cellbrowser Documentation

Release v0.5.21+16.g5b3bae6

Maximilian Haeussler, Lucas Seninge, Nikolai Markov

May 23, 2019

Contents

The UCSC Cell Browser is a fast, lightweight viewer for single-cell data. Cells are presented along withmetadata and gene expression, with the ability to color cells by both of these attributes. Additional information, such as cluster marker genes and selected dataset-relevant genes, can also be displayed using the Cell Browser.

There is a UCSC Cell Browser website available at http://cells.ucsc.edu, which includes a handful of datasets from repositories like HCA, CIRM, and GEO as well as user contributed ones. We are happy to add your favorite dataset to this, you will just need to send us the files or a link to where we can download them to cells@ucsc.edu.

The documentation on this website describes how you can create a Cell Browser for your own data and make it available through your own web server.

The UCSC cell browser is funded by grants from the California Institute for Regenerative Medicine and the Chan-Zuckerberg Initiative.

To report issues or view the source code, see GitHub.

This is early research software. You are likely to run into bugs. If you do run into any trouble, please open a Github issue or email us at cells@ucsc.edu, we can usually fix them quickly.

CHAPTER 1

Installation

1.1 Installation with pip

To install the Cell Browser using pip, you will need Python2.5+ or Python3+ and pip. With these setup, on a Mac or any Linux system, simply run:

```
sudo pip install cellbrowser
```

On Linux, if you are not allowed to run the sudo command, you can install the Cell Browser into your user home directory:

```
pip install --user cellbrowser
export PATH=$PATH:~/.local/bin
```

You can add the second command to your ~/.profile or ~/.bashrc, this will allow you to run the Cell Browser commands without having to specify their location.

On OSX, if running `sudo pip` outputs *command not found*, you will need to setup pip first by running:

```
sudo easy_install pip
```

1.2 Installation with conda

If you would prefer to install the Cell Browser through bioconda, you can run:

```
conda install -c bioconda ucsc-cell-browser
```

There should be conda versions for release 0.4.23 onwards. The conda version is managed by Pablo Moreno at the EBI and is often a few releases behind. Please indicate in any bug reports if you used conda to install.

1.3 Installation with git clone

Pip is not required to install the Cell Browser. As an alternative to pip or conda, you can also git clone the repo and run the command line scripts under cellbrowser/src:

```
git clone https://github.com/maximilianh/cellBrowser.git --depth=10
cd cellBrowser/src
```

1.4 Installation with just wget or curl

You don't use pip, conda or git? You can also download the current master branch:

```
wget https://github.com/maximilianh/cellBrowser/archive/master.zip
unzip master.zip
cellBrowser-master/src/cbBuild
```

CHAPTER 2

Basic usage

2.1 Overview

The UCSC Cell Browser tool set consists of a number of different scripts to help you set up your own. The primiary utility being the Python script `cbBuild` that will import a set of existing single-cell data from a directory of tab-separated files and configuration files to generate a directory of html, json, and css files that can be viewed on the web. The rest of the utilities will produce output than can be fed directly into `cbBuild`.

The utilities `cbSeurat` and `cbScanpy` run a very basic single-cell pipeline on your expression matrix and will output all the files needed to create a cell browser visualization. The `cbImport*` (`cbImportCellranger`, `cbImportScanpy`, etc.) tools convert files produced by Cellranger, Seurat, and Scanpy into a set of files that you can create a Cell Browser visualization from. Both the pipeline and import tools are covered in more detail under their respective sections (With Scanpy, With Seurat, and With Cellranger). There is also a collection of small tools (`cbTool`) to combine cell annotation files from different pipelines or convert expression matrices.

2.2 Using cbBuild to set up a Cell Browser

The main utility for building your own cell browser is `cbBuild`. It takes in a gene expression matrix and a set related files and converts them JSON and binary files outputting them to directory which can be put onto a web server or used with the built-in webserver. At this time, there is no backend server needed for a cell browser. You can place the output of `cbBuild` on any static web server at your University or the ones you can rent from companies will do.

