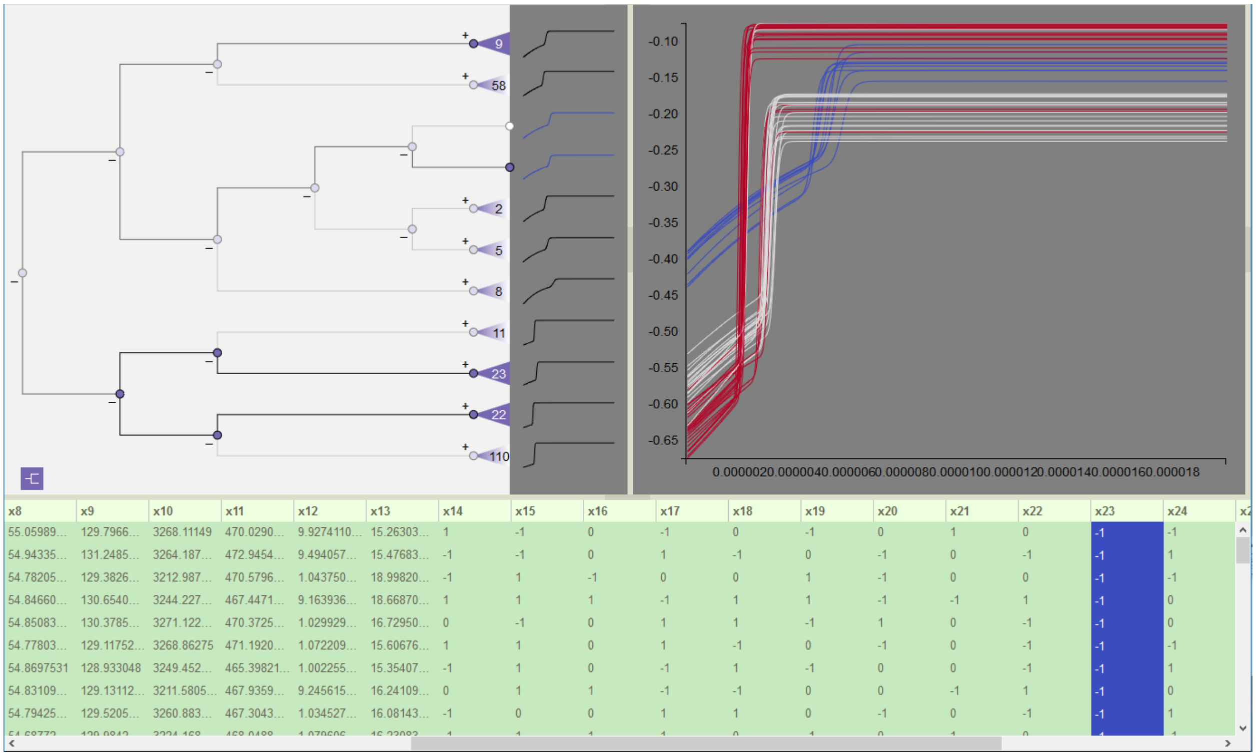
Sparklines

To the right of the subtree icons and leaf nodes are small graphs, called *sparklines*[1], providing a high-level representation of the general shape characteristics of the associated node/subtree. For a node, its *sparkline* is its time series plot rendered into a thumbnail image. For a subtree, its *sparkline* is the *sparkline* of the node closest to the centroid of the group in the subtree.

Sparklines for subtrees are drawn in black. *Sparklines* for leaf nodes are color-coded to match the line color of the corresponding run in the *Simulation View*, and the cell color of the corresponding simulation in the Variable Table. Beyond color-coding being linked between all three views, selection is also linked. Selection of a line (or lines) in the *Simulation View* or *Variable View* will highlight (darken) the *sparklines* of the associated subtrees and/or nodes to reveal their location within the hierarchy, as shown in Figure 64. Alternatively, clicking on a *sparkline* performs a group operation that selects the associated node set, highlighting the corresponding lines in the *Simulation View*, and the corresponding rows in the *Variable Table*. Figure 65 shows how clicking the *sparkline* for the 8 node subtree results in highlighting the eight associated lines and rows in the other two views.

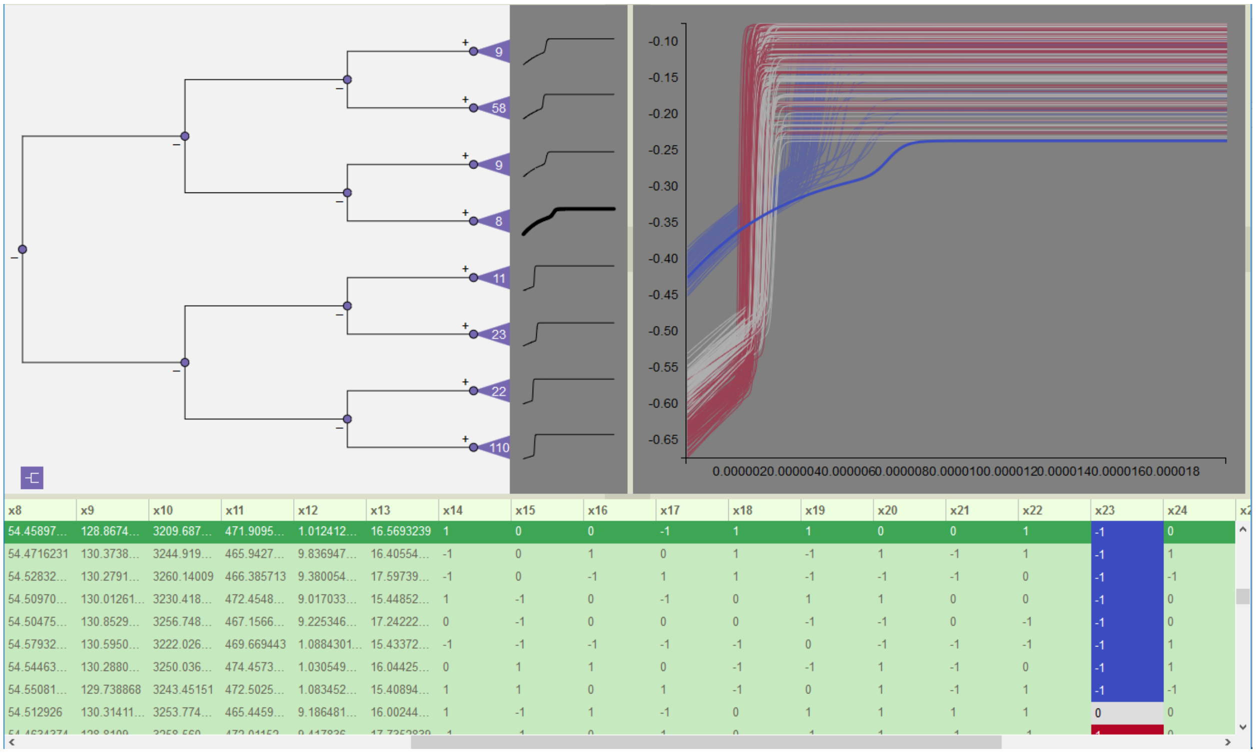
Highlighting and visibility are independent functions that may be combined. For example, in Figure 66, the set of runs that failed to peak are selected in the dendrogram. Highlighting is used to distinguish the runs in the subtree with 8 nodes from the subtree with 9 nodes. The 8 node subtree consists of runs that took longer to rise and had lower overall values than the 9 node group. Note that these differences are visible even in the *sparklines* for each subtree.

[1] Tufte, E., Beautiful Evidence, pp. 46-63, Graphics Press, Cheshire, Connecticut (2006).



