# 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

*SDCube Programming Library:* Software for the creation and manipulation of semantically-typed data hypercubes

# Contents



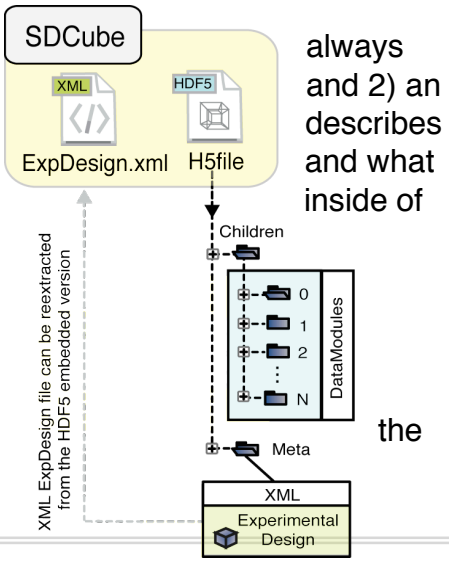
# 1 Introduction

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, called Data.h5, containing the acquired data values, always associated XML file called ExpDesign.xml, that and 2) an how each of the data samples have been perturbed describes was measured. These two files are located together and what an arbitrarily named parent folder that has a file inside of 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:




**How to extract embedded byte[] files from the Data.h5 file:**

```
// 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:**

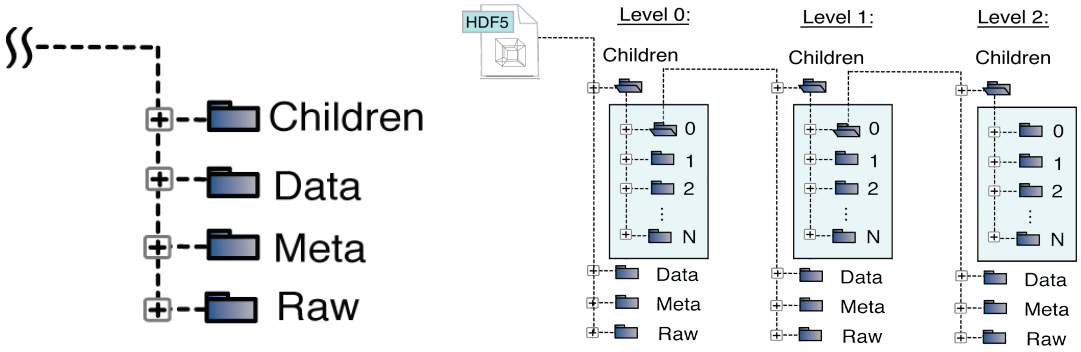
```
//Basic method calls for all DataObjects
long[] dimensions = data.getDimensions();
String type = data.getDataType();
String name = data.getName();
int rank = data.getRank();

//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.