Supplementary Figure 1

Image Data Storage Components: **~1.28 TB, 6.4×10^5 objects**
40,000-component RNAi library (416 x 96 well plates)
4 image fields per well
4 different wavelengths
640,000 image planes
= ~1.28 TB data (640,000 images * 2MB/image)

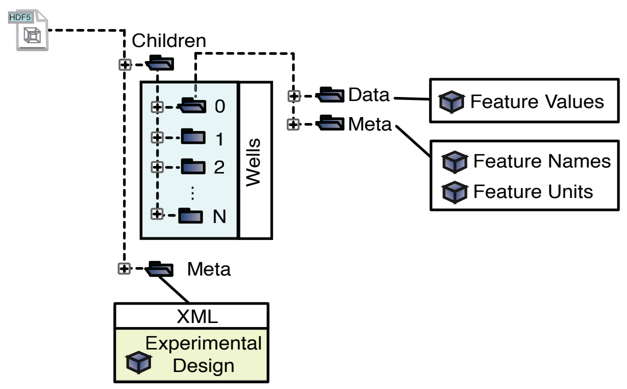
Extracted Data Storage Component: **~0.16 TB (12.5% of 1.28TB)**
4MB/well (empirically determined via an average dataset)
= 160 GB data (4MB/well * 40,000 wells)

Extracted Data Elements: **~8×10^8 objects (1250% of 6.4×10^5 objects)**
40,000 wells
4 fields/well
2,500 cells/field
2 compartments/cell
= 800,000,000 objects

Supplementary Figure 1: Details of calculations for full genome RNAi image-based screens.

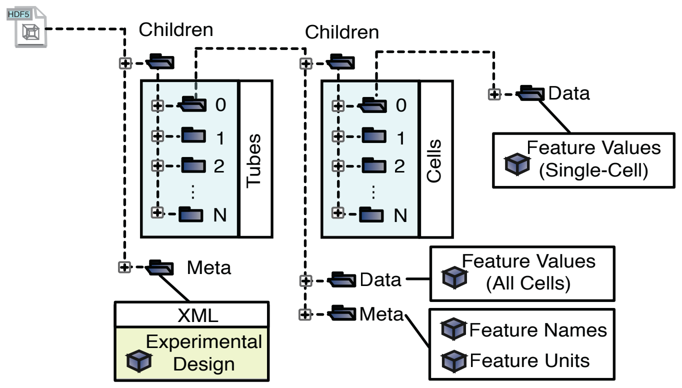
Supplementary Figure 2

a HT-Biochemical Assays/mRNA microarray



Hierarchy Level	0 Well
Data	well_means well_stdevs
Meta	XML feature_names feature_units
Raw	Source Files

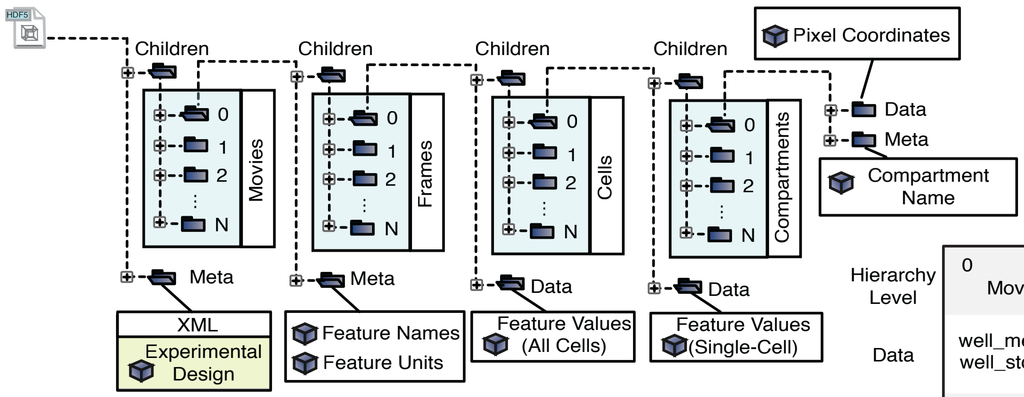
b Flow Cytometry



Hierarchy Level	0 Tube	1 Cell
Data	sample_mean sample_stdev	Feature Values (Single-Cell)
Meta	XML feature_names feature_units	
Raw	FCS file	

c

Live-Cell Microscopy

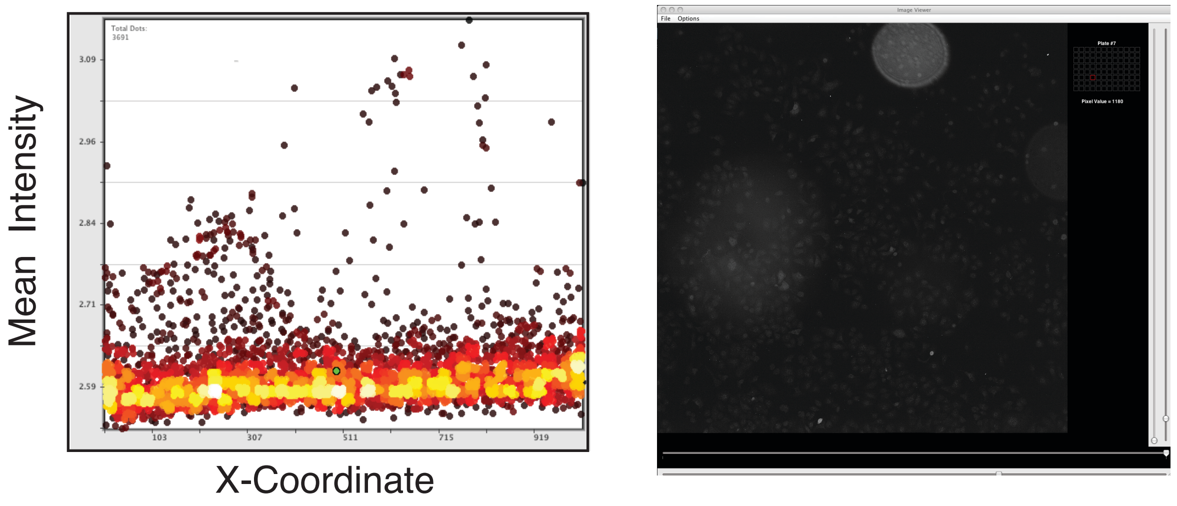


Hierarchy Level	0 Movie	1 Frame	2 Cell	3 Compartment
Data	well_means well_stdevs	frame_means frame_stdevs	feature_values	pixel_coords
Meta	XML plate_well feature_names feature_units			compartment name
Raw		TIFFs		

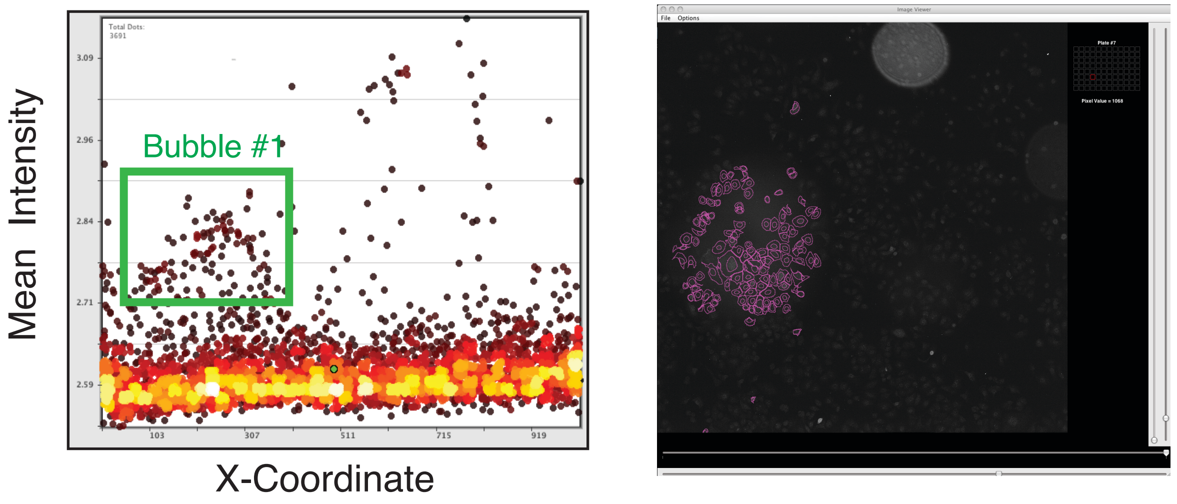
Supplementary Figure 2: Detailed description of various HDF5 structures taken from SDCubes applied to other biological assays in an illustration (left) and table format (right). **(a)** High-throughput biochemical assays and mRNA microarrays naturally fit into a single level of hierarchy, whereas assays with more data resolution such as **(b)** flow cytometry or **(c)** live-cell microscopy need multiple levels.

Supplementary Figure 3

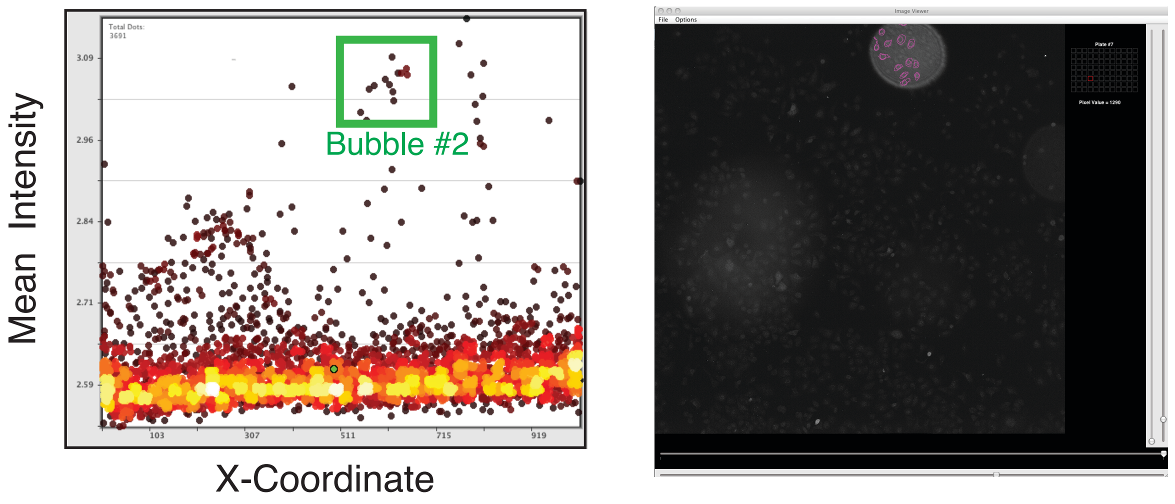
a



b



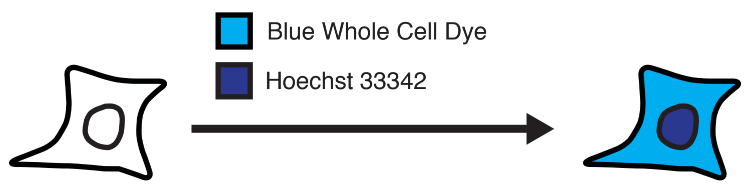
c



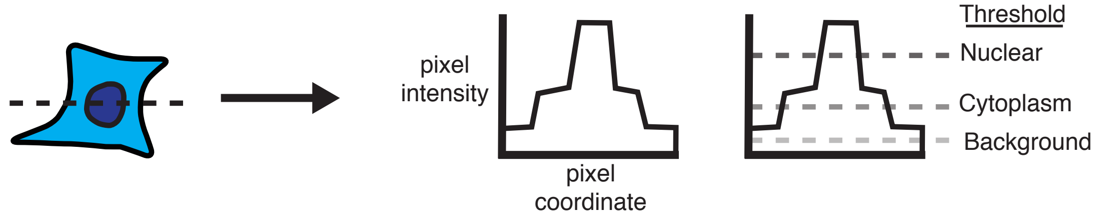
Supplementary Figure 3: Example of artifact filtering via scatter plots. **(a)** Scatter plot of cells from an image (right) plotting the x-coordinates of each cell on the X-axis and mean image intensity on the Y-axis. **(b)** Selection of a subset of cells in the scatter plot with elevated mean intensity values (green box) show they represent a bubble in the corresponding image (right). **(c)** Second example of cells that are actually an optical artifact caused by a bubble.