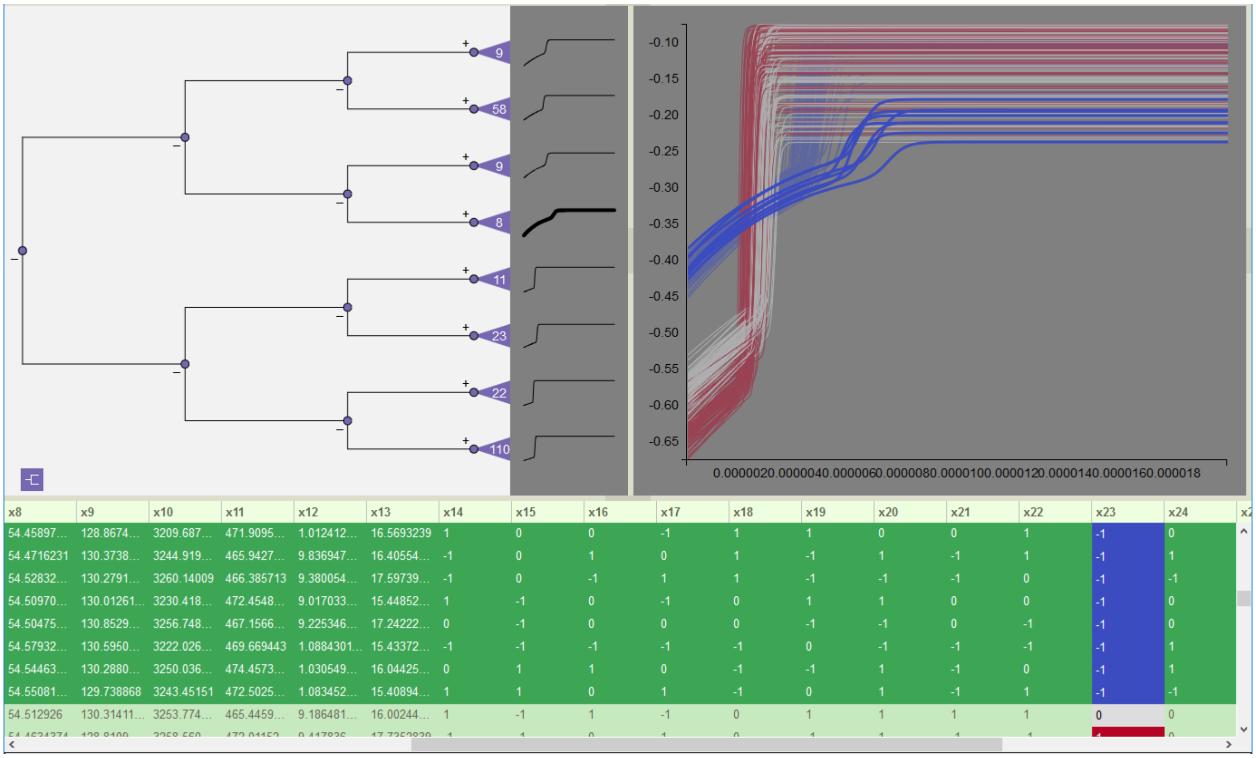
x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19	x20	x21	x22	x23	x24
55.05989...	129.7966...	3268.11149	470.0290...	9.9274110...	15.26303...	1	-1	0	-1	0	-1	0	1	0	-1	-1
54.94335...	131.2485...	3264.187...	472.9454...	9.494057...	15.47683...	-1	-1	0	1	-1	0	-1	0	-1	-1	1
54.78205...	129.3826...	3212.987...	470.5796...	1.043750...	18.99820...	-1	1	-1	0	0	1	-1	0	0	-1	-1
54.84660...	130.6540...	3244.227...	467.4471...	9.163936...	18.66870...	1	1	1	-1	1	1	-1	-1	1	-1	0
54.85083...	130.3785...	3271.122...	470.3725...	1.029929...	16.72950...	0	-1	0	1	1	-1	1	0	-1	-1	0
54.77803...	129.11752...	3268.86275	471.1920...	1.072209...	15.60676...	1	1	0	1	-1	0	-1	0	-1	-1	-1
54.8697531	128.933048	3249.452...	465.39821...	1.002255...	15.35407...	-1	1	0	-1	1	-1	0	0	-1	-1	1
54.83109...	129.13112...	3211.5805...	467.9359...	9.245615...	16.24109...	0	1	1	-1	-1	0	0	-1	1	-1	0
54.79425...	129.5205...	3260.883...	467.3043...	1.034527...	16.08143...	-1	0	0	1	1	0	-1	0	-1	-1	1

Fig. 75: **Figure 63: Complex visibility selection made through a combination of dendrogram expansion and using control-click to add nodes to the visible set. Paths from the root to the visible nodes are darkened, while the other paths are grayed out.**



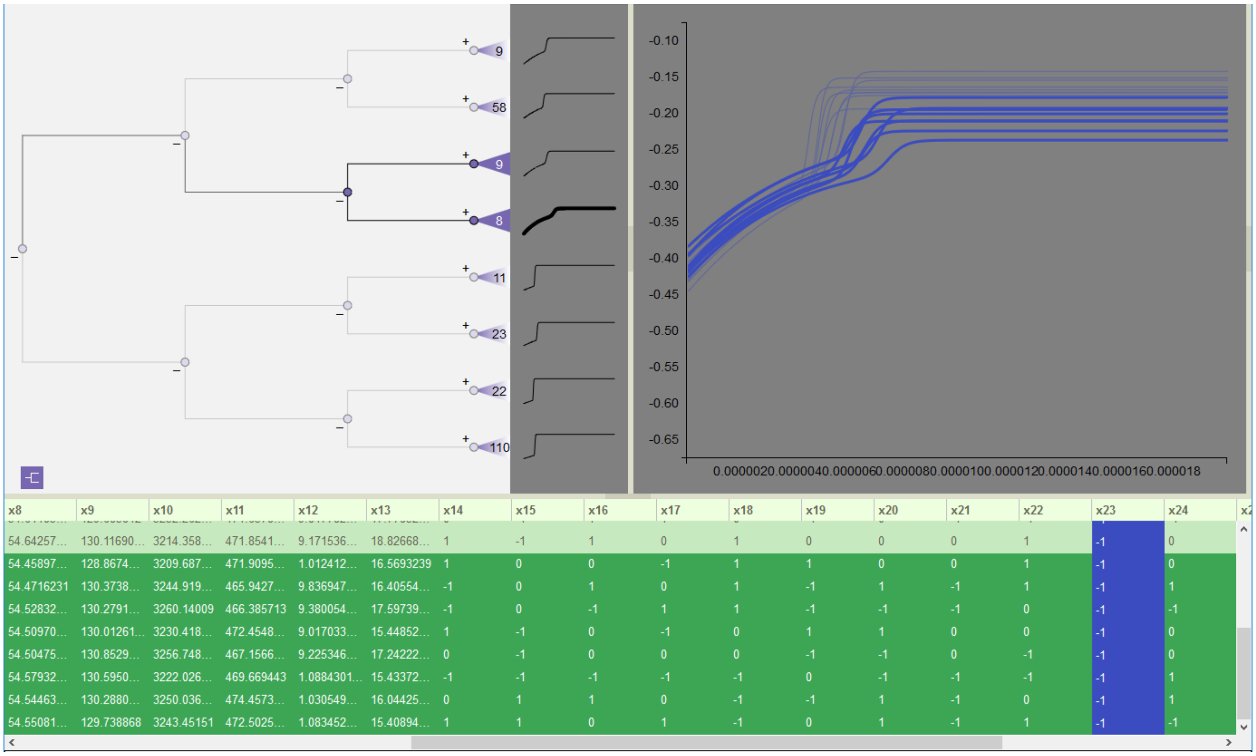
x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19	x20	x21	x22	x23	x24
54.45897...	128.8674...	3209.687...	471.9095...	1.012412...	16.5693239	1	0	0	-1	1	1	0	0	1	-1	0
54.4716231	130.3738...	3244.919...	465.9427...	9.836947...	16.40554...	-1	0	1	0	1	-1	1	-1	1	-1	1
54.52832...	130.2791...	3260.14009	466.385713	9.380054...	17.59739...	-1	0	-1	1	1	-1	-1	-1	0	-1	-1
54.50970...	130.01261...	3230.418...	472.4548...	9.017033...	15.44852...	1	-1	0	-1	0	1	1	0	0	-1	0
54.50475...	130.8529...	3256.748...	467.1566...	9.225346...	17.24222...	0	-1	0	0	0	-1	-1	0	-1	-1	0
54.57932...	130.5950...	3222.026...	469.669443	1.0884301...	15.43372...	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	1
54.54463...	130.2880...	3250.036...	474.4573...	1.030549...	16.04425...	0	1	1	0	-1	-1	1	-1	0	-1	1
54.55081...	129.738868	3243.45151	472.5025...	1.083452...	15.40894...	1	1	0	1	-1	0	1	-1	1	-1	-1
54.512926	130.31411...	3253.774...	465.4459...	9.186481...	16.00244...	1	-1	1	-1	0	1	1	1	1	0	0

