cellbrowser Documentation

Release v1.2.1+3.g0cb0a1c

Maximilian Haeussler, Lucas Seninge, Nikolai Markov

Sep 15, 2022

Contents

The UCSC Cell Browser is a fast, lightweight viewer for single-cell data. Cells are presented along with metadata 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. If you use the UCSC Cell Browser in your research, please cite our Bioinformatics paper. If you are also using data from a specific dataset we host, please also cite the original authors of that dataset (visible under 'Info & Download' while viewing that dataset).

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.

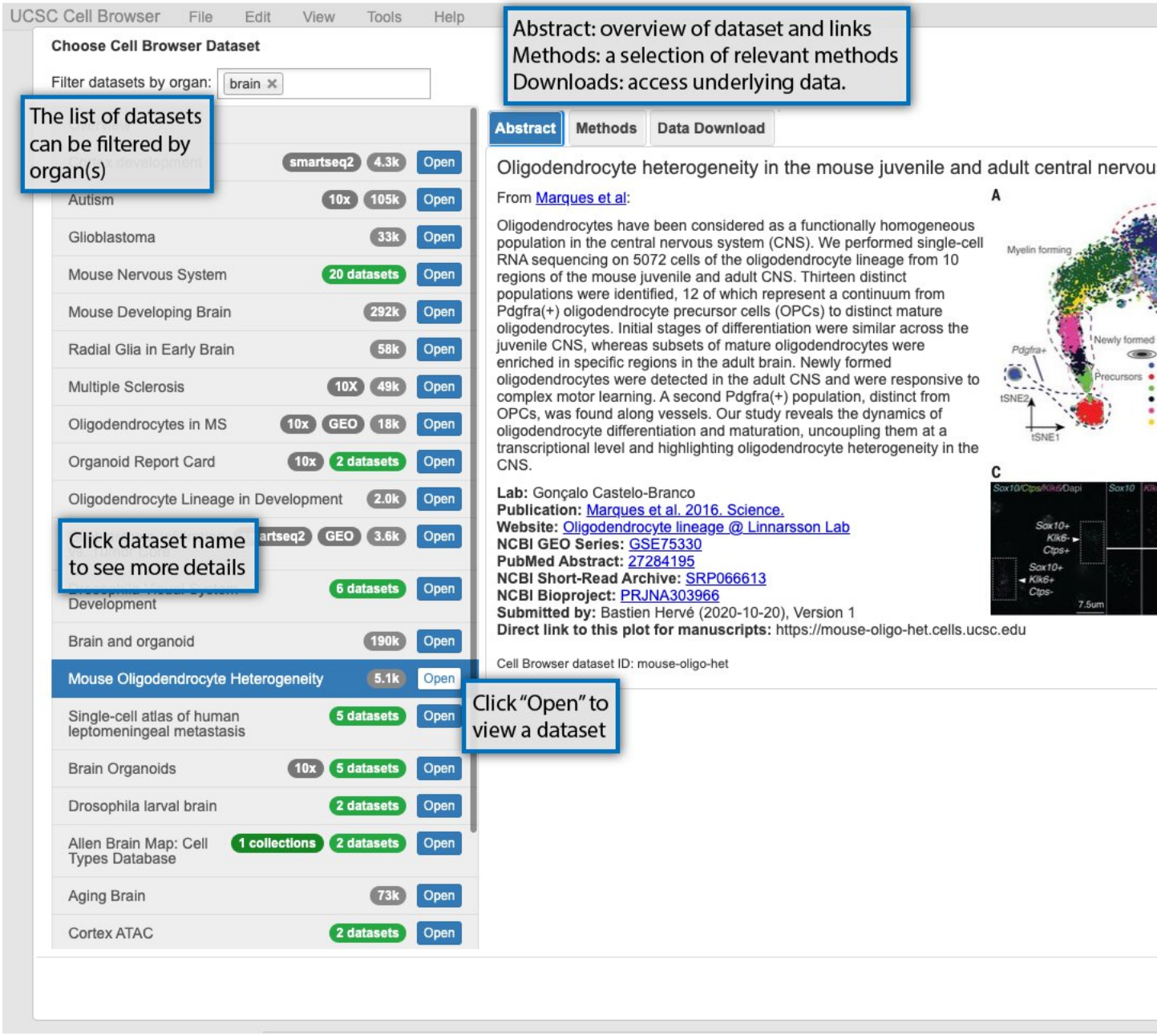
CHAPTER 1

Cell Browser Interface

Contents

1.1 Selecting a dataset and learning about it

When you first open https://cells.ucsc.edu or your local cell browser, you will be greeted by a dataset select screen. This image highlights some of these features, or read more about them below the image.



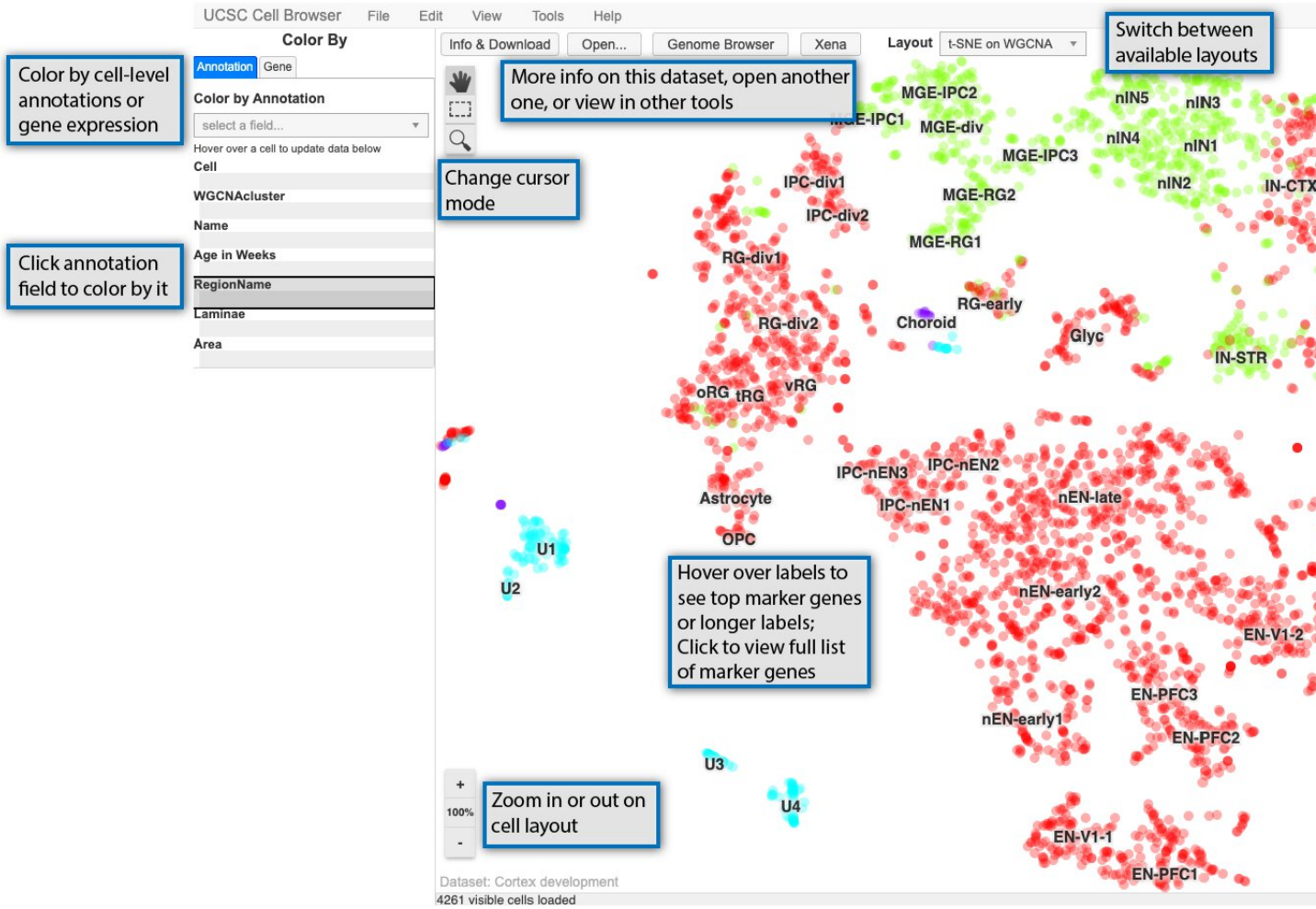
The datasets available are listed on the lefthand side of the screen. Datasets can be filtered using organ labels (e.g. skin, kidney) if available. If you select a dataset, information about it will be shown on the right. There are three informational tabs:

- The "Abstract" tab contains a short description of the dataset, typically the abstract of an associated paper, and links to the paper, GEO, or other related resources.
- The "Methods" tab contains a short description of the methods used to generate the dataset.
- The "Data Download" tab contains links to download the data underlying the dataset in the cell browser, such as expression matrix and metadata.

To open the cell browser for that dataset, double-click it in the dataset list or click "Open".