Supplementary Figure 4

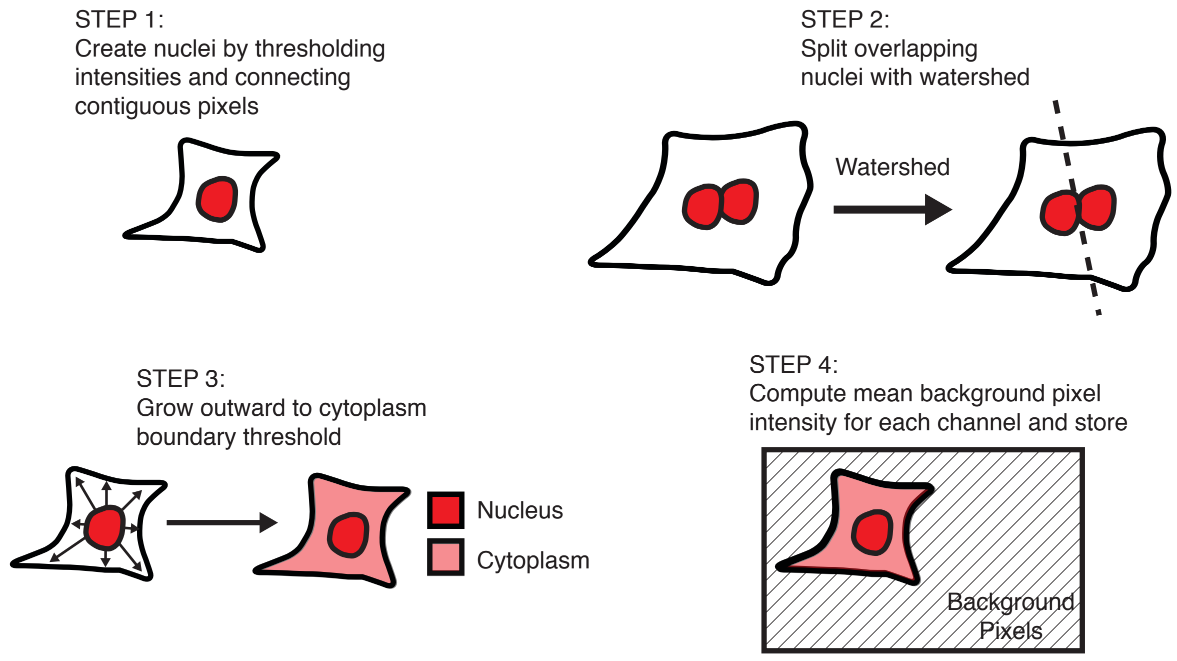
a



b

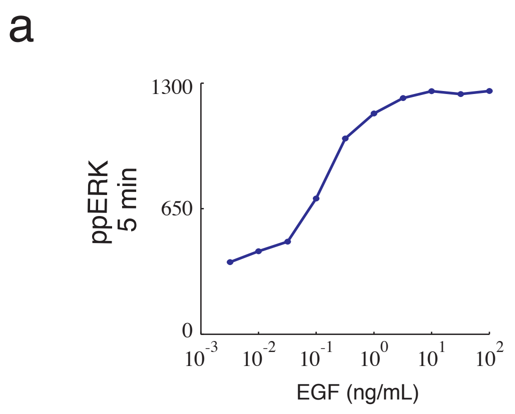


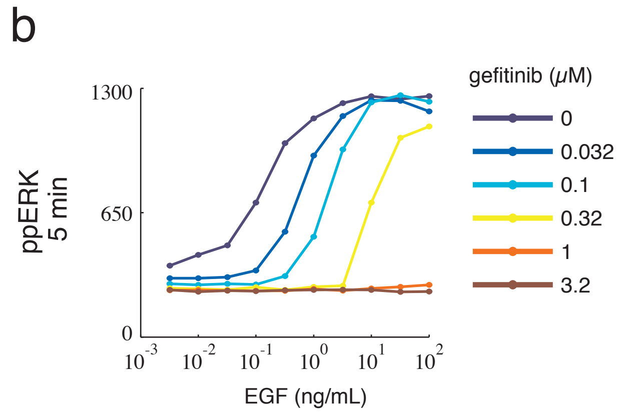
c

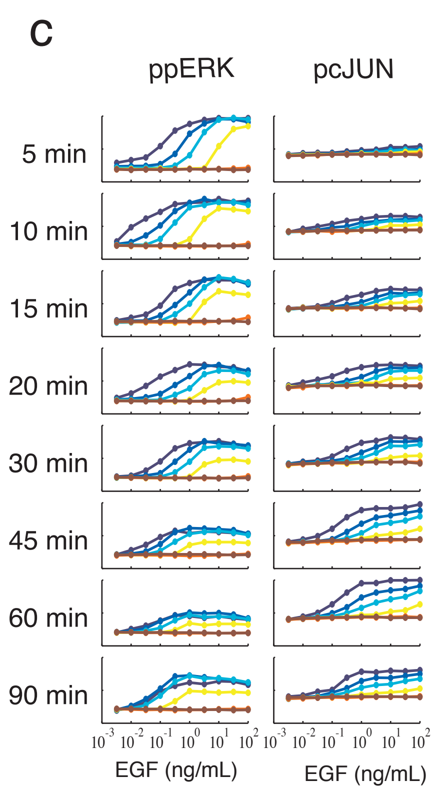

STEP 1:
Create nuclei by thresholding intensities and connecting contiguous pixels
STEP 2:
Split overlapping nuclei with watershed
Watershed
STEP 3:
Grow outward to cytoplasm boundary threshold
Nucleus
Cytoplasm
STEP 4:
Compute mean background pixel intensity for each channel and store
Background Pixels

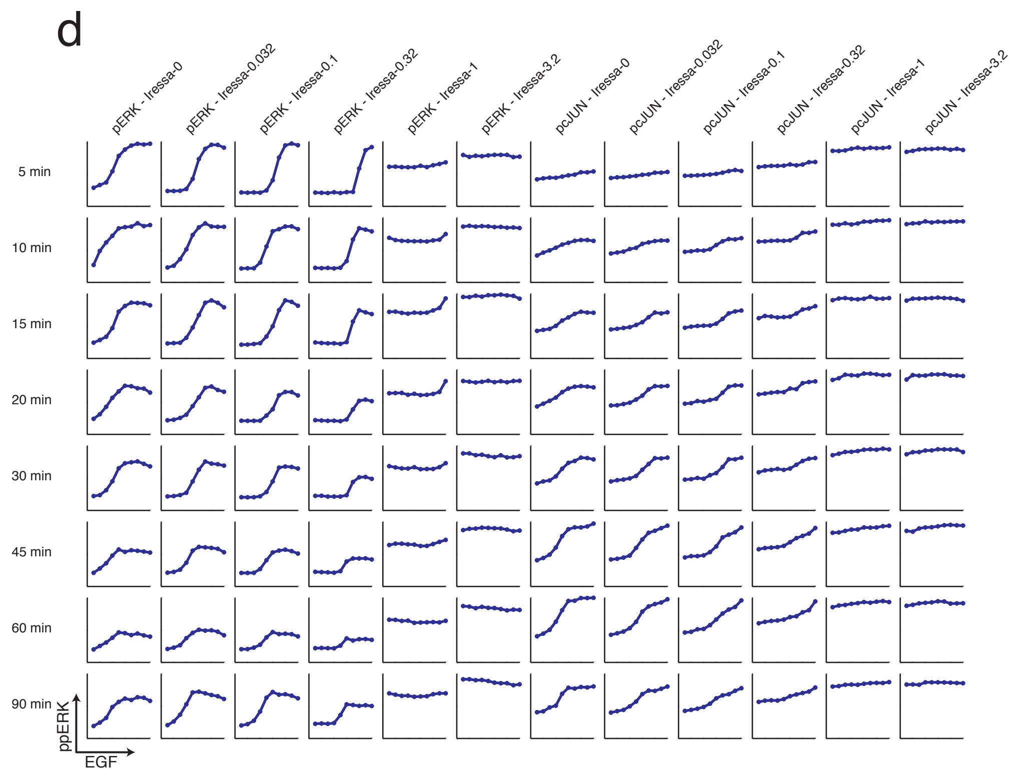
Supplementary Figure 4: Details of segmentation process used by *ImageRail*. **(a)** Cells are stained with a DNA and protein stain. **(b)** The stains emit at the same wavelength but have significantly different intensities at the dilutions used. This allows for different intensity thresholds to be identified for nuclei, cytoplasm and background within a single fluorescent channel. **(c)** The segmentation algorithm uses these threshold parameters to identify nuclei (step 1), attempt to cut apart overlapping nuclei (step 2), grow cytoplasms outward from the nuclei seeds to cytoplasm threshold levels (step 3) and compute the mean background intensities for each image channel of each field (step 4).

Supplementary Figure 5



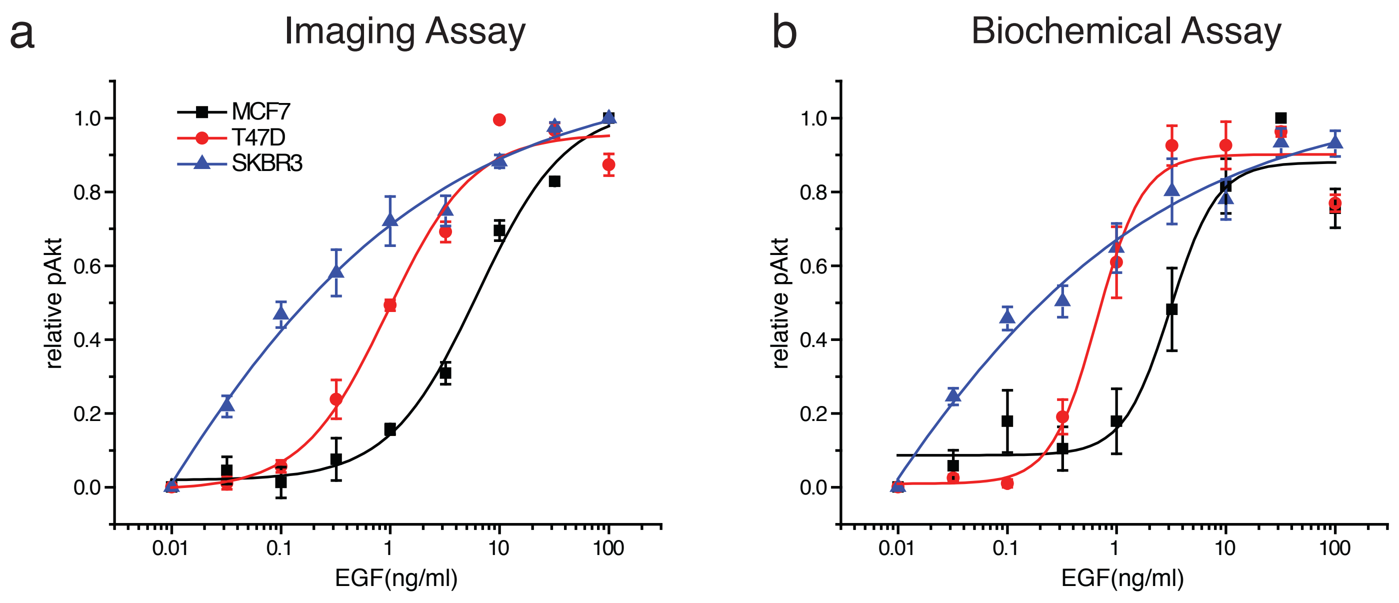


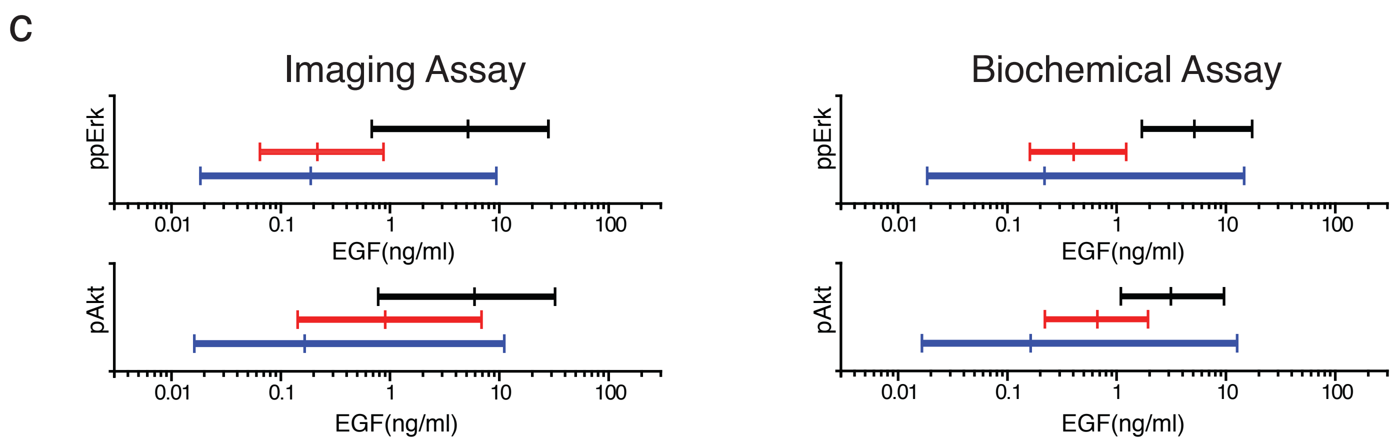




Supplementary Figure 5: Storage of high-throughput data in SDCubes facilitates storage of high dimensionality data sets and allows rapid plotting through the use of publicly available tools like DataPflex. (a) ppERK was measured in cells treated with increasing doses of EGF up to 100 ng/ml for 5 min. To enable DataPflex to plot all available data, controls not treated with EGF are plotted as receiving 0.003 ng/ml in all panels. (b) Cells treated as in a were pre-incubated with increasing doses of gefitinib. (c) Cells treated as in b were treated with EGF for different times. pcJUN was measured in addition to ppERK. (d) The same data as in c is presented as individual plots.