Fig. 76: **Figure 64: Sparkline highlights corresponding subtree for selected line in Simulation View,**



x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19	x20	x21	x22	x23	x24
54.45897...	128.8674...	3209.687...	471.9095...	1.012412...	16.5693239	1	0	0	-1	1	1	0	0	1	-1	0
54.4716231	130.3738...	3244.919...	465.9427...	9.836947...	16.40554...	-1	0	1	0	1	-1	1	-1	1	-1	1
54.52832...	130.2791...	3260.14009	466.385713	9.380054...	17.59739...	-1	0	-1	1	1	-1	-1	-1	0	-1	-1
54.50970...	130.01261...	3230.418...	472.4548...	9.017033...	15.44852...	1	-1	0	-1	0	1	1	0	0	-1	0
54.50475...	130.8529...	3256.748...	467.1566...	9.225346...	17.24222...	0	-1	0	0	0	-1	-1	0	-1	-1	0
54.57932...	130.5950...	3222.026...	469.669443	1.0884301...	15.43372...	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	1
54.54463...	130.2880...	3250.036...	474.4573...	1.030549...	16.04425...	0	1	1	0	-1	-1	1	-1	0	-1	1
54.55081...	129.738868	3243.45151	472.5025...	1.083452...	15.40894...	1	1	0	1	-1	0	1	-1	1	-1	-1
54.512926	130.31411...	3253.774...	465.4459...	9.186481...	16.00244...	1	-1	1	-1	0	1	1	1	1	0	0

Fig. 77: **Figure 65: Clicking the sparkline for the 8-node subtree highlights the corresponding set of lines and table rows.**



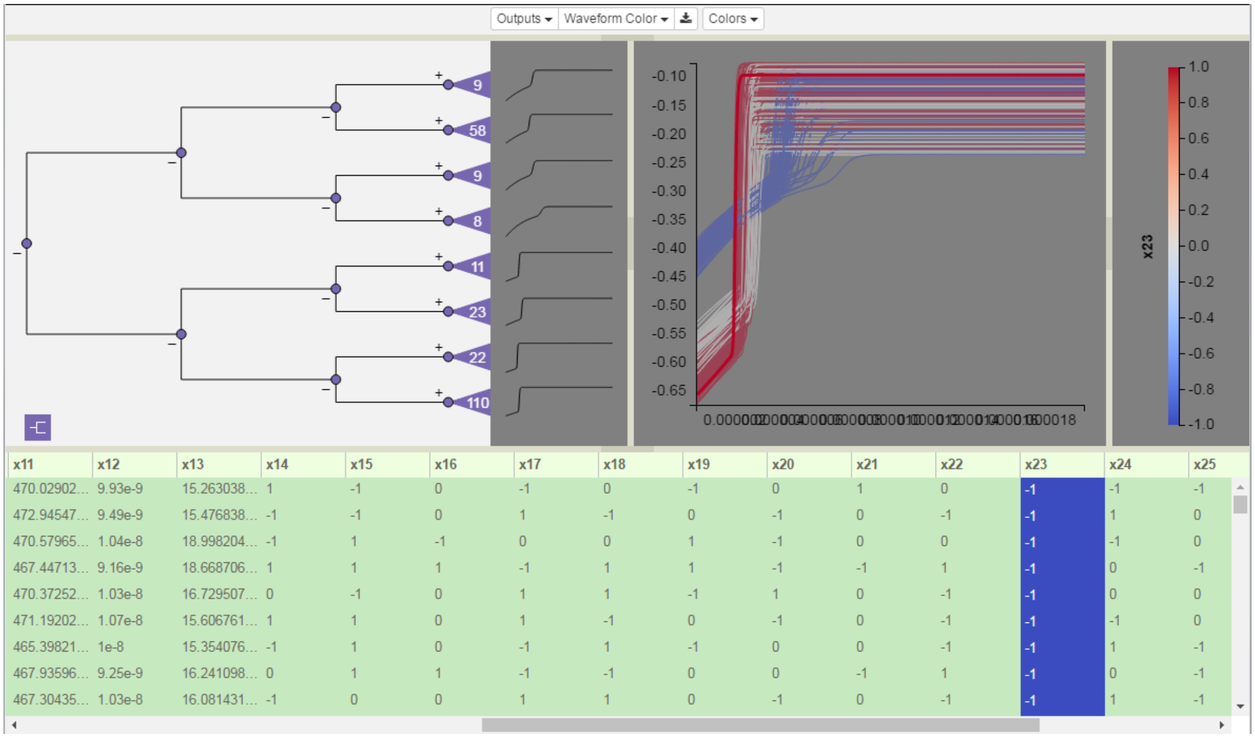
x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19	x20	x21	x22	x23	x24
54.64257...	130.11690...	3214.358...	471.8541...	9.171536...	18.82668...	1	-1	1	0	1	0	0	0	1	-1	0
54.45897...	128.8674...	3209.687...	471.9095...	1.012412...	16.5693239	1	0	0	-1	1	1	0	0	1	-1	0
54.4716231	130.3738...	3244.919...	465.9427...	9.836947...	16.40554...	-1	0	1	0	1	-1	1	-1	1	-1	1
54.52832...	130.2791...	3260.14009	466.385713	9.380054...	17.59739...	-1	0	-1	1	1	-1	-1	-1	0	-1	-1
54.50970...	130.01261...	3230.418...	472.4548...	9.017033...	15.44852...	1	-1	0	-1	0	1	1	0	0	-1	0
54.50475...	130.8529...	3256.748...	467.1566...	9.225346...	17.24222...	0	-1	0	0	0	-1	-1	0	-1	-1	0
54.57932...	130.5950...	3222.026...	469.669443	1.0884301...	15.43372...	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	1
54.54463...	130.2880...	3250.036...	474.4573...	1.030549...	16.04425...	0	1	1	0	-1	-1	1	-1	0	-1	1
54.55081...	129.738868	3243.45151	472.5025...	1.083452...	15.40894...	1	1	0	1	-1	0	1	-1	1	-1	-1

Fig. 78: **Figure 66: Select visibility combined with highlighting to explore differences in the set of runs that did not peak.**

Time Series Simulation View

The *Simulation View* is a line plot, where each line represents an ensemble member. The X axis is the shared range of temporal values between simulations, and the Y axis is the time series variable value. The lines are color-coded by using the value of a selected scalar variable (see Color-Coding Lines) to potentially reveal correlations between inputs and groups of similar output plots, as demonstrated by the examples of the previous section.

