
panoptes Documentation

Release 1.0-SNAPSHOT

Paul Vauterin, Ben Jeffery, Alistair Miles

July 31, 2015

1	Contents	3
1.1	Introduction	3
1.2	Installation and deployment guide	4
1.3	Loading data	7
1.4	Source files structure	29
1.5	Data import settings	32
2	License	45



Panoptes is a web application for exploration and visualisation of data. Many of the features, such as querying and browsing tables, are generic and can be used with any kind of data. There are also some specialised features for genomic and geospatial data.

Caution: Panoptes is still in an early stage of development. It should currently be considered as a prototype, unfit for production deployments.

We welcome early adopters, however please bear in mind that the design and configuration of the web application are still in flux and may change substantially. As our current focus is on developing the code, we can only offer limited support. For more information about the project please send an email to the public mailing list <panoptes-dev@googlegroups.com>.

- Source code: <https://github.com/cggh/panoptes>
- Documentation: <http://panoptes.readthedocs.org>
- Mailing list: <https://groups.google.com/forum/#!forum/panoptes-dev>

1.1 Introduction

Panoptes is a web application for exploration and visualisation of data. It was created by the [CGGH](#) software development team to assist with the visualisation of various types of data created by the project. It has a strong focus on population genetics data, but most of its tools are generic and can be used on a wide range of data.

1.1.1 List of basic features

Table viewer A cornerstone element of Panoptes is a paged table viewer that can serve tables of unlimited size.

Query builder Panoptes contains a graphical and interactive query builder that allows the user to create advanced queries in a simple, intuitive way. This query tool automatically hooks up to any other component of the software, such as the table viewer or the charting tools.

Charts The software contains a variety of chart visualisations, such as histograms, bar graphs, scatter plots, etc... . These charts are highly interactive, including colour overlays, tool tips, popups, and selection tools.

Geospatial and geotemporal visualisations A specific subset of charts deals with interactive visualisation of data points on a map, potentially combined with a time line.

Genome browser Data points that correspond to positions or regions on a genome (such as SNPs or other variants) can be visualised on a genome browser. Numerical properties can be shown as graphics tracks in that browser. An powerful feature of Panoptes is the concept of multiresolution data summarisation, which allows the browser to show properties over the genome in real-time, regardless the zoom level or genome size.

Genotype browser For a set of sequences, genotypes for a set of variants can be visualised as a matrix of variations in the genome browser. Both calls and allele depths are supported, with the ability to view other properties such as quality scores.

Visual Analytics Panoptes implements some basic concepts of Visual Analytics, offering near-realtime visualisations with a high level of interactivity, and extensive selection methods that can be used to drill down in the dataset.

Data sharing One of the fundamental design goals of the application is to serve as a data sharing tool and a collaborative platform between a group of scientists working on a common dataset. As an example, every visualisation that can be created in Panoptes, can be turned into a permanent link, ready to be shared by other users.

Data import Panoptes offers an easy and flexible data import path. It can grab source data from simple, TAB-delimited source files, augmented by settings files containing metadata that instruct the software how to treat these data.

1.2 Installation and deployment guide

1.2.1 Deployment on a new Ubuntu image

For testing purposes, a slightly easier way to obtain a running instance of Panoptes is to do a full deployment on a fresh Ubuntu 14.04.1 LTS image, e.g. on an EC2 virtual machine. A script is provided that performs a fully automatic installation, including

- Installation of all dependencies
- Deployment and configuration of MySQL
- Deployment and configuration of Apache2

Caution: This deployment option will aggressively override packages and settings on the machine. It is only intended to be used on a fresh image.

The following steps will create a fully working Panoptes instance on an Ubuntu 14.04.1 LTS image:

```
cd /
sudo wget https://raw.githubusercontent.com/cggh/panoptes/master/scripts/deploy_default/deployfull.sh
sudo chmod +x deployfull.sh
sudo ./deployfull.sh
```

The source data folder is set to */panoptes/sourcedata*. The application is accessible from *[ServerAddress]/index.html*.

1.2.2 Short debain/ubuntu guide for the temporarily challenged

```
sudo apt-get install mysql-server-5.6 mysql-client-5.6 git gcc gfortran python-dev python-virtualenv
wget https://github.com/cggh/panoptes/archive/master.zip
unzip master.zip
cd panoptes-master
cp config.py.example config.py
nano config.py #EDIT DB CREDENTIALS AND FILE PATHS
./scripts/build.sh
./scripts/run.sh
```

1.2.3 Basic installation

Download & dependencies

Download the code from the GitHub repository:

```
wget https://github.com/cggh/panoptes/archive/master.zip
unzip master.zip
cd panoptes-master
```

Panoptes needs a running MySQL version **5.6 or later** with permission to create and remove databases. The MySQL client tools also have to be installed on the machine running Panoptes. Install MySQL if you don't have it E.g. for debian-based Linuxes:

```
sudo apt-get install mysql-server-5.6 mysql-client-5.6
```


Caution: Note that if there are tables from other apps that name-collide with Panoptes dataset names then there will be data loss. **Use a separate MySQL install or set your MySQL permissions carefully!**

You will need to install the following packages (or equivalent) before Panoptes can be installed. E.g. for debian-based Linuxes:

```
sudo apt-get install git gcc gfortran python-dev python-virtualenv libblas-dev liblapack-dev cython
```

Build

In the directory where the code was unzipped, copy ‘config.py.example’ to ‘config.py’. Edit the file and specify the following components:

- MySQL setup (DBSRV, DBUSER, DBPASS).
- A directory Panoptes can use for storing files (BASEDIR, see further).
- A directory that will contain the source data files (SOURCEDATADIR, see further)
- Title of the deployment (TITLE)
- Extra JS for utilities and tracking such as rollbar etc. Note that google analytics can be set on a dataset level. (EXTRA_HEAD, EXTRA_TAIL)

The login credentials used need to have sufficient privileges to perform alterations such as database creation. .. note:

Changes in 'config.py' are fixed on build, so you will need to rebuild if they change.

To build run:

```
./scripts/build.sh
```

To build for development:

```
./scripts/build.sh DEV
```

to create a panoptes installation in ‘build’. Note that this deletes any existing build. This build copies the different components of the application, and merges them into a single file structure. Note that, during this process, a copy of *config.py* is put in the build folder. This copy is used by the actual server process. This will attempt to install the needed python packages and link Panoptes into the DQXServer framework which serves the app.

Server data file structure

Panoptes uses two file directories, and the location of both has to be specified in config.py (example: [config.py.sample](#)).

BASEDIR: This is the root directory for storing file-based server data. It should contain subdirectories “Summary-Tracks”, “Uploads” and “temp”. All should have write privileges for the user that runs the server.

SOURCEDATADIR: This directory contains the file-bases data sources that are used to import into the Panoptes datasets.

Note: Both paths have to be specified as absolute, starting from /. Do not use relative paths here.

See section [Loading data](#) for more information on how to populate the Panoptes instance with data.

Simple Server

The simplest way to run Panoptes is using:

```
./scripts/run.sh
```

by default, this serves Panoptes on <http://localhost:8000/index.html> using gunicorn. To run on your external network interface use (with the port you desire):

```
./scripts/run.sh 0.0.0.0:8000
```

Note that you will need internet access even if you run Panoptes locally due to google-hosted mapping tools.

Deployment on Apache2 (OPTIONAL)

Note: This section describes a deployment strategy where the static files (html, css, js) are also served through the WSGI interface. This allows one to protect the application using a CAS Single Sign-On service.

Install the Apache2 wsgi dependency *libapache2-mod-wsgi*.

Create a symbolic link in `/var/www/` to `[PanoptesInstallationPath]/build/DQXServer/wsgi_server.py`:

```
ln -s [PanoptesInstallationPath]/build/DQXServer/wsgi_server.py /var/www/.
```

The build script uses a virtualenv for the installation of Python dependencies, and the Apache2 WSGI configuration has to be instructed to use that virtualenv. An example VirtualHost config would be (note that the tokens need to be replaced by their proper values):

```
<VirtualHost *:80>
    DocumentRoot /var/www
    <Directory />
        Options FollowSymLinks
        AllowOverride None
    </Directory>
    WSGIDaemonProcess Panoptes processes=2 threads=25 python-path=[PanoptesInstallationPath]/build/p
    WSGIProcessGroup Panoptes
    WSGIScriptAlias / /var/www/wsgi_server.py
</VirtualHost>
```

In this configuration, the app is served from:

```
[ServerName]:80/
```

1.2.4 Authorization

Panoptes contains a simple authorization mechanism that can be used to grant or deny certain privileges on datasets. There are three levels of privileges:

- Read: View the data in a dataset.
- Edit: Add custom data properties to a workspace.
- Manage: All actions, including loading the dataset from the file source.

The authorization mechanism interacts with authentication systems implemented at the web server level, by reading the `REMOTE_USER` environment variable.

Specifically, Panoptes can integrate with a CAS Single Sign-On service. To enable this, specify the CAS service url in the `CAS_SERVICE` variable in `config.py`. In this case, authentication can also be based on user groups.

The file `PanoptesAuthDb` (<https://raw2.github.com/cggh/panoptes/master/servermodule/panoptesserver/PanoptesAuthDb>) is used to link user authentication information to privileges on specific datasets. The default installation grants all rights to everybody.

1.3 Loading data

Panoptes imports datasets into the server database from source data, consisting in a set of simple, structured files present on the server. Importing a dataset from source data does not happen automatically, and has to be initiated by the user (see [Importing datasets](#)). This import action creates all necessary components to serve the dataset in Panoptes (relational database, preprocessed files, etc...)

See also:

1.3.1 Panoptes data concepts

The data served by Panoptes is structured according to a number of central concepts:

Dataset

A complete set of data that can be loaded and visualized in a single Panoptes session. A dataset can consist of:

- One or more *workspaces*.
- A set of *data tables*.
- A *reference genome*, including annotation.
- A set of *summary values*, defined on the reference genome.
- *2D data table*.
- Various settings that define the structure of these components, and the interactions between them.

See also:

- *Creating a new dataset*
- *Dataset source files*

Workspace

Each *dataset* has one or more *workspaces* associated. The user always opens a specific *workspace*, and can add *custom data* to the *dataset* that is only visible in the context of this *workspace*.

In addition certain entities are specific to an individual *workspace*:

- Stored queries
- Subsets

See also:

- *Add a new workspace to a dataset*
- *Workspace source files*

Data table

A *data table* is a table of records, belonging to a *data set*, that corresponds to a type of information and be queried and visualized in Panoptes. A record in a *data table* is called a *data item*. Each *data table* has a number of columns called *properties*.

Optionally, a data table can be defines as containing a set of genomic positions, or genomic regions.

Examples: [samples table](#) and [variants table](#) .

See also:

- [Add a new data table to a dataset](#)
- [Data table source files](#)

Data item

A *data item* is an individual record in a *data table*. It has a number of *properties*, defined by columns in the *data table*. For example, it may correspond to an individual sample, or an individual variant.

On a running Panoptes instance, it is possible to format a url that creates a deep link into a view showing information about an individual data item:

```
[BaseUrl]?dataset=[DatasetId]&workspace=[WorkspaceId]&tableid=[TableId]&itemid=[DataItemId]
```

If the server only serves a single data set, the `dataset` token may be omitted. If the dataset contains only a single workspace, the `workspace` token may be omitted.