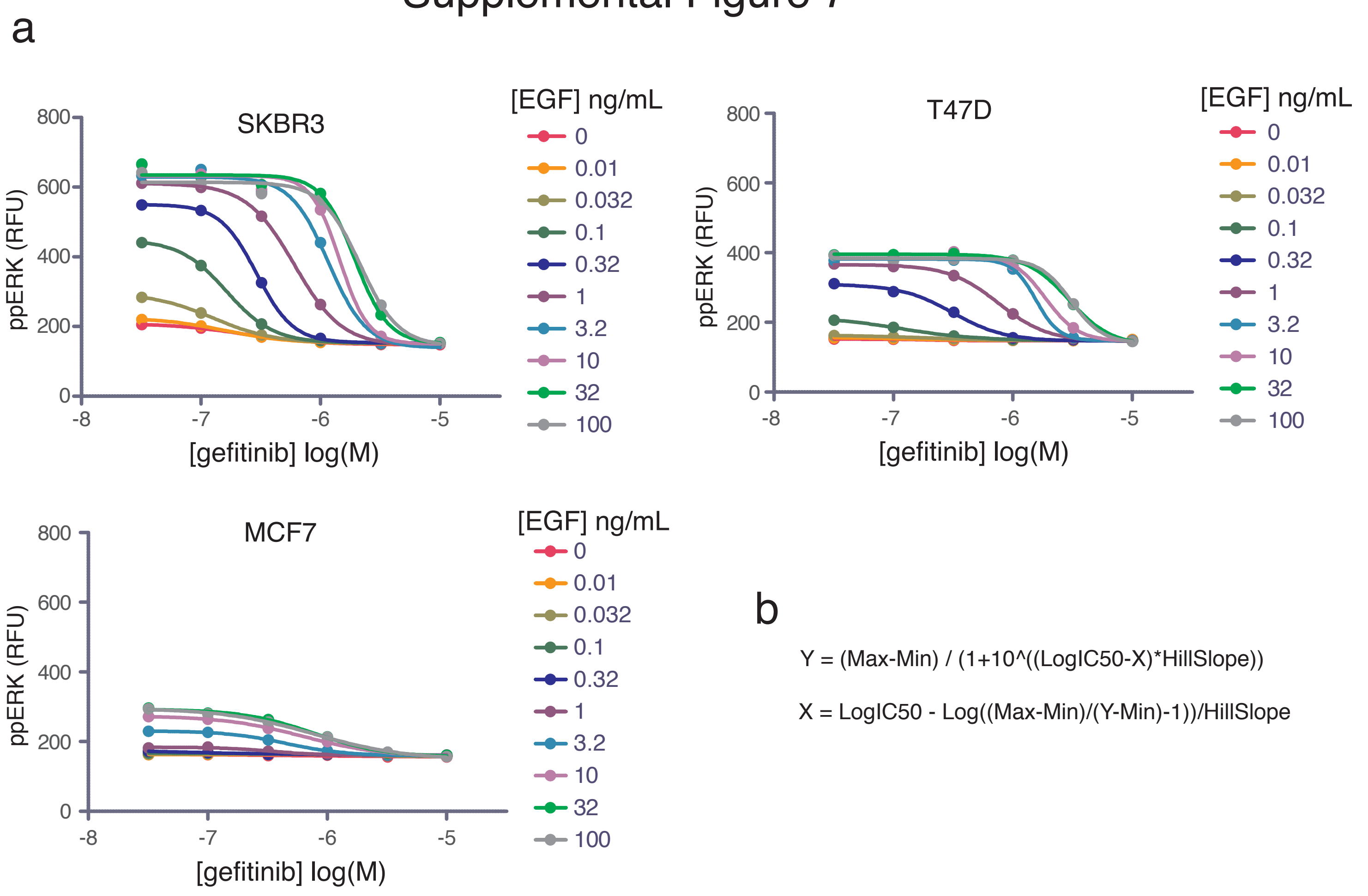
Supplementary Figure 6





Supplementary Figure 6: The high throughput imaging assay agrees well with a biochemical bead-based ELISA assay (Bioplex). **(a)** High-throughput microscopy measurements of EGF induced AKT kinase phosphorylation in three cell lines: SKBR3, T47D and MCF7, measured at 10 min post stimulation. **(b)** Corresponding measurements for same treatments as **a** measured by bead-based ELISA. **(c)** Comparison of the EC10, EC50 and EC90 values for both ppERK (shown in **Fig. 4c**) and pAKT displayed as high-mid-low whisker plots for both the imageWoRx and Bioplex.

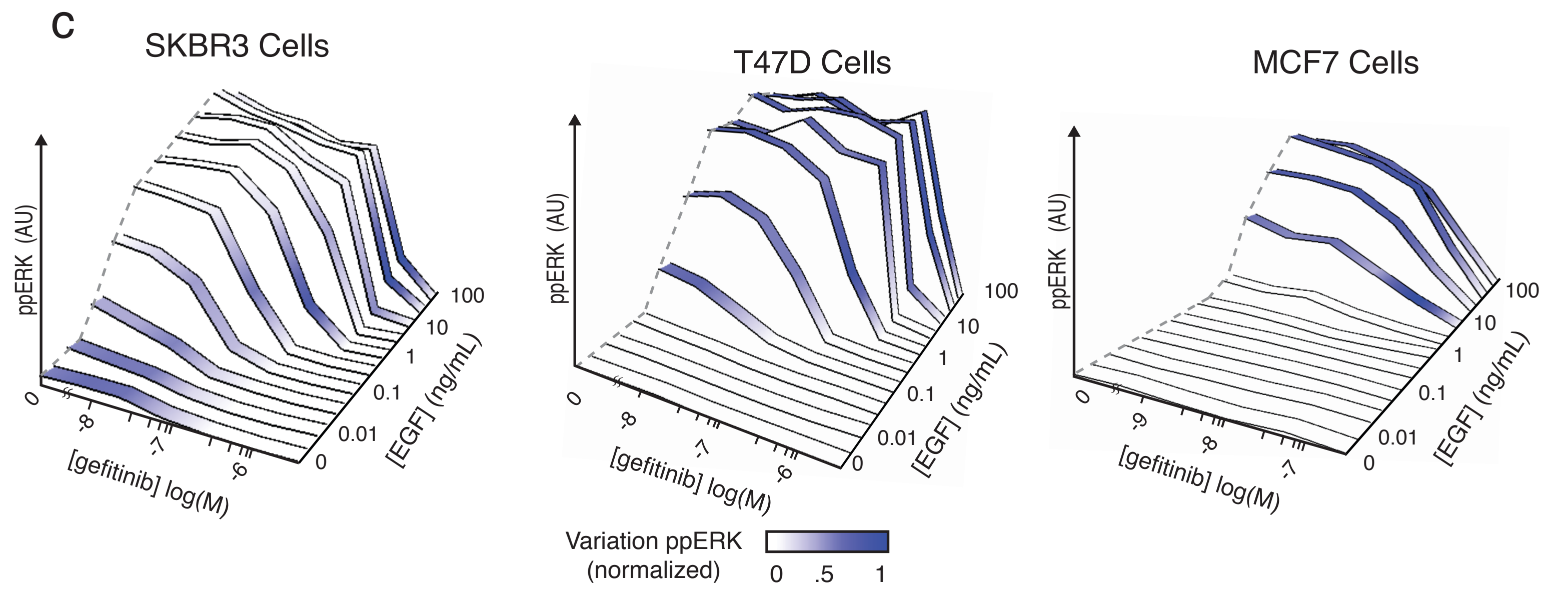
Supplemental Figure 7



b

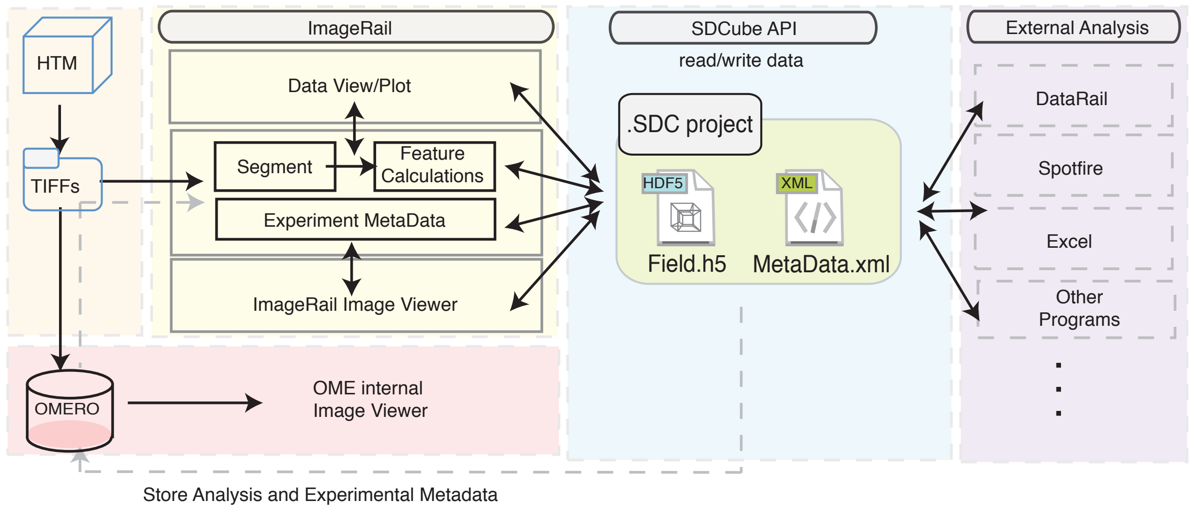
Y = (Max-Min) / (1+10^((LogIC50-X)*HillSlope))

X = LogIC50 - Log((Max-Min)/(Y-Min)-1))/HillSlope



Supplementary Figure 7: Details of curve fitting and IC10, IC50 and IC90 value extraction. **(a)** Gefitinib ppERK-inhibition dose response curves were fitted using a non-linear equation model using *GraphPad Prism* software (http://www.graphpad.com/prism/Prism.htm) for each dose of EGF in all cell lines (SKBR3, T47D and MCF7). **(b)** Equation used by software for the nonlinear inhibition curve fit (top). The equation was inverted (bottom) to determine the IC10, IC50, and IC90 concentration values. **(c)** Comparison of SKBR3, T47D and MCF-7 cell lines ERK phosphorylation response surfaces over concentration ranges of EGF and gefitinib where color represents extent of cell-to-cell variation.

Supplementary Figure 8



Supplementary Figure 8: The *SDCube* data format serves as an interface between analysis software programs with the input/output supported by the *SDCube Programming Library*. *ImageRail* (yellow) analyzes TIFF images produced by high-throughput microscopes (orange), segments cells, extracts cellular features and stores the resulting data into an *SDCube* project (blue). The *ImageRail SDCube* can be visualized and read either by internal *ImageRail* viewers or other software packages that help with data storage (pink) and data analysis (purple).

Supplementary Table 1

References to image analysis tools:

Definiens, http://www.definiens.com

Metamorph, http://www.moleculardevices.com/pages/software/metamorph.html

Cellomics, http://www.cellomics.com

ImageJ, http://rsbweb.nih.gov/ij/

CellProfiler, http://www.cellprofiler.org

ImageRail, http://code.google.com/p/sbpipeline/downloads/

Supplementary Note 1

SDCube

Programming Library Manual – v1.0.1

Software for the creation and manipulation of semantically typed data hypercubes (*SDCubes*)

Bjorn L Millard[1], Mario Niepel[1], Michael P Menden[1,3], Jeremy L Muhlich[1], and Peter K Sorger[1,2]

January 11, 2011

[1]Center for Cell Decision Processes, Department of Systems Biology, Harvard Medical School, Boston, MA 02115, USA, [2]Department of Biological Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139, USA [3]Current address: University of Applied Sciences Weihenstephan-Triesdorf

Email: imagerail-users@googlegroups.com
Web: http://www.semanticbiology.com/software/sdcube

License

Contents

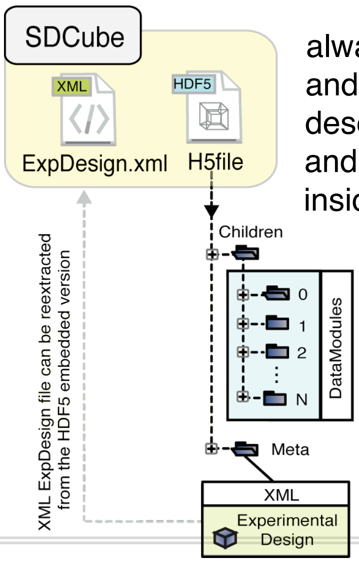
1 Introduction

This short document describes software for creating and semantically-typed data hypercubes (SDCubes), a data structure first described by Millard B, Niepel M, Menden M, Muhlich J, Sorger PK. *Adaptive informatics for multi-factorial and high content biological data. Nature Methods* (2011) Additional resources and periodic updates can be found at: http://www.semanticbiology.com/software. Further details on a software program that creates and manipulates SDCubes (*ImageRail*), can be found in the ImageRail Software Manual available at the same *Semantic Biology* Web site. Developers interesting in using *SDCubes* are advised to refer to *ImageRail* software for an example implementation (also open-source under GPL)

SDCubes are a data structure built from the HDF5 and XML file formats. SDCubes were developed to efficiently store data arising in molecular and cellular biology but are designed be adaptable to other data storage needs. However, the full benefits of SDCubes will become apparent when more programs can read and manipulate HDF5-encoded data and the XML-encoded experimental design data to analyze and navigate efficiently through the digitized single-cell data. Currently, ImageRail can create, modify and read SDCubes, but more advanced functional analysis and plotting of SDCubes will be handled by a sister program, DataRail v2.0, which is currently under development [2].