Line visibility in the *Simulation View* is controlled by selecting nodes and subtrees within the dendrogram (see Dendrogram View). Moving the mouse over the line plots, Slycat™ interactively provides feedback showing which line is being pointed at through a combination of highlighting the focus line and dimming all the other lines. Left-clicking on the line selects the simulation in all views, highlighting the *sparkline* next to the associated subtree in the dendrogram, highlighting the line in the line plot, and highlighting the associated row in the table. Multiple lines can be selected using control-click to toggle the selection state of lines (i.e. holding the control key while clicking adds unselected lines to the selection set, or removes previously selected lines from the set). Control-click selection is available in all three views, operating on *sparklines*, line plots, or table rows. To clear the current set of selections, click in the background of the *Simulation View* in any area away from the lines.



x11	x12	x13	x14	x15	x16	x17	x18	x19	x20	x21	x22	x23	x24	x25
470.02902...	9.93e-9	15.263038...	1	-1	0	-1	0	-1	0	1	0	-1	-1	-1
472.94547...	9.49e-9	15.476838...	-1	-1	0	1	-1	0	-1	0	-1	-1	1	0
470.57965...	1.04e-8	18.998204...	-1	1	-1	0	0	1	-1	0	0	-1	-1	0
467.44713...	9.16e-9	18.668706...	1	1	1	-1	1	1	-1	-1	1	-1	0	-1
470.37252...	1.03e-8	16.729507...	0	-1	0	1	1	-1	1	0	-1	-1	0	0
471.19202...	1.07e-8	15.606761...	1	1	0	1	-1	0	-1	0	-1	-1	-1	0
465.39821...	1e-8	15.354076...	-1	1	0	-1	1	-1	0	0	-1	-1	1	-1
467.93596...	9.25e-9	16.241098...	0	1	1	-1	-1	0	0	-1	1	-1	0	-1
467.30435...	1.03e-8	16.081431...	-1	0	0	1	1	0	-1	0	-1	-1	1	-1

Fig. 79: **Time Series model with Simulation View in middle upper right, and Legend on the far right.**

Color-Coding Lines

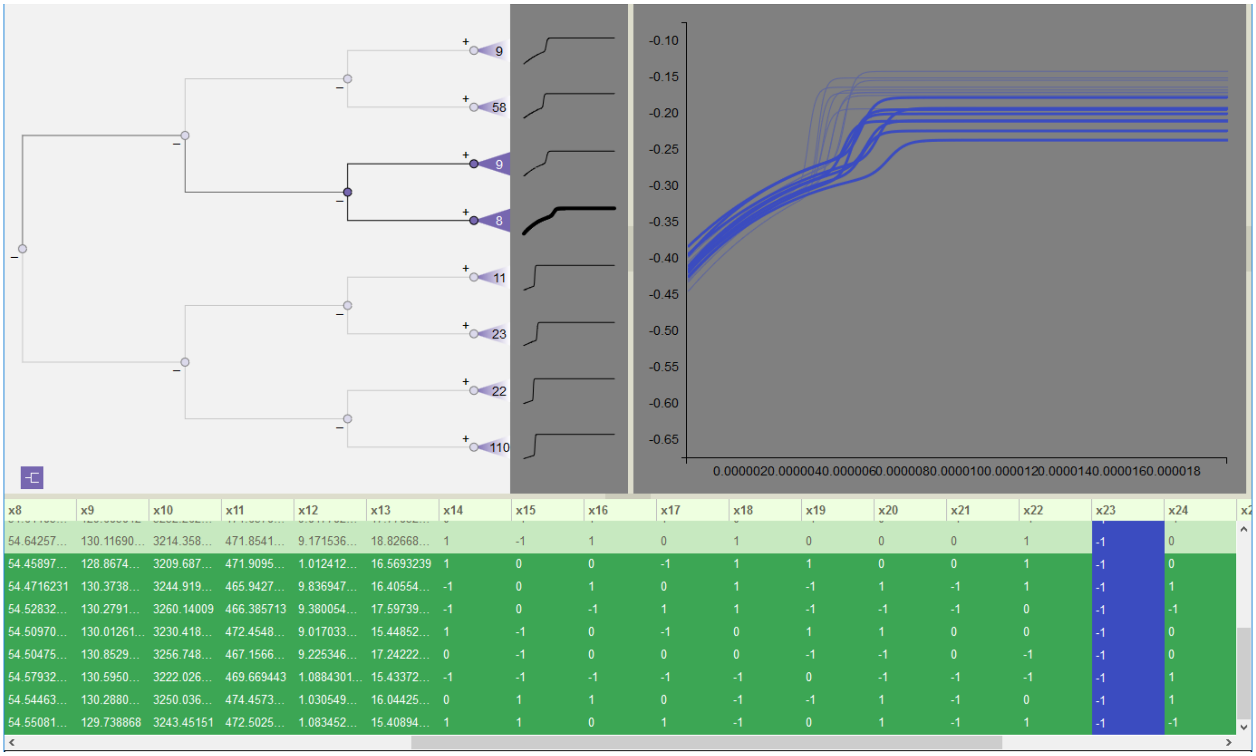
The first time a model is rendered, the lines are colored by their index number. There are two mechanisms for changing the variable selected for the color-coding: clicking on the column header in the *Variable Table* or selecting a variable from the *Line Color* dropdown list. Irrespective of the interface used, changing the color-mapping variable will lead to changes in all three views and the legend, including: recoloring the line plot using the new variable's values, coloring the cell backgrounds in that variable's table column, returning the cell backgrounds of the previously selected variable's column to the default color (green, lavender, or white depending on whether the variable is an input, output, or neither), recoloring any *sparklines* that are leaf nodes in the dendrogram, and relabeling and redefining the value range in the *Legend* (see below).

Legend

To the right of the line plot is the *Legend*. The *Legend* is in its own view, which can be resized or closed altogether. The *Legend* displays information about the current color-coding variable, including its name, range of values, and the mapping between values and colors. The color palette is defined by the current theme (see Color Themes).

Time Series Variable Table

The *Variable Table* is much the same as the table in the CCA model (see Variable Table). However, there is one additional option for row ordering in the table, which we refer to as *dendrogram ordering* (i.e. if the dendrogram were expanded out to the leaf level, the simulations associated with the rows in the table would correspond to those of the dendrogram's leaves). This is the default table order when the model is first visualized. The row ordering choice is explicitly shown by the color of the graph icon in the lower left corner of the *Dendrogram View*. When the graph icon is purple, as in the first figure below, the table is in *dendrogram order*. When the graph icon is lavender, as in the second figure below, the table is in *sorted order*. Clicking on the graph icon restores the table to *dendrogram order*, and returns the icon to purple. Sorting any of the table columns replaces this ordering with the *sorted order*, and changes the icon to lavender.