Property

A *property* is a column in a *data table*. As such, it defines a property of a *data item*. Examples are collection dates and geographical coordinates for samples. There are two types of properties:

Standard property: Provided in the *dataset*. These are visible in every *workspace*.

Custom property: Specified at the level of a *workspace* (see [Custom data](#)).

Reference genome

A *dataset* can have information that relates to a *reference genome*, such a genomic variants. In Panoptes, the reference genome may include the following elements:

Reference sequence: Imported from a FASTA file, and displayed as colour codes in the genome browser.

Annotation: Imported from a GFF file, and displayed as an annotation track in the genome browser.

See also:

- [Add reference genome annotation](#)
- [Add reference genome sequence](#)
- [Reference genome source files](#)

Summary value

A *summary value* is a filterbanked property defined over the *reference genome*. There are three types:

Standard summary value Provided as a *property* of a *data table*.

Custom summary value Specified as a *property* at the level of a *custom data source*.

Data table - related: A type of *summary value* that has an instance for each *data item* in a *data table*. Example: sequencing coverage info for all samples in a data table.

2D data table

A *2D data table* is a matrix of data that combines two (1D) *data tables*, and provides information for each combination of elements from both tables. One data table serves as the definition of the rows, the other as the definition of the columns of the matrix. Several properties can be defined for each cell. These properties can be thought of as layers of the matrix in a third dimension.

The typical use case is *genotype data*, information about the genomic variations found on a set of genomic sequences (called *samples*), over a set of *variants*. Both the *samples* and the *variants* have to be present in the dataset as *data tables*, with the *2D data table* containing the genotypes linked to them by the RowDataTable and ColumnDataTable. The *variants* data table should be defined as containing genomic positions (see also *Data table settings*).

The *genotype data* is visualised on the genome browser, with the *samples* as rows, and the *variants* as columns displayed at the corresponding genomic positions. A number of properties can be stored and visualised for each genotype, such as call, quality, and coverage.

Custom data

Panoptes allows the user to augment a *data table*, in the context of a specific *workspace*, with additional *properties*.


These extra properties will show up as additional columns for this data table in a Panoptes session that opens this workspace.

See also:

- [Add a custom data source to a workspace](#)
- [Custom data source files](#)

1.3.2 Importing datasets

Importing a dataset source does not happen automatically, and has to be initiated by the user. After installation, a number of sample dataset sources are copied into the source data folder, and are ready to be imported.

- Start the Panoptes app in a browser.
- In the intro screen, click on the hyperlink “Open admin page”. This creates a new tab in the browser, showing the administration section of the app.
- The administration section shows the available source data file sets as a tree. Click on the cog wheel icon () on the left of the dataset you want to import (e.g. “Samples_and_Variants”). An *Import dialog box* appears.
- Check the option “Full import”.
- Click the button “Import”.
- This initiates the data import. A progress box is shown during this action.

- Upon completion, a new item appears in the list “Server calculations”. Clicking on this shows a log of the import activities. If an error occurred, this can be useful for troubleshooting.
- Go back to the browser tab with the Panoptes intro screen, and reload the app by clicking the browser refresh button to retrieve the updated dataset information.
- The imported dataset should appear in the list.

See also:

Import dialog box

The “Import file source data” dialog box is used to initiate import of a dataset, interpreting the source data (see [Source files structure](#)), and creating the environment required for serving the dataset in Panoptes. This includes:

- Create the relational database containing tables for all data tables.
- Transform all necessary files such as GFF annotation data.
- If applicable, execute all filterbanking actions on genomic data.
- If applicable, transform genotyping data into a format suitable for serving.

In this dialog, one of the import types has to be chosen prior to activating the import:

Full import: This option executes a full import from scratch of the complete source data for the selected component. If an entire dataset is imported, the relational database used for serving will be dropped and rebuilt. Note that, for large datasets, this may take a long time to complete.

Update configuration only: This option does not import any data at all, but can be used to quickly update the settings on the served dataset, based on changes in the definitions in the source data settings files (e.g. descriptive texts, colours, etc...).

- **NOTE:** this option can only be used when a dataset was already imported earlier using the “Full import” or “Top X preview” option.
- **NOTE:** some setting changes (e.g. filterbanking definitions) do require an actual data import.

Top 1K, 10K, ... preview: This option executes a full import of a portion of the source data for the selected component. If an entire dataset is imported, the relational database used for serving will be dropped and rebuilt. Use this option to test run import of a large dataset on a subset of the source data. In this way, a much quicker turnaround can be achieved when debugging and tweaking the settings files. **NOTE:** if a top X import action is performed on a dataset where a full import was already executed earlier, the data not present in the top subset will be removed from the running dataset.

After selecting the appropriate import type, click the “Import” button to activate the import.

New source data can be added following two approaches:

1.3.3 Editing the source directory structure

The source data directory structure on the server can be directly manipulated: adding new directories, copying data files and setting files, etc... These source data files are located in the directory `[SOURCEDATADIR]/datasets` (`[SOURCEDATADIR]` as specified in `config.py`, see [config.py.sample](#)).

Note: This is a more low-level approach, which works best for large data sets and experienced data administrators.

See also:

Source files structure

Internally, Panoptes uses a combination of a set of MySQL databases and a file structure to serve the data. Data are loaded into this system by launching an **import action** that reads the data from a **source file location** (specified by `SOURCEDATADIR` in `config.py`, see also *Server data file structure*).

The formatting of the source data relies a few concepts:

- It is organised in a way that closely mimicks the basic concepts of the Panoptes data structures, using nested folders to reflect the structure.
- In most cases, data are provided using simple, TAB-delimited files. Exceptions are made in those cases where a widely accepted standard format is used for a specific type of information (e.g. GFF files for genome annotations).
- YAML (<http://www.yaml.org/about.html>) structured files are used to provide the necessary metadata to interpret and parse the data in the context of Panoptes. These metadata are provided in files called `settings`.

Caution: Many identifiers used in the source data structures (folder names, table column headers, etc..), are directly mapped to identifiers in the MySQL database tables. Therefore, they should be formatted as standard variable names (e.g. do not contain dashes, white spaces or other special characters, do not start with a number, ...)

Dataset source files

The `config.SOURCEDATADIR` folder should contain a folder `datasets`, serving as a root for all *datasets* being served by the Panoptes instance.

In this folder, a subfolder should be present for each *dataset*. The folder name is used as the unique identifier of this dataset. In the *dataset* folder, a `yaml settings` file should be present, specifying the displayed name of the dataset, and an optional description (see *General dataset settings*).

See also:

- *Dataset*
- *Source files structure*

Reference genome source files

A *dataset source folder* may optionally contain a subfolder `refgenome`, describing the *reference genome* used. It can contain the following files:

- `chromosomes` (required). A list of all chromosomes identifiers, and their lengths (in MB).
- `annotation.gff` (required). The annotation of the reference genome, in GFF format.
- `refsequence.fa` (optional). The reference genome sequence, as FASTA file.
- `settings` (required, `yaml` formatted). Various settings concerning the reference genome (see *Reference genome settings*).

Summary values source files The `refgenome` folder may contain an optional subfolder `summaryvalues`. Each subfolder in this folder represents a different (numerical) property defined over the genome that will be filter banked and can be displayed in the genome browser. The folder name serves as the identifier of the summary value. Each summary value folder should contain the following two files:

- `values`. A TAB-delimited file having three columns, and no header ([example file](#)):
 - column 1: Chromosome identifier
 - column 2: Position
 - column 3: Value
- `settings` (yaml formatted). Contains the displayed name of the summary value, and further guidelines on how to process the information ([sample numeric file](#)). ([sample categorical](#)).

See also:

- [Reference genome](#)
- [Source files structure](#)

Data table source files

In the [dataset source folder](#) folder, a subfolder `datatables` should be present. This is the root for a set of folders, each one describing an individual [data table](#), with the name of the folder serving as an identifier.

In each [data table](#) folder, a file `data` should be present, containing a list of all the [data items](#) in the table. Each line consists in a set of TAB-delimited [properties](#). The first line of the file serves as a header, specifying the identifiers for all [properties](#) ([example file](#)).

In addition, a yaml `settings` file should be present in the [data table](#) folder. This file can contain a number of settings, both at the level of the [data table](#), as at the level of individual [properties](#) (see [Data table settings](#)).

See also:

- [Data table](#)
- [Source files structure](#)

2D data table source files

In the [dataset source folder](#) folder, a subfolder `2D_datatables` should be present. This is the root for a set of folders, each one describing an individual [2D data table](#), with the name of the folder serving as an identifier.

In each [2D data table](#) folder, a file `data.hdf5` should be present, containing the arrays of properties. ([example file](#)).

In addition, a yaml `settings` file should be present in the [2D data table](#) folder (see [2D Datatable settings](#)).

HDF5 source file structure The source file `data.hdf5` should be structured according to the [HDF5 standard](#), and may contain the following arrays, which must be contained in the root of the HDF5 file:

Properties arrays One or more arrays specifying properties of the 2D data table. Note that these arrays can be 3D but the first two dimensions should be row and column.

Column index 1D array A 1D array listing the identifiers of all columns, in the order they are used in the properties matrices.

Row index 1D array A 1D array listing the identifiers of all rows, in the order they are used in the properties matrices.

Only scalar builtin dtypes (ie not structured with fields or user-defined) or strings currently permitted for HDF5 arrays.

Example python HDF5 creation code:


```
import h5py
outfile = h5py.File(filename, 'w', libver='latest')
call = outfile.create_dataset("call", (1000,10,2), dtype='i1')
call[:, :, :] = my_array_of_calls
allele_depth = outfile.create_dataset("allele_depth", (1000,10,3), dtype='i2')
allele_depth[:, :, :] = my_array_depth
quality = outfile.create_dataset("quality", (1000,10), dtype='i4')
quality[:, :] = my_array_of_quality
outfile.close()
```

We recommend using [VCFNP](#) for converting from VCF. See the [VCF example](#) for details of how to do this.

See also

- [Data table](#)
- [Source files structure](#)

Workspace source files

In the [dataset source folder](#), a subfolder `workspaces` should be present. This is the root for a set of subfolders, each one describing a [workspace](#) for this *dataset*. The folder name serves as identifier for the *workspace*.

In a *workspace* folder, a yaml structured `settings` file should be present, specifying the displayed name of the workspace (see [Workspace settings](#)).

In addition, a subfolder `customdata` should be present. This location is used to specify *Custom data*, which has the following basic properties:

- It only exists in the context of a specific *workspace*.
- It adds extra properties to a *data table* that already exists in the *dataset*.
- The primary key of the *data table* (as defined in the settings) is used to link the custom properties to the original table.

See also:

- [Workspace](#)
- [Source files structure](#)

Custom data source files

The `customdata` folder in a [workspace source folder](#) should have a subfolder for each *data table* it defines data for, and the folder name should be the *data table* identifier. In this data table - specific folder, a number of subfolder can be defined, each one specifying an individual [custom data source](#). Such a subfolder should contain two files:

- `data`. TAB-delimited file containing the custom property values ([example file](#)).
- `settings`. (yaml formatted). Specifies how the custom data should be interpreted (see [Custom data settings](#)).

See also:

- [Custom data](#)
- [Source files structure](#)

Documentation source files

A *dataset source folder* may optionally contain a subfolder `doc`, containing html files that can be displayed in Panoptes' internal documentation viewer.