After the installation, you should be able to run the cbBuild command and see the usage message:

```
cbBuild
```

2.2.1 Example Minimal Cell Browser

Below are some instructions to set up a cell browser using a small example dataset based on data from Nowakowski et al. 2017. and the cortex-dev dataset on cells.ucsc.edu. The expression matrix only includes 100 genes, but it does show off many of the features of the cell browser.

First, download and extract it to the directory `mini` with:

```
curl -s https://cells.ucsc.edu/downloads/samples/mini.tgz | tar xvz
```

Next, build a browser consisting of html and other files into the directory ~/public_html/cells/ and serve that directory on port 8888:

```
cd mini
cbBuild -o ~/public_html/cells/ -p 8888
```

Lastly, point your web browser to `http://localhost:8888` to view your minimal cell browser. If you're running this on a server and not your own computer, replace localhost with the address of your server. To stop the cbBuild web server, press Ctrl-C. To keep it running in the background, press Ctrl-Z and put it into the background with `bg`. If you have stopped the web server, you can always run the same `cbBuild` command to restart it. Restarting the web server will not re-export the entire expression matrix again if there is already one under `~/public_html/cells/sample`.

The optional to specify the port, `-p PORT`, is optional. If you only want to build html files and serve them with your own web server, do not specify this option and `cbBuild` will only build the output files, but won't start a web server.

The example cellbrowser.conf explains all the various settings that are available in this config file. Things you can change include the colors for different metadata attributes, explain cluster acronyms used in your cluster names, add file names, add alternative dimensionality reduction layouts, add more marker gene tables, and more.

One of the most important settings in cellbrowser.conf is the dataset name. For example, in this 'mini' example, the dataset name is 'sample'. When you run `cbBuild`, its output files will be written to ~/public_html/cells/sample. You can go to another directory with a different cellbrowser.conf file and a different dataset name, and if you run the same cbBuild command as above, the cell browser output files will be copied into a new subdirectory within ~/public_html/cells/. A single cbBuild output directory can contain multiple datasets.

CHAPTER 3

Putting it onto the internet

To deploy the result of `cbBuild` onto a webserver, simply copy all files and directories in the html output directory (in the examples on this site, that's usually *~/public_html/cells*) to an empty directory on a webserver and point your web browser to it. Many universities give their members webspace, sometimes in a directory called ~/public_html or on a special server. If you do not have one, contact us or use Cyverse, Amazon S3, Goole Cloud Storage, Microsoft Azure etc. to host your files. You cannot use online backup solutions like Dropbox, Box.com, iCloud OneDrive or Google Drive, they intentionally are not webservers. You can always send the result to cells@ucsc.edu, we are happy to add it to our cell browser site at cells.ucsc.edu.

To add more datasets, go to the other data directories and run cbBuild there, with the same output directory. cbBuild will then modify the index.html in the output directory to show all datasets. Note that the directory that you provide via -o (or the CBOUT environment variable) is the html directory. The data for each individual dataset will be copied into subdirectories under this html directory, one directory per dataset.

Instead of specifying "-o" all the time, you can also add a line like this to your ~/.bashrc to point to your html directory:

```
export CBOUT=/var/www
```

Alternatively, you can create a file ~/.cellbrowser.conf and assign a value to htmlDir:

```
echo 'htmlDir = "/var/www"' >> ~/.cellbrowser.conf
```

The -p 8888 is optional. A more permanent alternative to the -p option is to run a webserver on your machine and build directly into its web directory.

On a Mac you can use the Apache that ships with OSX:

```
sudo /usr/sbin/apachectl start
sudo cbBuild -o /Library/WebServer/Documents/cells/
```

Then you should be able to access your viewer at http://localhost/cells

On Linux, you would install Apache2 (with 'sudo yum install htppd' or 'sudo apt-get install apache2') and use the directory /var/www/ instead:

```
sudo cbBuild -o /var/www/
```

We hope you do not use this software on Windows. Email cells@ucsc.edu if you have to.

CHAPTER 4

With text files

The generic way to setup a Cell Browser is from .tsv or .csv text files.

Go to the directory with the expression matrix and the cell annotations. Start from a sample `cellbrowser.conf`:

```
cbBuild --init
```

The current list of all possible cellbrowser.conf statements can be found in *our Github example <https://github.com/maximilianh/cellBrowser/blob/master/src/cbPyLib/cellbrowser/sampleConfig/cellbrowser.conf>*.