1. SDCube

An *SDCube* consists of 2 files: 1) a single HDF5 file, always called Data.h5, containing the acquired data values, and 2) an associated XML file called ExpDesign.xml, that describes how each of the data samples have been perturbed and what was measured. These two files are located together inside of an arbitrarily named parent folder that has a file extension ".sdc". The associated XML file is *also* stored inside of the HDF5 file as a byte array dataset at the relative path: "./Meta/ExpDesign.xml". If the XML and HDF5 files ever get separated or if the XML integrity is questionable, the embedded XML can be extracted from the HDF5 with the given Java-SDCube API method call:



<u>How to extract embedded byte[] files from the Data.h5 file:</u>

```
// Use the H5IO object to pull out an HDF5 embedded file
H5IO io = new H5IO();
Io.readFileFromHDF5(hdfFilePath, relativePathInHDF5, destinationFilePath);
```

The SDCube is composed of a collection of SDCube_Samples objects in Java, but as 2 separate files (HDF5-XML) on disk. Below describes common Java code to create, manipulate, write and read SDCubes:

Creating an SDCube

```
 // Initialize a sdcube without a path
 SDCube sdc = new SDCube();
// Initialize a sdcube with a path
 SDCube sdc = new SDCube(sdcPath);
```

SDCube I/O

```
// Set the file path of the sdcube
 sdc.setPath(path);
// Write the SDCube to the currently set Path
 sdc.write();
// Write the SDCube to the given Path
 sdc.write(sdcubePath);
// Loads the SDCube from the currently set Path
 sdc.load();
// Loads the SDCube from the given Path
 sdc.load(sdcPath);
```

Getting info from SDCubes

```
 SDCube_DataModule data = sdc.getRootDataModule(sdcPath);
 ArrayList<ExpDesign_Sample> expDs = sdc.getExpDesigns();
```

Adding samples to the SDCube

```
 sdc.addSample(SDCube_Sample sample);
 sdc.addSamples(ArrayList<SDCube_Sample> samples);
```

Getting Samples from the SDCube

```
 sdc.getSampleIDs(sdcPath);
 sdc.getNumSamples(sdcPath);
 sdc.getSamplesWithDescriptorNames_AND(sdcPath, names);
 sdc.getSamplesWithDescriptorNames_OR(sdcPath, names);
       sdc.getSample(sdcPath, id);
```

2. DataObjects

Data stored in the Data.h5 file are contained within HDF datasets of various data types and dimensionality. The Java-SDCube API creates, writes and reads these datasets as Java data objects that implement the *DataObject* interface. The 1- and 2-dimensional data objects are called *Data_1D* and *Data_2D* respectively. Currently only 1- and 2-dimensional *DataObjects* have been implemented in this release of the API since they are capable of storing nearly all data encountered in biological applications when used in combination with HDF5 groups. It is trivial for other developers to create N-dimensional *DataObjects* through the use of the provided Java interface.

How to create and write DataObjects:

```
// Create an array or matrix of any given data type (not primitives!)
Float[] floatArray = ...
// Create the DataObject with the data, dataType, and dataName
Data_1D data1d = new Data_1D(floatArray, "FLOAT", "name");
//Write the dataset to the given relativePath of the given h5FilePath
new H5IO().writeDataset(h5FilePath, relativeDSpath, data1d);

//NOTE: the same process can also be done with Data_2D objects
```

How to read DataObjects from HDF5:

```
//NOTE: the same process can also be done with Data_2D objects
DataObject data = new H5IO().readDataset(h5FilePath, datasetPath);
```

How to get information from DataObjects:

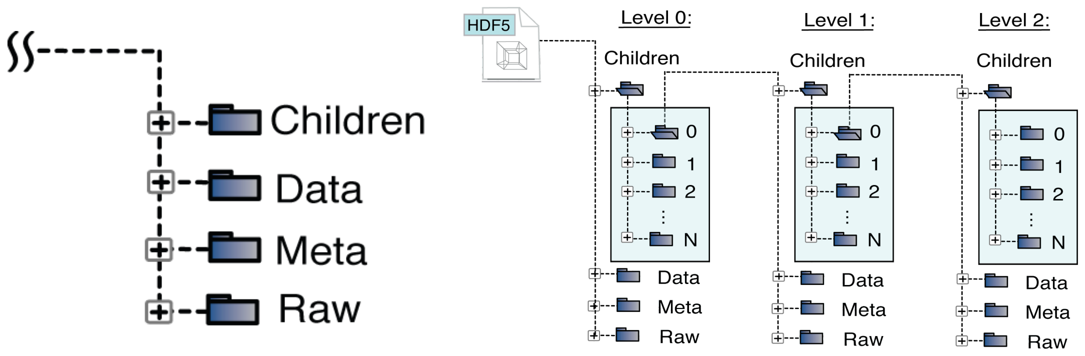
```
//Basic method calls for all DataObjects
long[] dimensions = data.getDimensions();
String type = data.getDataType();
String name = data.getName();
int rank = data.getRank();

//Once the rank is known, we can cast the DataObject appropriately
if(rank==1)
      Data_1D data1d = (Data_1D) data;
else if (rank==2)
      Data_2D data2d = (Data_2D) data;
```

How to get the data from DataObjects:

```
//Since we know it is a 1D object and a Float data type, we can cast it
Float[] floatArray = (Float[]) data1d.getData();
```

3 DataModules



The *DataModule* is the core building block of *SDCubes*. It is a data object capable of forming tree structures of arbitrary size and complexity. Each *DataModule* has 4 groups: Children, Data, Meta and Raw. The Data, Meta and Raw group store numerical and text datasets defining the *DataModule*. The Children group contains an arbitrary number of child *DataModules*. In this manner, it is trivial to create trees of *DataModules*. The root of the Data.h5 file is a single *DataModule*.

How to create an empty DataModule:

```
// Create a DataModule by giving it the path of the H5 and its location
SDCube_DataModule dataMod = new SDCube_DataModule(h5Path, relativeDMpath);

// Note: relativeDMpath looks like a normal file system path where
// the root of the path (“.”) is the top level of the HDF5 file.
```

Adding data to the DataModule:

```
// Creating a sample DataObject to add to our DataModule
Data_2D data2d_1 = new Data_2D(...);
Data_2D data2d_2 = new Data_2D(...);

// Adding data to the groups of this DataModule
dataMod.addData(data2d_1);
dataMod.addMeta(data2d_2);
dataMod.addRaw(byteArrayEncodedFile);
// Adding Children DataModules to this Parent
dataMod.addDataModule(new SDCube_DataModule(...));
```

Retrieving data from the DataModule:

```
// Retrieving data from DataModule
ArrayList<DataObject> data = dataMod.getDataGroup();
ArrayList<DataObject> meta = dataMod.getMetaGroup();
ArrayList<DataObject> raw = dataMod.getRawFileArrays();
ArrayList<DataModules> children = dataMod.getDataModules();
```

Write a DataModule to an HDF5 file:

```
// Writes to the H5FilePath to RelativePath already set
dataMod.write();

// Writes to the given H5FilePath and GroupPath if different from current
dataMod.write(H5FilePath, RelativeGroupPath);
```

Load a DataModule from an HDF5 file:

```
// Create and loads from the initialized path
SDCube_DataModule dataMod = new SDCube_DataModule(h5Path, relativeDMpath);
dataMod.load();

// Change H5path and groupPath of existing DataModule before loading
dataMod.setFilePath(H5path);
dataMod.setGroupPath(relativePath);
dataMod.load();
```

4. The Experimental Design

The experimental design for an SDCube sample is stored in an *ExpDesign_Sample* object when instantiated in *Java* and XML when stored on disk. Each *ExpDesign_Sample* contains a collection of *ExpDesign_Descriptors* that describe specifically how the experimental sample has been treated or what was measured. The following parameters describe a single *ExpDesign_Descriptor*:

Id = Text ID that links samples between XML/HDF5
Type = Either a "Treatment" or "Measurement"
Name = Text name of the descriptor
Value = Quantitative value associated with descriptor
Units = Units that correspond to the Value
Time = Time of treatment or measurement
Time_Units = Units that correspond to the Time

Example XML syntax to describe a Sample in the ExpDesign.xml file

```
<Sample id='ID_1'>
            <Descriptor>
                  <type>Treatment</type>
                  <name>EGF</name>
                  <value>1e-9</value>
                  <units>M</units>
                  <time>0</time>
                  <time_units>min</time_units>
            </Descriptor>
      <!--Add as many Descriptors or this sample as needed -->
</Sample>
```

Creating and Writing an ExpDesign Sample

```
// Initialize Descriptor with all parameters
ExpDesign_Descriptor desc = new ExpDesign_Descriptor(
      sample_id, type, name, value, units, timeValue, timeUnits);

      . . .

/ ... or set them one at a time
ExpDesign_Descriptor desc = new ExpDesign_Descriptor();
desc.setId(id);
desc.setName(name);
desc.setType(type);
desc.setValue(value);
desc.setUnits(units);
desc.setTime(time);
desc.setTimeUnits(tUnits);


// Creating a Sample and adding our descriptor to it
ExpDesign_Sample sample = new ExpDesign_Sample();
sample.addDescription(desc)

// Adding this sample to the collection of samples for the SDCube
ArrayList<ExpDesign_Sample> allSamples = new ...
allSamples.add(sample);

// Writing all the samples to the given XML path
ExpDesign_IO.write(xmlFilePath, allSamples);
```

Reading samples from XML

```
ArrayList<ExpDesign_Samples> parsedSamples;
// Reading all the samples from the given XML path
parsedSamples = ExpDesign_IO.parseSamples(xmlFilePath);
// Reading all the samples that have the given ID
parsedSamples = ExpDesign_IO.parseSamplesWid(xmlFilePath, id);
// Reading all the samples that have one of the given (String[])ids
parsedSamples = ExpDesign_IO.parseSamplesWids(xmlFilePath, ids);
// Reading samples that contain descriptors with ALL the dNames
parsedSamples = ExpDesign_IO.parseSamplesWithDescriptorNames_AND(
      String xmlPath, String[] dNames))
// Reading samples that contain descriptors of at least 1 dName
parsedSamples = ExpDesign_IO.parseSamplesWithDescriptorNames_OR(
      String xmlPath, String[] dNames))
```

5. The SDCube Sample

The SDCube is composed of one or more SDCube_Sample objects. A Sample contains two components: the HDF5 encoded data and the XML experimental design. A String ID links the both components of the SDCube_Sample. Refer to prior sections for the specifics of these individual components. Here the SDCube_Sample is described:

Creating an SDCube Sample

```
// Initialize SDCube_Sample
SDCube_Sample sample = new SDCube_Sample(
      SDCube_DataModule rootDM, ExpDesign_Sample expD, String id);
```

Getting information from an SDCube Sample

```
// Getting the Sample ID
String id = sample.getID();
// Getting the root DataModule
SDCube_DataModule data = sample.getDataModule();
// Getting the ExpDesign_Sample
ExpDesign_Sample expD = sample.getExpDesign();
```

Acknowledgements
This work was supported by National Institute of Health (NIH) grants HG006097, HG005693 and GM68762.