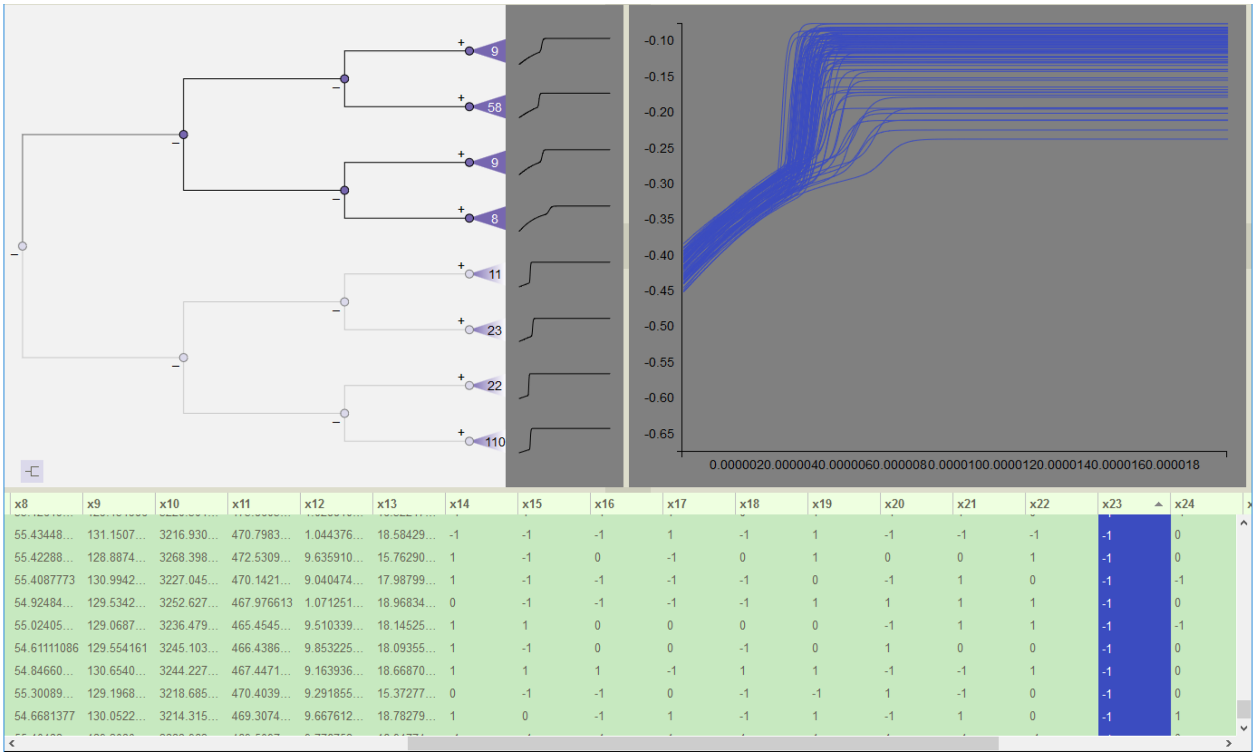


x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19	x20	x21	x22	x23	x24
54.64257...	130.11690...	3214.358...	471.8541...	9.171536...	18.82668...	1	-1	1	0	1	0	0	0	1	-1	0
54.45897...	128.8674...	3209.687...	471.9095...	1.012412...	16.5693239	1	0	0	-1	1	1	0	0	1	-1	0
54.4716231	130.3738...	3244.919...	465.9427...	9.836947...	16.40554...	-1	0	1	0	1	-1	1	-1	1	-1	1
54.52832...	130.2791...	3260.14009	466.385713	9.380054...	17.59739...	-1	0	-1	1	1	-1	-1	-1	0	-1	-1
54.50970...	130.01261...	3230.418...	472.4548...	9.017033...	15.44852...	1	-1	0	-1	0	1	1	0	0	-1	0
54.50475...	130.8529...	3256.748...	467.1566...	9.225346...	17.24222...	0	-1	0	0	0	-1	-1	0	-1	-1	0
54.57932...	130.5950...	3222.026...	469.669443	1.0884301...	15.43372...	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	1
54.54463...	130.2880...	3250.036...	474.4573...	1.030549...	16.04425...	0	1	1	0	-1	-1	1	-1	0	-1	1
54.55081...	129.738868	3243.45151	472.5025...	1.083452...	15.40894...	1	1	0	1	-1	0	1	-1	1	-1	-1

Fig. 80: **Variable Table is in dendrogram order.**

2.1.5 Acknowledgements

Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This user manual was approved for release as SAND2018-0620 R.

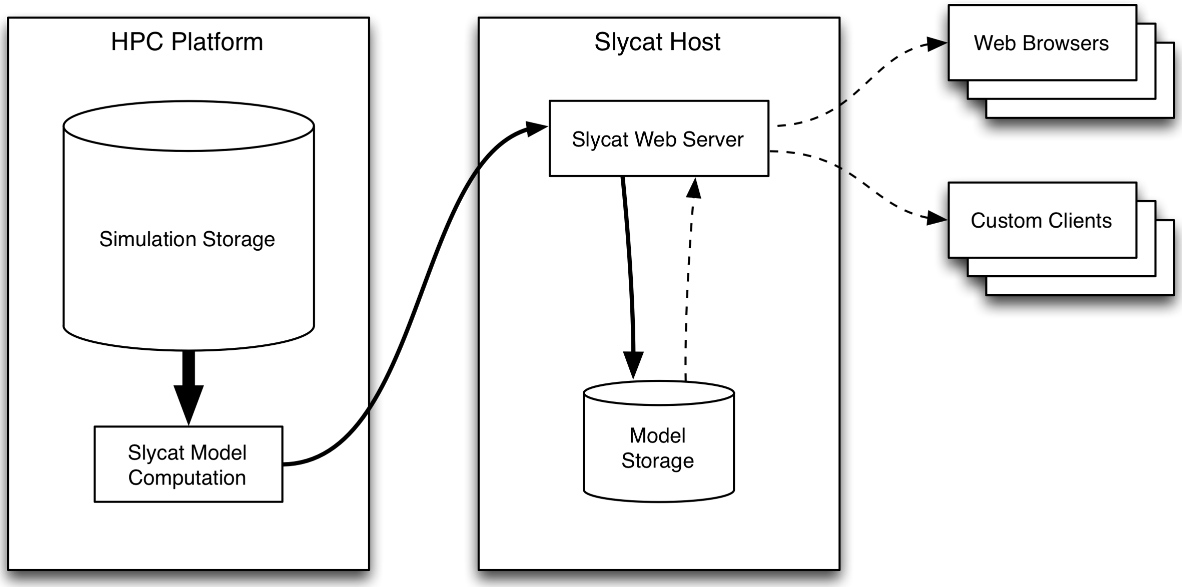


x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19	x20	x21	x22	x23 ▲	x24
55.43448...	131.1507...	3216.930...	470.7983...	1.044376...	18.58429...	-1	-1	-1	1	-1	1	-1	-1	-1	-1	0
55.42288...	128.8874...	3268.398...	472.5309...	9.635910...	15.76290...	1	-1	0	-1	0	1	0	0	1	-1	0
55.4087773	130.9942...	3227.045...	470.1421...	9.040474...	17.98799...	1	-1	-1	-1	-1	0	-1	1	0	-1	-1
54.92484...	129.5342...	3252.627...	467.976613	1.071251...	18.96834...	0	-1	-1	-1	-1	1	1	1	1	-1	0
55.02405...	129.0687...	3236.479...	465.4545...	9.510339...	18.14525...	1	1	0	0	0	0	-1	1	1	-1	-1
54.61111086	129.554161	3245.103...	466.4386...	9.853225...	18.09355...	1	-1	0	0	-1	0	1	0	0	-1	0
54.84660...	130.6540...	3244.227...	467.4471...	9.163936...	18.66870...	1	1	1	-1	1	1	-1	-1	1	-1	0
55.30089...	129.1968...	3218.685...	470.4039...	9.291855...	15.37277...	0	-1	-1	0	-1	-1	1	-1	0	-1	0
54.6681377	130.0522...	3214.315...	469.3074...	9.667612...	18.78279...	1	0	-1	1	-1	1	-1	1	0	-1	1

Fig. 81: **Variable Table is in sorted order.**

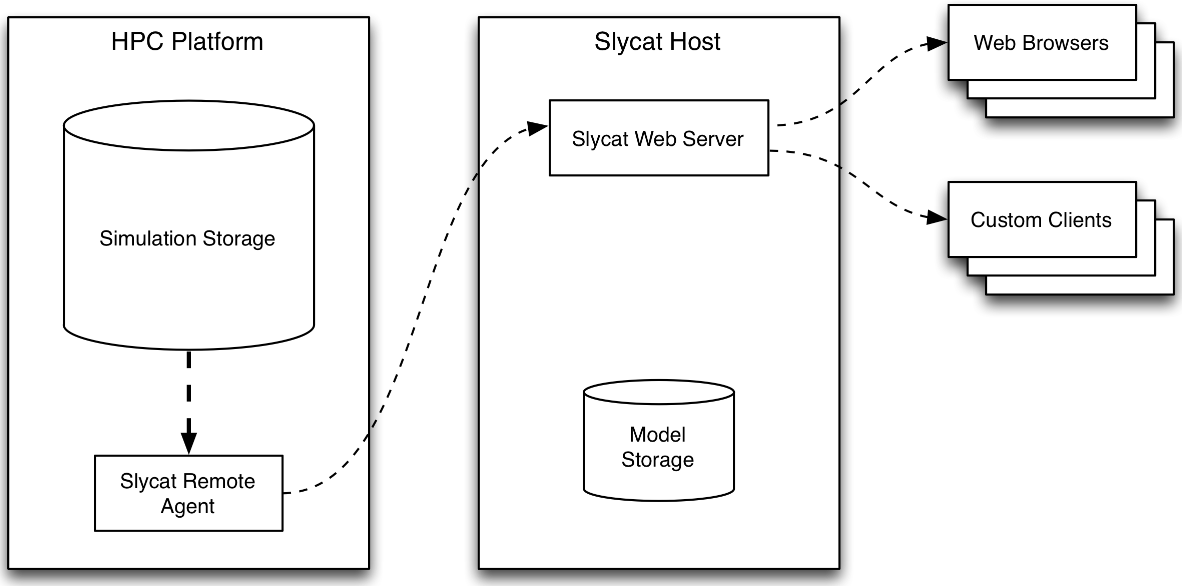
2.2 Design

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.