Then you need at least three but ideally four data files, they can be in .tsv or .csv format:

1. The expression matrix, one row per gene, ideally gzipped. The first column must be the gene identifier or gene symbol, or ideally geneId|symbol. ENSG and ENSMUSG gene identifiers will be translated automatically to symbols. The other columns are expression values as numbers, one per cell. The number type will be auto-detected (float or int). The file must be a header line that describes the columns with the identifiers for the cells.

2. The cell annotation meta data table, one row per cell. No need to gzip this relatively small file. The first column is the name of the cell and it has to match the cell name in the expression matrix. There should be at least two columns: one with the name of the cell and one with the name of the cluster. Ideally your expression matrix is a tab-separated file and has as many cell columns as you have rows in the meta data file and they appear in the same order in both files, as cbBuild doesn't have to trim the matrix then or reorder the meta file. The meta file has a header line, the names of the columns from this line are the ones refered to in the cellbrowser.conf file.

3. The coordinates of the cells, often t-SNE or UMAP coordinates. This file always has three columns, (cellName, x, y). The cellName must be the same as in the expression matrix and cell annotation meta data file. You can provide multiple files in this format, if you have run multiple dimensionality reduction algorithms. You can also specify only a subset of the cells here. In this way, you can use a single dimensionality reduction algorithm, but multiple subsets of the cells, e.ge. one coord file per tissue. If R has changed your cell identifiers (adding dots), you may be able to fix them with `cbTool metaCat`.

4. The (optional) table with cluster-specific marker genes. The first column is the cluster name (from the cell annotation meta file), the second column contains the gene symbol (or Ensembl gene ID, will be mapped to symbol) and the third column is some numeric score (e.g. p-Value or FDR). You can add as many other columns as you like with additional information about this gene or run your table through cbMarkerAnnotate to add

information from various gene-centric databases to your existing table. Alternatively you can also provide the raw Seurat marker gene output. There can be multiple files with cluster-specific marker genes, e.g. in case that you are also doing differential gene expression analysis or have results from multiple algorithms. See Annotate genes on how to add link to external gene-databases (like Allan Brain Atlas or OMIM) to your marker genes.

Make sure that all your input files have Unix line endings and fix the line endings if necessary with mac2unix or dos2unix:

```
file *.txt *.csv *.tsv *.tab
```

Edit cellbrowser.conf. Enter the name of the three files with the config statements exprMatrix, meta, coordFiles. If you have a table with cluster specific genes, put that into clusterFiles. Enter the value of your cluster name field from the meta annotation file for the tags labelField and clusterField.

From the directory where your cellbrowser.conf is located, run:

```
cbBuild -o /tmp/cb -p 8888
```

Point your internet browser to the name of the server (or localhost, if you're running this on your own machine) followed by :8888, e.g. http://localhost:8888.

The output directory (/tmp/cb in the example) can hold multiple datasets. If you have a second dataset in another directory that contains cellbrowser.conf, just make sure that the other cellbrowser.conf specifies a different dataset name with *name=xxx*. Then run *cbBuild -o /tmp/cb -p 8888* in the other directory to add the second dataset to your browser output directory /tmp/cb.

CHAPTER 5

With Seurat

If you are an RStudio user and have a Seurat object, you can convert it to html directly without going to the Unix command line with the `ExportToCellbrowser()` R function. Users of large servers may prefer to import a Seurat `rds` file with the Unix command line tool `cbImportSeurat`. If you have an expression matrix and no knowledge of Seurat, you can use our default Seurat pipeline `cbSeurat` to create a Cell Browser.

