

# J-Express Scripting



## Internal J-Express Objects

Some classes exist in J-Express that gives access to various framework objects such as the internal desktop and the project tree. Most structures used in J-Express are available from the *main* object. The other central object is the *data* object which contains all available information about the selected dataset.

Here are the most important fields in the two framework classes and the Group class for group information:

### Main

Field	Type	Description
MW.JDesktopPanel	JDesktopPane	The main desktoppane containing all InternalFrames
addNode(TreeNode child,TreeNode parent)	void	Adds a project node (for instance a dataset) to the parent node
getProjectTree()	JTree	The main project tree
getProperties()	Hashtable	Settings that are saved when J-Express closes and loaded at startup. Remember that objects put into this table must be serializable.
getSelected()	Object	The selected object (node) in the project tree. Are usually casted to DataSet (This is done automatically in jython: for instance Dat = main.getSelected() where Dat is used a DataSet object)
getTreeModel()	DefaultTreeModel	The project tree TreeModel
getTreeRoot()	TreeNode (DataSet)	The root of the project tree
numFormat	java.text.DecimalFormat	The decimalformatter used in all objects

Examples:

```
#Create a JInternalFrame and put it into the main desktop
dialog = JInternalFrame("Script")
dialog.getContentPane().add(new JLabel("test"))
dialog.setLocation(300,300)
dialog.pack()
dialog.setClosable(1)
dialog.setResizable(1)
```

```
dialog.setIconifiable(1)
```

```
main.MW.JDesktopPane1.add(dialog)
dialog.setVisible(1)
#store some values in the Hashtable storage
Hashtable hash = main.getProperties()
hash.put("aninteger",new Integer(3))
hash.put("aString", "this is a string")
```

```
#Next time J-Express starts, the following script is valid:
Hashtable hash = main.getProperties()
Integer aninteger = hash.get("aninteger")
aString astr = hash.get("aString")
print astr
```

## Data

FileId	Type	Description
<code>DataSet(double[][] data,String[][] infos,String[][] colinfos)</code>	Creator	Creates a new dataset from a double[][] array of data, a String[][] array of row(gene) annotation and a String[][] array of column annotation.
<code>addColumnGroup(Group gr,boolean last)</code>	void	Add a group of samples (columns) to the dataset, see description for the Group Object below
<code>addGroup(Group gr,boolean last)</code>	void	Add a group of genes (rows) to the dataset, see description for the Group Object below
<code>extract(Vector members)</code>	DataSet	The main method for creating subsets of data. The members Vector should contain Integers where each integer is a row that should be in the result dataset. This dataset is initially linked (contains only pointers to data from the parent dataset). To unlink the dataset (create its own <i>data[][]</i> vector) call the <i>setParentDataSet( DataSet parent )</i> first and then <i>unLink(0)</i> . To connect it to a dataset int the project tree, call <i>main.addNode(TreeNode child,TreeNode parent)</i>
<code>extractColumns(Vector members)</code>	DataSet	Same as above for columns.
<code>fireSelectionChangeEvent (Object source)</code>	void	Fires a change event so that all listener listening for selection changes are

		updated. Use this together with <i>setSelectedRows(int[] selectedRows)</i> or <i>setSelectedColumns(int[] selectedCols)</i>
getColInfos()	String[][]	Annotation for all samples
getColumnGroups()	Vector of Group objects	All sample groups (see Group object)
getData()	double[][]	The actual expression matrix
getLength()	int	Number of rows (genes) in the dataset
getWidth()	int	Number of Columns (Samples) in the dataset
getFile()	String	The name of the dataset (appears in the project tree)
getGroups()	Vector of Group objects	All gene groups (see Group object)
getIconImage()	ImageIcon	The icon of the dataset
getInfo()	String	The info field
getInfoHeaders()	String[]	Row annotation headers
getInfos()	String[][]	The gene (row) annotation
getSelectedColumns()	int[]	The column selection
getSelectedRows()	int[]	The row selection
getStructures()	Hashtable	A hashtable to store anything together with the dataset. Remember that object in this hash must be serializable
getNulls()	boolean[][]	The missing values in this dataset
hasNaN()	boolean	True if there is any NaN values in the data
linked()	boolean	True if this dataset does not contain data of it's own, but only has indices to the parent dataset
reLink(boolean showWarnings)	void	Removes data and links to the parent dataset
setColumnInfoHeaders(String[] colinfoHeaders)	Void	Set the headers for column annotation
setColInfos(String[][] colinfos)	Void	Set column annotation
setColumnGroups(Vector classes)	Void	Reset all column groups (Vector of Group)
setData(double[][] data)	Void	Set the data
setFile(String file)	Void	File is the name of the dataset
setGroups(Vector classes)	Void	Reset all row (gene) groups
setIcon(ImageIcon icon)	Void	Set the icon for this dataset
setInfo(String info)	Void	Set the info field
setInfoHeaders(String[] headers)	Void	Set headers for row (gene) annotation
setInfos(String[][] infos)	Void	Set row (gene) annotation

setMetaList(expresscomponents.Documentation.MetaInfo List MetaList)	Void	Meta list is the list of meta info
setStructures(Hashtable structures)	Void	Structures is a hashtable that is saved with the dataset. It can be used to store any kind of serializable object
setnulls(boolean[][] nulls)	Void	Set a matrix of missing values (true represents a missing value)
unLink(boolean showWarnings)	Void	Unlink the dataset from its parent dataset. This will copy all data overlapping between this dataset and its parent to this dataset.