2.3 Tutorial

2.3.1 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*.

2.3.2 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*.

2.3.3 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*.

2.3.4 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*.

2.3.5 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!

2.4 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.

2.4.1 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

2.4.2 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.

2.4.3 See Also

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

2.5 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.

2.5.1 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!

2.5.2 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.

2.5.3 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.

2.6 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:

2.6.1 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 sandialabs/slycat-dev
```

- 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.

2.6.2 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
```

2.6.3 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,reconnect,defer_permissions,negative_vncache,volname=slycat-
↪container
```

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.

2.7 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/

2.7.1 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 libsasl2-dev libssl-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 Mexico/L=Albuquerque/O=The Slycat Project/OU=QA/CN=Slycat"
```

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/O=The Slycat Project/OU=QA/CN=localhost"
$ openssl x509 -req -in web-server.csr -CA root-ca.cert -CAkey root-ca.key -
↪CAcreateserial -out web-server.cert -days 365
```

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 tornado-couchdb selenium pyvirtualdisplay
```

2.7.2 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 }
```

(continues on next page)

(continued from previous page)

```
  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="packages/slycat/web/server/*" -m behave -i "(agent|hyperchunks|rest-
↪api|slycat-web-server|slycat-project)"
```

2.7.3 Running Coverage

To run the coverage report:

```
$ coverage report
```

2.7.4 Modifying Tests

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

2.8 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.

2.9 Plugins

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

2.9.1 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.

2.9.2 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.

2.9.3 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=datetime.datetime.utcnow().isoformat(), progress=1.0, message=
↪"")

  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: %(default)s")
parser.add_argument("--project-name", default="Hello World Project", help="New␣
↪project name.  Default: %(default)s")
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.

2.9.4 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.

2.9.5 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.com:389", "user_dn":"uid={},dc=example,dc=com"}}
```

2.10 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/

2.10.1 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.

2.10.2 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.

2.10.3 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.

2.11 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:

2.11.1 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.

2.12 REST API

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

2.12.1 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.

2.12.2 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
```

(continues on next page)

(continued from previous page)

```
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*

2.12.3 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)*

2.12.4 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)*

2.12.5 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/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 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)*

2.12.6 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*

2.12.7 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*

2.12.8 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*

2.12.9 GET Home

GET /

Returns a redirect to /projects.

Status Codes

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

See Also

- *GET /projects*

2.12.10 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=little HTTP/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
```

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*

2.12.11 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%3b0/1 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
```

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":
  [
    {
```

(continues on next page)

(continued from previous page)

```
      "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*

2.12.12 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)*

2.12.13 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.

2.12.14 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)*

2.12.15 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)*

2.12.16 GET Model Table Chunk

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

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

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=0,1,2,3,4,5,6,7,8,9&columns=0 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: 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*

2.12.17 GET Model Table Metadata

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

> **Warning:** This request is deprecated. Use *GET /models/(mid)/arraysets/(aid)/metadata* 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", "float32", "float64", "string"],
  "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", "float32", "float64", "string"],
  "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*

2.12.18 GET Model Table Sorted Indices

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

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

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*

2.12.19 GET Model Table Unsorted Indices

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

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

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*

2.12.20 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": "",
```

(continues on next page)

(continued from previous page)

```
  "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)*

2.12.21 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)*

2.12.22 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

2.12.23 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)*

2.12.24 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*

2.12.25 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)*

2.12.26 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&max-width=64
```

See Also

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

2.12.27 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)`*

2.12.28 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)*

2.12.29 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",
}
```

2.12.30 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*

2.12.31 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*

2.12.32 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*

2.12.33 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",
```

(continues on next page)

(continued from previous page)

```
  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*

2.12.34 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
```

2.12.35 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*

2.12.36 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=60000;Path=/"
Location: http://localhost:8092/projects/505d0e463d5ed4a32bb6b0fe9a000d36
Server: CherryPy/3.2.2

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

See Also

- *DELETE /logout*

2.12.37 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)*

2.12.38 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)*

2.12.39 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)*

2.12.40 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)*

2.12.41 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*

2.12.42 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", "2015-03-02T21:03:50"],
  "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

2.12.43 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*

2.12.44 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
```

(continues on next page)

(continued from previous page)

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

See Also

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

2.12.45 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*

2.12.46 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)*

2.12.47 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)*

2.12.48 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*

2.12.49 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
```

(continues on next page)

(continued from previous page)

```
--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)*

2.12.50 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*

2.12.51 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)*

2.12.52 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".

2.12.53 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],
```

(continues on next page)

(continued from previous page)

```
  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)*

2.12.54 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)*

2.12.55 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": [{"user": "bar"}]},
  "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)*

2.12.56 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*

2.13 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*.

2.13.1 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.

2.13.2 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,
```

(continues on next page)

(continued from previous page)

```
  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.

2.13.3 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,
```

(continues on next page)

(continued from previous page)

```
  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.

2.13.4 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.

2.13.5 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.

2.13.6 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.

2.14 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.

2.14.1 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.

2.14.2 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.

2.14.3 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*)

Return statistics describing one 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*

2.14.4 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.

2.14.5 slycat.table

2.14.6 slycat.timeseries

2.14.7 slycat.timeseries.segmentation

2.14.8 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.

2.14.9 slycat.web.client

class `slycat.web.client.`**`ArgumentParser`**(**arguments, **keywords*)

Bases: `argparse.ArgumentParser`

Return an instance of argparse.ArgumentParser, pre-configured with arguments to connect to a Slycat server.

`parse_args`()

class `slycat.web.client.`**`Connection`**(*host='http://localhost:8092'*, ***keywords*)

Bases: `object`

Encapsulates a set of requests to the given host. Additional keyword arguments must be compatible with the Python Requests library, http://docs.python-requests.org/en/latest

`delete_model`(*mid*)

Delete an existing model.

Parameters **`mid`** (*`string, required`*) – The unique model identifier.

See also:

`DELETE /models/(mid)`

`delete_project`(*pid*)

Delete an existing project.

Parameters **`pid`** (*`string, required`*) – The unique project identifier.

See also:

`DELETE /projects/(pid)`

`delete_project_cache_object`(*pid*, *key*)

Delete an existing project cache object.

Parameters

- **`pid`** (*`string, required`*) – The unique project identifier.
- **`key`** (*`string, required`*) – Unique cache object key.

See also:

`DELETE /projects/(pid)/cache/(key)`

`delete_reference`(*rid*)

Delete an existing reference.

Parameters **`rid`** (*`string, required`*) – The unique reference identifier.

See also:

`DELETE /references/(rid)`

`delete_remote`(*sid*)

Delete an existing remote session.

Parameters **`sid`** (*`string, required`*) – The unique remote session identifier.

See also:

`DELETE /remotes/(sid)`

`find_or_create_project`(*name*, *description=''*)

Return a project identified by name, or newly created.