These files should follow proper XML formatting, and contain a `<body>` element. The html may contain hyperlinks to other documentation files in the same source directory, or to external links.

These documents may be referred to in other components of the source data, such as descriptions of the dataset or data tables. Referring happens through a hyperlink with the structure `hyperlink display name`, with `[docid]` the file name of the document file *without the .html extension*.

On the deployment, this will render as a hyperlink that leads to an in-app popup showing the documentation in the source file.

1.3.4 Using the Panoptes admin web frontend


Panoptes has an **admin** web frontend that can be used to manipulate the source data. This admin section can be opened in several ways:

- From the Panoptes web app, when the startup menu is show with a choice of datasets: click on “*Open admin page*”.
- From a Panoptes dataset session: click on the Panoptes logo in the top left corner, and then on “*Open admin page*” in the popup.

Note: The user must have sufficient privileges in order to have access to this admin section (see also *Authorization*).

See also:

Creating a new dataset

Using the admin web frontend, a new *dataset* can be created by clicking the  icon, next to the top branch “*Datasets*”.


In the dialog box, specify a unique identifier for this new dataset.


Note: This identifier will be used internally, and is different from the display name which is specified in the settings. The data will be imported into a database that has a name equal to this identifier.


Caution: Make sure that the identifier is a valid variable name (see *Valid data identifiers*).

This action creates a new source data directory for this dataset on the server (see also *Dataset source files*).


Follow-up actions

Modify the settings: Click on the  icon left of the dataset label (see *Data import settings* and *General dataset settings*).

Add a data table to this dataset: Click on the  icon right of the “*Datatables*” label in the dataset section (see *Add a new data table to a dataset*).

Import the source data: Importing a dataset from source the data to the server database does not happen automatically, and has to be initiated by the user. Click on the  icon left of the dataset label to initiate this import. (see [Import dialog box](#)).

Add a new data table to a dataset

Using the admin web frontend, a new [data table](#) can be created by clicking the button , next to the label “*Datatables*”, under the tree branch that represents the dataset.

A dialog box appears that allows one to pick a TAB-delimited file from the local computer, and upload it to the Panoptes server as a source file for a new data table ([example file](#)).

- Click “*Choose file*” to select the local file that contains the data for this data table.
- Click “*Create data table*” to create a new data table source, based on this file.

Notes


- The name of the local source file will be used as an internal identifier for this custom data table. During import, these data will be loaded into a relational database table that has a name equal to this identifier.
- The columns in the source file will be mapped to properties of the data table.
- The column header names will be used as the property unique ID’s.


Caution: Make sure that the source file name, and the column headers are valid variable names (see [Valid data identifiers](#)).

This action creates a new data table directory for this dataset on the server, and uploads the file as data source (see also [Data table source files](#)).


Follow-up actions

Review the uploaded data file content: Click on the  icon left of the data table label to bring up a top N row view of the uploaded source data.

Modify the settings: Click on the  icon left of the data table label (see [Data import settings](#) and [Data table settings](#)).

Import the source data: Updating a dataset from source the data to the server database does not happen automatically, and has to be initiated by the user. Click on the  icon left of the dataset label (see [Import dialog box](#)).

Add a new workspace to a dataset

Using the admin web frontend, a new [workspace](#) can be created by clicking the button , next to the label “*Workspaces*”, under the tree branch that represents the dataset.


In the dialog box, specify a unique identifier for this new workspace.

Note: This identifier will be used internally, and is different from the display name which is specified in the settings.


Caution: Make sure that the identifier is a valid variable name (see [Valid data identifiers](#)).

This action creates a new workspace directory for this dataset on the server (see also [Workspace source files](#)).


Follow-up actions

Modify the settings: Click on the  icon left of the workspace label (see [Data import settings](#)).

Add custom data to this workspace: Click on the  icon right of the “Custom data” label in the workspace section (see [Add a custom data source to a workspace](#) and [Workspace settings](#)).

Re-import the source data: Updating a dataset from source the data to the server database does not happen automatically, and has to be initiated by the user. Click on the  icon left of the dataset label (see [Import dialog box](#)).

Add a custom data source to a workspace

Using the admin web frontend, [custom data](#) can be added by clicking the button , next to the label “Custom data”, under the tree branch that represents the workspace.

A dialog box appears that allows one to pick a TAB-delimited file from the local computer, and upload it to the Panoptes server as a source file for a new custom data source ([example file](#)).

- Click “Choose file” to select the local file that contains the data for this custom data source.
- Select the data table identifier to which this custom data should be attached to
- Click “Create custom data file” to create a new custom data source, based on this file.

This action creates a new custom data directory for this workspace on the server, and uploads the file as data source (see also [Custom data source files](#)).


Notes

- The name of the local source file will be used as an internal identifier for this custom data source.
- The columns in the source file will be mapped to extra properties of the data table, only visible in the context of the workspace this data source is loaded in.
- The column header names will be used as the property unique ID’s.
- The property ID’s defined in this custom data source must be globally unique for this data table, and all other custom data sources associated with this data table.
- The custom data source **must** have a column with a name that corresponds to the primary key defined for this data table. The custom data records will be merged to the data table by joining the values of this primary key.



Caution: Make sure that the source file name, and the column headers are valid variable names (see [Valid data identifiers](#)).

Follow-up actions

Review the uploaded data file content: Click on the  icon left of the data table label to bring up a top N row view of the uploaded source data.


Modify the settings: Click on the  icon left of the custom data label (see [Data import settings](#) and [Custom data settings](#)).

Import the source data: Updating a dataset from source the data to the server database does not happen automatically, and has to be initiated by the user.

- If the dataset was not yet been imported or is outdated, click on the  icon left of the dataset label. This will initiate the import of the entire data set, including custom data.
- If the dataset was already imported earlier, the custom data can be added by clicking on the  icon left of the custom data label. This will only import the custom data, augmenting the already deployed database. (see [Import dialog box](#)).

Add reference genome annotation

Using the admin web frontend, [reference genome](#) annotation data can be added to a dataset by


1. Clicking the  icon to the left of the label “**Reference genome**”, under the tree branch that represents the dataset.
2. Click the button “Annotation” in the popup.

A dialog box appears that allows one to pick a GFF annotation file from the local computer, and upload it to the Panoptes server as a source file for genome annotation.


- Click “*Choose file*” to select the local file that contains the genome annotation.
- Click “*Create annotation*” to define this file as the source of annotation.


Follow-up actions

Modify the reference genome settings:

1. Click on the  icon left of the label “**Reference genome**”
2. Click on the button “Edit Settings” in the popup. (see [Data import settings](#) and [Reference genome settings](#))


Edit the chromosome definitions:

1. Click on the  icon left of the label “**Reference genome**”
2. Click on the button “Edit Chromosome definition” in the popup.

Import the source data: Updating a dataset from source the data to the server database does not happen automatically, and has to be initiated by the user. Click on the  icon left of the dataset label to initiate this import. (see [Import dialog box](#)).

Add reference genome sequence

Using the admin web frontend, a [reference genome](#) sequence can be added to a dataset by


1. Clicking the  icon to the left of the label “**Reference genome**”, under the tree branch that represents the dataset.
2. Click the button “Reference genome” in the popup.

A dialog box appears that allows one to pick a FASTA sequence file from the local computer, and upload it to the Panoptes server as a source file for the reference genome sequence.


- Click “*Choose file*” to select the local file that contains the reference genome sequence.
- Click “*Create reference genome*” to define this file as the source of the reference genome sequence.


Follow-up actions

Modify the reference genome settings:


1. Click on the  icon left of the label “**Reference genome**”
2. Click non the button “Edit Settings” in the popup. (see [Data import settings](#) and [Reference genome settings](#))

Edit the chromosome definitions:

1. Click on the  icon left of the label “**Reference genome**”
2. Click non the button “Edit Chromosome definition” in the popup.

Import the source data: Updating a dataset from source the data to the server database does not happen automatically, and has to be initiated by the user. Click on the  icon left of the dataset label to initiate this import. (see [Import dialog box](#)).

Data import settings

Most source data resources have settings that are specified through a [YAML](#) definition file. Using the admin web frontend, these settings can be edited by clicking the  icon, left of the label that identifies a source data resource.

Note: The sample dataset [Samples_and_Variants](#) contains settings files that are fully commented, and can serve as a starting point to explore the possible options. There is also a [VCF example](#) which shows data imported from VCF. Additional comments are provided in other sample datasets as well, wherever concepts are introduced that are not present in this dataset.

General dataset settings

This [YAML](#) file contains settings for a *dataset*. See also:

- [Data import settings](#)
- [Creating a new dataset](#)
- [Example file](#)

Possible keys

Name *Text (required).* The visible name of the dataset, as it appears on the intro page.

NameBanner *Text.* Visible name of the dataset, as it appears on the top banner of the app. Note: this text may contain html markup.

Description *Text*. A description of the dataset that appears on the start page. Note: this text may contain html markup, and documentation links (see [Documentation source files](#)). A longer description can be split over several lines by writing a > sign on the key line, and indent subsequent lines:

```
Description: >
  This web application provides an interactive view
  on the data ...
```

DataTables *List*. A list of the data table identifiers in the dataset. These names should correspond to directory names in the *datatables* source directory (see [Data table source files](#)). This can be included in the settings in order to provide an explicit ordering of the data tables in the app. If this key is not provided, a default ordering will be used.

2D_DataTables *List*. List the 2D data tables that should be exposed in the app.

IntroRightPanelFrac *Value*. Controls the proportion of left and right columns on the start page. If set to zero, the right column will be absent.

IntroSections *List*. Enumerates sections on the intro page that can contain quick start buttons to specific views in the app. Buttons can be added to these sections by (1) clicking on the “Get Link” button in the top right corner of the app, (2) clicking on one of the “Add to start page” options, and (3) entering the right section id in the “Section” edit box. Similarly, a button displaying a plot can be created by clicking the link button in the plot popup. The block can contain the following keys:

Id *Text*. Unique identifier of the section.

Name *Text*. Displayed title.

Content *Text*. Intro text of the section, appearing above the buttons. This text can be HTML formatted.

RightPanel *Boolean*. If set, the section will appear in the right column, replacing the default content of this column.

GoogleAnalyticsId *Text*. .

Data table settings

This [YAML](#) file contains settings for a *data table*. See also:

- [Data import settings](#)
- [Add a new data table to a dataset](#)
- [Example file](#)

Possible keys

NameSingle *Text (required)*. Display name referring to a single table item (single, without starting capital).

NamePlural *Text (required)*. Display name referring to several table items (plural, without starting capital).

Description *Text*. Default:. A short description of this data table. This text will appear on the intro page, and on the table view page of this data table. Note: this text may contain documentation links (see [Documentation source files](#)).

Icon *Text*. Specifies an icon that will be associated with the data table. The icon name can be chosen from the list specified in <http://fortawesome.github.io/Font-Awesome/icons/>.

IsHidden *Boolean*. If set to true, the data table will not be displayed as a standalone entity (i.e. not mentioned on the intro page and no tab).

PrimKey *PropertyID (required)*. The primary key *property ID* for this table. A data item *property* is a column in the TAB-delimited source file *data*, and the *ID* corresponds to the column header. The primary key should refer to a column containing a unique value for each record in the table. Optionally, this parameter can be set to 'AutoKey' to instruct the software to automatically generate a primary key.