1.2 Basic Cell Browser features

After you have opened a dataset, you will be taken to the main way of visualizing datasets in the Cell Browser, an annotated scatterplot derived from UMAP, tSNE, or some other dimensionality reduction method. The image below highlights some of the main features of this core visualization:



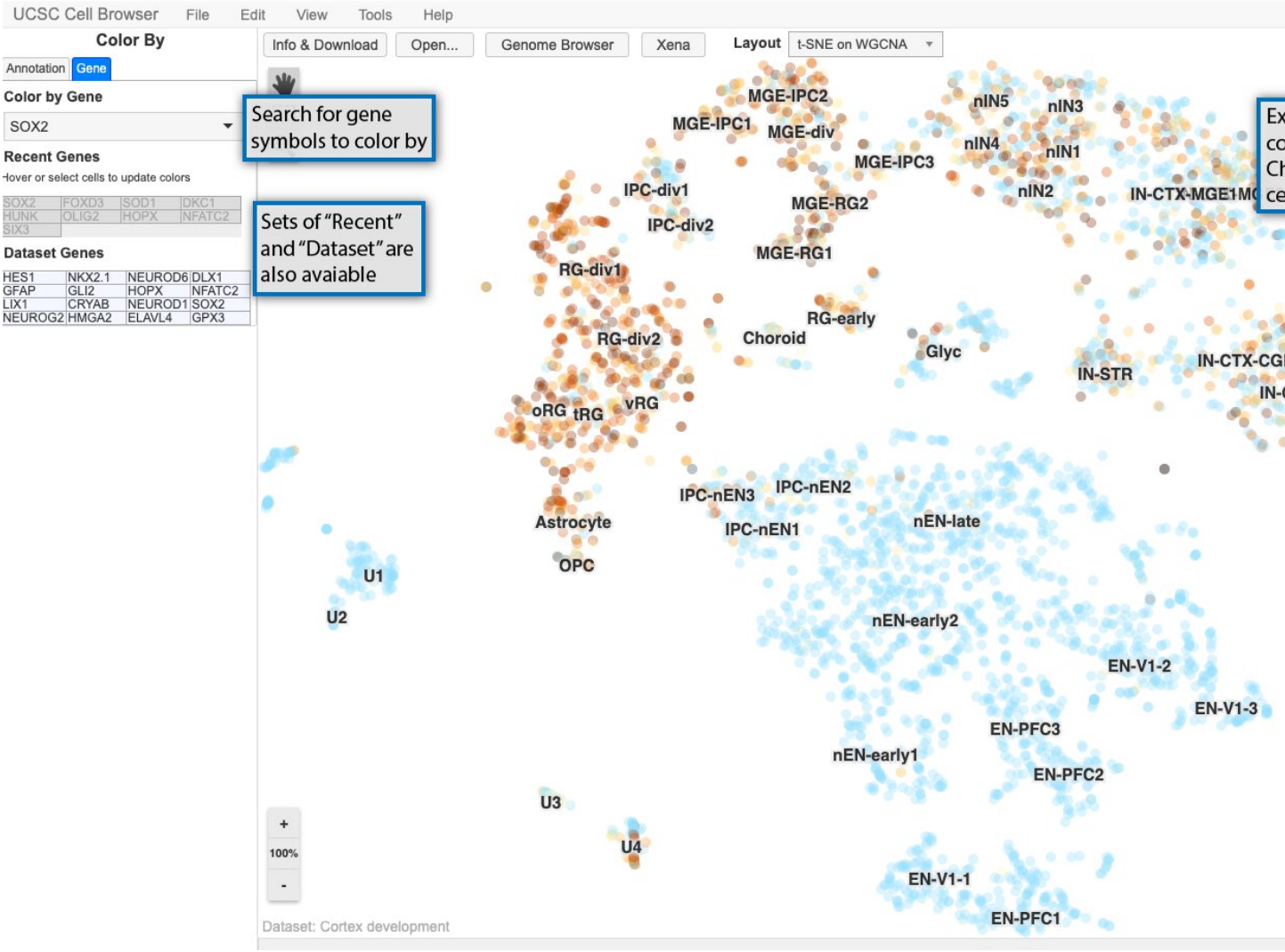
The main scatterplot will be colored by default using a field from the "Annotation" tab on the lefthand side of the screen and can be changed by clicking on a new field. The values in the selected field are shown in the "Legend" on the righthand side of the screen. You can zoom in or out on this plot using the buttons in the lower left of the screen or your center mouse wheel with the "100%" button or space bar returning you to the default zoom. Depending on the cursor mode, you can: * pan around the view (the hand) * select cells (dashed outline rectangle) * select and then zoom in on the plot (magnifying glass)

For many datasets, the plot in the center of the screen will include cluster labels. These labels can be hovered over with your mouse cursor to see more details if they are present, such as top marker genes or an expansion of acronym used in the label. Clicking on the clusters labels, will show you the marker genes pop-up described in the next section.

Finally, along the top, there are links to open the dataset description or switch the layout among other things. Some options are only available if certain conditions are met, such as viewing related data in Xena or the UCSC Genome Browser (if available) or navigating between datasets in a collection.

1.3 Color the display by gene expression

In addition to being able to color by metadata as described in the previous section, you can color the scatterplot by gene expression. Start by selecting the "Gene" tab on the lefthand side and searching for a gene symbol in the search box.



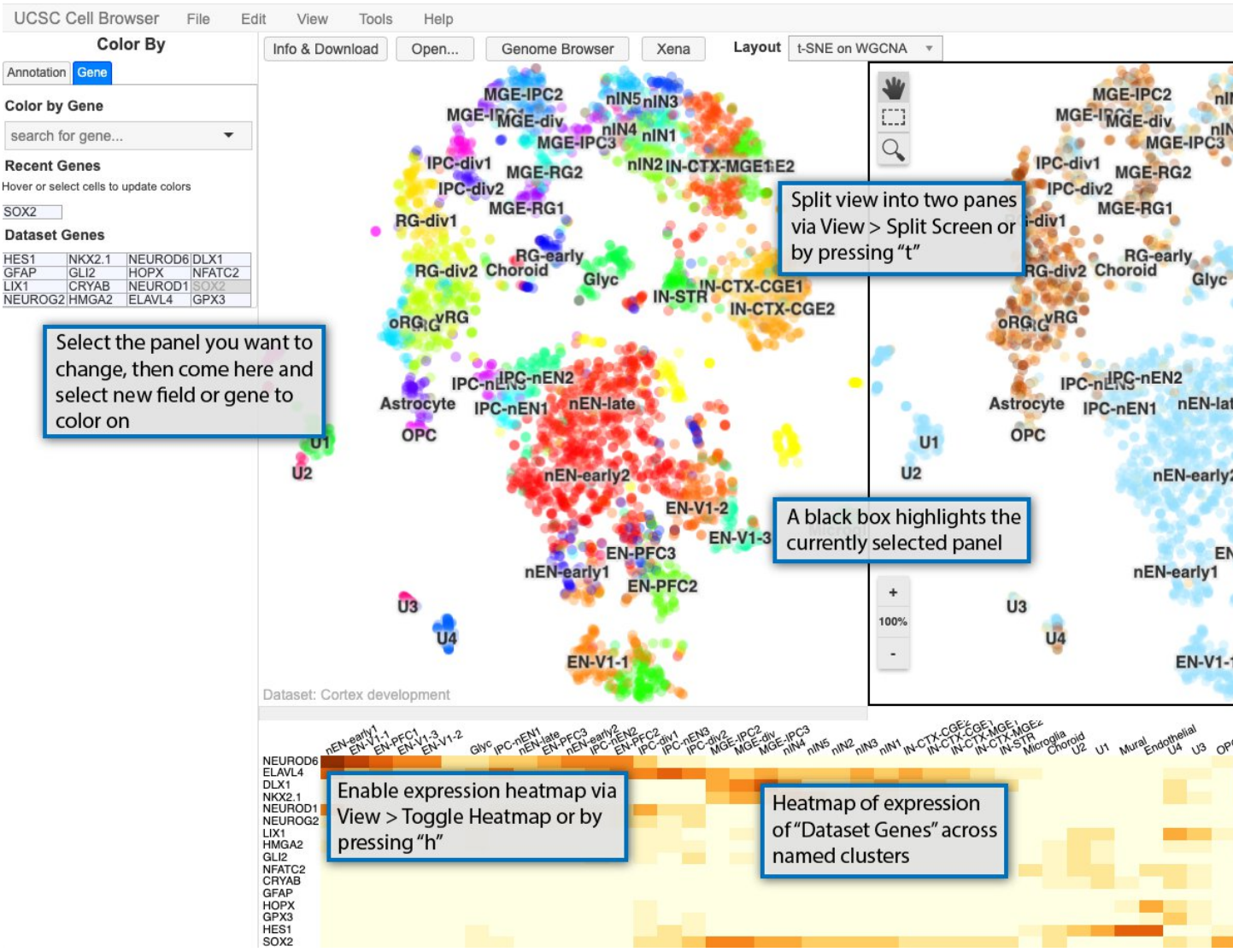
Expression values are divided up into bins and associated with gradient of colors going from light to dark as shown in the legend on the right.

Cells can be selected using the checkboxes in the legend or using the square select cursor mode. After cells are selected, a violin plot shows the expression for the currently colored gene in the selected cells versus all other cells.

Cells can also be selected and set as the "background cells" to compare against. First, select cells as described previously, and then go to Tools > Set as background cells or type bs on your keyboard. Those cells will then set as your background and gene expression in any new selection of cells will be compared against these background cells in the violin plot. To reset the background cells, go to Tools > Reset background cells or type br.

1.4 View expression heatmap or split the display

The Cell Browser also provides other methods for exploring datasets: split-screen mode and expression heatmap for the dataset genes.