5.1 Convert a Seurat2 .rds file

You can use the program `cbImportSeurat2` to convert a `rds` file to a Cell Browser. You can create an .rds file in R as described in the Seurat tutorial:

```
saveRDS(pbmc, "pbmc3k_small.rds")
```

Then, on the Unix command line, you specify the input .rds file and the output directory (the name in the cell browser defaults to the output directory name, but you can change this with -n):

```
cbImportSeurat2 -i pbmc3k_small.rds -o pbmc3kImport
```

Then go into the directory *pbmc3kImport* and run cbBuild to create the Cell Browser html files:

```
cd pbmc3kImport
cbBuild -o ~/public_html/cb
```

5.2 Convert a Seurat object from R

The function ExportToCellbrowser() is already part of Seurat 3. You can install pre-release Seurat3 like this:

```
install.packages("devtools")
devtools::install_github("satijalab/seurat", ref = "release/3.0")
```

For Seurat 2, you have to load the function with this command:

```
source("https://raw.githubusercontent.com/maximilianh/cellBrowser/master/src/cbPyLib/
↪cellbrowser/R/ExportToCellbrowser-seurat2.R")
```

You can then write a Seurat object to a directory from which you can run cbBuild:

```
ExportToCellbrowser(pbmc_small, dir="pbmcSmall", dataset.name="pbmcSmall")
```

Or immediately convert the files to html files in the directory `htdocs` and serve the result on port 8080 via http and open a web browser from R:

```
ExportToCellbrowser(pbmc_small, dir="pbmcSmall", cb.dir="htdocs", dataset.name=
↪"pbmcSmall", port=8080)
```

Writing the expression matrix is somewhat slow. If you have already exported into the same output directory before and just updated a part of the cell annotation data (e.g. clustering), you can use the argument *skip.matrix=TRUE* to save some time:

ExportToCellbrowser(pbmc_small, dir="pbmcSmall", dataset.name="pbmcSmall", skip-matrix=TRUE)

5.3 Run a basic Seurat pipeline

If you have never used Seurat before and just want to process an expression matrix as quickly as possible, this section is for you.

If you do not have R installed yet, we recommend that you install it via conda. Follow these instructions to install the miniconda installer: https://conda.io/projects/conda/en/latest/user-guide/install/index.html#regular-installation

When conda is installed, install R:

```
conda install r
```

Then, again using conda, install Seurat:

```
conda install -c bioconda r-seurat
```

To process an example dataset now, download the 10X pbmc3k expression matrix:

```
rsync -Lavzp genome-test.gi.ucsc.edu::cells/datasets/pbmc3k/ ./pbmc3k/ --progress
```

Create a default file `seurat.conf`:

```
cbSeurat --init
```

You can modify seurat.conf but the default values are good for this dataset. Now run the expression matrix *filtered_gene_bc_matrices/hg19/matrix.mtx* through Seurat like this:

```
cbSeurat -e filtered_gene_bc_matrices/hg19 --name pbmc3kSeurat -o seuratOut
```

This will create a script seuratOut/runSeurat.R, run it through Rscript and will fill the directory seuratOut/ with everything needed to create a cell browser. Now you can build your cell browser from the Seurat output:

```
cd seuratOut
cbBuild -o ~/public_html/cells
```

You can modify the file seurat.conf and rerun the cbSeurat command above.

CHAPTER 6

With Scanpy

If have an expression matrix and would like to do standard preprocessing, embedding and clustering, you can use our minimal Scanpy pipeline through the command line toold `cbScanpy`.

If you already have a .h5ad file, you can use the program `cbImportScanpy` to export the data to a directory and then create the Cell Browser html directory with the `cbBuild` command.

If you are already using Scanpy, you can convert your anndata Scanpy objects directly to Cell Browser format and start a webserver, e.g. from Jupyter, directly with the Python3 function `cellbrowser.scanpyToCellbrowser(ad, outDir, datasetname)`.

6.1 A standard Scanpy pipeline

Requirements: Python3 with Scanpy installed, see https://scanpy.readthedocs.io/en/latest/installation.html. Please make sure that you install the igraph library. It's a requirement for the most basic scanpy features, but it's not an official requirement of scanpy. The command *pip install scanpy[louvain]* will make sure that igraph is installed.