ItemTitle *Text*. A [handlebars](#) template. Defaults to the primary key. The rendered template will be used when a data item title is needed.

SortDefault *PropertyID*. Specifies the property ID (i.e. column name in the *data* source file) used as the default sort field..

CacheWorkspaceData *Boolean*. If set, a materialised table will be created in the relational database for this data in each workspace. For large data tables (>1M records), this option is faster than the standard option, which uses a JOIN statement.

MaxCountQueryRecords *Value*. Default:200000. Defines the maximum number of records that will be downloaded to the client. This limit influences views that display individual data items, such as scatter plots and geographical map views. If not specified, this defaults to 200,000.

MaxCountQueryAggregated *Value*. Default:1000000. Defines the maximum number of records that will be queried on the server for views that present data items in an aggregated way, such as histograms and bar graphs. If not specified, this defaults to 1,000,000.

FetchRecordCount *Boolean*. Default:False. .

QuickFindFields *PropertyIDs*. The list of properties will be used by some tools in the software that allow the user to quickly find a (set of) item(s).

ColumnIndexField *Text*. When this table is linked to a 2D data table setting this value to the same as that in the 2D settings provides a performance improvement for large data sets.

DisableSubsets *Boolean*. If set, there will be no subsets options for this data table.

DisablePlots *Boolean*. If set, there will be no options to create plots for this data table.

DisableNotes *Boolean*. If set, it will not be possible to define notes for items in this data table.

PropertyGroups *List*. Each item in the list specifies a group of properties. Property groups are used to combine sets of related properties into logical sections in the app. The block can contain the following keys:

Id *Text (required)*. a unique identifier for the group.

Name *Text (required)*. a display name.

AutoScanProperties - deprecated - please use scripts/mksettings.sh to generate a skeleton settings.gen file and use that to create a settings file .. _Properties: Properties

List (required). Each list item defines a [property](#), linked to a column in the TAB-delimited source file *data*. See [Datatable property settings](#) for an overview of the keys that can be used for each property in this list.

DataItemViews *List*. Definitions of custom views that will appear in the popup for an individual data table item. The block can contain the following keys:

Type *Text (required)*. Identifier of the custom view type

(can be Overview, PropertyGroup, FieldList, ItemMap, PieChartMap) See [DataItemViews settings](#) for more details about defining custom data item views.

ExternalLinks *List*. Each item in the list specifies a link for a data item to an external url. These links show up in the app as buttons in the data item popup window. The block can contain the following keys:

Url

Text (required). Url for this link. This may include tokens property ID's between curly braces.

These tokens will be expanded to their actual content for a specific data item. Example:

`http://maps.google.com/maps?q={Latitude},{Longitude}`.

Name *Text (required).* Display name for this external link.

ListView *Boolean.* Default:False. Replaces the normal table view with a list view, showing rows on left and a single selected row on the right.

IsPositionOnGenome *Boolean.* Default:False. Instructs Panoptes that records in this data table should be interpreted as genomic positions. In this case, the *Chromosome* and *Position* keys should be defined.

IsRegionOnGenome *Boolean.* Default:False. Instructs Panoptes that records in this datatable should be interpreted as genomic regions. In this case, the *Chromosome*, *RegionStart* and *RegionStop* keys should be defined.

BrowserTrackHeightFactor *Value.* Specifies a relative size factor for the genome browser track height (only applicable if *IsPositionOnGenome* or *IsRegionOnGenome* is set).

Chromosome *PropertyID.* Specifies the table column ID that contains the chromosome (only to be used if *IsPositionOnGenome* or *IsRegionOnGenome* is set). Note that the values in this column should correspond to the content of the `chromosomes` source file (see [Reference genome source files](#)).

Position *PropertyID.* Specifies the table column ID that contains the position on the chromosome (only to be used if *IsPositionOnGenome* is set).

RegionStart *PropertyID.* Specifies the table column ID that contains the start position of the region (only to be used if *IsRegionOnGenome* is set).

RegionStop *PropertyID.* Specifies the table column ID that contains the end position of the region (only to be used if *IsRegionOnGenome* is set).

GenomeMaxViewportSizeX *Value.* Specifies the maximum genome browser viewport size (in bp) for which individual data points from this table will be displayed in the tracks. (only to be used if *IsPositionOnGenome* or *IsRegionOnGenome* is set).

BrowserDefaultVisible *Boolean.* For genomic regions: specifies the default visibility status of this data table in the genome browser (only to be used if *IsRegionOnGenome* is set). Note that, for genomic position, default visibility is specified on a per-property basis.

AllowSubSampling *Boolean.* Default:False. .

MaxTableSize *Value.* Default:None. .

BrowserDefaultLabel *PropertyID.* Specifies the default label that is used in the genome browser, used for genomic regions. None indicates that no label is displayed by default.

TableBasedSummaryValues *List.* Declares that numerical genome values for are available for each item in the table. Panoptes will process these using the multiresolution filterbanking, and the user can display these as tracks in the genome browser. A typical use case is if the data table contains samples that were sequenced, and there is coverage data available

Approach 1

There should be a subdirectory named after the identifier of this track in the data table source data folder. For each data item, this directory should contain a data file with the name equal to the primary key (see [example](#)). The input files should not contain a header row

The Id is the identifier of this set of per-data-item genomic values i.e. the name of the subdirectory

Approach 2

This approach is more like the way the table based data files are processed. In this case multiple tracks can be stored in the same input file. The Id corresponds to the column name instead of the directory name with the directory details given in the FilePattern expression The name is the first match in the FilePattern expression

The block can contain the following keys:

Id *Text (required)*. Identifier of this set of per-data-item genomic values - name of subdirectory or Identifier of this set of per-data-item genomic values - name of the column in the matching files.

FilePattern *Text*. A glob (regular expression) containing a relative path to the file(s).

Name *Text (required)*. Display name of the property.

MinVal *Value (required)*. Default:0. Value used for lower extent of scales.

MaxVal *Value (required)*. Value used for upper extent of scales.

BlockSizeMin *Value (required)*. Default:1. Minimum block size used by the multiresolution summariser (in bp).

BlockSizeMax *Value (required)*. Maximum block size used by the multiresolution summariser (in bp).

ChannelColor *Text*. Colour used to display these tracks as a genome browser track. Formatted as "rgb(r, g, b)".

Datatable property settings An overview of the possible keys than can be defined for an individual property in the *Properties* block of the data table settings.

Id *Text (required)*. Identifier of the property, equal to the corresponding column header in the TAB-delimited source file data.

DataType *Text (required)*. Data type of the values in the property. Possible values:

- Text: text strings.
- Value: numerical values (integer of decimal; the distinction is made by the key *DecimDigits*). Absent values can be coded by an empty string, "NA", "None", "NULL", "null", "inf" or "-".
- HighPrecisionValue: same as Value, with higher precision.
- Boolean: Yes/No binary states. Possible values according to YAML: y|Y|yes|Yes|YES|n|N|no|No|NO|true|True|TRUE|false|False|FALSE|on|On|ON|off|Off|OFF. Absent values are coded by an empty string.
- GeoLongitude: longitude part of a geographical coordinates (in decimal degrees). Absent values are coded by an empty string.
- GeoLatitude: latitude part of a geographical coordinates (in decimal degrees). Absent values are coded by an empty string.
- Date: calendar dates, ISO formatted (i.e. YYYY-MM-DD). Absent values are coded by an empty string.

Name *Text (required)*. Display name of the property.

Description *Text*. Brief description of the property. This will appear in hover tool tips and in the popup box if a user clicks a property info button.

GroupId *Text*. Id of the Property group this property belongs to.

ExternalUrl *Text*. A url that should be opened when the user clicks on a value of this property. The url should be formatted as a template, with {value} interpolated to the property value. For example: `http://www.ebi.ac.uk/ena/data/view/{value}`.

IsCategorical *Boolean*. Instructs Panoptes to treat the property as a categorical variable. For example, a combo box with the possible states is automatically shown in queries for this property. Categorical properties are automatically indexed.

CategoryColors *Block*. Specifies display colours for the categorical states of this property. Each key in the block links a possible value of the property to a color (example: `Accepted: rgb(0,192,0)`). The special value `_other_` can be used to specify a color for all other property values that are not listed explicitly.

MaxColumnWidth *Value*. Specifies the maximum width (in pixels) used for the column representing this property in a table view. Longer text will be abbreviated with ellipsis.

BarWidth *Value*. Draws a bar in the background of the table, indicating the value. Requires *MinVal* & *MaxVal* to be defined(only applies if *DataType* is ['Value', 'HighPrecisionValue']).

MinVal *Value*. Default:0. For *Value* types, upper extent of scale(only applies if *DataType* is ['Value', 'HighPrecisionValue']).

MaxVal *Value*. Default:1.0. For *Value* types, lower extent of scale(only applies if *DataType* is ['Value', 'HighPrecisionValue']).

MaxLen *Value*. Default:0. If present used to specify the maximum size of the database column - otherwise it is calculated.

DecimDigits *Value*. For *Value* types, specifies the number of decimal digits used to display the value(only applies if *DataType* is ['Value', 'HighPrecisionValue']).

MaxDecimDigits *Value*. (Not currently used) For *Value* types, specifies the number of decimal digits used to store the value in the database(only applies if *DataType* is ['Value', 'HighPrecisionValue']).

Index *Boolean*. Default:False. If set, instructs Panoptes to create an index for this property in the relational database. For large datasets, this massively speeds up queries and sort commands based on this property.

Search *Text*. Default:None. Indicates that this field can be used for text search in the find data item wizard. Possible values:

- None: .
- Match: only exact matched are searched for.
- StartPattern: searches all text that starts with the string typed by the user.
- Pattern: searches all text that contains the string typed by the user.

Relation *Block*. Defines a many-to-one foreign relation to a parent data table. The parent table should contain a property with the same name as the primary key property in the child table. The block can contain the following keys:

TableId *DatatableID (required)*. Data table ID of the relation parent table.

ForwardName *Text (required)*. Default:belongs to. Display name of the relation from child to parent.

ReverseName *Text (required)*. Default:has. Display name of the relation from parent to child.

ReadData *Boolean*. Default:True. If set to false, this property will not be imported from the TAB-delimited source file.

CanUpdate *Boolean*. Default:False. If set to true, this property can be modified by the user. (*NOTE: under construction*).

ShowInTable *Boolean*. If set, this property will appear by default in data table grids in the application.

ShowInBrowser *Boolean*. If set, this property will automatically appear as a track in the genome browser (only applies if *IsPositionOnGenome* is specified in database settings).

BrowserDefaultVisible *Boolean*. Indicates that the track will activated by default in the genome browser (only applies if *ShowInBrowser* is True).

BrowserShowOnTop *Boolean*. Indicates that the track will be shown in the top (non-scrolling) area of the genome browser. In this case, it will always be visible (only applies if *ShowInBrowser* is True).

ChannelName

Text. **Name of the genome browser track this property will be displayed in.** Properties sharing the same track name will be displayed in overlay (only applies if *ShowInBrowser* is True).

ChannelColor *Text*. Colour used to display this property in the genome browser. Formatted as "rgb(r,g,b)" (only applies if *ShowInBrowser* is True).