Supplementary Note 2

imagerail

Software Manual – v1.2.1

High-throughput image analysis software using *SDCubes* for single-cell data storage and adaptive experimental design

Bjorn L Millard[1], Mario Niepel[1], Michael P Menden[1,3], Jeremy L Muhlich[1], and Peter K Sorger[1,2]

January 11, 2011

[1]Center for Cell Decision Processes, Department of Systems Biology, Harvard Medical School, Boston, MA 02115, USA, [2]Department of Biological Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139, USA [3]Current address: University of Applied Sciences Weihenstephan-Triesdorf

Email: imagerail-users@googlegroups.com
Web: http://www.semanticbiology.com/software/imagerail

License

Contents

1 Introduction

This short document is a manual for *ImageRail* software first described by Millard et al. [1]. Additional resources and periodic updates can be found at: http://www.semanticbiology.com/software. This website also offers further details on SDCubes, the storage format used by ImageRail, including a separate manual and software library.

The goal of *ImageRail* software is to provide an open-source, stand-alone program built around the *SDCube* data structure that 1) facilitates high-throughput image analysis, including segmentation and feature computation, 2) provides basic data exploration tools such as line plots, scatter plots, histograms and heat maps, and 3) encodes and manipulates experimental design metadata and links it to extracted data from the source images.

ImageRail data and metadata are stored in *semantically-typed data hypercubes* (*SDCubes*), a data structure built from the HDF5 and XML file formats. *SDCubes* were developed to efficiently store large amounts of experimental biology data (in this case, single-cell data acquired via high-throughput microscopy; HTM) and to link it to digitized experimental metadata describing how each sample was perturbed experimentally and then measured. Although *ImageRail* data is stored primarily in *SDCubes*, user-defined subsets of the data can be exported in CSV format for analysis in other software programs such as *Excel, MatLab* and *Spotfire.*

The default *ImageRail* segmentation algorithm is implements a simple watershed-style method. Currently, alternative algorithms can be substituted by performing some *Java* programming. We are currently creating new programming APIs that will create a direct interface with open-source image analysis programs *ImageJ.*

The full benefits of *SDCubes* will become apparent when more programs can read and manipulate HDF5-encoded data and the XML-encoded experimental design data to analyze and navigate efficiently through the digitized single-cell data. Currently, *ImageRail* can create, modify and read *SDCubes*, but more advanced functional analysis and plotting of *SDCubes* will be handled by a sister program, *DataRail v2.0*, which is currently under development [2].

2 Installation and Quick Start

ImageRail downloads can be found at: http://www.semanticbiology.com/software/imagerail

Windows:

1) Download the Windows installer .exe file
2) Double click the installer file to initiate installation
3) Optional: To make a Desktop shortcut, drag the ImageRail icon from the Start menu Programs folder to your desktop while holding the Ctrl key.
4) ImageRail may be launched from the Start menu's Programs folder, or your desktop shortcut if you chose to create one.

Mac OS X (OS X 10.6 - Snow Leopard or greater is required)

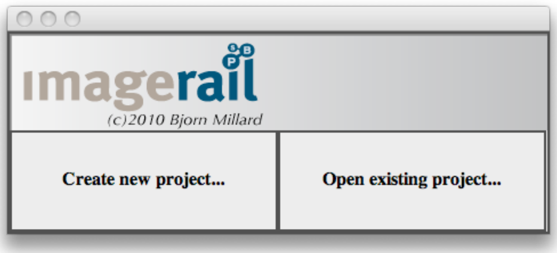
1) Download the Mac OS X .dmg
2) Double-click the .dmg to mount it and display its contents
3) Drag the ImageRail application to your Applications folder
4) Optional: Drag the instruction manual PDF file to your Documents folder or wherever you would like to keep it.
5) To launch ImageRail, open your Applications folder and double-click the ImageRail application.

3 The *ImageRail* Project

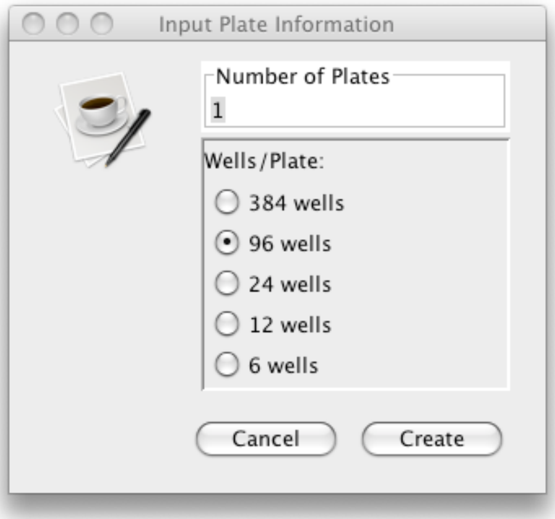
After the *ImageRail* splash screen goes away, a dialog box should appear (Figure 1a). This box will let you create a new *ImageRail* project or re-open an existing project.

Figure 1

a



b


Input Plate Information

Number of Plates
1

Wells/Plate:
( ) 384 wells
(•) 96 wells
( ) 24 wells
( ) 12 wells
( ) 6 wells

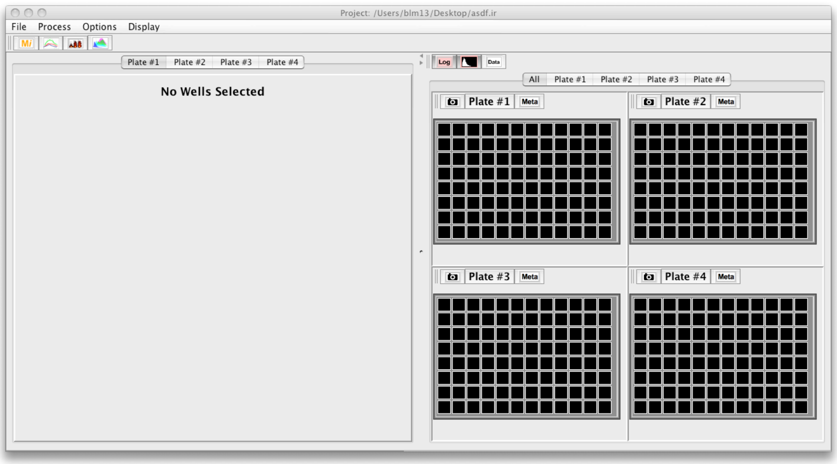
3.1 Creating New Projects

To create a new project, you must specify how many plates and how many wells per plate you want to initialize using the appropriate dialog box (Figure 1b).

3.2 Opening Existing Projects

If you want to re-open an existing project, a file chooser will appear so you can select the directory and project of interest. Once a project has been initialized or opened, a graphical user interface should appear (Figure 2). This is an example of a new project created with four 96 well plates.

Figure 2

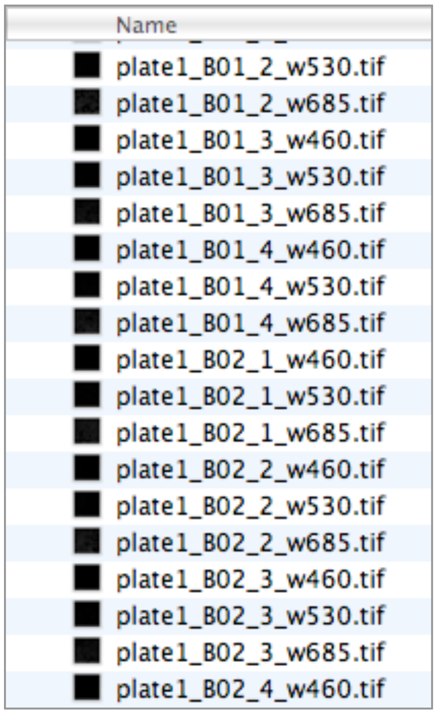


4 Images

4.1 Image Formats and Conventions

Currently, *ImageRail* reads grayscale-TIFF images (different images correspond to different filter combinations in the usual manner, thereby enabling multi-spectral analysis). Before getting started you must make sure that all TIFF images for a given plate are put into their own directory folder. All the images representing each field for all wells and various wavelengths for each individual plate will be mixed together in this directory. (Figure 3). *ImageRail* will parse through and organize the images based on a simple file naming convention.

Figure 3



Name
plate1_B01_2_w530.tif
plate1_B01_2_w685.tif
plate1_B01_3_w460.tif
plate1_B01_3_w530.tif
plate1_B01_3_w685.tif
plate1_B01_4_w460.tif
plate1_B01_4_w530.tif
plate1_B01_4_w685.tif
plate1_B02_1_w460.tif
plate1_B02_1_w530.tif
plate1_B02_1_w685.tif
plate1_B02_2_w460.tif
plate1_B02_2_w530.tif
plate1_B02_2_w685.tif
plate1_B02_3_w460.tif
plate1_B02_3_w530.tif
plate1_B02_3_w685.tif
plate1_B02_4_w460.tif

Each image file name has 4 key properties:

1. Prefix string
2. Well identifier
3. Field number identifier
4. Wavelength identifier

An example file name would look like:

experiment1_A01_1_w488.tif

In this example, the prefix→ "experiment1", well identifier→ "A01", field number→ "1", and wavelength→ "w488". Note that all these identifiers are separated by an underscore ("_"). Also note that the label for an image arising from well B01 must contain a “01;” “B1” will not work.

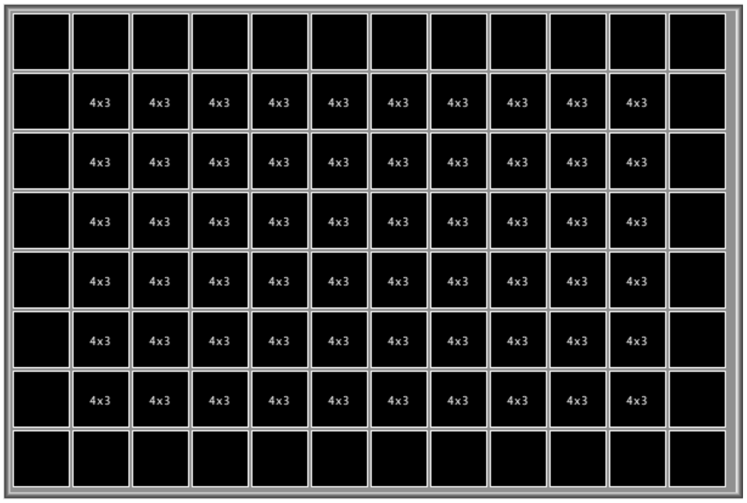
4.2 Loading Images

To begin your analysis, you will first load images into each plate. Please refer to section 4.1 to learn more about current image format requirements. We are actively working to make image formats compliant with Open Microscopy Environment (OME) standards.

Once images conform to the required naming convention and each directory contains images from one plate, drag and drop each directory into the appropriate GUI-plate.

If successfully loaded, the wells of the plate that contain images will now display a NxM indicator, where N represents the number of fields acquired and M represents the number of channels per field (Figure 4). In this example, the outer wells were not imaged.

Figure 4



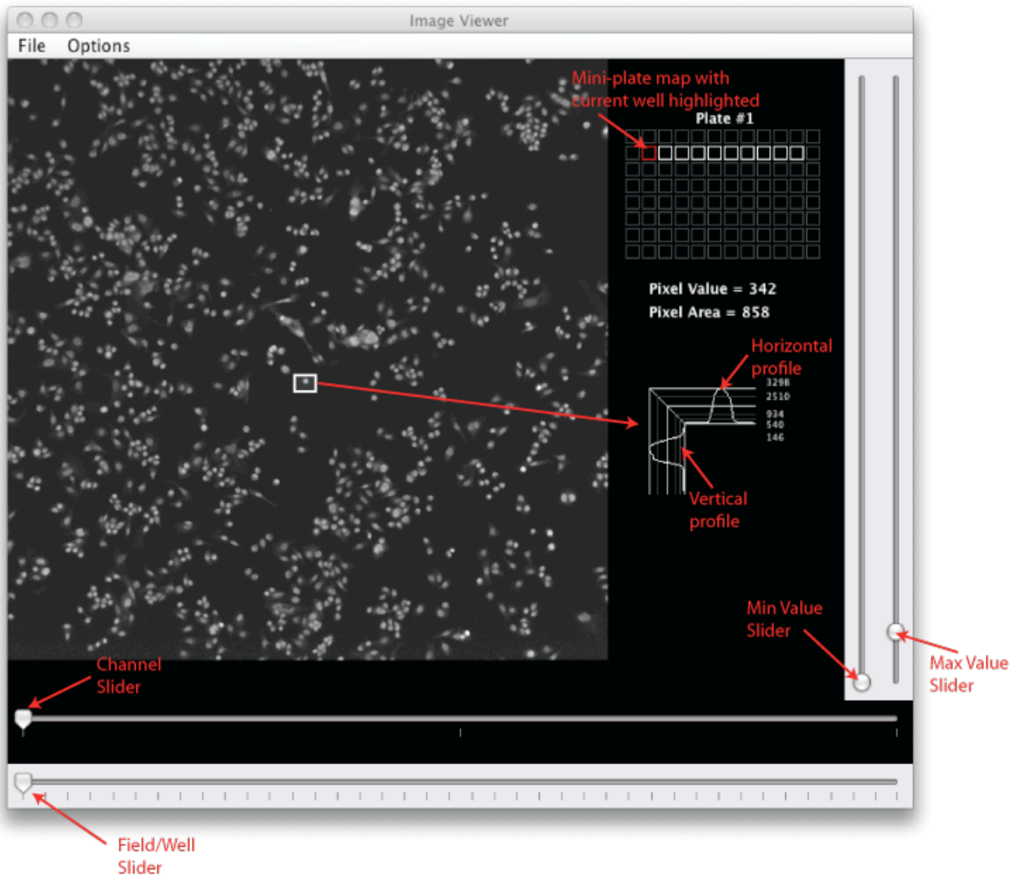
Thus the total number of image files within each well is equal to: NumFields x

NumChannels. In the above example we have 4 fields and 3 channels for a total of 12 separate TIFF images per well.