We provide a wrapper around Scanpy which runs filtering, PCA, nearest-neighbors, clustering, t-SNE and UMAP and formats them for cbBuild. An example file is on our downloads server:

```
mkdir ~/cellData
cd ~/cellData
rsync -Lavzp genome-test.gi.ucsc.edu::cells/datasets/pbmc3k/ ./pbmc3k/ --progress
cd pbmc3k
```

Write an empty **scanpy.conf**:

```
cbScanpy --init
```

Edit the *scanpy.conf* file and adapt it to your needs or just keep the default values. Then run commands like this:

```
# process the matrix and write results to scanpyout/
cbScanpy -e filtered_gene_bc_matrices/hg19/matrix.mtx -o scanpyout -n pbmc3k
```

(continues on next page)

(continued from previous page)

```
# build the html directory from scanpyout/
cd scanpyout
cbBuild -o ~/public_html/cb -p 8888
```

6.2 Convert a Scanpy .h5ad

The tool for this task is called cbImportScanpy. Specify the input file and the output directory, then run `cbBuild` on the output directory. There is an example input file in the cellbrowser github repository:

```
cd sample_data/pbmc_small
cbImportScanpy -i anndata.h5ad -o pbmc3kImportScanpy
cd pbmc3kImportScanpy
cbBuild -o ~/public_html/cb
```

6.3 Convert a Scanpy object

From Jupyter or Python3, create a data directory with the tab-sep files and a basic cellbrowser.conf:

```
import cellbrowser.cellbrowser as cb
cb.scanpyToCellbrowser(adata, "scanpyOut", "myScanpyDataset")
```

Then, build the cell browser from this output directory into a html directory:

```
cb.build("scanpyOut", "~/public_html/cells")
```

If you don't have a webserver running already, start an http server to serve this directory:

```
cb.serve("~/public_html/cells", 8888)
```

You can later stop this http server:

```
cb.stop()
```

Or from a Unix Shell, build and start the http server:

```
cd scanpyOut
cbBuild -o ~/public_html/cells/ -p 8888
```

CHAPTER 7

With CellRanger

Find the cellranger OUT directory, it contains an *analysis* directory and also a subdirectory *filtered_gene_bc_matrices*. The *OUT* directory is the one for our tool `cbImportCellranger`. The tool converts the cellranger files to tab-separated files, then you can run cbBuild on these.

As we are reading Cellranger *mtx* files, we need the scipy package (add –user if you are not admin on your machine):

```
pip install scipy
```

Let's use an example, the pbmc3k cellranger output files from the 10x website:

```
rsync -Lavzp genome-test.gi.ucsc.edu::cells/datasets/pbmc3kCellranger/ ./
↪pbmc3kCellranger/ --progress
cbImportCellranger -i pbmc3kCellranger -o cellrangerOut --name pbmc3k_cellranger
cd cellrangerOut
cbBuild -o ~/public_html/cells -p 9999
```

CHAPTER 8

Advanced Topics

For a reference of all cellbrowser.conf statements, see the example file https://github.com/maximilianh/cellBrowser/blob/master/src/cbPyLib/cellbrowser/sampleConfig/cellbrowser.conf

To update only the javascript files and re-create the index.html, you can use the command line tool cbUpgrade.

To add Google Analytics tracking to your cell browser, create a file `.cellbrowser.conf` in your home directory and add a line like this:

```
gaTag = "UA-11231232-1"
```

Then cbBuild or cbUpgrade and your index.html should contain the Google Analytics tracking code.

The html directory can be defined in all tools with the option `-o`. If that becomes cumbersome, you can also permanently set it through the environment variable CBOUT or by adding a line like this to `~/.cellbrowser.conf`:

```
htmlDir = "/data/www/cb/"
```

Your webserver should support byte-range requests. Smaller datasets work without that, but for datasets with files larger than 30MB, a warning message will be shown once. Byte-ranges are active by default in Apache but may need to be activated in nginx.

If you have meta fields with very long names, you can reduce the font size. Configure them like this:

metaOpt = {'Cluster_field' : {'fontSize': '10px'}}

In your meta.tsv, you can have URLs to images. These will be shown on mouse over in the left annotation bar.

If you set the default coloring field to 'None' (without the quotes), then there is no coloring at all when the cell browser starts.