ConnectLines *Boolean*. Indicate that the points will be connected with lines in the genome browser (only applies if *ShowInBrowser* is True).

DefaultVisible *Boolean*. Default:True. .

Order *Value*. Default:-1. Only used for reference genome tracks.

SummaryValues *Block*. Instructs Panoptes to apply a multiresolution summary algorithm for fast display of this property in the genome browser at any zoom level(only applies if *ShowInBrowser* is True). The block can contain the following keys:

BlockSizeMin *Value*. Default:1. Minimum summary block size (in bp).

BlockSizeMax *Value (required)*. Maximum summary block size (in bp).

ChannelColor *Text*. Colour of the channel, for numerical channels. Formatted as "rgb(r,g,b)".

MaxDensity *Value*. For categorical properties this set the scale for the summary track in rows/bp. Defaults to 1/bp.

DataItemViews settings The key *Type* for member of the data table settings key *DataItemViews* can have the following values:

Type *Text (required)*. Identifier of the custom view type (can be Overview, PropertyGroup, FieldList, ItemMap, PieChartMap) See DataItemViews settings for more details about defining custom data item views. Possible values:

- Overview: Specifies the default data item view of Panoptes, including all fields.
- PropertyGroup: Displays all properties that are member of a specific property group.
- FieldList: Displays a selection of properties for the data item.
- ItemMap: Displays the data item as a pin on a geographical map. Requires the presence of properties with data type GeoLongitude and GeoLatitude.
- PieChartMap: Defines a view that shows a set of pie charts on a geographic map (see example). This is achieved by combining information from two data tables:

A locations data table. Each item in this data table defines a location where a pie chart is displayed. The current data table (where the view is defined), which contains the sizes of the pies for each data item as column values.

A set of properties of the current table is used to define pie sizes on all pie charts. For each pie and location combination there s

- Template: A view that is defined by a template that is filled with row item properties.

Overview Specifies the default data item view of Panoptes, including all fields Name

Text (required). Display name of this view.

Template A view that is defined by a template that is filled with row item properties Content

Text (required). A [handlebars](#) template(only applies if *Type* is Template).

PropertyGroup Displays all properties that are member of a specific property group GroupId

Text (required). Identifier of the property group to display(only applies if *Type* is PropertyGroup).

FieldList Displays a selection of properties for the data item Introduction

Text. A static text that will be displayed on top of this view(only applies if *Type* is FieldList).

Fields *PropertyIDList (required).* Each item in this list specifies a property ID(only applies if *Type* is FieldList).

ItemMap Displays the data item as a pin on a geographical map. Requires the presence of properties with data type GeoLongitude and GeoLatitude MapZoom

Value (required). Start zoom factor of the map (integer, minimum value of 0)(only applies if one of the following is true:(*Type* is ItemMap)(*Type* is PieChartMap)).

PieChartMap Defines a view that shows a set of pie charts on a geographic map (see [example](#)). This is achieved by combining information from two data tables:

- A locations data table. Each item in this data table defines a location where a pie chart is displayed.
- The current data table (where the view is defined), which contains the sizes of the pies for each data item as column values.

A set of properties of the current table is used to define pie sizes on all pie charts. For each pie and location combination there should be a property in the data table, containing the relative size of that specific pie PieChartSize

Value (required). Displayed size of the largest pie chart(only applies if *Type* is PieChartMap).

MapCenter *Block (required).* Specifies the map center in the start view(only applies if *Type* is PieChartMap). The block can contain the following keys:

Longitude *Value (required).* Geographic longitude.

Latitude *Value (required).* Geographic latitude.

DataType *Text (required).* Type of values used to create the pie chart(only applies if *Type* is PieChartMap). Possible values:

- Fraction:.

PositionOffsetFraction *Value (required).* An offset between the pie chart location and the actual chart, used to achieve a nice (ideally non-overlapping) view(only applies if *Type* is PieChartMap).

LocationDataTable *Text (required).* ID of the data table containing the locations (this table should have properties with GeoLongitude and GeoLatitude data types)(only applies if *Type* is PieChartMap).

LocationSizeProperty *Text (required).* Property ID of the locations data table containing the size of the pie chart(only applies if *Type* is PieChartMap).

LocationNameProperty *Text (required).* Property ID of the locations data table containing the name of the pie chart(only applies if *Type* is PieChartMap).

ComponentColumns *List (required).* Enumerates all the pies displayed on the pie charts, and binds them to properties of this data table (one for each combination of component x location)(only applies if *Type* is PieChartMap). The block can contain the following keys:

Pattern *Text (required)*. Property ID of the column providing the data. NOTE: the token {locid} will be replaced by the primary key value of the records in the locations data table.

Name *Text (required)*. Display name of the pie.

Color *Text (required)*. Color of the pie. Format: `rgb(r,g,b)`.

ResidualFractionName *Text*. Name of the pie representing residual fraction (only applicable if the fractions do not sum up to 1)(only applies if *Type* is PieChartMap).

2D Datatable settings

This [YAML](#) file contains settings for a *2D data table*. See also:

- [Data import settings](#)
- [2D data table source files](#)
- [Example file](#)

Possible keys

NameSingle *Text (required)*. Display name referring to data of an individual cell (single, without starting capital).

NamePlural *Text (required)*. Display name referring to data of several cells (plural, without starting capital).

Description *Text*. Default:. A short description of this 2D data table. Note: this text may contain documentation links (see [Documentation source files](#)).

ColumnDataTable *Text (required)*. Identifier of the (1D) data table defining the columns of the matrix (In case of genotype data: the variants). This links the 2D data table to the 1D data table containing the column information.

ColumnIndexField *Text (required)*. The property ID in the `ColumnDataTable` data table that maps into the `ColumnIndexArray` array in the HDF5 source file. `ColumnIndexField` and `ColumnIndexArray` together establish the link between the column data table values, and the data present in the HDF5 source file. Alternatively `ColumnIndexArray` can be omitted implying that the columns in HDF5 are in the same order as `ColumnIndexField` sorted. Note that “AutoKey” can be used if your rows do not have Unique IDs.

ColumnIndexArray *Text*. 1D Array in the HDF5 source file that gives the value of `ColumnIndexField` for each column. If this is omitted then it is assumed that the HDF5 columns are in the same order as the `ColumnDataTable` data table, sorted by the `ColumnIndexField` property.

RowDataTable *Text (required)*. Identifier of the (1D) data table defining the rows of the matrix (in case of genotype data: the samples). This links the 2D data table to the 1D data table containing the row information.

RowIndexField *Text (required)*. The property ID in the `RowDataTable` data table that maps into `RowIndexArray` array in the HDF5 source file. `RowIndexField` and `RowIndexArray` together establish the link between the row data table values, and the data present in the HDF5 source file. Alternatively `RowIndexArray` can be omitted implying that the rows in HDF5 are in the same order as `RowIndexField` sorted. Note that “AutoKey” can be used if your rows do not have Unique IDs.

RowIndexArray *Text*. 1D Array in the HDF5 source file that gives the value of `RowIndexField` for each row. If this is omitted then it is assumed that the HDF5 columns are in the same order as the `RowDataTable` data table, sorted by the `RowIndexField` property.

FirstArrayDimension *Text*. Either ‘row’ or ‘column’ to indicate the first dimension in the HDF5 array. ‘column’ will generally perform better. Possible values:

- `row:.`
- `column:.`

SymlinkData *Boolean*. Default:False. If true then the HDF5 source file will not be copied but only symlinked. Note that if your HDF5 doesn't have small enough chunking (max few MB per chunk) then performance will suffer. The default of False copies and rechunks the HDF5.

ShowInGenomeBrowser *Block*. If this key is present, the data will be visualised as a channel in the genome browser. This requires that data table used as `ColumnDataTable` is defined as "IsPositionOnGenome" (see [Data table settings](#)) This key contains the following subkeys, Either 'Call' or 'AlleleDepth' or both must be present. The block can contain the following keys:

Call *PropertyID*. Reference to the 2D data table property that contains call information.

AlleleDepth *PropertyID*. Reference to the 2D data table property that contains depth information.

ExtraProperties *PropertyIDList*. A list of the extra 2D data table properties that are displayed in the genotype channel. This will populate options for alpha and height control.

GenomeMaxViewportSizeX *Value*. Maximum size of the genome browser viewport (in bp) for which genotype calls will be displayed.

Properties *List (required)*. Contains a list of all properties defined for each cell of the 2D data table. The block can contain the following keys:

Id *Text (required)*. Identifier of the property, and name of the dataset in the HDF5 source file.

Name *Text*. Display name of the property.

Description *Text*. Short description of this property.

MinVal *Value*. For continuous properties the lower level at which values will be clipped on display.

MaxVal *Value*. For continuous properties the upper level at which values will be clipped on display.

Workspace settings

This [YAML](#) file contains settings for a [workspace](#). See also:

- [Data import settings](#)
- [Add a new workspace to a dataset](#)
- [Example file](#)

Possible keys

Name *Text (required)*. Display name of the workspace.

Reference genome settings

This [YAML](#) file contains settings for the [reference genome](#). See also:

- [Data import settings](#)
- [Add reference genome annotation](#)
- [Add reference genome sequence](#)
- [Example file](#)

Possible keys

GenomeBrowserDescr *Text*. Descriptive text that will be displayed in the genome browser section of the main page.

AnnotMaxViewPortSize *Value*. Maximum viewport (in bp) the genome browser can have in order to show the genome annotation track.

RefSequenceSumm *Boolean*. If set, a summary track displaying the reference sequence will be included in the genome browser.

Annotation *Block*. Directives for parsing the annotation file (annotation.gff). The block can contain the following keys:

Format

Text. **File format. Possible values** GFF = Version 3 GFF file GTF = Version 2 GTF file

GeneFeature *Text or List*. Feature id(s) used to identify genes.

Example: [gene, pseudogene].

ExonFeature *Text or List*. Feature id(s) used to identify exons.

GeneNameAttribute *Text*. Attribute id used to identify gene names.

GeneNameSetAttribute *Text or List*. Attribute id(s) used to identify gene name sets.

Example: [Name, Alias].

GeneDescriptionAttribute *Text or List*. Attribute id(s) used to identify gene descriptions.

ExternalGeneLinks *List*. Each item in the list specifies a link for a gene to an external url. These links will show up as buttons in the gene popup window. The block can contain the following keys:

Url *Text (required)*. Url for this link. This may include a token {Id} to refer to the unique gene identifier. Example: <https://www.google.co.uk/search?q={Id}>.

Name *Text (required)*. Display name for this external link.

Custom data settings

This [YAML](#) file contains settings for a *custom data source*. See also:

- [Data import settings](#)
- [Add a custom data source to a workspace](#)
- [Example file](#)

Possible keys

AutoScanProperties *Boolean*. If set, Panoptes will try to automatically obtain property definitions from the TAB-delimited source data file.

PropertyGroups *List*. Each item in the list specifies a group of properties. It should contain two keys: “Id” representing a unique identifier for the group, and “Name” representing a display name. Property groups can be used to combine sets of related properties into sections in the app.

Properties *List (required)* The data table yaml should contain a key “Properties”, which contains a list of descriptions for all columns used in the app for this custom data table. See [Datatable property settings](#) for an overview of the keys that can be used for each individual item in this list.