Note that the original images are ***moved*** into the *ImageRail* project and not copied, so be sure to make a copy of your images if you want to store them elsewhere as well.

4.3 Viewing Images

Figure 5



4.3.1 Field Viewer

To display the loaded images, click or drag on the plate to select the wells of interest and then either select menu item *Display→Display Images* or press *ctrl+d*.

An image browser will open displaying an image, four slider bars and a mini-plate map (Figure 5). NOTE: Mousing over the image will display the pixel intensity on the right margin below the mini-plate.

1. Bottom-most Slider - Changing fields and wells:

The default start image is from the upper left-most of the selected wells in the plate, with its first field and lowest wavelength. The bottom-most slider will first scroll through any other fields of the first well, then move on to the first field of the second well and so forth. The sliding order of the wells is always left to right, top to bottom. The plate map will indicate the current well being viewed, and clicking on a white-highlighted well will skip directly to the first field of that well.

2. Bottom-top Slider - Changing Wavelengths:

Scrolls through the various channels acquired for the currently selected field.

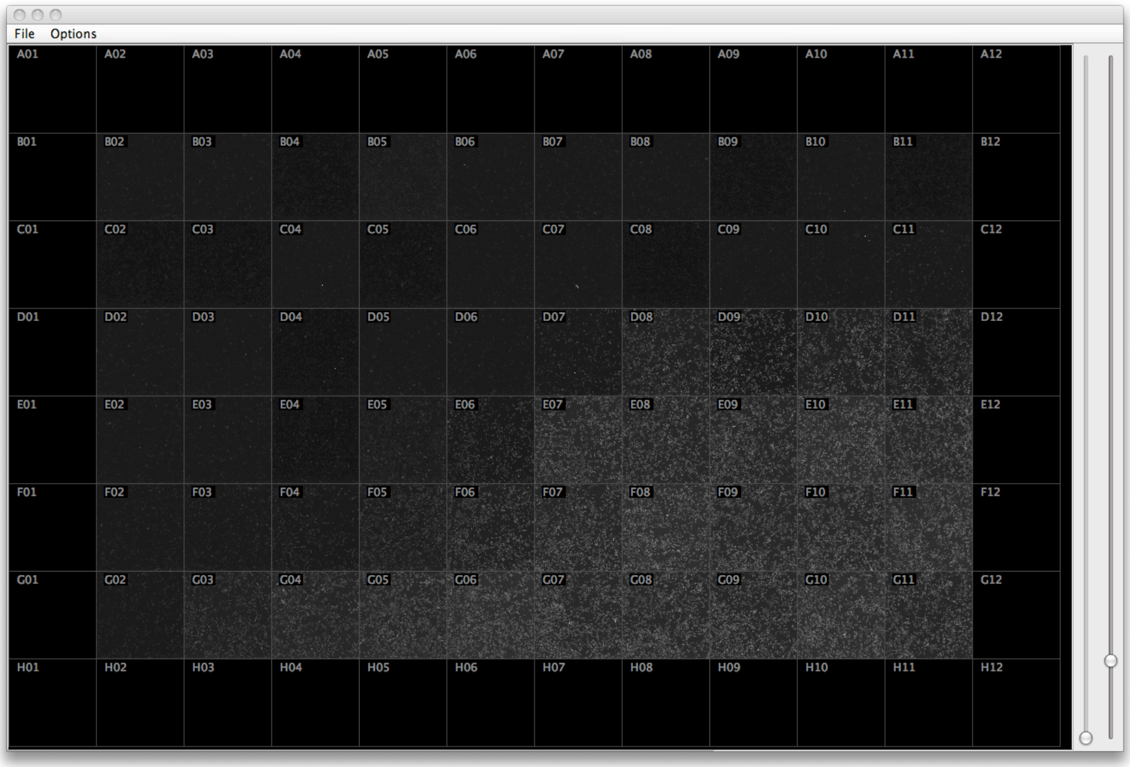
3. Right (1st-left) Slider - Min Value Scaling:

Changes the minimum value of the image; pixels with values lower than this limit are set to 0 (black).

4. Right (2nd-right) Slider - Max Value Scaling:

Changes the maximum value of the image to make it brighter or dimmer; pixels with values higher than this limit are saturated and displayed white.

Figure 6


File
Options
A01 A02 A03 A04 A05 A06 A07 A08 A09 A10 A11 A12
B01 B02 B03 B04 B05 B06 B07 B08 B09 B10 B11 B12
C01 C02 C03 C04 C05 C06 C07 C08 C09 C10 C11 C12
D01 D02 D03 D04 D05 D06 D07 D08 D09 D10 D11 D12
E01 E02 E03 E04 E05 E06 E07 E08 E09 E10 E11 E12
F01 F02 F03 F04 F05 F06 F07 F08 F09 F10 F11 F12
G01 G02 G03 G04 G05 G06 G07 G08 G09 G10 G11 G12
H01 H02 H03 H04 H05 H06 H07 H08 H09 H10 H11 H12

4.3.2 Montage Viewer

The *montage* button on top of each plate panel will produce a montage image viewer that displays an image from one field from each well (Figure 6). By default, the viewer is initialized with the first wavelength and the first field for each well. The *Options* menu allows selection of other channels and fields. The mouse wheel controls the image zoom. If zoomed, mouse dragging will pan the viewer.

5 Image Processing

Preprocessing

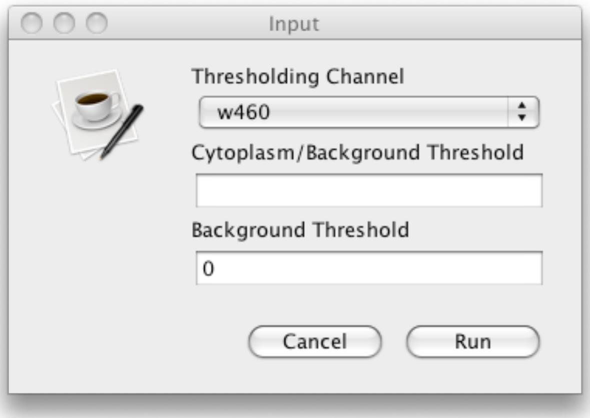
In its default form, *ImageRail* does not perform image pre-processing such as image flat-field calibration, contrast enhancements or convolution filtering. These features as well as other segmentation algorithms can be implemented as plug-ins. When we need preprocessing for our purposes, we use the open-source NIH software program *ImageJ* (http://rsbweb.nih.gov/ij).

5.1 Well Means

If you are only interested in population average fluorescent intensities, then there is no need for single-cell segmentation. Well-mean computations can be done quickly. This measurement would be similar to the data acquired from standard fluorescence-based plate readers that specifically ignores image regions without cells, as opposed to performing single-cell analysis and then averaging the data over all cells in the plate.

To begin, highlight the wells in the plate you want to process and use the menu option: *Process→Well Means*

Figure 7


Input

Thresholding Channel

w460

Cytoplasm/Background Threshold

Background Threshold

0

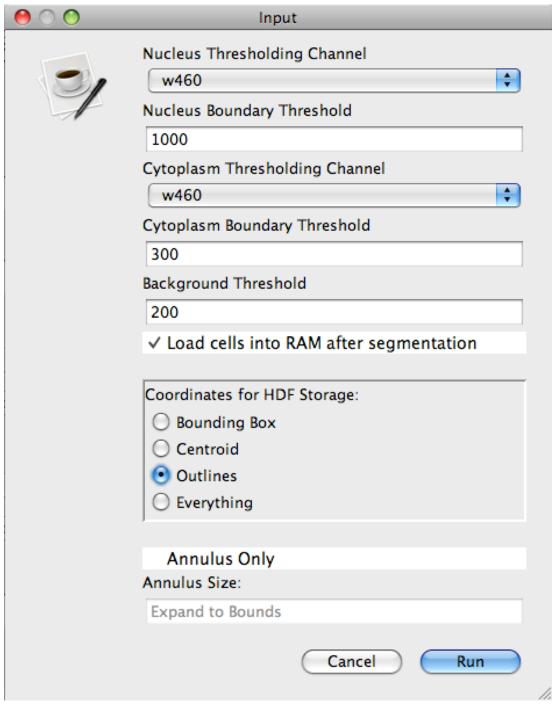
A dialog box will prompt the input of three parameters: 1) channel for cell identification, 2) intensity threshold, which sets a minimum defining where cells exist, and 3) background threshold, which establishes an upper bound for defining background pixels (Figure 7). Refer to section 5.3 to learn how to determine optimal parameters.

5.2 Single-Cell Segmentation

The default *ImageRail* segmentation algorithm is simple yet effective for the 2D tissue cultures for which it was originally designed. When cells grow on top of each other or 3D samples are to be analyzed, more sophisticated segmentation is required. It is currently necessary to do additional programming to integrate new processing routines.

To perform single-cell segmentation of loaded images, use the menu item *Process→Single Cells*

Figure 8



Nucleus Thresholding Channel: w460

Nucleus Boundary Threshold: 1000

Cytoplasm Thresholding Channel: w460

Cytoplasm Boundary Threshold: 300

Background Threshold: 200

✓ Load cells into RAM after segmentation

Coordinates for HDF Storage: Bounding Box, Centroid, [✓ Outlines], Everything

Annulus Only

Annulus Size: Expand to Bounds

Depending on the version of *ImageRail* or the segmentation algorithm you are using, the dialog window may look different (Figure 8), but in the original version there are three essential user supplied parameters:

1. Nucleus Threshold
2. Cytoplasm Threshold
3. Background Threshold

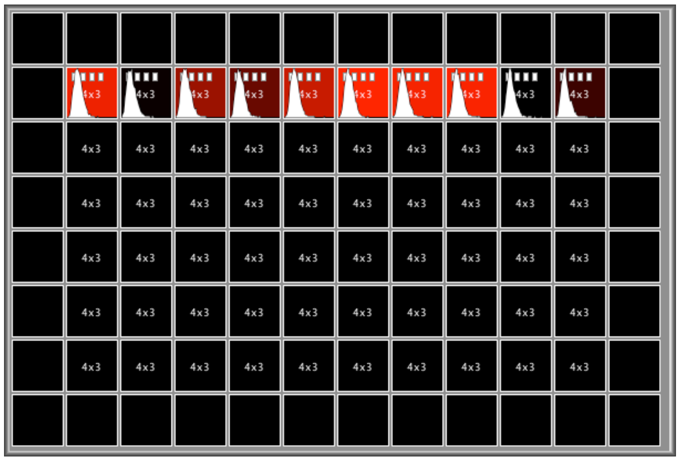
For both the nucleus and cytoplasm thresholds, the user must specify which channel should be used. NOTE: the channel used to threshold the nucleus can be different from the channel used to threshold the cytoplasm.

Refer to section 5.3 to learn how to determine optimal parameters.

5.2.1 Load Cells into RAM after Segmentation:

Unless this box is checked, the single-cell data for each processed well will be cleared from the RAM and only cached to the HDF5 file on the hard drive. If processed for single-cells, the plate will display the number of image fields that have data processed and ready for analysis. These are displayed as icons that look like white document icons within the wells, where each icon represents data for a single field as shown in the top row of the plate (Figure 9).

Figure 9



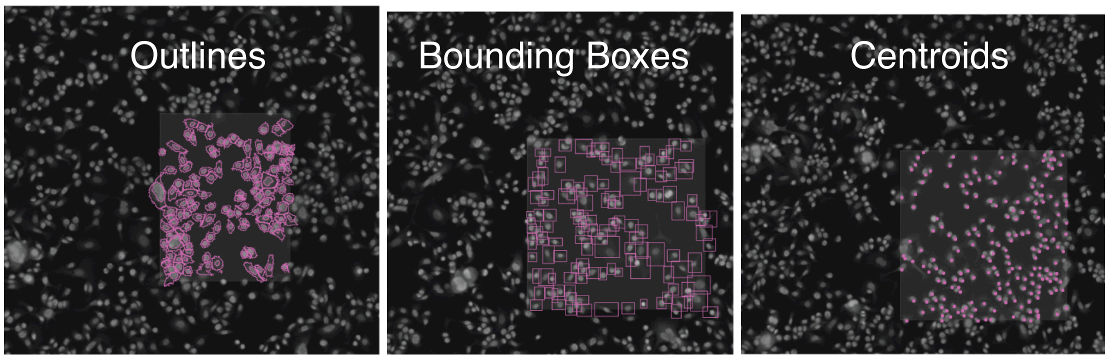
In this example, the single-cell data has not been loaded. By using the menu option, *Options*→*Load Cells*, or by clicking the "*Load Cells into RAM after Segmentation*" checkbox before processing the wells, the cells will be loaded into RAM and a mini-histogram of all cells will be displayed for the selected feature. Note that there is an option to only load the single-cell data without the pixel coordinates as an alternative to loading both. This will save RAM space if no Data←→Image link is required. See section 6.3 for more details.