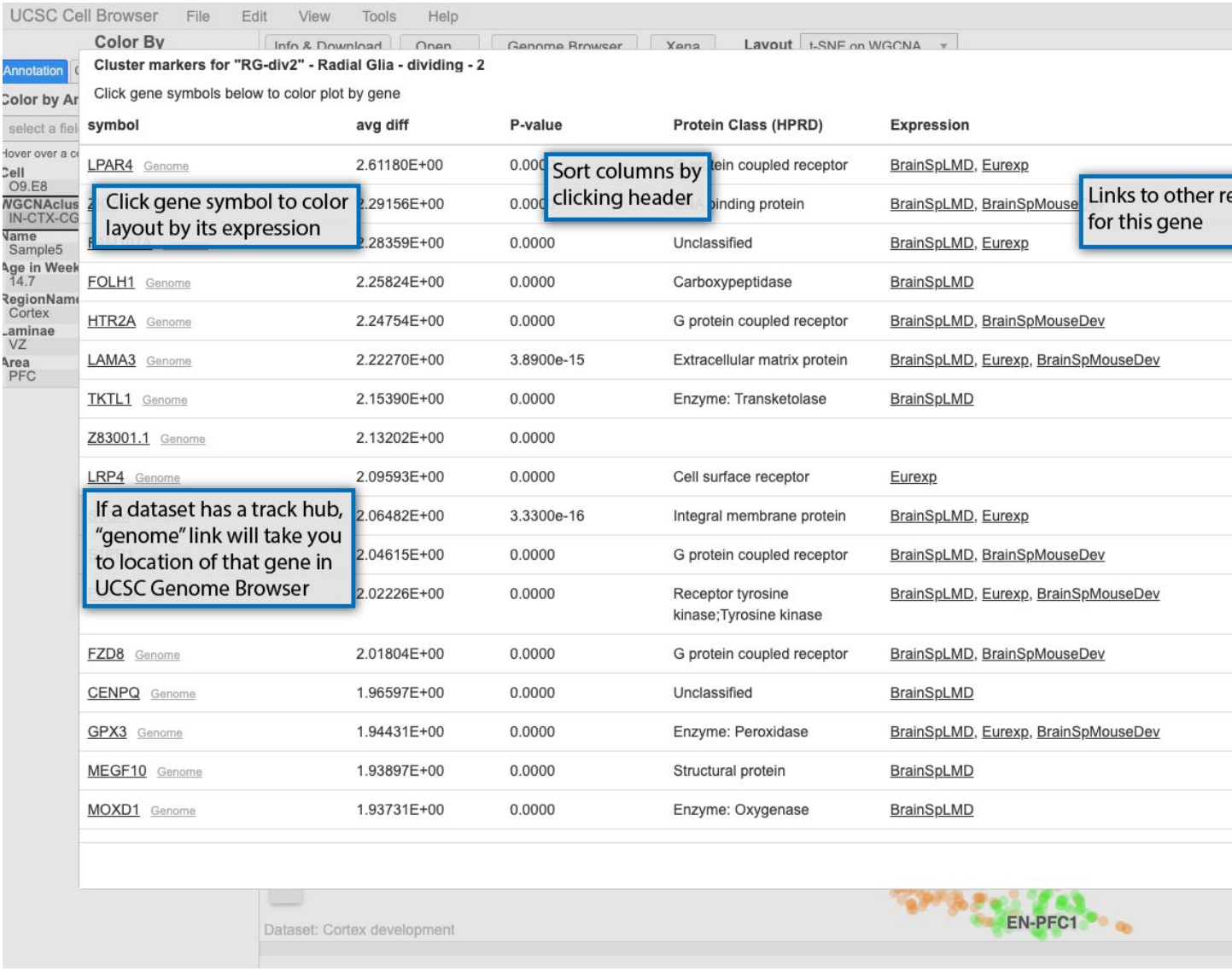


The main Cell Browser view can also be split into two panes for easy comparison of two different metadata fields or gene expression patterns or a combination of the two. Enable split-screen using View > Split screen or typing t. The currently selected panel is boxed in black with the legend reflecting the current panel. If you want to change the coloring for a panel, select it by clicking it, and then changing the coloring to whatever annotation or gene you want.

Additionally, a heat map of the expression of the "dataset genes" across the cluster names displayed in the scatterplot. The size of the heatmap can be adjusted by clicking on the bar between the heat map and scatterplot and dragging it to the desired size.

1.5 View cluster marker genes

If available, clicking on the cluster labels in the main scatterplot view will bring up a list of marker genes for that cluster.



symbol	avg diff	P-value	Protein Class (HPRD)	Expression
LPAR4 Genome	2.61180E+00	0.000	ein coupled receptor	BrainSpLMD, Eurexp
	2.29156E+00	0.000	inding protein	BrainSpLMD, BrainSpMouse
	2.28359E+00	0.0000	Unclassified	BrainSpLMD, Eurexp
FOLH1 Genome	2.25824E+00	0.0000	Carboxypeptidase	BrainSpLMD
HTR2A Genome	2.24754E+00	0.0000	G protein coupled receptor	BrainSpLMD, BrainSpMouseDev
LAMA3 Genome	2.22270E+00	3.8900e-15	Extracellular matrix protein	BrainSpLMD, Eurexp, BrainSpMouseDev
TKTL1 Genome	2.15390E+00	0.0000	Enzyme: Transketolase	BrainSpLMD
Z83001.1 Genome	2.13202E+00	0.0000		
LRP4 Genome	2.09593E+00	0.0000	Cell surface receptor	Eurexp
	2.06482E+00	3.3300e-16	Integral membrane protein	BrainSpLMD, Eurexp
	2.04615E+00	0.0000	G protein coupled receptor	BrainSpLMD, BrainSpMouseDev
	2.02226E+00	0.0000	Receptor tyrosine kinase;Tyrosine kinase	BrainSpLMD, Eurexp, BrainSpMouseDev
FZD8 Genome	2.01804E+00	0.0000	G protein coupled receptor	BrainSpLMD, BrainSpMouseDev
CENPQ Genome	1.96597E+00	0.0000	Unclassified	BrainSpLMD
GPX3 Genome	1.94431E+00	0.0000	Enzyme: Peroxidase	BrainSpLMD, Eurexp, BrainSpMouseDev
MEGF10 Genome	1.93897E+00	0.0000	Structural protein	BrainSpLMD
MOXD1 Genome	1.93731E+00	0.0000	Enzyme: Oxygenase	BrainSpLMD

By default, the genes are sorted by p-value, but can be sorted by any of the other columns as well. At minimum, the pop-up will include the gene symbol and a score, but can be augmented with other scores or even links out to other resources like OMIM. If you click on the gene symbol in the first column, it will color the scatterplot by the expression of that gene. If a dataset includes a UCSC Genome Browser hub, small "genome" links will appear next to the gene symbols and clicking those will take you a genome browser view centered on that gene with the hub tracks displayed.

CHAPTER 2

Submitting data to the UCSC Cell Browser

At this time, we are happy to host pretty much any single-cell dataset, regardless of the library prepartion (10x, Smart-seq2, etc), organism (human, mouse, zebrafish, etc), or analysis method (Seurat, Scanpy, Monocle, etc).

A cell browser requires at minimum three things:

- Expression matrix
- Metadata with cell names and cluster field
- 2D Layout coordinates

If you can provide these alongside some description of the dataset, we'll host it. Contact us at cells@ucsc.edu to get started.

2.1 Preparing and sharing your files

Before we can make a cell browser for you, you have to share the data with us. We accept the following file types:

- Seurat RDS, Rdata, or Robj files
- Scanpy h5ad or Loom files
- A collection of tsv or csv files
- The output directory of one of our cbImport* tools

After you have your data in one of the formats above, you will have to share the data with us.

We prefer data to be shared in a way that is easy for us to download with something like wget, so the following methods are ideal:

- University or other institutional server space
- AWS
- GEO
- Dropbox (if shared via link, e.g. https://www.dropbox.com/s/NN/my_seurat.rds)

We do accept files via other methods, although they take a little more work for us to move to our server. These include methods such as:

- Box
- Google Drive
- Dropbox, other methods of sharing data

2.2 Other things we want from you

2.2.1 Dataset description

Alongside your submission, it would be great if you filled out a desc.conf file. At the very least, it should have the abstract, methods, and title filled out. However, you are welcome to fill out more fields and make it as complete as you would like. You can run `cbBuild --init` to copy an example desc.conf into your current directory or you can copy one from our Github repo.

If you have a collection of datasets, please provide basic desc.conf for each of the datasets in the collection. These don't have to be as comprehensive as the top-level one for the collection itself. At minimum, they should include two tags: (1) title and (2) abstract. However, feel free to fill out as many of the desc.conf fields as you think is helpful for that dataset.

2.2.2 Dataset shortname

It would be great if you could suggest a dataset shortname at the time of your submission, although we're happy to make one up for you. An ideal short name fulfills the following requirements

- All lowercase
- Words separated by dashes ("-")
- Four words or less (don't be afraid to abreviate words, e.g. development -> dev)
- Informative

A great example is cortex-dev - it's all lowercase, the two words are separated by dashes, it's short at only two words long, and informs you that the dataset is focused on cortex development. It fulfills all four points above.

Other great examples:

- mouse-nervous-system
- skeletal-muscle
- mouse-oligo-het
- covid-hypertension

The short name doesn't have to be perfect, but good enough to communicate something about your dataset in a few words.

2.2.3 "Quick Genes"

This is a list of 10-15 genes that you think are important to your dataset(s). In addition to the list of gene symbols, it is great to have a word or two about why it's in the list (e.g. "Fst, Paraxial Mesoderm"; "HES1, Fig1D"). If you have a collection of datasets, you can have one set of genes for every dataset in the collection or a different set for each.

This list and descriptions should be in CSV or TSV format.