DataItemViews *List*. Definitions of custom views that will appear in the popup for an individual datatable item. The views defined at the level of this custom data source will be added to the standard data item popup. Each item in the list should contain the following key:

Type *Text (required)*. Identifier of the custom view type (can be `Overview`, `PropertyGroup`, `FieldList`, `ItemMap`, `PieChartMap`) See [DataItemViews settings](#) for more details about these custom views.

Valid data identifiers

Many identifiers used in the source data structures (resource identifiers, file and folder names, table column headers, etc..), are directly mapped to identifiers in the MySQL database tables. Therefore, they should be formatted as standard variable names:

- Do not contain dashes, white spaces or other special characters.
- Do not start with a number.

1.4 Source files structure

Internally, Panoptes uses a combination of a set of MySQL databases and a file structure to serve the data. Data are loaded into this system by launching an **import action** that reads the data from a **source file location** (specified by `SOURCEDATADIR` in `config.py`, see also [Server data file structure](#)).

The formatting of the source data relies a few concepts:

- It is organised in a way that closely mimicks the basic concepts of the Panoptes data structures, using nested folders to reflect the structure.
- In most cases, data are provided using simple, TAB-delimited files. Exceptions are made in those cases where a widely accepted standard format is used for a specific type of information (e.g. GFF files for genome annotations).
- YAML (<http://www.yaml.org/about.html>) structured files are used to provide the necessary metadata to interpret and parse the data in the context of Panoptes. These metadata are provided in files called `settings`.

Caution: Many identifiers used in the source data structures (folder names, table column headers, etc..), are directly mapped to identifiers in the MySQL database tables. Therefore, they should be formatted as standard variable names (e.g. do not contain dashes, white spaces or other special characters, do not start with a number, ...)

1.4.1 Dataset source files

The `config.SOURCEDATADIR` folder should contain a folder `datasets`, serving as a root for all [datasets](#) being served by the Panoptes instance.

In this folder, a subfolder should be present for each *dataset*. The folder name is used as the unique identifier of this dataset. In the *dataset* folder, a `yaml settings` file should be present, specifying the displayed name of the dataset, and an optional description (see [General dataset settings](#)).

See also:

- [Dataset](#)
- [Source files structure](#)

1.4.2 Reference genome source files

A *dataset source folder* may optionally contain a subfolder `refgenome`, describing the *reference genome* used. It can contain the following files:

- `chromosomes` (required). A list of all chromosomes identifiers, and their lengths (in MB).
- `annotation.gff` (required). The annotation of the reference genome, in GFF format.
- `refsequence.fa` (optional). The reference genome sequence, as FASTA file.
- `settings` (required, yaml formatted). Various settings concerning the reference genome (see [Reference genome settings](#)).

Summary values source files

The `refgenome` folder may contain an optional subfolder `summaryvalues`. Each subfolder in this folder represents a different (numerical) property defined over the genome that will be filter banked and can be displayed in the genome browser. The folder name serves as the identifier of the summary value. Each summary value folder should contain the following two files:

- `values`. A TAB-delimited file having three columns, and no header ([example file](#)):
 - column 1: Chromosome identifier
 - column 2: Position
 - column 3: Value
- `settings` (yaml formatted). Contains the displayed name of the summary value, and further guidelines on how to process the information ([sample numeric file](#)). ([sample categorical](#)).

See also:

- [Reference genome](#)
- [Source files structure](#)

1.4.3 Data table source files

In the *dataset source folder* folder, a subfolder `datatables` should be present. This is the root for a set of folders, each one describing an individual *data table*, with the name of the folder serving as an identifier.

In each *data table* folder, a file `data` should be present, containing a list of all the *data items* in the table. Each line consists in a set of TAB-delimited *properties*. The first line of the file serves as a header, specifying the identifiers for all *properties* ([example file](#)).

In addition, a yaml `settings` file should be present in the *data table* folder. This file can contain a number of settings, both at the level of the *data table*, as at the level of individual *properties* (see [Data table settings](#)).

See also:

- [Data table](#)
- [Source files structure](#)

1.4.4 2D data table source files

In the *dataset source folder*, a subfolder `2D_datatables` should be present. This is the root for a set of folders, each one describing an individual *2D data table*, with the name of the folder serving as an identifier.

In each *2D data table* folder, a file `data.hdf5` should be present, containing the arrays of properties. ([example file](#)).

In addition, a `yaml settings` file should be present in the *2D data table* folder (see [2D Datatable settings](#)).

HDF5 source file structure

The source file `data.hdf5` should be structured according to the [HDF5 standard](#), and may contain the following arrays, which must be contained in the root of the HDF5 file:

Properties arrays One or more arrays specifying properties of the 2D data table. Note that these arrays can be 3D but the first two dimensions should be row and column.

Column index 1D array A 1D array listing the identifiers of all columns, in the order they are used in the properties matrices.

Row index 1D array A 1D array listing the identifiers of all rows, in the order they are used in the properties matrices.

Only scalar builtin dtypes (ie not structured with fields or user-defined) or strings currently permitted for HDF5 arrays.

Example python HDF5 creation code:

```
import h5py
outfile = h5py.File(filename, 'w', libver='latest')
call = outfile.create_dataset("call", (1000,10,2), dtype='i1')
call[:, :, :] = my_array_of_calls
allele_depth = outfile.create_dataset("allele_depth", (1000,10,3), dtype='i2')
allele_depth[:, :, :] = my_array_depth
quality = outfile.create_dataset("quality", (1000,10), dtype='i4')
quality[:, :] = my_array_of_quality
outfile.close()
```

We recommend using [VCFNP](#) for converting from VCF. See the [VCF example](#) for details of how to do this.

See also

- [Data table](#)
- [Source files structure](#)

1.4.5 Workspace source files

In the *dataset source folder*, a subfolder `workspaces` should be present. This is the root for a set of subfolders, each one describing a *workspace* for this *dataset*. The folder name serves as identifier for the *workspace*.

In a *workspace* folder, a `yaml structured settings` file should be present, specifying the displayed name of the workspace (see [Workspace settings](#)).

In addition, a subfolder `customdata` should be present. This location is used to specify *Custom data*, which has the following basic properties:

- It only exists in the context of a specific *workspace*.
- It adds extra properties to a *data table* that already exists in the *dataset*.

- The primary key of the *data table* (as defined in the settings) is used to link the custom properties to the original table.

See also:

- [Workspace](#)
- [Source files structure](#)

1.4.6 Custom data source files

The `customdata` folder in a *workspace source folder* should have a subfolder for each *data table* it defines data for, and the folder name should be the *data table* identifier. In this data table - specific folder, a number of subfolder can be defined, each one specifying an individual *custom data source*. Such a subfolder should contain two files:

- `data`. TAB-delimited file containing the custom property values ([example file](#)).
- `settings`. (yaml formatted). Specifies how the custom data should be interpreted (see [Custom data settings](#)).

See also:

- [Custom data](#)
- [Source files structure](#)

1.4.7 Documentation source files


A *dataset source folder* may optionally contain a subfolder `doc`, containing html files that can be displayed in Panoptes' internal documentation viewer.

These files should follow proper XML formatting, and contain a `<body>` element. The html may contain hyperlinks to other documentation files in the same source directory, or to external links.

These documents may be referred to in other components of the source data, such as descriptions of the dataset or data tables. Referring happens through a hyperlink with the structure `hyperlink display name`, with `[docid]` the file name of the document file *without the .html extension*.

On the deployment, this will render as a hyperlink that leads to an in-app popup showing the documentation in the source file.

1.5 Data import settings

Most source data resources have settings that are specified through a [YAML](#) definition file. Using the admin web frontend, these settings can be edited by clicking the  icon, left of the label that identifies a source data resource.

Note: The sample dataset [Samples_and_Variants](#) contains settings files that are fully commented, and can serve as a starting point to explore the possible options. There is also a [VCF example](#) which shows data imported from VCF. Additional comments are provided in other sample datasets as well, wherever concepts are introduced that are not present in this dataset.

1.5.1 General dataset settings

This **YAML** file contains settings for a *dataset*. See also:

- [Data import settings](#)
- [Creating a new dataset](#)
- [Example file](#)

Possible keys

Name *Text (required)*. The visible name of the dataset, as it appears on the intro page.

NameBanner *Text*. Visible name of the dataset, as it appears on the top banner of the app. Note: this text may contain html markup.

Description *Text*. A description of the dataset that appears on the start page. Note: this text may contain html markup, and documentation links (see [Documentation source files](#)). A longer description can be split over several lines by writing a `>` sign on the key line, and indent subsequent lines:

```
Description: >
  This web application provides an interactive view
  on the data ...
```

DataTables *List*. A list of the data table identifiers in the dataset. These names should correspond to directory names in the *datatables* source directory (see [Data table source files](#)). This can be included in the settings in order to provide an explicit ordering of the data tables in the app. If this key is not provided, a default ordering will be used.

2D_DataTables *List*. List the 2D data tables that should be exposed in the app.

IntroRightPanelFrac *Value*. Controls the proportion of left and right columns on the start page. If set to zero, the right column will be absent.

IntroSections *List*. Enumerates sections on the intro page that can contain quick start buttons to specific views in the app. Buttons can be added to these sections by (1) clicking on the “Get Link” button in the top right corner of the app, (2) clicking on one of the “Add to start page” options, and (3) entering the right section id in the “Section” edit box. Similarly, a button displaying a plot can be created by clicking the link button in the plot popup. The block can contain the following keys:

Id *Text*. Unique identifier of the section.

Name *Text*. Displayed title.

Content *Text*. Intro text of the section, appearing above the buttons. This text can be HTML formatted.

RightPanel *Boolean*. If set, the section will appear in the right column, replacing the default content of this column.

GoogleAnalyticsId *Text*. .

1.5.2 Data table settings

This **YAML** file contains settings for a *data table*. See also:

- [Data import settings](#)
- [Add a new data table to a dataset](#)

- [Example file](#)

Possible keys

NameSingle *Text (required)*. Display name referring to a single table item (single, without starting capital).

NamePlural *Text (required)*. Display name referring to several table items (plural, without starting capital).

Description *Text*. Default:.. A short description of this data table. This text will appear on the intro page, and on the table view page of this data table. Note: this text may contain documentation links (see [Documentation source files](#)).

Icon *Text*. Specifies an icon that will be associated with the data table. The icon name can be chosen from the list specified in <http://fontawesome.github.io/Font-Awesome/icons/>.

IsHidden *Boolean*. If set to true, the data table will not be displayed as a standalone entity (i.e. not mentioned on the intro page and no tab).

PrimKey *PropertyID (required)*. The primary key *property ID* for this table. A data item *property* is a column in the TAB-delimited source file data, and the *ID* corresponds to the column header. The primary key should refer to a column containing a unique value for each record in the table. Optionally, this parameter can be set to 'AutoKey' to instruct the software to automatically generate a primary key.

ItemTitle *Text*. A [handlebars](#) template. Defaults to the primary key. The rendered template will be used when a data item title is needed.

SortDefault *PropertyID*. Specifies the property ID (i.e. column name in the data source file) used as the default sort field..

CacheWorkspaceData *Boolean*. If set, a materialised table will be created in the relational database for this data in each workspace. For large data tables (>1M records), this option is faster than the standard option, which uses a JOIN statement.