5.2.2 Coordinates for HDF5 Storage:

Once a full segmentation has been performed, all features are computed for each cell using the complete coordinate list. See Supplementary Fig. 4 of *Millard et al.* [1] for a more detailed description of the "segmentation and feature computation" process.

For long-term storage and data sharing, a complete coordinate list for all cells may not be necessary: in many situations a centroid or bounding box for each cell is sufficient and significantly reduces the amount of data to be stored on the hard drive (Figure 10).

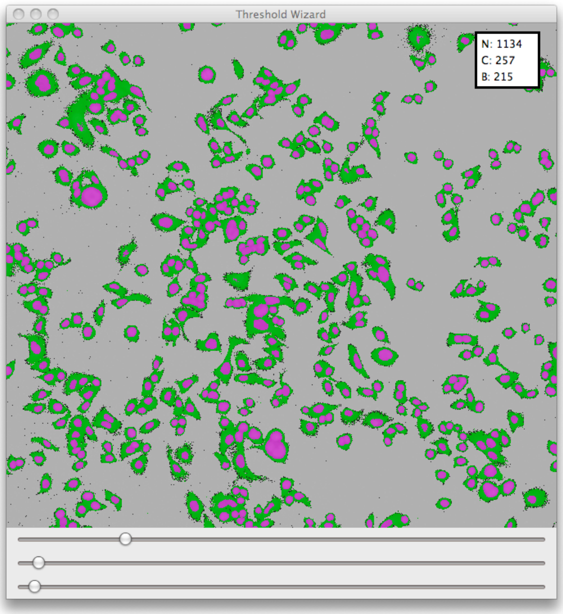
Figure 10



5.3 Parameter Selection

To determine the numerical values for the segmentation parameters, open a sample image of the wells you want to segment. Highlight a region in the image that contains representative cells in the channel that will be used for segmentation and select: *Options*→*Threshold Wizard*. A separate window will open and display the selected region of the original image and three sliders at the bottom (Figure 11). The top slider changes the nuclear threshold values by coloring pixels greater than the selected value pink. The middle slider does the same for the cytoplasm, coloring pixels greater than the selected threshold green (excluding any enclosed nuclear pixels which remain pink). The bottom slider colors pixels *lower* than the current value white, which represents background pixels. The currently selected threshold values from all sliders are displayed in the upper right corner of the window, where N = nucleus, C = cytoplasm and B = background. Note that even though the green and pink pixel regions may appear contiguous in the Threshold Wizard, this does not represent the final segmentation result; the segmentation algorithms will subsequently attempt to cut overlapping cells apart with a standard watershed procedure.

Figure 11



In this example, images were acquired using Hoechst stain to identify DNA and a blue whole protein dye to mark the cytoplasm. Note that the stains have drastically different intensity values, such that it is possible to select two different thresholds for the different compartments while using a single fluorescent channel.

In this example, the following parameters were selected:

1. Nucleus = 1000
2. Cytoplasm = 250
3. Background = 200

6 Data Viewing

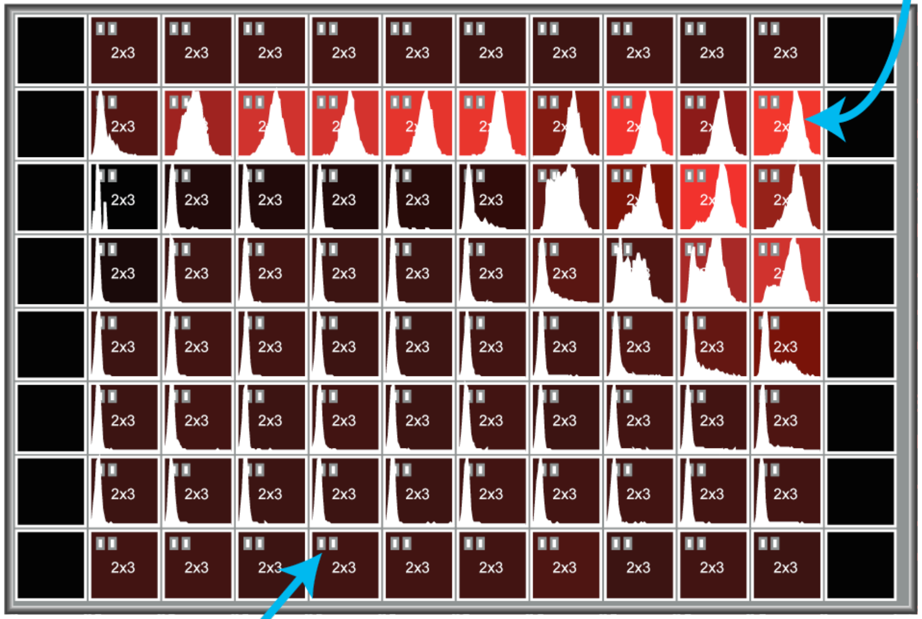
6.1 Plate Graph

The right panel of the main *ImageRail* GUI contains all plates included in the project currently loaded (Figure 12). The graphical plate interface acts both as a controller for actions to be performed on specific wells of specific plates and as a plate-wide data display that includes well heat maps and mini-histograms.

Figure 12

The well background color represents a normalized heat map of the feature selected in the combo-box directly above the plate GUI.

A white histogram represents the single-cell distribution for each well. The y-axis represents number of cells and the x-axis the value of the feature selected in the combo-box above the plate.



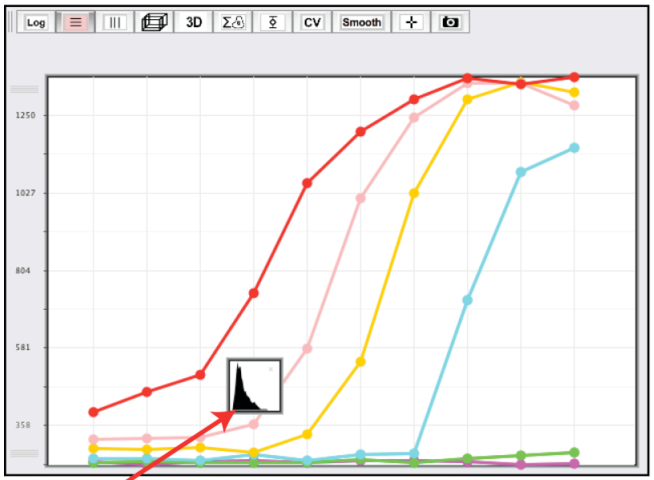
A white file icon shows the number of image fields that have single-cell data processed and available to be loaded into the GUI.

The number of loaded images in each well is indicated by an NxM text string, where N = #Fields and M = #Channels

6.2 Line Plot

Well mean features can be plotted as contiguous well data series in a line plot (Figure 13). Currently, data series can be constructed based on contiguous wells as rows, columns, or across the same well of multiple plates (trans-plate). Note that if experimental metadata have been entered to denote how each well has been treated, the line plot will attempt to create proper x-axis labels and series legend labels.

Figure 13



Double-clicking on a data point exposes the underlying single-cell population histogram.
NOTE: histograms can be moved via dragging

Toolbar Items:

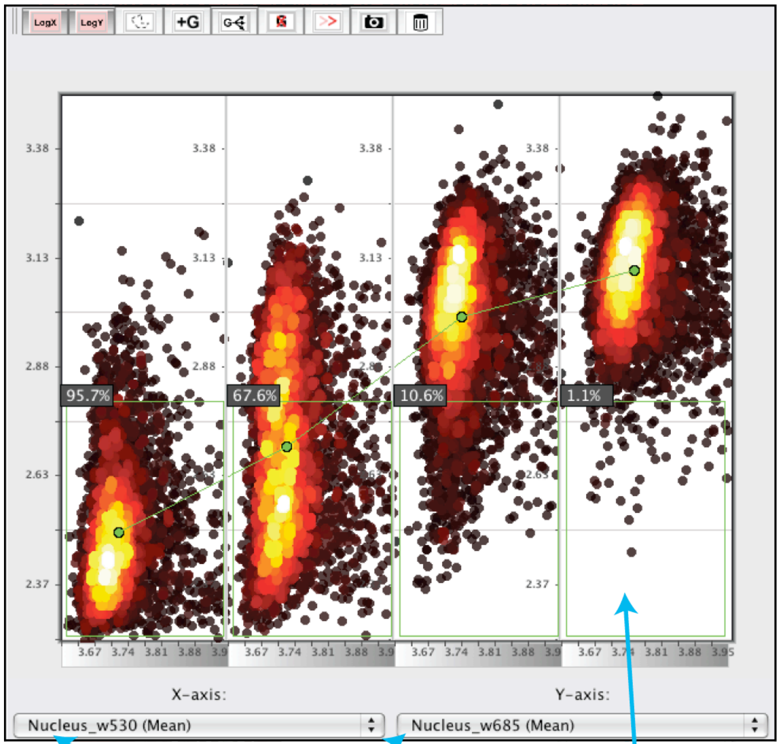
Log	Log transform the y-axis
☰	Creates row data series
⦀	Creates column data series
	Creates trans-plate data series
3D	Creates a 3D plot of the data series
Σ	Plots number of cells in each well
	Plots the population stdev bars
Smooth	Performs a 3-point smoothing
CV	Plots the population CV
✛	Interpolating tracking marker
📷	Image capture (JPG, PNG, or SVG)

6.3 Scatter Plot

Single-cell data can be plotted in a 2D scatter plot, where each dot represents one cell with an X-Y coordinate in the plot based on computed features whose identities are selected at the bottom of the plot window (Figure 14).

Figure 14

Cells from four selected wells are plotted in a side-by-side scatter plot for direct comparison. This view also connects the centroids of the distributions by a green line so mean value trends can be visualized.



X-axis: Nucleus_w530 (Mean)

Y-axis: Nucleus_w685 (Mean)

X-axis feature variable
Y-axis feature variable
Rectangular or freeshape gates added to the plots compute the percentage of bound cells.

Toolbar Items:

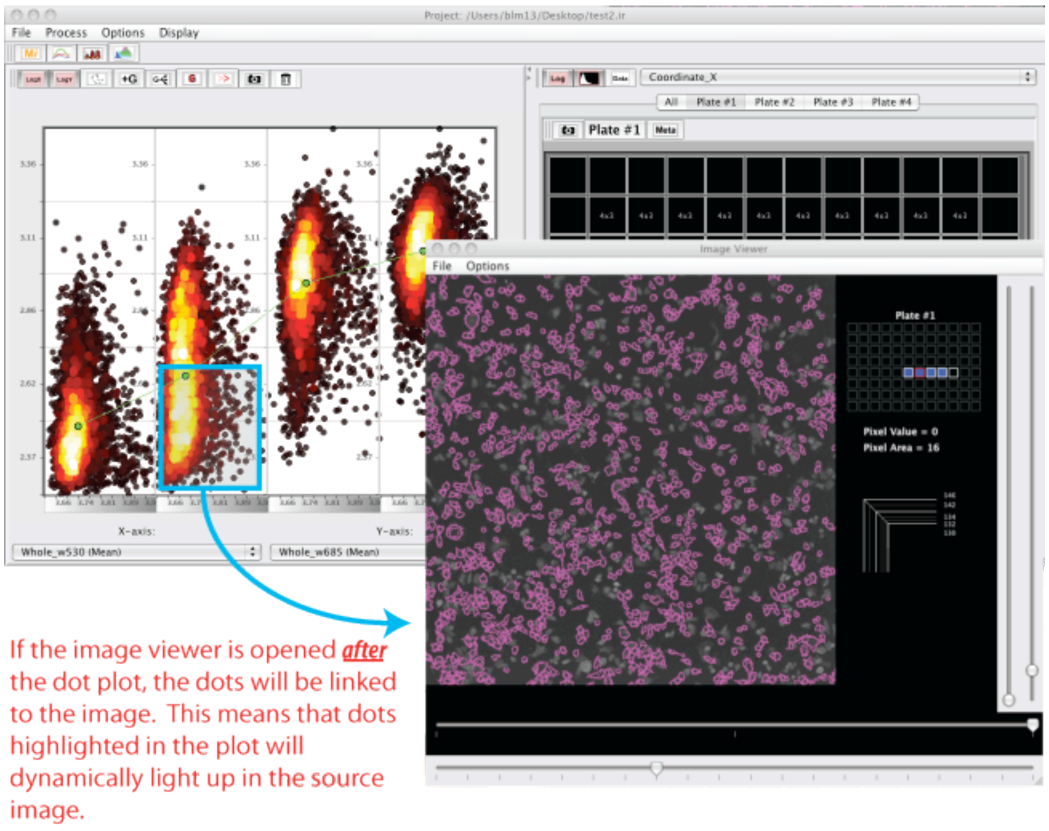
LogX	Log transform the x-axis
LogY	Log transform the y-axis
	Freeshape gate tool
+G	Add gate
	Delete all gates
>>	Save gate data to CSV file
	Image Capture
	Delete highlighted cells

If multiple wells are highlighted, then multiple plots will be constructed in a “side-by-side” fashion. The plots are ordered based on well location within the plate: left-to-right, top-to-bottom. A green line connects the centroids of distributions. Note that each well’s scatter plot is a collection of the cells from all image fields within that well.