2.3 Getting your URL

After you submit your dataset to us, we will import the data and make a preliminary version available on our development server. We will work with you to iterate and make improvements to this version first. Once you give your final approval, we will push the data to our main site, cells.ucsc.edu. Once there, you will recieve the final URL, e.g. cortex-dev.cells.ucsc.edu. This is the URL you should place in your paper, link to from your lab website, tweet about, etc. Please **do not** put the url to our development server in your paper, since it is under active development, we occasionaly break it.

2.4 FAQs

2.4.1 Can I share the output of cbBuild with you?

If you are going to share the output of one of our cbImport* tools, we prefer the directory containing the cellbrowser.conf, desc.conf, etc. The output of cbBuild is optimized for web access and display, which makes it difficult if not impossible to make changes to the cell browser at a later date (e.g. correcting spelling mistakes). If we have access to the desc.conf, cellbrowser.conf, and other files, we can easliy make these changes and rebuild the cell browser if needed.

2.4.2 Can I keep my dataset private until a later date, but still accessible to reviewers?

Yes, we offer limited methods for keeping datasets private. We can hide datasets from being listed alongside the others we host. This means that someone would need to know the URL or dataset name to be able to access your dataset. For example, this means that someone would need the URL cells.ucsc.edu/?ds=cortex-dev or know the name (cortex-dev) to access the dataset.

CHAPTER 3

Installation

Table of Contents

There are currently no required Pythom modules for the core Cell Browser script: `cbBuild`. Installation instructions for a variety of methods are below, with pip being the recommended method.

3.1 Basic installation

3.1.1 With pip (recommended)

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
```

3.1.2 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. Please indicate in any bug reports if you used conda to install.

3.1.3 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
```

3.1.4 With 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
```

3.1.5 Notes on Windows Installation

First install the Windows Linux subsystem. Then open the Windows Linux Subsystem bash terminal and run these commands:

```
sudo apt-get update
sudo apt-get install python-pip
sudo pip install cellbrowser
```

3.2 Optional modules

To take advantage of some of the advanced features or specialized scripts such as `cbScanpy` or `cbSeurat`, you will need to install some extra packages or tools.

3.2.1 Image sizes

To get the image sizes, cbBuild uses either the "file" command or the "identify" command (for JPEGs). You may have to install the ImageMagick package to get the identify command.

3.2.2 Custom Colors

In your cellbrowser.conf you can specify a file with custom colors for your metadata values. If this file contains HTML color names instead of color codes, you have to install the module webcolors:

```
pip install webcolors
```

3.2.3 Matrices in mtx format

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

```
pip install scipy
```

3.2.4 `cbScanpy` and `cbSeurat`

`cbScanpy` requires that Scanpy is installed.

`cbSeurat` requires that both R and Seurat are installed .

CHAPTER 4

Quick Start Guide

These steps will walk through downloading and setting a minimal cell browser using an example dataset. It is assumed that you are carrying out these steps locally on your computer.

It is based on Nowakowski et al. 2017. and the corresponding https://cortex-dev.cells.ucsc.edu dataset. The expression matrix only includes 100 genes, but it does show off many of the features of the cell browser.

4.1 Step 1: Download the example dataset

Download the data using curl:

```
curl -kO https://cells.ucsc.edu/downloads/samples/mini.tgz
```

This file contains all the pieces necessary to create a simple cell browser.

4.2 Step 2: Unpack the dataset

Next, we need to take this file and unpack it's contents:

```
tar -xvzf mini.tgz
```

This will create a new `mini` directory contain the cell browser files.

4.3 Step 3: Enter the *mini* directory

Now, enter the `mini` directory so that we can build the cell browser next:

```
cd mini
```

4.4 Step 4: Build the cell browser

Then, build the cell browser:

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

The `-o` option specifies the output directory and `-p` defines the port to used for the webserver. A bunch of information will be displayed on the screen, mostly just reporting its progress in building the cell browser.

4.5 Step 5: Check it out!

Finally, point your web browser to http://localhost:8888 to view your minimal cell browser.

4.6 Step 6: Stop webserver

When you are done playing with your minimal cell browser, stop the web server started by cbBuild using `Ctrl-C`.

CHAPTER 5

Setup your own

5.1 Overview

The UCSC Cell Browser tool set consists of a number of different scripts to help you set up your own. The primary 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.

5.2 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
```

5.2.1 On your local computer

If you are running the cell browser on your local computer, you will likely need to have cbBuild start up a webserver for you. You can use the `-p PORT` option to specify on which port the webserver will run:

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

Pointint your web browser to `http://localhost:8888` to view your cell browser. 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/my-dataset`.

5.2.2 On a webserver

If you are on a webserver, you likely only only want to build the cell browser html files into a web-accessible directory:

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

Specifying the port is optional is you are running this on a server that is already web-accessible. To view your cell browser, navigate to the address for your webserver.

5.3 Customizing your cell browser

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. When you run `cbBuild`, the output will be written to OUTPUT_DIR/dataset-name. If you specify the same `-o OUTPUT_DIR` when running cbBuild for a different dataset with a different dataset name and cellbrowser.conf, the output will be put into a new subdirectory within `OUTPUT_DIR`. A single cbBuild output directory can contain multiple datasets.

CHAPTER 6

Putting it onto the internet

6.1 Basics

Deploying your cell browser on the web is as simple as copying the output of `cbBuild`, including all files and directories, into to an empty directory on a web server. The cell browser should be able to be deployed on almost any web server, including:

- One provided by your university (often through a public_html directory in your home directory)
- Github.io provides free hosting for 1 GB of webspace and it is fast enough, see https://sansomlab.github.io/
- You can also use commercial cloud providers, such as Amazon S3, Google Cloud Storage, Microsoft Azure, though they will need a credit card.

Please consider also sending the output files to cells@ucsc.edu, we are more than happy to add it to our public Cell Browser website. You can choose a dataset prefix, and then can add myDataset.cells.ucsc.edu to your manuscript. Unfortunately, online backup solutions such as Dropbox, Box.com, iCloud, OneDrive or Google Drive will not work; they are intentionally designed to not be usable as web servers.

6.2 Adding multiple datasets to your cell browser

To add more datasets to the same cell browser, navigate 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.

6.3 Specifying a default output directory for `cbBuild`

The output directory for `cbBuild` can be controlled using environment or .conf variables. This allows you to run `cbBuild` in a directory without needing to specify an output directory using the "-o" option.

To control the output directory using an environment variable, add the following line to your ~/.bashrc to point to your html directory:

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

Replace `/var/www` with whatever you want the default output directory to be.

Alternatively, you can create a file called `.cellbrowser.conf` in your home directory and assign a value to htmlDir:

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

Again, replace `/var/www` with your own dorectory.

6.4 Notes on setting up a permanent cell browser on a local machine

The port option, e.g. `-p 8888`, is optional. When this option is specified, it will start up its own web server. If you are running this on your local machine, a more permanent alternative to the -p option is to run a web server on your machine and then 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/
```

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

On Linux, you will need to install Apache2 (with `sudo yum install httpd` or `sudo apt-get install apache2`) and use the directory `/var/www/` instead:

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

Windows is usually not used as a web server, just use the built-in web-server via `-p (portNumber)` on Windows.

CHAPTER 7

How To. . .

7.1 How to create a cell browser using a Seurat RDS file

You can go from an RDS file to cell browser for a dataset in three easy steps:

7.1.1 Step 1: Export an RDS file of your Seurat object

From within R, run this command to create an RDS file fo your dataset:

```
saveRDS(objName, "myDataset.rds")
```

7.1.2 Step 2: Use cbImportSeurat to export

Next, you will use cbImportSeurat to create the files needed for a cell browser using the data in the RDS file:

```
cbImportSeurat -i myDataset.rds -o myRdsImport -n seurat-import
```

Note: cbImportSeurat will work with RDS files from Seurat v2 or v3. When importing data, you need to have installed the same version of Seurat that was used to create the RDS file.

7.1.3 Step 3: Build a Cell Browser

Lastly, go into the output directory specified in the cbImportSeurat command and run cbBuild to create the cell browser:

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

Or, if you don't have a webserver already, use the built-in one:

```
cbBuild -o /myHtmlFiles -p 8888
```

You should now be able to access your cell browser from the web.

7.2 How to use the cell browser export function in Seurat3

It is a simple, single-line command to build a web-accessible cell browser from a Seurat object from within R:

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

7.3 How to run a basic Seurat pipeline using cbSeurat

Going from an expression matrix to a cell browser by running our basic Seurat pipeline takes two steps:

7.3.1 Step 1: Run cbSeurat on your expression matrix

First, run a Seurat pipeline on your expression matrix using `cbSeurat`:

```
cbSeurat --exprMatrix=myExpressionMatrix.tsv.gz --name=myDataset --outDir=seurat-out
```

7.3.2 Step 2: Build a Cell Browser

Next, go into the output directory specified in the cbImportSeurat command and run cbBuild to create the cell browser:

```
cd seurat-out
cbBuild -o ~/public_html/cb
```

Or, if you don't have a webserver already, start the built-in one:

```
cbBuild -o /myHtmlFiles -p 8888
```

7.4 How to configue a basic cbSeurat pipeline

Running `cbSeurat` will run a basic Seurat pipeline with the default settings. `cbSeurat` can be configured through a seurat.conf.

7.4.1 Step 1: Copy a seurat.conf

cbSeurat can be used to copy down an example seurat.conf:

```
cbSeurat --init
```

7.4.2 Step 2: Edit your seurat.conf

Now that you have a seurat.conf in your current directory, open it up and edit it! If this file is in the same directory where you are running `cbSeurat`, it will be automatically picked up.

7.5 How to create a cell browser using a Scanpy h5ad file

Going from an h5ad file to cell browser for a dataset takes two steps:

7.5.1 Step 1: Use cbImportScanpy to export

First, you will use cbImportScanpy to create the files needed for a cell browser using the data in the RDS file:

```
cbImportScanpy -i myDataset.h5ad -o scanpy-import -n my-dataset
```

7.5.2 Step 2: Build a Cell Browser

Then, go into the output directory specified in the cbImportSeurat command and run cbBuild to create the cell browser:

```
cd scanpy-import
cbBuild -o ~/public_html/cb
```

Or, if you don't have a webserver already, start the built-in one:

```
cbBuild -o /myHtmlFiles -p 8888
```

You should now be able to access your cell browser from the web or your local computer.

7.6 How to convert a Scanpy object within Python

It a few simple commands to build a `cellbrowser.conf` and all the files you need for a cell browser. This is particularly useful for Jupyter notebooks.

7.6.1 Step 1: Export the data needed

Load the cell browser package and export the files from the scanpy object:

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

7.6.2 Step 2: Build the cell browser

Next, build the dataset:

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

7.6.3 Step 3: Start (and stop) web server (optional)

This step is only necessary if you don't already have a web server running that is servering up the output of step 2.

Start the web server:

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

Stop the webserver when you're done:

```
cb.stop()
```

7.7 How to run a basic Scanpy pipeline using cbScanpy

Going from an expression matrix to a cell browser by running our basic Scanpy pipeline takes two steps:

7.7.1 Step 1: Run cbScanpy on your expression matrix

First, run a Scanpy pipeline on your expression matrix using cbSeurat:

```
cbScanpy -e myExpressionMatrix.tsv.gz -n my-scanpy-dataset -o scanpy-out -m cell-
↪annotations.tsv
```

7.7.2 Step 2: Build a Cell Browser

Next, go into the output directory specified in the `cbScanpy` command and build your cell browser:

```
cd scanpy-out
cbBuild -o ~/public_html/cb
```

7.8 How to configue a basic cbScanpy pipeline

Running `cbSeurat` will run a basic Scanpy pipeline with the default settings. `cbScanpy` can be configured through a scanpy.conf.

7.8.1 Step 1: Copy a scanpy.conf

cbSeurat can be used to copy down an example scanpy.conf:

```
cbScanpy --init
```

7.8.2 Step 2: Edit your seurat.conf

Now that you have a scanpy.conf in your current directory, open it up and edit it! If this file is in the same directory where you are running `cbScanpy`, it will be automatically picked up.

7.9 How to export the data from Monocle for use in the Cell Browser

Monocle is an R package that can be used to reconstruct transcriptional trajectories. You can export the coordinates, expression data, and metadata from a Monocle object and then use those files to build a cell browser. These steps assume that you have your Monocle object loaded into R already.

7.9.1 Step 1: Export expression matrix

First, export data in MTX format, since it can handle large matrix sizes. MTX consists of three files: (1) a sparse matrix, (2) a file of column names, and (3) a file of row names.

(1) MTX sparse matrix:

```
writeMM(exprs(monocle_obj), 'matrix.mtx')``
```

(2) Row names (genes):

```
write.table(as.data.frame(cbind(rownames(exprs(monocle_obj)), rownames(exprs(monocle_
↪obj)))), file='features.tsv', sep="\t", row.names=F, col.names=F, quote=F)
```

(3) Column names (samples/cells):

```
write(colnames(exprs(monocle_obj)), file = 'barcodes.tsv')
```

7.9.2 Step 2: Export cell annotations

Next, export the cell metadata annotations, which includes Monocle's calculated 'pseudotime':

```
write.table(as(monocle_obj@phenoData,"data.frame"), file='meta.tsv', quote=FALSE, sep=
↪'\t', col.names = NA)
```

7.9.3 Step 3: Export cell coordinates

Then, export the cell coordinates:

```
write.table(t(monocle_obj@reducedDimS), file='monocle.coords.tsv', quote=FALSE, sep=
↪'\t', col.names = NA)
```

7.9.4 Step 4: Set up your cellbrowser.conf

Finally, create the cellbrowser.conf file for your dataset. You can use `cbBuild --init` to place an example cellbrowser.conf (and desc.conf) into your current directory.

You will specifically need to edit these lines to point to the flies that you exported in steps 1-3 above:

```
exprMatrix="matrix.mtx"
meta="meta.tsv"

coords=[
  {
    "file":"monocle.coords.tsv",
    "shortLabel":"Monocle Trajectory",
    "flipY":True,
  },
]

defColorField="Pseudotime"
```

You will still need to set the other required settings in your cellbrowser.conf as well

7.10 How to export the tree and data from URD for use in the Cell Browser

URD is an R package that can be used to reconstruct transcriptional trajectories and then displaying this trajectory as a branching tree. You can export the tree diagram, expression data, and metadata from an URD object from within R and then use the resulting files to build a cell browser.

7.10.1 Step 1: Export cell coordinates for the tree

First, we need the coordinates for the cells in relation to the tree:

```
write.table(urd_obj@tree$cell.layout, file='urd.coords.tsv', quote=FALSE, sep='\t',␣
↪col.names = NA)
```

7.10.2 Step 2: Export line coordinates for the tree

Next, we need the coordinates for the lines that make up the tree:

```
write.table(urd_obj@tree$tree.layout, file='urd.lines.tsv', quote=FALSE, sep='\t',␣
↪col.names = NA)
```

7.10.3 Step 3: Export expression matrix

Export data in MTX format, since it can handle large matrix sizes. MTX consists of three files: (1) a sparse matrix, (2) a file of column names, and (3) a file of row names.

(1) MTX sparse matrix:

```
writeMM(urd_obj@count.data, 'matrix.mtx')``
```

(2) Row names (genes):

```
write.table(as.data.frame(cbind(rownames(urd_obj@count.data), rownames(urd_obj@count.
↪data))), file='genes.tsv', sep="\t", row.names=F, col.names=F, quote=F)
```

(3) Column names (samples/cells):

```
write(colnames(urd_obj@count.data), file = 'barcodes.tsv')
```

7.10.4 Step 4: Convert MTX to tsv.gz

It's easiest to specify a single exprMatrix.tsv.gz file in your cellbrowser.conf later, so we'll convert our exported MTX to tsv via `cbTool mtx2tsv`:

```
cbTool mtx2tsv matrix.mtx genes.tsv barcodes.tsv exprMatrix.tsv.gz
```

7.10.5 Step 5: Export metadata

Metadata annotations are also needed for a cell browser:

```
write.table(urd_obj@meta, file='meta.tsv', quote=FALSE, sep='\t', col.names = NA)
```

7.10.6 Step 6: Export tSNE (optional)

The cell coordinates and lines from steps one and two above satisfy the cell browser's need for a layout, however, URD can generate a tSNE layout as part of it's run. You can export these coordinates for use in the cell browser:

```
write.table(urd_obj@tsne.y, file='tsne.coords.tsv', quote=FALSE, sep='\t', col.names
↪= NA)
```

7.10.7 Step 7: Create your cellbrowser.conf

Next create the cellbrowser.conf file for your dataset. You can use `cbBuild --init` to place an example cellbrowser.conf (and desc.conf) into your current directory.

You will specifically need to edit these lines to point to the flies that you exported in steps 1-5 above:

```
exprMatrix="exprMatrix.tsv.gz"

meta="meta.tsv"

coords=[
  {
    "file":"urd.coords.tsv",
    "lineFile":"urd.lines.tsv",
    "shortLabel":"URD Trajectory",
    "flipY":True,
    "lineFlipY": True
  },
  {
    "file": "tsne.coords.tsv",
    "shortLabel":"tSNE"
  }
]
```

You will still need to set the other required settings in your cellbrowser.conf as well

7.11 How to start the webserver without building datasets

If you have stopped the built-in webserver and want to start it again, without rebuilding the entire dataset, use the cbUpgrade tool:

```
cbUpgrade -o /myHtmlFiles -p 8888
```

7.12 How to visualize single-cell ATAC-seq data in the Cell Browser

The Cell Browser supports single-cell ATAC-seq data. It requires the same files that a standard dataset needs with the added requirement of knowing the gene models to enable searching for peaks around genes. Typically ATAC-seq data includes inferred gene signal analysis as well, so the gene models used for that should be the same used here.

7.12.1 Step 1: Gather required files

You will the following three files:

- Expression matrix with cell names as columns and peak ranges as rows.
- Cell annotations/metadata
- Layout coordinats (e.g. UMAP)

7.12.2 Step 2: Determine GENCODE Gene Model version (optional)

If you don't know the GENCODE version used, cbGenes can determine the most likely version used from a list of gene identifiers:

```
cbGenes guess exprMatrix.tsv.gz human
```

The first column of this file should be the gene symbols of GENCODE gene IDs (e.g. ENSG00000184779).

7.12.3 Step 3: Download the gene model files

Once you know the version, download the appropriate files to your cellbrowserData directory:

```
cbGenes fetch gencode-34         # geneId -> symbol mapping for human gencode relase 34
cbGenes fetch hg38.gencode-34    # gene -> chrom mapping for human gencode relase 34
```

Both files are required for this to work.

7.12.4 Step 4: Set up your cellbrowser.conf

You will need to add the following lines to your `cellbrowser.conf`:

```
atacSearch = "hg38.gencode-34" # Version downloaded in Step 3 combined with the UCSC␣
↪assembly name
geneLabel = "Peak"
```

You will still need to set the other required settings in your cellbrowser.conf as well

7.12.5 Step 5: Build your Cell Browser

After all is set up, build your cell browser:

```
cbBuild -o alpha
```

CHAPTER 8

With text files

The generic way to set up your own cell browser is to start from tab-separated (tsv) or comma-separated (csv) format text files. The steps on this page assume that you have already gone through the process of clustering your cells.

8.1 The files you will need

You will need the first three files described below in tsv or csv format, the fourth is optional:

1. **Expression matrix**: one row per gene and one column per cell, ideally gzipped. The first column must be the gene identifier or gene symbol, or ideally geneId|symbol. Ensembl/GENCODE gene identifiers starting with ENSG and ENSMUSG 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 first line of the file must be a header that includes the cell identifiers.
2. **Cell annotation metadata 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 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. To speed up processing of both your expression matrix and metadata file, these files should describe the same numbers of cells and be in the same order. This allows cbBuild process these files without needing to trim the matrix and reorder the metadata file. The metadata file also must have a header line.
3. **Cell coordinates**, 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 metadata file. If you have run multiple dimensionality reduction algorithms, you can specify multiple coordinate files in this format. The number rows in these coordinates doesn't need to match that of your expression matrix or metadata files, allowing you to specify only a subset of the cells. In this way, you can use a single dimensionality reduction algorithm, but include multiple subsets and view of the cells, e.g. one coordinates file per tissue. Note, if R has changed your cell identifiers (e.g. by adding dots), you may be able to fix them by running `cbTool metaCat`.
4. **Cluster-specific marker genes** (optional). The first column is the cluster name (from the cell annotation metadata file), the second column contains the gene symbol (or Ensembl gene ID, which will automatically be mapped to the gene 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. You can also run

`cbMarkerAnnotate` on this file to add information from various gene-centric databases and link-outs to other resources to your existing table. See Annotate genes on how to add link to external gene-databases (like Allan Brain Atlas or OMIM) to your marker genes. If you used Seurat for your clustering, you can just provide the raw Seurat marker gene output. You can also specify multiple files of cluster-specific marker genes, e.g. in case that you are also doing differential gene expression analysis or have results from multiple algorithms.

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
```

8.2 Setting up your `cellbrowser.conf`

After you have all of these files in place, go to the directory go to the directory containing all of these files and run the following command to copy a sample `cellbrowser.conf` into your current directory:

```
cbBuild --init
```

This sample `cellbrowser.conf` includes the required settings plus some other useful settings. The current list of all possible cellbrowser.conf statements can be found in our example cellbrowser.conf.

In your cellbrowser.conf, replace the default values in the config statements:

- `exprMatrix` - expression matrix file name
- `meta` - cell annotation metadata file name
- `coords` - coordinate file names with a layout method label for each
- `markers` - cluster-specific marker gene file namew with a label for each
- `labelField` and `clusterField` - name of cluster field from header line of metadata file

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 cell browser output directory (/tmp/cb in this 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 cell browser output directory /tmp/cb.

CHAPTER 9

With Seurat

There are a number of ways to create a cell browser using Seurat:

- **Import a Seurat rds file** - create a cell browser with the Unix command line tool `cbImportSeurat`.
- **Using RStudio and a Seurat object** - create a cell browser directly using the `ExportToCellbrowser()` R function.
- **Run our basic Seurat pipeline** - with just an expression matrix, you can run our `cbSeurat` pipeline to create a cell browser.

Each of these methods are described in more detail below.

9.1 Convert a Seurat `rds` or ".rdata' file

First, create an .rds file in R as described in the Seurat tutorial:

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

Next, on the Unix command line, use the `cbImportSeurat` script to convert this `rds` file into a cell browser:

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

This works with objects created by versions 2 and 3 of Seurat. `cbImportSeurat` can read both `.rds` and `.rdata` files, for *.rdata* it assume the first object is the Seurat object. Make sure that you have the same major version of Seurat installed that was used to create the object. You cannot open Seurat2 objects with Seurat3 or vice versa. (We often need to switch between Seurat versions and found conda environments very helpful for this.)

The `-i` option specifies the input `rds` file and the `-o` option specifies a name for the output directory. You can use the `-n` option to change the dataset name in the cell browser; if it is not specified, it will default to the output directory name.

A Seurat object does not contain the marker genes by default, as FindAllMarkers() does not save its output. You can add it to the object when you save the .rds file with a command like this:

```
object@misc$markers <- FindAllMarkers(object)
```

`cbImportSeurat` will then use these markers. Otherwise, if `misc$markers` is not present in the object, it will run FindAllMarkers with the default values (Wilcoxon and 0.25 as the cutoff). Alternatively, you can also save the markers to a tab-separated file yourself and provide this file with the `--markerFile` option.

Lastly, go into the `pbmc3kImport` directory and run `cbBuild` to create the cell browser output files:

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

Alternatively, you can use the `--htmlDir` option for `cbImportSeurat` to automatically run `cbBuild` for you:

```
cbImportSeurat -i pbmc3k_small.rds -o pbmc3kImport --htmlDir=~/public_html/cb
```

9.2 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, you can build a cell browser from this dataset into the `htdocs` directory, serve the result on port 8080 via http, and open a web browser from within 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)

9.3 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. To install miniconda, follow their installation instructions.

After setting up conda, install R:

```
conda install r
```

Then, 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
```

Now run the expression matrix `filtered_gene_bc_matrices/hg19/matrix.mtx` through Seurat:

```
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. After the `cbSeurat` script completes, you can build your cell browser from the output:

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

9.3.1 Changing the defaults using `seurat.conf`

This set of steps will run a basic Seurat pipeline with the default settings. You can modify the settings for Seurat by creating a `seurat.conf` file:

```
cbSeurat --init
```

You can edit the settings in `seurat.conf` and re-run the `cbSeurat` command to generate a new set of Seurat output using these new settings.

CHAPTER 10

With Scanpy

There area few different ways to create a cell browser using Scanpy:

- **Run our basic Scanpy pipeline** - with just an expression matrix and `cbScanpy`, you can the standard preprocessing, embedding, and clustering through Scanpy.
- **Import a Scanpy h5ad file** - create a cell browser from your `h5ad` file using the command-line program `cbImportScanpy`.
- **Use a few Python 3 function** - you can build a cell browser from a Scanpy `h5ad` file and start a web server, e.g. from Jupyter, with the Python3 function `cellbrowser.scanpyToCellbrowser(ad, outDir, datasetname)`.

10.1 A standard Scanpy pipeline

Requirements: Python3 with Scanpy installed, see their installation instructions for information about setting up Scanpy. As part of the Scanpy installion process, ensure that the igraph library is also installed. It's needed for the most basic scanpy features even though it's not an official requirement. The command `pip install scanpy[louvain]` will make sure that igraph is installed.

We provide a wrapper around Scanpy, named `cbScanpy`, which runs filtering, PCA, nearest-neighbors, clustering, t-SNE, and UMAP. The individual steps are explained in more detail in the Scanpy PBMC3k tutorial.

The output of `cbScanpy` is formatted to be directly usable to build a cell browser with `cbBuild`.

You can test `cbScanpy` yourself using the following set of steps. To process an example dataset, download the 10x pbmc3k expression matrix from our servers:

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

Next, run the expression matrix `filtered_gene_bc_matrices/hg19/matrix.mtx` through Scanpy:

```
cbScanpy -e filtered_gene_bc_matrices/hg19/matrix.mtx -o scanpyOut -n pbmc3k
```

This will run Scanpy and will fill the directory `scanpyOut/` with everything needed to create a cell browser. After the `cbScanpy` script completes, you can build your cell browser from the output:

```
cd scanpyout
cbBuild -o ~/public_html/cb -p 8888
```

10.1.1 Changing the defaults using `scanpy.conf`

This set of steps will run a basic Scanpy pipeline with the default settings. You can modify the settings for Scanpy by creating a `scanpy.conf`:

```
cbScanpy --init
```

You can edit the settings in `scanpy.conf` and re-run the `cbScanpy` command to generate a new set of Scanpy output using these new settings.

10.2 Convert a Scanpy `h5ad`

If you have run Scanpy and have an output `h5ad` file, you can import it into a cell browser using the command `cbImportScanpy`.

The steps in this section walk you through the process of importing data from a Scanpy file and then building a cell browser from the output. The steps use an example `h5ad` file available for a small pbmc dataset from our Github repo: anndata.h5ad.

First, use `cbImportScanpy` to extract the data from the `h5ad`:

```
cbImportScanpy -i anndata.h5ad -o pbmc3kImportScanpy
```

The `-i` option specifies the input `h5ad` file and the `-o` option specifies a name for the output directory. You can use the `-n` option to change the dataset name in the cell browser; if it is not specified, it will default to the output directory name.

The output of `cbImportScanpy` will be formatted so that you can immediately build a cell browser from it. Go into the pbmc3kImportScanpy directory and run `cbBuild` to create the cell browser output files:

```
cd pbmc3kImportScanpy
cbBuild -o ~/public_html/cb
```

Alternatively, you can use the `--htmlDir` option for `cbImportScanpy` to automatically run cbBuild for you:

```
cbImportScanpy -i anndata.h5ad -o pbmc3kImportScanpy --htmlDir=~/public_html/cb
```

10.3 Convert a Scanpy object

From Jupyter or Python3, you can create a data directory with the necessary tsv files and a basic `cellbrowser.conf`:

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

Here `adata` is your Scanpy object, `scanpyOut` is your output directory, and `myScanpyDataset` is your dataset name.