To change the coloring/label field automatically when the user activates some coordinates (layout), use the option "colorOnMeta" to specify the field:

coords=[{"file":"tsne.coords.tsv", "shortLabel":"t-SNE on WGCNA"}, # you can force coloring of some other meta data field when a layout is changed to another one {"file":"subset.coords.tsv", "shortLabel":"neural cells", colorOnMeta="neuralCluster"},

]

In very rare cases, it can be necessary to tell cbBuild that the numbers in the matrix are floating point numbers. The setting looks like this:

matrixType = “float”

CHAPTER 9

Annotate Genes

If you load your existing marker genes into the cell browser, they will by default only be imported as symbols, with no annotations. You may want to annotate the marker genes with information about their gene expression profile (Allan Brain Atlas), the diseases they have been linked to (OMIM, HPO, Sfari), their protein class (HPRD) or other annotations (number of PubMed publications).

To this end, run the tool cbMarkerAnnotate. The syntax is very simple:

```
cbMarkerAnnotate inFname outFname
```

The format for inFname is the same as for the cellbrowser marker gene files, a tab-sep or comma-sep table with at least three columns, in this order: cluster, gene, score. Typical scores are "avg_diff" or "p-Value" or similar. Gene can be a gene symbol or Ensembl gene ID, with or without the version.

cbMarkerAnnotate will map Ensembl gene IDs to symbols and then lookup various gene-related databases to add more columns to inFname and write the result to outFname, in a format that the Cell Browser can easily display.

CHAPTER 10

Combine results

You can use *cbTool metaCat* to merge the meta.tsv files from different pipelines (cbScanpy, cbSeurat, etc) into a single one, like this:

```
cbTool metaCat myMeta.tsv seuratOut/meta.tsv scanpyOut/meta.tsv ./newMeta.tsv --fixDot
```

The option –fixDot will work around R's strange habit of replacing special characters in the cell identifiers with ".". Directories created with ExportToCellbrowser() from R should not have this problem, but others may.

You can start with one of the auto-generated cellbrowser.conf files or start from a fresh one with *cbBuild –init*. In this cellbrowser.conf, add all the coordinates files from all your pipelines.

——— REQUIRED SETTINGS ————

example config file with all possible settings # For a minimal file, see minimal.conf

internal short name, only visible in the URL # same as the output directory name # no special chars, no whitespace, please name = "sample"

priority determines the order of the datasets # smallest comes first priority = 10

tags are shown in the dataset browser # current tags: # smartseq2,10x tags = ["smartseq2"]

human-readable name of this dataset shortLabel="CellBrowser 100-genes demo"

name of the expression matrix file, genes are rows exprMatrix="exprMatrix.tsv.gz"

"gencode-human", "gencode-mouse" or "symbol" # For "symbol" you can specify which database to use to check # symbols or, for cbHub, how to map them to the genome. # 'auto' will automatically detect Ensembl human/mouse IDs # and translate to symbols geneIdType="auto"

name of the meta data table ("samplesheet). One sample per row. First row is name of sample. meta="meta.tsv"

we try to auto-detect the field type of fields in the meta data. # Sometimes, this doesn't work, e.g. when your cluster ID is a numer # or your C1 chip ID is a number, but you don't want them binned, you want # to treat as if they were categories enumFields = ["c1_cell_id"]

tsv files with coordinates of every sample in format <sampleId, x, y> # first the name of the file, then a human readable description coords=[

{"file":"tsne.coords.tsv", "shortLabel":"t-SNE on WGCNA"}, # you can force coloring of some other meta data field when a layout is changed to another one {"file":"subset.coords.tsv", "shortLabel":"neural cells", "colorOnMeta":"neuralCluster"},

]

——— OPTIONAL SETTINGS ————

default field in the meta data table with the name of the cluster clusterField="WGCNAcluster"

default field in the meta data table used for the label of the clusters shown by default labelField="WGCNAcluster"

tsv files with marker gene lists for the clusters # format is (clusterName, geneSymbol, pValue, enrichment) + any additional fields or URLs you want to show markers=[