6.3 Scatter Plot / Image Linkage

The single-cell data within the scatter plot can be directly linked to the source image if the data was loaded via: *Options*→ *Load Cells*→ *Data & Coords*. Once the scatter plot has been displayed with the appropriate wells highlighted, initiate the image link by opening the image viewer via the menu: *Display→Display Images*. Once the image viewer is open, highlighting dots within the scatter plot will dynamically display the corresponding cell within the image (Figure 15).

Figure 15



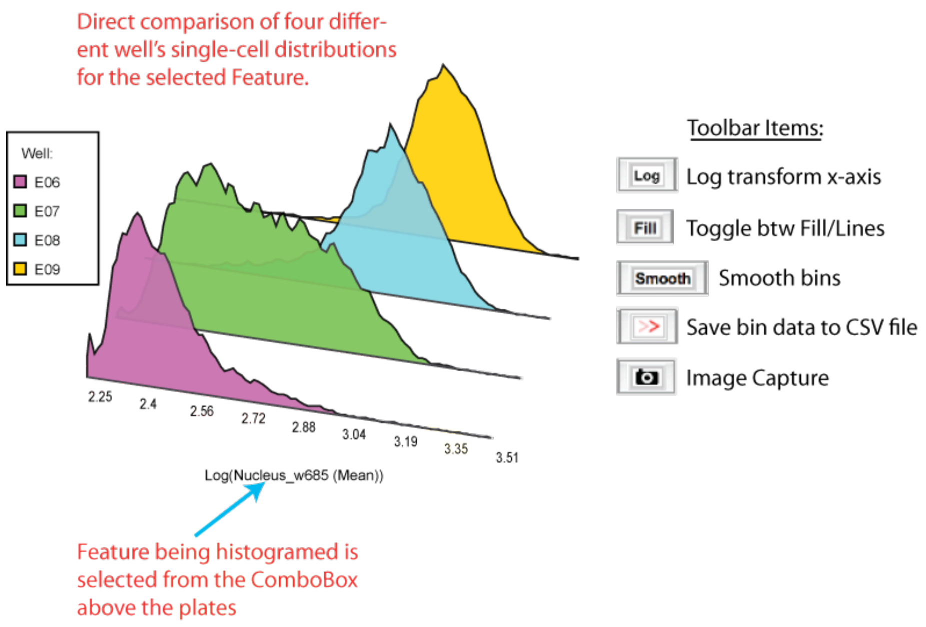
If the image viewer is opened ***after*** the dot plot, the dots will be linked to the image. This means that dots highlighted in the plot will dynamically light up in the source image.

Note that the extent of coordinates stored for this linkage was determined during the segmentation step (see Section 5.2.2). Users can decide to store from as little as a centroid up to all the pixel coordinates that represent the location of each cell. Most often a centroid will suffice for post-segmentation analysis, whereas viewing *outlines* is most useful to confirm proper parameters for segmentation (this is usually done with a test well before switching to centroids for the rest of the plate).

6.4 Histogram Plot

In addition to scatter plots, single-cell data can also be plotted as 1D histograms (Figure 16) of the feature selected in the combo-box at the top of the plate panel. The plots are ordered based on well location within the plate: left-to-right, top-to-bottom. The plot can be translated, zoomed and rotated by holding down *Shift* or *Control* with left-button drag or a right-button mouse drag.

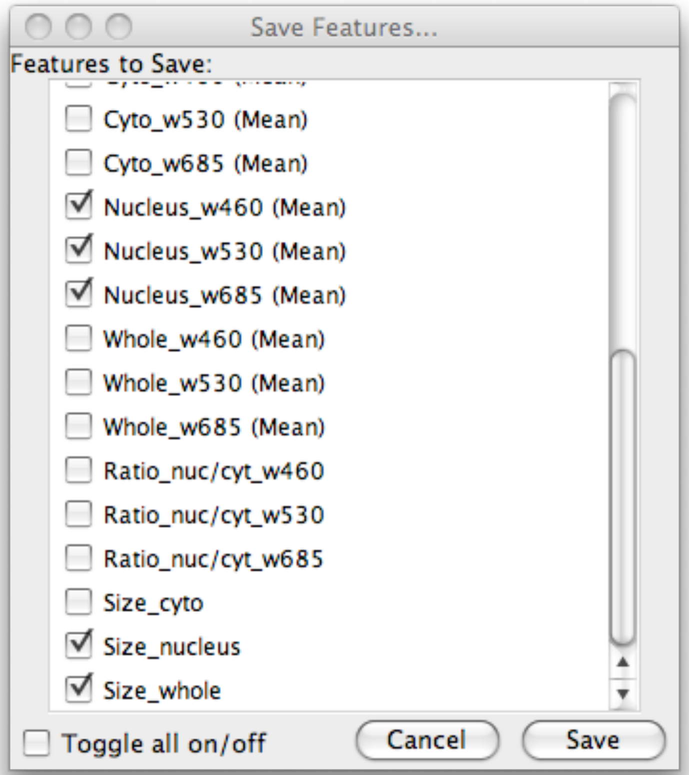
Figure 16



7 Data Export

7.1 CSV – Well Means

Figure 17


Save Features...

Features to Save:

- [ ] Cyto_w530 (Mean)
- [ ] Cyto_w685 (Mean)
- [x] Nucleus_w460 (Mean)
- [x] Nucleus_w530 (Mean)
- [x] Nucleus_w685 (Mean)
- [ ] Whole_w460 (Mean)
- [ ] Whole_w530 (Mean)
- [ ] Whole_w685 (Mean)
- [ ] Ratio_nuc/cyt_w460
- [ ] Ratio_nuc/cyt_w530
- [ ] Ratio_nuc/cyt_w685
- [ ] Size_cyto
- [x] Size_nucleus
- [x] Size_whole

Mean value and Standard Deviation data can be exported to a comma-separated value file (CSV) for external analysis in other software programs such as *Excel* and *Spotfire*. To perform the export, use the menu option *File→ Save as CSV*. A dialog will be displayed that allows the user to select which features to export to CSV (Figure 17). This file will contain blocks of data for each feature organized in an NxM block, where N=number of rows, M=number of columns. The left block represents the well mean feature values and the right block represents the well population standard deviation.

Note: The number of cells for each well will be appended to the bottom of the CSV export file by default. No selection for cell number is necessary in the dialog window for export.

7.2 MIDAS file formats

MIDAS is a CSV file format capable of encoding experimental data that are coupled to the corresponding experimental design metadata. A complete description is provided in the reference section of publication [2].

7.2.1 Well Means

The MIDAS file format was originally developed for well mean, multiplexed data. To export well means from your plates, highlight the wells you want to export and select the menu item, *File→Save as MIDAS→Well Means*. The dialog box shown at the header of section 7 will allow selection of features that you would like to export. Note that MIDAS export can be done without metadata input.

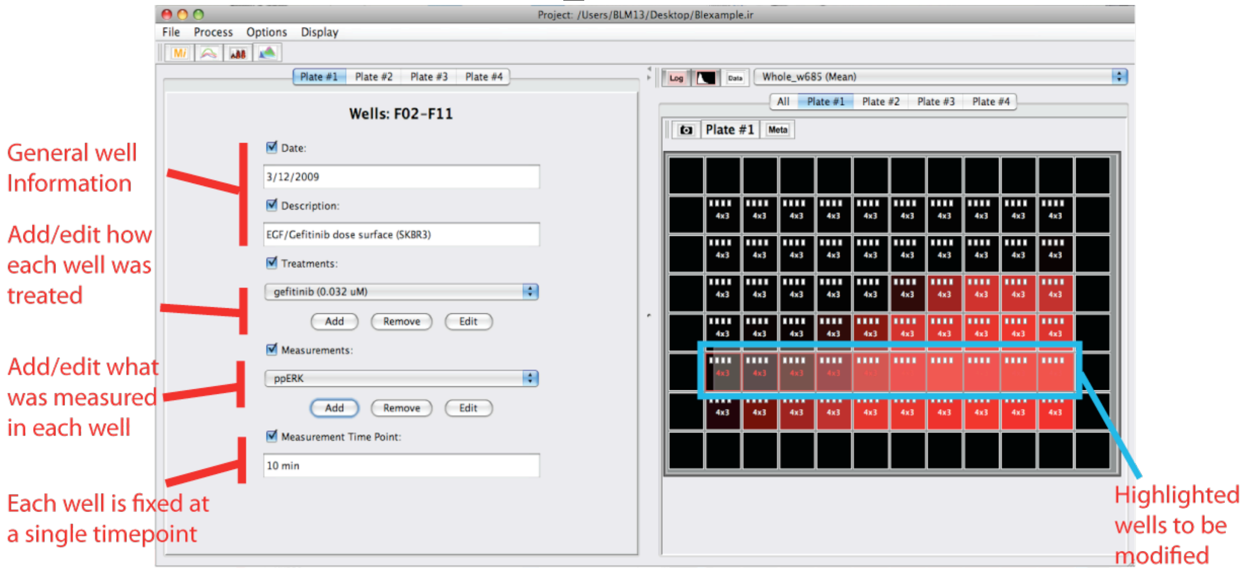
7.2.2 Single-Cells

Although the MIDAS file format was originally developed for well mean values it is capable of storing small amounts of single-cell data. To export single cell data from your plates, highlight the wells you want to export and select the menu item, *File→Save as MIDAS→Single-Cells*. The dialog box shown at the header of section 7 will allow selection of desired features to export. Note that MIDAS export can be done without metadata input.

8 Experimental Design Metadata

To enter experimental design metadata select the yellow “*Mi*” button on the top toolbar of the left panel (Figure 18). *Mi* stands for MIDAS, a file standard for experimental design that we have previously described [2]. First select the wells you wish to annotate in the plate view at right. Then enter the desired metadata in the left MIDAS panel. To modify each field, the checkbox above that field must be selected.

Figure 18


Project: /Users/BLM13/Desktop/Blexample.ir
File Process Options Display
Plate #1 Plate #2 Plate #3 Plate #4
Log Data Whole_w685 (Mean)
All Plate #1 Plate #2 Plate #3 Plate #4
Plate #1 Meta
Add Remove Edit
General well Information
Add/edit how each well was treated
Add/edit what was measured in each well
Each well is fixed at a single timepoint
Highlighted wells to be modified

Wells: F02-F11

Date:
3/12/2009

Description:
EGF/Cefitinib dose surface (SKBR3)

Treatments:
gefitinib (0.032 uM)

Measurements:
ppERK

Measurement Time Point:
10 min

The top two text fields allow for entry of a free-text based Date and Description. Treatments can be added, edited or deleted. A treatment is composed of a compound name, value, units, time of treatment and time units. Measurements for each well can be added to document what each wavelength of the images acquired represents. For example, each fluorescently tagged secondary antibody typically binds a primary antibody specific to a particular protein epitope. Thus fluorescence from each channel is a surrogate read-out for the presence of specific biological components. Note that the addition of a measurement description in the metadata is only for documentation purposes and does not change the exported value names, which continue to be based on the specific feature that was computed (ex: size_nuclear or Nucleus_w685 (Mean)). Finally, since each well has been fixed at a specific time point, the time of sample acquisition can be added in the bottom text field. Both a time value and time unit can be added (*e.g.*, 10 min), and *ImageRail* will parse the value and the units accordingly.

9 Acknowledgments

We thank Emily Pace, Gerard Ostheimer, Deborah Flusberg, John Albeck, Laura Kleiman and Somponnat Sampattavanich for testing *ImageRail* and providing valuable development insight. This work was supported by National Institute of Health (NIH) grants HG006097, HG005693 and GM68762.

References

[1] Millard B, Niepel M, Menden M, Muhlich J, Sorger PK. Adaptive informatics for multi-factorial and high content biological data. *Nature Methods* 2011

[2] Saez-Rodriguez J, Goldsipe A, Muhlich J, Alexopoulos L, Millard B, Lauffenburger D, Sorger PK. Flexible informatics for linking experimental data to mathematical models via DataRail. *Bioinformatics* 2008