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 web server running already, use this function start one to serve up this directory:

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

You can stop the web server with the function:

```
cb.stop()
```

Or from a Unix shell, you can build and start a web server using `cbBuild`:

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

CHAPTER 11

With Cell Ranger

Find the `cellranger` **OUT** directory, it should contain an `analysis` directory and a subdirectory `filtered_gene_bc_matrices`. The **OUT** directory will be the input for our tool `cbImportCellranger`. The tool converts the `cellranger` files to ones formatted for `cbBuild`.

As we are reading Cell Ranger `mtx` files, we need the scipy package (add `--user` if you are not the admin on your machine):

```
pip install scipy
```

The example below use the pbmc3k `cellranger` output files from the 10x website. First, download the files with the command:

```
rsync -Lavzp genome-test.gi.ucsc.edu::cells/datasets/pbmc3kCellranger/ ./
↪pbmc3kCellranger/ --progress
```

Next, run `cbImportCellranger` to convert the `cellranger` files into something that can be used to build a cell browser:

```
cbImportCellranger -i pbmc3kCellranger -o cellrangerOut --name pbmc3k_cellranger
```

The `-i` option specifies the input `cellranger` directory and the `-o` option specifies a name for the output directory. You can use the `-n` option to change the dataset name in the cell browser; if it is not specified, it will default to the output directory name.

Lastly, go into the `cellrangerOut` directory and run `cbBuild` to create a cell browser:

```
cd cellrangerOut
cbBuild -o ~/public_html/cells -p 9999
```

CHAPTER 12

Advanced Topics

12.1 Cell browser updates and web server configuration

New features and bug fixes are being added to the UCSC Cell Browser software all the time. You can update the javascript files and re-create the index.html using the command line tool `cbUpgrade`. You need to use the `--code` option and the `-o` to specify the output directory, e.g. `cbUpgrade --code -o /var/www/cellbrowser/`.

Your web server should support byte-range requests. This isn't important for smaller, but for datasets with files larger than 30MB, a warning message will be shown once. Most web servers and web hosters support them by default. For Apache, byte-range requests are enabled by default but may need to be activated in some installations of nginx.

12.2 Default output directory for building cell browsers

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 (e.g. in your `~/.bashrc`) or by adding a line like this to `~/.cellbrowser.conf`:

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

12.3 Google Analytics

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 run `cbBuild` or `cbUpgrade` to rebuild your index.html, after which it should contain your Google Analytics tracking code.

12.4 Various `cellbrowser.conf` configurations

For a reference of all possible `cellbrowser.conf` statements, see the example conf

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 13

Annotate Genes

When you load a basic set of marker genes into your cell browser, they will be imported as gene symbols along with an associated score, with no extra annotations. To make the list of markers genes more useful, you can add extra annotations using the `cbMarkerAnnotate` tool. This tool will add information about in other resources that describe gene expression profiles (Allen Brain Atlas), diseases they have been linked to (OMIM, HPO, Sfari), and protein class (HPRD).

Running this script on your marker genes file is very simple:

```
cbMarkerAnnotate inFname outFname
```

The format for `inFname` is the same as for standard cell browser marker gene files, a tsv or csv table with at least three columns, in this order:

1. cluster - needs to match `labelField` in `cellbrowser.conf`.
2. gene - can be a gene symbol or Ensembl gene ID, with or without the version.
3. score - scores are typically "avg_diff" or "p-Value" or similar. Gene

`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`. You can then change the marker genes file described by the `markers` parameter in your `cellbrowser.conf` to point to `outFname`.

You can also add your own annotations (e.g. number of associated PubMed articles) to your marker genes files. These will be displayed alongside any other annotations that you may have added using `cbMarkerAnnotate`.

CHAPTER 14

cbTool: combine and convert your data

The script cbTool included in the Cell Browser package includes a number of utilties for combining or converting your data. These different functions and how to use them are described below.

14.1 Combining results

14.1.1 Metadata

You can use the cbTool metaCat utility to merge the metadata files from different sources or pipelines (e.g. cbScanpy or cbSeurat) into a single one. A command to do combine a metadata set of metadata files from Scanpy and Seurat with a separate one might look like this:

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

The resulting file will include the columns from all three of the original files combined into a new metadata file. Note that cbTool metaCat assumes that the first column of each file contains the same cell identifier that it can use to join them.

14.1.2 Matrices

Similar to metaCat for combining metadata files, cbTool matCat can be used to combine expression matrices from different sources. A command combine the two different matrixes would look like this:

```
cbTool matCat mat1.tsv.gz mat2.tsv.gz exprMatrix.tsv.gz
```

14.2 Converting mtx to tsv

Using cbTool mtx2tsv, you can convert your expression matrix in matrix market to tsv format (with one gene per line and one cell per column):

```
cbTool mtx2tsv matrix.mtx genes.tsv barcodes.tsv exprMatrix.tsv.gz
```

14.3 Fixing R Seurat output

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

——— REQUIRED SETTINGS ———

example config file with all possible settings

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", "flipY" : False, # R/Matplotlib files need to be flipped on the Y-axis "shortLabel":"t-SNE on WGCNA"

}, {

"file":"subset.coords.tsv", "shortLabel":"neural cells", # you can overlay lines onto the cells, table has to have columns named x1, x2, y1, y2 "lineFile" : "lines.tsv", # you can flip the y-axis of just the lines, relative to the points # This was necessary for a user when using the files produced by the URD pseudotime package #"lineFlipY" : True, # you can automatically switch on coloring on a meta data field whenever a layout is activated "colorOnMeta":"neuralCluster"

},

]

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"

——— OPTIONAL SETTINGS ———

The settings below are used to create filters on a Cell Browser instance (e.g. cells.ucsc.edu). # Note, all filter values can be a list (e.g. body_parts = ["brain","cortex"]) body_parts = [""] # body_parts = ["heart"] organisms = [""] # organisms = ["Human (H. sapiens)", "Mouse (M. musculus)"] diseases = [""] # diseases = ["Healthy"] projects = [""] # projects = ["GTEx", "Human Cell Atlas", "hca"]

genes that are highlighted in your paper can be pre-loaded and are shown as a clickable table on the left # this is optional but we highly recommend that you define at least 2-3 quick genes, it makes the browser a lot # more intuitive for users quickGenesFile = "quickGenes.csv"

if you want to enforce some order of the values of your enums, e.g. your cluster annotation should be sorted # in a given order in the display, supply a text file with the values in the right order, one per line. # You can supply one text file per meta data field # enumOrder = { "WGCNAcluster" : "clusterorder.txt" }

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"}

]

do not show this dataset on the dataset list. This can be used for pre-publication data. # visibility="hide"

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 acronymFile = "acronyms.tsv"

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"

format of the numbers in the matrix. # 'auto' works in 99% of the cases. Otherwise you can use 'int' for integers and 'float' for floating point numbers. # Use 'forceInt' if your matrix contains only integers but in a format like 3.123e10 # or the matrix has only integers expressed like 100.000, 200.000, 300.00, ... matrixType='auto'

rarely needed: if your expression matrix does not contain genes, but something # else, like "lipids" or "plankton", you can replace the word "gene" in the # user interface with another word # geneLabel = "Lipid"

the default color palettes for this dataset. By default, we use Paul Tol's # but you could use other ones, see the URL when you change the palette to see possible values # defQuantPal = "viridis" # defCatPal = "rainbow"

you can optionally show little images for clusters on the tooltip. # For now, they have to be PNGs. # For now, you will have to copy these images to the source destination htdocs directory manually # right now, only brain-lipids/all-lipids is using this # clusterPngDir = "clusterImgs"

Enter organs/organ parts here in a comma-separated list. Organs entered will show up as filter options for dataset list. # Really only useful if you have a cell browser with multiple datasets covering multiple organs (e.g. cells.ucsc.edu) # If no datasets have 'body_parts setting, filter won't be displayed. # body_parts=["eye","cornea"]

CHAPTER 15

Describing datasets

The file `desc.conf` is a key-value file, similar to `cellbrowser.conf`, but it describes the dataset.

A sample file can be created with the command `cbBuild --init` or be copied from our Github repo. The tags in the file refer to either HTML files or directly contain the relevant text, URL, accession or in rare cases key/value information.

The most important tags are `title`, `abstract`, `methods`, `unitDesc`, `image`, `pmid`, `paper_url`, `lab` and `submitter`. All tags are described below and they are sorted by content.

These tags contain longer text and 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.

These tags contain a file name:

- `image`: usually a 400px-wide thumbnail of the dimensionality reduction. You can use a command like `convert graphical_abstract.png -sampling-factor 4:2:0 -strip -resize 400 -quality 85 -interlace JPEG -colorspace sRGB thumb.jpg` to create `thumb.jpg`, a version of the image only 400 pixels wide that loads faster.
- `rawMatrixFile`: the file name of the raw unprocessed matrix. Usually a .zip or .gz file. Also see `rawMatrixNote`.

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 (remember that in addition to a string, they can also be a list of strings, in the usual JSON format e.g. ['123', '234']):

- `pmid`: Pubmed ID of the publication (CIRM TagsV5)
- `geo_series`: NCBI GEO series ID (CIRM TagsV5)
- `sra`: NCBI SRA accession
- `arrayexpress`: EBI Arrayexpress accession
- `ena_project`: EBI ENA project accession, ENAPxxxx
- `sra_study`: NCBI SRA SRPxxxx accession
- `doi`: DOI of paper fulltext
- `dbgap`: NCBI dbGaP accession, starts with phs
- `bioproject`: NCBI Bioproject accession, PRJNAxxxx. Can be included with or without the PRJNA prefix.
- `ega_study`: EGA accession
- `cirm_dataset`: CIRM CDW dataset name

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
- `rawMatrixNote`: text to describe the raw matrix, see `rawMatrixFile`
- `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
- `wrangler` and `shepherd`: these are mostly used at UCSC. We store the name of the person of our team who loaded the data (wrangler) and sometimes the name of the person who was in contact with the lab and did quality control on the data (shepherd).

