Microarray Scan Simulator (MSS)

The MSS can generate images that are similar to those that would come from scanning microarrays.nn There are 3 ways to use this tool:

1. A GUI interface that collects data from a user to run the generator (this is the easiest).
2. A command-line interface, where all parameters to run the generator are specified on the command line.
3. An API, which can be invoked from custom programs, to give maximum flexibility.

Few users will need to access the API, because the command-line interface provides a lot of power/flexibility.

An image generation request is encoded in an XML file which can be complete in itself or which can refer to external parameters. These external parameters allow for different images to be generated from a shared XML file. Commonly, the name of a features intensities file is provided as one external parameter. But, many other things can be parameterized too, such as feature size, pixel size (resolution), and random number seeds (to allow for reproducible pseudo-random results). For the external parameters to be used, the XML file must be set up with external parameter references, each one with a unique tag (name). Then if these external parameters are set up, they are required.

Individual parameter values can be specified on the command line with a '-D' flag or multiple parameters can be put in a properties file (file with tag/value pairs) which is in turn specified on the command line.

An optional “extensions” jar file can be specified on the command line; this would contain software extensions that can be referred to in the XML. Classes added to this extensions jar file would be subclasses of XMLHandler and X_OperationJAI to add data types and JAI operations, respectively. These are loaded before parsing the XML.

GUI Interface

The GUI interface has turned the interface upside-down, in a sense, because it requires both a properties file and an intensities file, and the name of the XML (which is in fact required by the MSS) is hidden in the properties file. If a file named “extensions.jar” is placed in the run directory, then the GUI will load it and handle the extensions classes. If the jar file is invoked with no parameters, it brings up the GUI:

```
java -jar MSS.jar
```

Generating the XML Schema file

The following command will send to STDOUT the XML schema defining the flavor of XML used to describe an images request:

```
java -jar MSS.jar -schema
```

If you want to create XML, this is your bible; there are even some comments on the elements (datatypes) supported.

Command Line Help

```
java -jar MSS.jar -help
```

Feature Intensities

Whether embedded in the XML file or read from an tab-delimited file that is external to the XML, intensities are encoded using floating point numbers between 0.0 and 1.0, where zero is black and one is white. For each feature, there is one number per channel. Microarrays are layed out in rows and columns with zero based indexing (by default). The indexing origin is in the upper left corner.

The microarray can be divided into zones which if on a grid use metarow, metacol to index a zone. Then within the zone are rows and columns, each zone with an origin 0, 0 feature.

JAI Background Recommended for Understanding the MSS XML

To understand this image generator sufficiently well to create/edit the XML or to write code that calls the API, it's recommend that you first understand some of the concepts from the JAI. It would be good to first understand the following classes from J2SE and JAI:

- JAI
- RenderedImage
- RenderedOp
- ParameterBlock
- ParameterBlockJAI

"Pending Resolution" Images

This API introduces a new concept of classes whose image manifestation is pending a specification of image resolution. These are used to describe the complete recipe for how to draw an image in a way that's analogous to how it would be done with the aforementioned classes, but where coordinates, dimensions, etc. are specified in units of microns rather than pixels. These micron coordinates are translated to pixel coordinates when an image is to be manifested for a particular resolution (parameterized as size of pixel in microns). These "pending resolution" classes only support a float sample models and the conversion of standard integer sample model images into the float paradigm. These float pixel values can either express intensities where black=0 and white=1 or can be used in operands in pixel intensity arithmetic. Given that these intensities can be manipulated in the execution of the image recipe, values outside of the range [0,1] might be meaningful in stages prior to the final image.

Class name	***Description***
PendingRenderedOp	Analogous to RenderedOp, this class is a node in a JAI DAG*, a recipe for building an image However, this DAG* is not fully actualized until the pixel resolution is provided via a call to the method: RenderedOp manifestResolution(float pixelSize) The pixel size is manifested recursively through the entire DAG*.
PendingParameterBlockJAI	Analogous to ParameterBlockJAI, this class describes the actual parameters to a JAI operation. However parameters are specified in units of microns; they are translated to pixel coordinates in a call to following method: ParameterBlockJAI manifestResolution(float pixelSize) The pixel size is manifested through all of the parameters and image sources (type PendingRenderedOp). Each image source recursively manifestss the resolution through its DAG*.
* **DAG** = directed acyclic graph	

Extensions classes

You can add extensions to this system, specifically subclasses of XMLHandler or X_OperationJAI. XMLHandler is the base class for datatypes that can appear as elements in the XML. For each such class, methods must be implemented to generate XML schema describing the element for the datatype and correspondingly to parse the XML and instantiate an object of that type. These classes must be put in a jar file, which is indicated on the command line with the '-extensions' option or put in a file named 'extensions.jar'

Features Image

At the core of the image generation is the “features image”, a conceptual abstraction that separates aspects of the image generation that are related to individual features (such as location, shape and intensities of features) from global effects of the image (such as overall background, streaks, etc.). This functionality has been implemented as a JAI operation (“featuresimage”) to facilitate incorporation of the features image into a JAI recipe (directed acyclic graph built with JAI operations).