Parameters

- **`name`** (*`string, required`*) – The name of the project to return (or create).
- **`description`** (*`string, optional`*) – Description to use for the new project (if a new project is created).

Returns **pid** – Unique identifier of the matching (or newly created) project.

Return type string

Raises `Exception` – If more than one project matches the given name.

See also:

`post_projects()`

`find_project`(*name*)

Return a project identified by name.

Parameters **`name`** (*`string, required`*) – The name of the project to return.

Returns **project**

Return type The matching project, which is an arbitrary collection of JSON-compatible data.

Raises `Exception` – If a project with a matching name can't be found, or more than one project matches the name.

See also:

`find_or_create_project()`, *`get_projects()`*

`get_bookmark`(*bid*)

Retrieve an existing bookmark.

Parameters **`bid`** (*`string, required`*) – The unique bookmark identifier.

Returns **bookmark** – The bookmark object, which is an arbitrary collection of JSON-compatible data.

Return type object

See also:

`GET /bookmarks/(bid)`

`get_configuration_markings`()

Retrieve marking information from the server.

Returns **markings**

Return type server marking information.

See also:

`GET /configuration/markings`

`get_configuration_parsers`()

Retrieve parser plugin information from the server.

Returns **parsers**

Return type server parser plugin information.

See also:

`GET /configuration/parsers`

`get_configuration_remote_hosts`()

Retrieve remote host information from the server.

Returns **parsers**

Return type server remote host information.

See also:

`GET /configuration/remote-hosts`

get_configuration_support_email()

Retrieve support email information from the server.

Returns **parsers**

Return type server support email information.

See also:

`GET /configuration/support-email`

get_configuration_version()

Retrieve version information from the server.

Returns **version**

Return type server version information.

See also:

`GET /configuration/version`

get_configuration_wizards()

Retrieve wizard plugin information from the server.

Returns **version**

Return type server wizard plugin information.

See also:

`GET /configuration/wizards`

get_global_resource(*resource*)

get_model(*mid*)

Retrieve an existing model.

Parameters **mid** (*string, required*) – The unique model identifier

Returns **model** – The model object, which is an arbitrary collection of JSON-compatible data.

Return type object

See also:

GET /models/(mid)

get_model_arrayset_metadata(*mid*, *aid*, *arrays=None*, *statistics=None*, *unique=None*)

Retrieve metadata describing an existing model arrayset artifact.

Parameters

- **mid** (*string, required*) – The unique model identifier.
- **aid** (*string, required*) – The unique artifact identifier.
- **arrays** (*string, optional*) – A set of arrays, specified using HQL.
- **statistics** (*string, optional*) – A set of attributes, specified using HQL.
- **unique** (*string, optional*) – A set of attributes, specified using HQL.

Returns **metadata** – The arrayset metadata, which is an arbitrary collection of JSON-compatible data.

Return type object

See also:

GET /models/(mid)/arraysets/(aid)/metadata

get_model_file(*mid*, *aid*)

get_model_parameter(*mid*, *aid*)

Retrieve a model parameter artifact.

Model parameters are JSON objects of arbitrary complexity. They are stored directly within the model as part of its database record, so they should be limited in size (larger data should be stored using arraysets or files).

Parameters

- **mid** (*string, required*) – Unique model identifier.
- **aid** (*string, required*) – Unique (within the model) artifact id.

Returns **parameter**

Return type JSON-compatible object

See also:

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

get_model_resource(*mtype*, *resource*)

get_project(*pid*)

Retrieve an existing project.

Parameters **pid** (*string, required*) – Unique project identifier.

Returns **project**

Return type Arbitrary collection of JSON-compatible data.

See also:

GET /projects/(pid)

get_project_cache_object(*pid*, *key*)

Retrieve an object from a project cache.

Parameters

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

Returns **content**

Return type Cached object content.

See also:

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

get_project_models(*pid*)

Returns every model in a project.

get_project_references(*pid*)

Returns every reference in a project.

get_projects()

Retrieve all projects.

Returns **projects**

Return type List of projects. Each project is an arbitrary collection of JSON-compatible data.

See also:

GET /projects

get_remote_file(*sid*, *path*, *cache=None*, *project=None*, *key=None*)

Retrieve a file using a remote session.

Parameters

- **sid** (*string, required*) – Unique remote session identifier.
- **path** (*string, required*) – Remote filesystem path (must be absolute).
- **cache** (*string, optional*) – Optional server-side cache for the retrieved file. Must be *None* or "project".
- **project** (*string, optional*) – If *cache* is set to "project", this must specify a unique project identifier.
- **key** (*string, optional*) – if *cache* is set to "project", this must specify a unique key for the cached object.

Returns file

Return type Remote file contents.

See also:

GET /remotes/(sid)/file(path)

get_remote_image(*sid*, *path*, *cache=None*, *project=None*, *key=None*)

Retrieve an image using a remote session.

Parameters

- **sid** (*string, required*) – Unique remote session identifier.
- **path** (*string, required*) – Remote filesystem path (must be absolute).
- **cache** (*string, optional*) – Optional server-side cache for the retrieved image. Must be *None* or "project".
- **project** (*string, optional*) – If *cache* is set to "project", this must specify a unique project identifier.
- **key** (*string, optional*) – if *cache* is set to "project", this must specify a unique key for the cached object.

Returns image

Return type Remote image contents.

See also:

GET /remotes/(sid)/image(path)

get_user(*uid=None*)

Retrieve directory information about an existing user.

Parameters uid (*string, optional*) – Unique user identifier. If unspecified, returns information about the user making the call.

Returns user

Return type Arbitrary collection of JSON-compatible data.

See also:

GET /users/(uid)

get_wizard_resource(*wtype*, *resource*)

join_model(*mid*)

Wait for a model to complete before returning.

A Slycat model goes through several distinct phases over its lifetime:

1. The model is created.
2. Input artifacts are pushed into the model.
3. The model is marked "finished".
4. Optional one-time computation is performed on the server, storing output artifacts.
5. The model is complete and ready to be viewed.

Use this function in scripts that have performed steps 1, 2, and 3 and need to wait until step 4 completes.

Parameters **mid** (*string, required*) – Unique model identifier.

Notes

A model that hasn't been finished will never complete - you should ensure that post_model_finish() is called successfully before calling join_model().

See also:

post_model_finish()

post_events(*path*, *parameters={}*)

post_model_files(*mid*, *aids*, *files*, *parser*, *input=True*, *parameters={}*)

Stores a model file artifacts.

post_model_finish(*mid*)

Notify the server that a model is fully initialized.

When called, the server will perform one-time computation for the given model type.

Parameters **mid** (*string, required*) – Unique model identifier.

See also:

POST /models/(mid)/finish

post_project_bookmarks(*pid*, *bookmark*)

Store a bookmark.

Parameters

- **pid** (*string, required*) – Unique project identifier.
- **bookmark** (*object*) – Arbitrary collection of JSON-compatible data.

Returns **bid** – Unique bookmark identifier.

Return type string

See also:

POST /projects/(pid)/bookmarks

post_project_models (*pid*, *mtype*, *name*, *marking=”*, *description=”*)

Creates a new model, returning the model ID.

post_project_references (*pid*, *name*, *mtype=None*, *mid=None*, *bid=None*)

Store a project reference.

Parameters

- **pid** (*string, required*) – Unique project identifier.
- **name** (*string, required*) – Reference name.
- **mtype** (*string, optional*) – Optional model type.
- **mid** (*string, optional*) – Optional model identifier.
- **bid** (*string, optional*) – Optional bookmark identifier.

Returns **rid** – Unique reference identifier.