The following tags contain key-value information:

```
- ``custom``: anything about the dataset that does not have an existing tag, e.g. {
↪'taxon_id':'9606'}
- ``urls``: any url you want to show, e.g. { 'Raw data on synapse':'https://www.
↪synapse.org/#!Synapse:syn21560407' }.
- ``algParams``: algorithm parameters, e.g. { 'louvainRes':'0.7' }. This tag is␣
↪generated by cbScanpy.
```

The following tags contain a list of key-value information:: - `supplFiles`: additional files that should be copied and be shown in the 'Download' section, like protocols, Seurat

or Scanpy files, e.g. `supplFiles=[{'file':'seurat3.rds', 'label':'Seurat3 RDS'}]`

The folllowing tags contain only `True` or `False`:

```
- ``hideDownload=True``: do not show the download instructions.
```

CHAPTER 16

Dataset Hierarchies

The Cell Browser allows you to group related datasets into a hierarchy, where datasets are grouped into collections, like files are grouped into directories. When you open a collection, it will show you all of the datasets within it.

This requires your datasets to be arranged in directories on disk. Let's say you have two directories with data files, one in directory `data1` and one in directory `data2`, each with their own cellbrowser.conf files, then these two directories must be both subdirectories of a parent directoryn named e.g. `dataParent`. The names of all the datasets are the names of their directories, not the names specified via the `name` statement in their `cellbrowser.conf` files anymore. Specified names from cellbrowser.conf are ignored when dataset hierarchies are used and are replaced with their directory names.

To enable dataset hierarchies, you only have to add a single line pointing to the top-level parent directory where all of your single-cell data lives. Add a statement like the following to your `~/.cellbrowser.conf`:

```
dataRoot='/celldata/'
```

Alternatively, `dataRoot` can be set using the `CBDATAROOT` environment variable:

```
export CBDATAROOT='/celldata/'
```

Then, create a "stub" cellbrowser.conf into this directory, it should only contain a single line like `shortLabel="some description"`. You can describe your collection as discussed under the **Describing datasets** section. Put the `desc.conf` file into the same directory as the `cellbrowser.conf` you just created. Define at least the statements `title` and `description`. They will be shown at the top of your dataset list. This directory can be called the top-level collection.

Arrange your dataset directories under this directory. You can add empty directories, which will become collections, by creating dataset directories in them and put a `cellbrowser.conf` and `desc.conf` into it, e.g. like this:

```
mkdir -p /celldata/organoids
cd /celldata/organoids
echo 'name="organoids"' > cellbrowser.conf
echo 'shortLabel="Brain Organoids"' >> cellbrowser.conf
echo 'tags=["10x"]' >> cellbrowser.conf
```

Now you can run `cbBuild` in each subdirectory of a collection. in the collection. Or you can rebuild in all subdirectories using `cbBuild -r`.

If you view the cell browser now using a web browser, you should see this new collection present. When viewing a dataset in a collection, you can move quickly to any other dataset in the same collection using the "Collection" dropdown menu in the toolbar.

CHAPTER 17

Loading a dataset

Once you have found a dataset of interest on https://cells.ucsc.edu, it is very easy to load it into your favorite analysis environment. (Let us know if the commands below do not work in your environment.)

First, download the expression matrix and the meta data, usually in a Unix terminal:

```
mkdir adultPancreas
cd adultPancreas
wget https://cells.ucsc.edu/adultPancreas/exprMatrix.tsv.gz
wget https://cells.ucsc.edu/adultPancreas/meta.tsv
```

Replace "quakePancreas" above with the dataset name of interest, it is shown in the URL when you open a dataset after "ds=" or in the download instructions or on the dataset page as the "CellBrowser dataset identifier".

Then open your favorite tool (e.g. RStudio or Jupyter) and follow the instructions below.

17.1 Seurat

Run these commands if you have downloaded the file as above:

```
require(Seurat)
require(data.table)
setwd("adultPancreas")
mat <- fread("exprMatrix.tsv.gz")
meta <- read.table("meta.tsv", header=T, sep="\t", as.is=T, row.names=1)
genes = mat[,1][[1]]
genes = gsub(".+[|]", "", genes)
mat = data.frame(mat[,-1], row.names=genes)
so <- CreateSeuratObject(counts = mat, project = "adultPancreas", meta.data=meta)
```

Or you can download directly into R, without wget, by replacing the fread and read.table commands above in line 4 and 5 with these:

```
mat <- fread("https://cells.ucsc.edu/adultPancreas/exprMatrix.tsv.gz")
meta <- data.frame(fread("https://cells.ucsc.edu/adultPancreas/meta.tsv"), row.
↪names=1)
```

If your version of data.tables does not support .gz yet, the fread commands can be changed to this:

```
# from current directory
mat <- fread("zcat < exprMatrix.tsv.gz")
# or direct download:
mat <- fread("curl https://cells.ucsc.edu/adultPancreas/exprMatrix.tsv.gz | zcat")
```

If the matrix name is not `exprMatrix.tsv.gz` but `matrix.mtx`, you have to use Seurat's MTX loader. In addition to `matrix.mtx`, make sure to also download the files `barcodes.tsv` and `genes.tsv` sometimes called `features.tsv`. If you downloaded these three files and `meta.tsv` into a directory `downloadDir`, load them like this:

```
require(Seurat)
setwd("downloadDir")
mat = Read10X(".")
meta = read.table("meta.tsv", header=T, sep="\t", as.is=T, row.names=1)
so <- CreateSeuratObject(counts = mat, project = "myProjectName", meta.data=meta)
```

17.2 Scanpy

To create an anndata object in Scanpy if the expression matrix is a .tsv.gz file:

```
import scanpy as sc
import pandas as pd
ad = sc.read_text("exprMatrix.tsv.gz")
meta = pd.read_csv("meta.tsv", sep="\t")
ad.var = meta
```

If the expression matrix is an MTX file:

```
import scanpy as sc
import pandas as pd
ad = sc.read_mtx("matrix.mtx.gz")
meta = pd.read_csv("meta.tsv", sep="\t")
ad.var = meta
```

Some datasets use the format identifier|symbol for the ad.obs gene names (e.g. "ENSG0123123.3|HOX3"). To keep only the symbol:

ad.obs.index = [x.split("|")[1] for x in ad.obs.index.tolist()]

CHAPTER 18

Bulk downloads

18.1 The dataset hierarchy

There is no search API yet but a systematic way to download:: 1) http://cells.ucsc.edu/dataset.json contains the list of top-level datasets in the "datasets" attribute. 2) every object in the "datasets" list is a child dataset, note the "name" attribute. 3) every dataset has a <datasetname>/dataset.json file again, e.g. http://cells.ucsc.edu/adultPancreas/dataset.json 4) datasets with a "datasets" attribute have children themselves, e.g. http://cells.ucsc.edu/xena/dataset.json

18.2 Rsync download

If you want to download all or selected datasets, use our `rsync` server. It is still being tested, please email us for more info.

CHAPTER 19

Microscopy images

The Cell Browser has some basic support for showing microscopy images for your dataset. The basic model is the one of sets of images (e.g. stainings for different markers of the same section) arranged into categories (e.g. stages or tissues). Some images that can be offered for download-only (e.g. TIFFS) and some images can be shown on the screen, these are typically reduced size versions in JPEG format with sizes of around 1-4k pixels. You provide a JSON file with the description of the images. The Cell Browser then shows an overview of the images on the "Info & Downloads" dialog box.

The default filename is `images.json` and has to be placed in the same directory as the desc.conf file. Here is an example images.json file. Contact us if you have questions or feedback on this basic system.

In the example below, there is only one image category called "main category", (there could be several, by duplicating the top-most object and changing the categoryLabel in the copy). This category has two categoryImageSets, the first has the label "DAPI SOX2 TBR2 DCX CTIP2". This imageSet offers two files for download and one file for display. The next imageSet has the label "AnotherImageSet" and also two files for download and one for display.

```
"categoryLabel": "main category", "categoryImageSets": [
  { "setLabel": "DAPI SOX2 TBR2 DCX CTIP2", "downloads": [
      { "label": "all merged", "file": "images/cs13-dapi-488-dcx-546-sox2-594-
        tbr2-647-ctip.tif"
      }, {
        "label": "DAPI", "file": "images/cs13-DAPI-dcx-sox2-tbr2-ctip2.tif"
      }
    ], "images": [
      { "label": "all merged", "file": "images/cs13-dapi-488-dcx-546-sox2-594-
        tbr2-647-ctip.5000.jpg" "thumb": "images/cs13-dapi-488-dcx-546-sox2-
        594-tbr2-647-ctip.500.jpg"
      }
    ],
  }, {
```

```
        "setLabel": "AnotherImageSet", "downloads": [
          { "label": "dicom file1", "file": "images/test1.dicom"
          }, {
            "label": "dicom file2", "file": "images/test2.dicom"
          }
        ], "images": [
          { "label": "overview", "file": "images/testMerge.jpg" "thumb": "images/testMerge.thumb.jpg"
          }
        ]
      }
]
```