{"file":"markers.tsv", "shortLabel":"Cluster-specific markers"}

]

optional: UCSC track hub with the BAM file reads and expression values # Alternatively, you can also provide a full link to a UCSC Genome Browser session here hubUrl="http://cells.ucsc.edu/cortex-dev/hub/hub.txt"

optional: table with <name><color> for any meta data values # color is a six-digit hexcode # name is a any value in the meta data table, e.g. cluster name. Canb be a .tsv or .csv file. colors="colors.tsv"

should the cluster labels be shown by default (default: true) showLabels=True

the radius of the circles. If not specified, reasonable defaults will be used #radius = 5 # the alpha/transparency of the circles. If not specified, reasonable defaults will be used. #alpha = 0.3

you need short names for your clusters, as there is little space on the plot # but cell types have complicated and long names # So you can provide a table with two columns: 1) short cluster name 2) long version # e.g. EC, endothelial cells # can be a .tsv or .csv file acroFname = "acronyms.tsv"

genes that are highlighted in your paper can be pre-loaded and are shown as a clickable table on the left quickGenesFile = "quickGenes.csv"

the unit of the values in the expression matrix # any string, shown on genome browser and violin y-Axis # typical values are: "read count/UMI", "log of read count/UMI", "TPM", "log of TPM", "CPM", "FPKM", "RPKM" unit = "TPM"

— The following options are only used by cbHub — hubName = "100 Genes Sample Hub" # name of hub (optional, default is value of 'shortLabel') ucscDb = "hg38" # UCSC genome ID of the BAM files, required bamDir = "bam" # directory with .bam files, optional. If not present, don't do bam merging #clusterOrder = "clusterOrder.txt" # file with cluster names to order the tracks (default is alphabetical)

CHAPTER 11

Describing datasets

A dataset can be described with three HTML files, summary.html, methods.html and downloads.html. You can put these in the same directory where `cellbrowser.conf` is stored and they will get copied along to the webserver and shown in the `File > Open Dataset...` dialog.

Howver, when you have many datasets, writing the html files gets repetitive. This is where desc.conf is handy, it's a key-value file with the description of the dataset in a standardized format.

A sample file can be created with the command `cbBuild --init`.

The following lists all tags that are currently supported.

These tags contain longer text that can include HTML markup:

- `title`: title of the dataset, often the paper title
- `abstract`: a big picture summary of the dataset, as a string
- `methods`: the methods for the dataset, as a string
- `unitDesc`: a description of the values / the unit in the expression matrix (e.g. 'TPM' or 'log'ed counts')

Instead of long strings with HTML content for `abstract` and `methods`, you can also create the files `abstract.html` and `methods.html`, they will be used instead. Or use the statements `abstractFile` and `methodsFile` to specify other file names. In the HTML, you can use text like `<section>some subtitle</section>` to split the text into sections.

This tag contains an image file name: - `image`: usually a 400px-wide thumbnail of the dimensionality reduction

The following tags can contain URLs and optionally, separated with a space, a label for the link. If you do not specify the label, a default label will be used (e.g. 'Biorxiv Preprint'):

- `biorxiv_url`: URL of the pre-print
- `paper_url`: URL to any website with the fulltext
- `other_url`: URL to a website that describes the dataset

The following tags contain accession IDs and will be translated to links:

- `pmid`: Pubmed ID of the publication (CIRM TagsV5)

- `geo_series`: NCBI GEO series ID (CIRM TagsV5)
- `sra`: NCBI SRA accession
- `sra_study`: NCBI SRA SRPxxxx accession
- `doi`: DOI of paper fulltext
- `dbgap`: NCBI dbGaP accession, starts with phs

The following tags contain just text:

- `submitter`: name and/or email of submitter
- `lab`: lab and University of submitter
- `submission_date`: ideally in format year-month-day
- `version`: version of dataset, a simple number (1,2,3,...) that should be increased each time a major change (usually meta data) was received from the lab

CHAPTER 12

Optional Python modules

There are currently no required Pythom modules for the basic Cell Browser script cbBuild.

In cellbrowser.conf you can specify a color file, the format is .tsv or .csv and it has two columns, clusterName<tab>colorCode. If this file contains html color names instead of color codes, you have to install the module webcolors:

```
pip install webcolors
```

To read expression matrices in .mtx format, you have to install scipy:

```
pip install scipy
```

cbScanpy require that scanpy is installed. cbSeurat requires that the R that is run when you type *Rscript* has Seurat installed.