Return type string

See also:

`POST /projects/(pid)/references`

post_projects (*name*, *description=”*)

Creates a new project, returning the project ID.

post_remote_browse (*sid*, *path*, *file_reject=None*, *file_allow=None*, *directory_allow=None*, *directory_reject=None*)

post_remotes (*hostname*, *username*, *password*, *agent=None*)

put_model (*mid*, *model*)

put_model_arrayset (*mid*, *aid*, *input=True*)

Starts a new model array set artifact, ready to receive data.

put_model_arrayset_array (*mid*, *aid*, *array*, *dimensions*, *attributes*)

Starts a new array set array, ready to receive data.

put_model_arrayset_data (*mid*, *aid*, *hyperchunks*, *data*, *force_json=False*)

Write data to an arrayset artifact on the server.

Parameters

- **mid** (*string, required*) – Unique model identifier.
- **aid** (*string, required*) – Unique (to the model) arrayset artifact id.
- **hyperchunks** (*string, required*) – Specifies where the data will be stored, in *Hyperchunks* format.
- **data** (*iterable, required)*) – A collection of numpy.ndarray data chunks to be uploaded. The number of data chunks must match the number implied by the *hyperchunks* parameter.
- **force_json** (*bool, optional)*) – Force the client to upload data using JSON instead of the binary format.

See also:

`PUT /models/(mid)/arraysets/(aid)/data`

put_model_inputs (*source*, *target*)

put_model_parameter(*mid*, *aid*, *value*, *input=True*)

Store a model parameter artifact.

Model parameters are JSON objects of arbitrary complexity. They are stored directly within the model as part of its database record, so they should be limited in size (larger data should be stored using arraysets or files).

To get the value of a parameter artifact, use `get_model()` and read the value directly from the model record. An artifact id *foo* will be accessible in the record as *model["artifact:foo"]*.

Parameters

- **mid** (*string, required*) – Unique model identifier.
- **aid** (*string, required*) – Unique (within the model) artifact id.
- **value** (*object, required*) – An arbitrary collection of JSON-compatible data.
- **input** (*boolean, optional*) – Marks whether this artifact is a model input.

See also:

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

put_project(*pid*, *project*)

Modifies a project.

request(*method*, *path*, ***keywords*)

Makes a request with the given HTTP method and path, returning the body of the response. Additional keyword arguments must be compatible with the Python Requests library, http://docs.python-requests.org/en/latest

update_model(*mid*, ***kwargs*)

Update model state.

This function provides a more convenient alternative to `put_model()`.

See also:

`put_model()`

slycat.web.client.**connect**(*arguments*, ***keywords*)

Factory function for client connections that takes an option parser as input.

2.14.10 slycat.web.server

slycat.web.server.**check_https_get_remote_ip**()

checks that the connection is https and then returns the users remote ip :return: remote ip

slycat.web.server.**check_rules**(*groups*)

slycat.web.server.**check_user**(*session_user*, *apache_user*, *couchdb*, *sid*, *session*)

check to see if the session user is equal to the apache user raise 403 and delete the session if they are not equal :param session_user: user_name in the couchdb use session :param apache_user: user sent in the apache header "authuser" :param couchdb: hook to couch :param sid: session id :param session: session object from couch :return:

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.`**`clean_up_old_session`**`()`

try and delete any outdated sessions for the user if they have the cookie for it :return:no-op

`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.`**`create_single_sign_on_session`**`(`*remote_ip*, *auth_user*`)`

WSGI/RevProxy no-login session creations. Successful authentication and access verification, create a session and return. :return: not used

`slycat.web.server.`**`decode_username_and_password`**`()`

decode the url from the json that was passed to us :return: decoded url and password as a tuple

`slycat.web.server.`**`delete_model_parameter`**`(`*database*, *model*, *aid*`)`

Delete a model parameter in the couch database :param database: :param model: model from the couchdb :param aid: artifact id :return: not used

`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_lock`**`(`*model_id*`)`

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

`slycat.web.server.`**`get_password_function`**`()`

`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.`**`get_remote_file_server`**`(`*client*, *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*, *client=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.`**`response_url`**`()`

get the resonse_url and clean it to make sure that we are not being spoofed :return: url to route to once signed in

`slycat.web.server.`**`ssh_connect`**`(`*hostname=None*, *username=None*, *password=None*`)` 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"

2.14.11 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.

2.14.12 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*

2.14.13 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*)

Determine if the specified record is to be logged.

Is the specified record to be logged? Returns 0 for no, nonzero for yes. If deemed appropriate, the record may be modified in-place.

`slycat.web.server.engine.`**start**(*root_path*, *config_file*)

2.14.14 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_model_parameter**(*mid*, *aid*)

delete a model artifact :param mid: model Id :param aid: artifact id :return:

`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*)

cleans up an upload session throws 409 if the session is busy :param uid: :return: not used

`slycat.web.server.handlers.`**get_bookmark**(*bid*)

`slycat.web.server.handlers.`**get_checkjob**(*hostname*, *jid*)

`slycat.web.server.handlers.`**get_configuration_ga_tracking_id**()

`slycat.web.server.handlers.`**get_configuration_injected_code**()

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*)
returns either and array of projects or html for displaying the projects :param _: :return:

slycat.web.server.handlers.**get_projects_list**(*_=None*)
returns either and array of projects or html for displaying the projects :param _: :return:

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_job_status**(*hostname*, *jid*)

slycat.web.server.handlers.**get_remote_show_user_password**()
checks to see if the application needs to show password :return: json {show:bool, msg:msg}

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_root**()
Redirect all requests to "/" to "/projects" Not sure why we used to do that, but after conversion to webpack this is no longer needed, so I changed the projects-redirect config parameter in web-server-config.ini to just "/"

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.`**`open_id_authenticate`**`(`***params*`)`

takes the openid parameter sent to this function and logs in a user :param params: openid params as a dictionary :return: not used

`slycat.web.server.handlers.`**`post_events`**`(`*event*`)`

`slycat.web.server.handlers.`**`post_log`**`()`

send post json {"message":"message"} to log client errors onto the client server :return:

`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_command`**`(`*hostname*`)`

run a remote command from the list of pre-registered commands that are located on a remote agent. :param hostname: name of the hpc host :return: {

> "message": a message that is supplied by the agent, "command": an echo of the command that was sent to the server and the agent, "error": boolean describing if there was an agent error, "available_scripts": [{
>
> > "name": script_name, "description": script_description, "parameters": [{

“name”: parameter_name as string, “description”: description of the param string, “example”:example usage string, “type”: field type eg string, int…

}]

}]list of available scripts from the agent

}

`slycat.web.server.handlers.`**`post_remote_launch`**`(`*hostname*`)`

`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*, *byteorder=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*, *hostname=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*`)`

2.14.15 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.

2.14.16 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.

2.14.17 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_remote_job_status(*jid*)
check of the status of a job running on an agent with a hostanemd session :param jid: job id :return:

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

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

run_remote_command(*command*)

run a remote command from an HPC source running a slycat agent. the command could be things such as starting an hpc script or batch job or something as simple as moving files. the only requirement is that the script is in our list of trusted scripts.

this_func()->calls agent_command_func()->which runs_shell_command() -> which launches_script()-> sends_response_to_agent()->sends_response_to_server() ->sends_status_response_to_client()

Parameters

- **self** –
- **command** – json form of a command to be run

{ “scripts”: //pre defined scripts that are registerd with the server [{

> “script_name”:”script_name”, // key for the script lookup “parameters”: [{key:value},. . .] // params that are fed to the script

},. . .] “hpc”: // these are the hpc commands that may be add for thing such as slurm {

> “is_hpc_job”:bol, // determins if this should be run as an hpc job “parameters”:[{key:value},. . .] // things such as number of nodes

}

} :return: {“msg”:”message from the agent”, “error”: boolean}

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.remote.**get_session_server**(*client*, *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. :param client:

Return type *slycat.web.server.remote.Session*

2.14.18 slycat.web.server.template

`slycat.web.server.template.`**`render`**(*path*, *context*)

Render an HTML template using Mustache syntax.

2.15 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

HTTP Routing Table

Python Module Index

Index