MaxCountQueryRecords *Value*. Default:200000. Defines the maximum number of records that will be downloaded to the client. This limit influences views that display individual data items, such as scatter plots and geographical map views. If not specified, this defaults to 200,000.

MaxCountQueryAggregated *Value*. Default:1000000. Defines the maximum number of records that will be queried on the server for views that present data items in an aggregated way, such as histograms and bar graphs. If not specified, this defaults to 1,000,000.

FetchRecordCount *Boolean*. Default:False. .

QuickFindFields *PropertyIDs*. The list of properties will be used by some tools in the software that allow the user to quickly find a (set of) item(s).

ColumnIndexField *Text*. When this table is linked to a 2D data table setting this value to the same as that in the 2D settings provides a performance improvement for large data sets.

DisableSubsets *Boolean*. If set, there will be no subsets options for this data table.

DisablePlots *Boolean*. If set, there will be no options to create plots for this data table.

DisableNotes *Boolean*. If set, it will not be possible to define notes for items in this data table.

PropertyGroups *List*. Each item in the list specifies a group of properties. Property groups are used to combine sets of related properties into logical sections in the app. The block can contain the following keys:

Id *Text (required)*. a unique identifier for the group.

Name *Text (required)*. a display name.

AutoScanProperties - deprecated - please use scripts/mksettings.sh to generate a skeleton settings.gen file and use that to create a settings file .._Properties: Properties

List (required). Each list item defines a *property*, linked to a column in the TAB-delimited source file data. See *Datatable property settings* settings for an overview of the keys that can be used for each property in this list.

DataItemViews *List.* Definitions of custom views that will appear in the popup for an individual data table item. The block can contain the following keys:

Type *Text (required).* Identifier of the custom view type

(can be Overview, PropertyGroup, FieldList, ItemMap, PieChartMap) See *DataItemViews settings* for more details about defining custom data item views.

ExternalLinks *List.* Each item in the list specifies a link for a data item to an external url. These links show up in the app as buttons in the data item popup window. The block can contain the following keys:

Url

Text (required). Url for this link. This may include tokens property ID's between curly braces.

These tokens will be expanded to their actual content for a specific data item. Example:

`http://maps.google.com/maps?q={Latitude},{Longitude}`.

Name *Text (required).* Display name for this external link.

ListView *Boolean.* Default:False. Replaces the normal table view with a list view, showing rows on left and a single selected row on the right.

IsPositionOnGenome *Boolean.* Default:False. Instructs Panoptes that records in this data table should be interpreted as genomic positions. In this case, the *Chromosome* and *Position* keys should be defined.

IsRegionOnGenome *Boolean.* Default:False. Instructs Panoptes that records in this datatable should be interpreted as genomic regions. In this case, the *Chromosome*, *RegionStart* and *RegionStop* keys should be defined.

BrowserTrackHeightFactor *Value.* Specifies a relative size factor for the genome browser track height (only applicable if *IsPositionOnGenome* or *IsRegionOnGenome* is set).

Chromosome *PropertyID.* Specifies the table column ID that contains the chromosome (only to be used if *IsPositionOnGenome* or *IsRegionOnGenome* is set). Note that the values in this column should correspond to the content of the chromosomes source file (see *Reference genome source files*).

Position *PropertyID.* Specifies the table column ID that contains the position on the chromosome (only to be used if *IsPositionOnGenome* is set).

RegionStart *PropertyID.* Specifies the table column ID that contains the start position of the region (only to be used if *IsRegionOnGenome* is set).

RegionStop *PropertyID.* Specifies the table column ID that contains the end position of the region (only to be used if *IsRegionOnGenome* is set).

GenomeMaxViewportSizeX *Value.* Specifies the maximum genome browser viewport size (in bp) for which individual data points from this table will be displayed in the tracks. (only to be used if *IsPositionOnGenome* or *IsRegionOnGenome* is set).

BrowserDefaultVisible *Boolean.* For genomic regions: specifies the default visibility status of this data table in the genome browser (only to be used if *IsRegionOnGenome* is set). Note that, for genomic position, default visibility is specified on a per-property basis.

AllowSubSampling *Boolean.* Default:False. .

MaxTableSize *Value.* Default:None. .

BrowserDefaultLabel *PropertyID*. Specifies the default label that is used in the genome browser, used for genomic regions. None indicates that no label is displayed by default.

TableBasedSummaryValues *List*. Declares that numerical genome values for are available for each item in the table. Panoptes will process these using the multiresolution filterbanking, and the user can display these as tracks in the genome browser. A typical use case is if the data table contains samples that were sequenced, and there is coverage data available

Approach 1

There should be a subdirectory named after the identifier of this track in the data table source data folder. For each data item, this directory should contain a data file with the name equal to the primary key (see [example](#)). The input files should not contain a header row

The Id is the identifier of this set of per-data-item genomic values i.e. the name of the subdirectory

Approach 2

This approach is more like the way the table based data files are processed. In this case multiple tracks can be stored in the same input file. The Id corresponds to the column name instead of the directory name with the directory details given in the FilePattern expression The name is the first match in the FilePattern expression

.

The block can contain the following keys:

Id *Text (required)*. Identifier of this set of per-data-item genomic values - name of subdirectory or Identifier of this set of per-data-item genomic values - name of the column in the matching files.

FilePattern *Text*. A glob (regular expression) containing a relative path to the file(s).

Name *Text (required)*. Display name of the property.

MinVal *Value (required)*. Default:0. Value used for lower extent of scales.

MaxVal *Value (required)*. Value used for upper extent of scales.

BlockSizeMin *Value (required)*. Default:1. Minimum block size used by the multiresolution summariser (in bp).

BlockSizeMax *Value (required)*. Maximum block size used by the multiresolution summariser (in bp).

ChannelColor *Text*. Colour used to display these tracks as a genome browser track. Formatted as "rgb(r, g, b)".

Datatable property settings

An overview of the possible keys than can be defined for an individual property in the *Properties* block of the data table settings.

Id *Text (required)*. Identifier of the property, equal to the corresponding column header in the TAB-delimited source file data.

DataType *Text (required)*. Data type of the values in the property. Possible values:

- Text: text strings.
- Value: numerical values (integer of decimal; the distinction is made by the key *DecimDigits*). Absent values can be coded by an empty string, "NA", "None", "NULL", "null", "inf" or "-".
- HighPrecisionValue: same as Value, with higher precision.

- **Boolean:** Yes/No binary states. Possible values according to YAML: y|Y|yes|Yes|YES|n|N|no|No|NO|true|True|TRUE|false|False|FALSE|on|On|ON|off|Off|OFF. Absent values are coded by an empty string.
- **GeoLongitude:** longitude part of a geographical coordinates (in decimal degrees). Absent values are coded by an empty string.
- **GeoLatitude:** latitude part of a geographical coordinates (in decimal degrees). Absent values are coded by an empty string.
- **Date:** calendar dates, ISO formatted (i.e. YYYY-MM-DD). Absent values are coded by an empty string.

Name *Text (required).* Display name of the property.

Description *Text.* Brief description of the property. This will appear in hover tool tips and in the popup box if a user clicks a property info button.

GroupId *Text.* Id of the Property group this property belongs to.

ExternalUrl *Text.* A url that should be opened when the user clicks on a value of this property. The url should be formatted as a template, with {value} interpolated to the property value. For example: `http://www.ebi.ac.uk/ena/data/view/{value}`.

IsCategorical *Boolean.* Instructs Panoptes to treat the property as a categorical variable. For example, a combo box with the possible states is automatically shown in queries for this property. Categorical properties are automatically indexed.

CategoryColors *Block.* Specifies display colours for the categorical states of this property. Each key in the block links a possible value of the property to a color (example: `Accepted: rgb(0,192,0)`). The special value `_other_` can be used to specify a color for all other property values that are not listed explicitly.

MaxColumnWidth *Value.* Specifies the maximum width (in pixels) used for the column representing this property in a table view. Longer text will be abbreviated with ellipsis.

BarWidth *Value.* Draws a bar in the background of the table, indicating the value. Requires *MinVal* & *MaxVal* to be defined(only applies if *DataType* is ['Value', 'HighPrecisionValue']).

MinVal *Value.* Default:0. For *Value* types, upper extent of scale(only applies if *DataType* is ['Value', 'HighPrecisionValue']).

MaxVal *Value.* Default:1.0. For *Value* types, lower extent of scale(only applies if *DataType* is ['Value', 'HighPrecisionValue']).

MaxLen *Value.* Default:0. If present used to specify the maximum size of the database column - otherwise it is calculated.

DecimDigits *Value.* For *Value* types, specifies the number of decimal digits used to display the value(only applies if *DataType* is ['Value', 'HighPrecisionValue']).

MaxDecimDigits *Value.* (Not currently used) For *Value* types, specifies the number of decimal digits used to store the value in the database(only applies if *DataType* is ['Value', 'HighPrecisionValue']).

Index *Boolean.* Default:False. If set, instructs Panoptes to create an index for this property in the relational database. For large datasets, this massively speeds up queries and sort commands based on this property.

Search *Text.* Default:None. Indicates that this field can be used for text search in the find data item wizard. Possible values:

- None: .
- Match: only exact matched are searched for.
- StartPattern: searches all text that starts with the string typed by the user.
- Pattern: searches all text that contains the string typed by the user.

Relation *Block*. Defines a many-to-one foreign relation to a parent data table. The parent table should contain a property with the same name as the primary key property in the child table. The block can contain the following keys:

TableId *DatatableID (required)*. Data table ID of the relation parent table.

ForwardName *Text (required)*. Default:belongs to. Display name of the relation from child to parent.

ReverseName *Text (required)*. Default:has. Display name of the relation from parent to child.

ReadData *Boolean*. Default:True. If set to false, this property will not be imported from the TAB-delimited source file.

CanUpdate *Boolean*. Default:False. If set to true, this property can be modified by the user. (*NOTE: under construction*).

ShowInTable *Boolean*. If set, this property will appear by default in data table grids in the application.

ShowInBrowser *Boolean*. If set, this property will automatically appear as a track in the genome browser (only applies if *IsPositionOnGenome* is specified in database settings).

BrowserDefaultVisible *Boolean*. Indicates that the track will activated by default in the genome browser (only applies if *ShowInBrowser* is True).

BrowserShowOnTop *Boolean*. Indicates that the track will be shown in the top (non-scrolling) area of the genome browser. In this case, it will always be visible (only applies if *ShowInBrowser* is True).

ChannelName

Text. **Name of the genome browser track this property will be displayed in.** Properties sharing the same track name will be displayed in overlay (only applies if *ShowInBrowser* is True).

ChannelColor *Text*. Colour used to display this property in the genome browser. Formatted as "rgb(r,g,b)" (only applies if *ShowInBrowser* is True).

ConnectLines *Boolean*. Indicate that the points will be connected with lines in the genome browser (only applies if *ShowInBrowser* is True).

DefaultVisible *Boolean*. Default:True. .

Order *Value*. Default:-1. Only used for reference genome tracks.

SummaryValues *Block*. Instructs Panoptes to apply a multiresolution summary algorithm for fast display of this property in the genome browser at any zoom level(only applies if *ShowInBrowser* is True). The block can contain the following keys:

BlockSizeMin *Value*. Default:1. Minimum summary block size (in bp).

BlockSizeMax *Value (required)*. Maximum summary block size (in bp).

ChannelColor *Text*. Colour of the channel, for numerical channels. Formatted as "rgb(r,g,b)".

MaxDensity *Value*. For categorical properties this set the scale for the summary track in rows/bp. Defaults to 1/bp.

DatalItemViews settings

The key *Type* for member of the data table settings key *DatalItemViews* can have the following values:

Type *Text (required)*. Identifier of the custom view type (can be Overview, PropertyGroup, FieldList, ItemMap, PieChartMap) See *DatalItemViews* settings for more details about defining custom data item views. Possible values:

- **Overview:** Specifies the default data item view of Panoptes, including all fields.
- **PropertyGroup:** Displays all properties that are member of a specific property group.
- **FieldList:** Displays a selection of properties for the data item.
- **ItemMap:** Displays the data item as a pin on a geographical map. Requires the presence of properties with data type GeoLongitude and GeoLatitude.
- **PieChartMap:** Defines a view that shows a set of pie charts on a geographic map (see example). This is achieved by combining information from two data tables:

A locations data table. Each item in this data table defines a location where a pie chart is displayed. The current data table (where the view is defined), which contains the sizes of the pies for each data item as column values.

A set of properties of the current table is used to define pie sizes on all pie charts. For each pie and location combination there s

- **Template:** A view that is defined by a template that is filled with row item properties.

Overview Specifies the default data item view of Panoptes, including all fields Name

Text (required). Display name of this view.

Template A view that is defined by a template that is filled with row item properties Content

Text (required). A [handlebars](#) template(only applies if *Type* is Template).

PropertyGroup Displays all properties that are member of a specific property group GroupId

Text (required). Identifier of the property group to display(only applies if *Type* is PropertyGroup).

FieldList Displays a selection of properties for the data item Introduction

Text. A static text that will be displayed on top of this view(only applies if *Type* is FieldList).

Fields *PropertyIDList (required).* Each item in this list specifies a property ID(only applies if *Type* is FieldList).

ItemMap Displays the data item as a pin on a geographical map. Requires the presence of properties with data type GeoLongitude and GeoLatitude MapZoom

Value (required). Start zoom factor of the map (integer, minimum value of 0)(only applies if one of the following is true:(*Type* is ItemMap)(*Type* is PieChartMap)).

PieChartMap Defines a view that shows a set of pie charts on a geographic map (see [example](#)). This is achieved by combining information from two data tables:

- A locations data table. Each item in this data table defines a location where a pie chart is displayed.
- The current data table (where the view is defined), which contains the sizes of the pies for each data item as column values.

A set of properties of the current table is used to define pie sizes on all pie charts. For each pie and location combination there should be a property in the data table, containing the relative size of that specific pie PieChartSize

Value (required). Displayed size of the largest pie chart(only applies if *Type* is PieChartMap).

MapCenter *Block (required)*. Specifies the map center in the start view(only applies if *Type* is PieChartMap). The block can contain the following keys:

Longitude *Value (required)*. Geographic longitude.

Latitude *Value (required)*. Geographic latitude.

DataType *Text (required)*. Type of values used to create the pie chart(only applies if *Type* is PieChartMap). Possible values:

- **Fraction**:.

PositionOffsetFraction *Value (required)*. An offset between the pie chart location and the actual chart, used to achieve a nice (ideally non-overlapping) view(only applies if *Type* is PieChartMap).

LocationDataTable *Text (required)*. ID of the data table containing the locations (this table should have properties with GeoLongitude and GeoLatitude data types)(only applies if *Type* is PieChartMap).

LocationSizeProperty *Text (required)*. Property ID of the locations data table containing the size of the pie chart(only applies if *Type* is PieChartMap).

LocationNameProperty *Text (required)*. Property ID of the locations data table containing the name of the pie chart(only applies if *Type* is PieChartMap).

ComponentColumns *List (required)*. Enumerates all the pies displayed on the pie charts, and binds them to properties of this data table (one for each combination of component x location)(only applies if *Type* is PieChartMap). The block can contain the following keys:

Pattern *Text (required)*. Property ID of the column providing the data. NOTE: the token {locid} will be replaced by the primary key value of the records in the locations data table.

Name *Text (required)*. Display name of the pie.

Color *Text (required)*. Color of the pie. Format: `rgb(r, g, b)`.

ResidualFractionName *Text*. Name of the pie representing residual fraction (only applicable if the fractions do not sum up to 1)(only applies if *Type* is PieChartMap).

1.5.3 2D Datatable settings

This [YAML](#) file contains settings for a [2D data table](#). See also:

- [Data import settings](#)
- [2D data table source files](#)
- [Example file](#)

Possible keys

NameSingle *Text (required)*. Display name referring to data of an individual cell (single, without starting capital).

NamePlural *Text (required)*. Display name referring to data of several cells (plural, without starting capital).

Description *Text*. Default:.. A short description of this 2D data table. Note: this text may contain documentation links (see [Documentation source files](#)).

ColumnDataTable *Text (required)*. Identifier of the (1D) data table defining the columns of the matrix (In case of genotype data: the variants). This links the 2D data table to the 1D data table containing the column information.

ColumnIndexField *Text (required)*. The property ID in the `ColumnDataTable` data table that maps into the `ColumnIndexArray` array in the HDF5 source file. `ColumnIndexField` and `ColumnIndexArray` together establish the link between the column data table values, and the data present in the HDF5 source file. Alternatively `ColumnIndexArray` can be omitted implying that the columns in HDF5 are in the same order as `ColumnIndexField` sorted. Note that “AutoKey” can be used if your rows do not have Unique IDs.

ColumnIndexArray *Text*. 1D Array in the HDF5 source file that gives the value of `ColumnIndexField` for each column. If this is omitted then it is assumed that the HDF5 columns are in the same order as the `ColumnDataTable` data table, sorted by the `ColumnIndexField` property.

RowDataTable *Text (required)*. Identifier of the (1D) data table defining the rows of the matrix (in case of genotype data: the samples). This links the 2D data table to the 1D data table containing the row information.

RowIndexField *Text (required)*. The property ID in the `RowDataTable` data table that maps into `RowIndexArray` array in the HDF5 source file. `RowIndexField` and `RowIndexArray` together establish the link between the row data table values, and the data present in the HDF5 source file. Alternatively `RowIndexArray` can be omitted implying that the rows in HDF5 are in the same order as `RowIndexField` sorted. Note that “AutoKey” can be used if your rows do not have Unique IDs.

RowIndexArray *Text*. 1D Array in the HDF5 source file that gives the value of `RowIndexField` for each row. If this is omitted then it is assumed that the HDF5 columns are in the same order as the `RowDataTable` data table, sorted by the `RowIndexField` property.

FirstArrayDimension *Text*. Either ‘row’ or ‘column’ to indicate the first dimension in the HDF5 array. ‘column’ will generally perform better. Possible values:

- row: .
- column: .

SymlinkData *Boolean*. Default:False. If true then the HDF5 source file will not be copied but only symlinked. Note that if your HDF5 doesn’t have small enough chunking (max few MB per chunk) then performance will suffer. The default of False copies and rechunks the HDF5.

ShowInGenomeBrowser *Block*. If this key is present, the data will be visualised as a channel in the genome browser. This requires that data table used as `ColumnDataTable` is defined as “IsPositionOnGenome” (see [Data table settings](#)) This key contains the following subkeys, Either ‘Call’ or ‘AlleleDepth’ or both must be present. The block can contain the following keys:

Call *PropertyID*. Reference to the 2D data table property that contains call information.

AlleleDepth *PropertyID*. Reference to the 2D data table property that contains depth information.

ExtraProperties *PropertyIDList*. A list of the extra 2D data table properties that are displayed in the genotype channel. This will populate options for alpha and height control.

GenomeMaxViewportSizeX *Value*. Maximum size of the genome browser viewport (in bp) for which genotype calls will be displayed.

Properties *List (required)*. Contains a list of all properties defined for each cell of the 2D data table. The block can contain the following keys:

Id *Text (required)*. Identifier of the property, and name of the dataset in the HDF5 source file.

Name *Text*. Display name of the property.

Description *Text*. Short description of this property.

MinVal *Value*. For continuous properties the lower level at which values will be clipped on display.

MaxVal *Value*. For continuous properties the upper level at which values will be clipped on display.

1.5.4 Workspace settings

This **YAML** file contains settings for a *workspace*. See also:

- *Data import settings*
- *Add a new workspace to a dataset*
- *Example file*

Possible keys

Name *Text (required)*. Display name of the workspace.

1.5.5 Reference genome settings

This **YAML** file contains settings for the *reference genome*. See also:

- *Data import settings*
- *Add reference genome annotation*
- *Add reference genome sequence*
- *Example file*

Possible keys

GenomeBrowserDescr *Text*. Descriptive text that will be displayed in the genome browser section of the main page.

AnnotMaxViewportSize *Value*. Maximum viewport (in bp) the genome browser can have in order to show the genome annotation track.

RefSequenceSumm *Boolean*. If set, a summary track displaying the reference sequence will be included in the genome browser.

Annotation *Block*. Directives for parsing the annotation file (annotation.gff). The block can contain the following keys:

Format

Text. File format. Possible values GFF = Version 3 GFF file GTF = Version 2 GTF file

.

GeneFeature *Text or List*. Feature id(s) used to identify genes.

Example: [gene, pseudogene].

ExonFeature *Text or List*. Feature id(s) used to identify exons.

GeneNameAttribute *Text*. Attribute id used to identify gene names.

GeneNameSetAttribute *Text or List*. Attribute id(s) used to identify gene name sets.

Example: [Name, Alias].

GeneDescriptionAttribute *Text or List*. Attribute id(s) used to identify gene descriptions.

ExternalGeneLinks *List*. Each item in the list specifies a link for a gene to an external url. These links will show up as buttons in the gene popup window. The block can contain the following keys:

Url *Text (required)*. Url for this link. This may include a token {Id} to refer to the unique gene identifier. Example: `https://www.google.co.uk/search?q={Id}`.

Name *Text (required)*. Display name for this external link.

1.5.6 Custom data settings

This **YAML** file contains settings for a *custom data source*. See also:

- [Data import settings](#)
- [Add a custom data source to a workspace](#)
- [Example file](#)

Possible keys

AutoScanProperties *Boolean*. If set, Panoptes will try to automatically obtain property definitions from the TAB-delimited source data file.

PropertyGroups *List*. Each item in the list specifies a group of properties. It should contain two keys: “Id” representing a unique identifier for the group, and “Name” representing a display name. Property groups can be used to combine sets of related properties into sections in the app.

Properties *List (required)* The data table yaml should contain a key “Properties”, which contains a list of descriptions for all columns used in the app for this custom data table. See [Datatable property settings](#) for an overview of the keys that can be used for each individual item in this list.

DataItemViews *List*. Definitions of custom views that will appear in the popup for an individual datatable item. The views defined at the level of this custom data source will be added to the standard data item popup. Each item in the list should contain the following key:

Type *Text (required)*. Identifier of the custom view type (can be Overview, PropertyGroup, FieldList, ItemMap, PieChartMap) See [DataItemViews settings](#) for more details about these custom views.

License

Panoptes © Copyright 2014, CGGH <info@cggh.org>

This program is free software: you can redistribute it and/or modify it under the terms of the GNU Affero General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Affero General Public License for more details.

A copy of the license is at <http://opensource.org/licenses/AGPL-3.0>