## Group

FileId	Type	Description
Group()	Creator	Creates a new empty group
Group( boolean active, String name,Color color,boolean[] members, LineMark lineMark, String description)	Creator	Creates a new group. The members are representing the row or columns included in this group. This number must be the same dimension as the number of rows or columns. DataSet.addGroup(Group group, Boolean last) or DataSet.addColumnGroup(Group group, Boolean last) can be used to add the group to a dataset.
setActive(boolean active)	Void	Turn this group on or of
getColor()	Color	The group Color
setColor(Color color)	Void	Set the group Color
setDescription(String description)	Void	Set the group description
setMembers(boolean[] members)	Void	The rows or columns that should be a member of this group.
setName(java.lang.String name)	Void	Set the name of the group
getCopy()	Group	Get a clone of this group
getDescription()	String	Get the description n of this group
getGroupCount()	String	The number of members in this group
getMembers()	boolean[]	The group members
getName()	String	The name of the group
isActive()	boolean	True if this group is active
isMember(int row)	boolean	True if "row" is a member of this group

## **Importing objects**

Objects that are already in the J-Express classpath can be created by including their package in an import statement like:

```
from myclasses import myclass
```

## **Adding new libraries**

By putting a jar-file in the J-Express lib-folder, the library will automatically be included in the class path. Objects can then be created by including them in import statements.

*Example:*

A library called *mylib.jar* has a class called *myclass* in package *myclasses* with a constructor *myclass(String str)* and method *String mymethod(int anint)*.

We can use the library by putting it into the J-Express lib-folder. The following script is then valid:

```
from myclasses import myclass

aclass = myclass("a string")
anotherstring = aclass(55)

print anotherstring
```

For instance, the jFreeChart library is already present in the J-Express lib folder. New charts can be generated in the following way (for complete reference of the JFreechart API, please refer to <http://www.jfree.org/jfreechart/javadoc/>):

This script calculates the 3 first principal components using the Jama library and plots them in a JFreechart line chart. (The script is available as an example script in the J-Express script folder).

```
from java.lang import *
from org.jfree.chart import *
from org.jfree.chart.plot import *
from Jama import *
from expresscomponents import JDoubleSorter
from org.jfree.chart.renderer.category import *
from org.jfree.chart.axis import *
from org.jfree.data.category import *
from java.awt import Rectangle
from javax.swing import JFrame

sel = [4,5,6,7,90,12,44,43,43,22,11]
dat = data.getData()
m=data.getDataWidth()
```

```

colinfos = data.getColInfos()
rowinfos = data.getInfos()

A = Matrix(dat);
SVD =SingularValueDecomposition(A)

R = SVD.getSingularValues()

m2 = len(R)

dataset = DefaultCategoryDataset();

M=SVD.getV()

ARR = M.getArray()
r1 = M.getRowDimension()
c1 = M.getColumnDimension()

princ1=[]

#This is the principal component containing most of the variance
for j in range(0,m):
    #princ1=princ1+[ARR[j][0]]
    princ1=princ1+[ARR[j][2]]
    dataset.addValue(ARR[j][0],String("PC1"),String(String.valueOf(j)
))
    dataset.addValue(ARR[j][1],String("PC2"),String(String.valueOf(j)
))
    dataset.addValue(ARR[j][2],String("PC3"),String(String.valueOf(j)
))

chr = ChartFactory.createLineChart( "", "", "", dataset,
PlotOrientation.VERTICAL, 1,0,0 );

plot = chr.getCategoryPlot()

domainAxis = plot.getDomainAxis()
domainAxis.setCategoryLabelPositions(
CategoryLabelPositions.createUpRotationLabelPositions(Math.PI / 6.0)
);

data.setSelectedRows(sel)
data.fireSelectionChangeEvent(data)

rend = plot.getRenderer()
rend.setShapesVisible(1);

#Create the panel
panel = ChartPanel(chr)

#Create the dialog frame
dialog = JInternalFrame("Script")
dialog.getContentPane().add(panel)
dialog.setLocation(300,300)

```

```

dialog.pack()
dialog.setClosable(1)
dialog.setResizable(1)
dialog.setIconifiable(1)

main.MW.JDesktopPane1.add(dialog)
dialog.setVisible(1)

```

## Creating Plugins

J-Express plugins no longer need to be subclasses of the plugin classes. Instead, they are initiated from a jython script. All jar-files placed in the J-Express plugins folder at startup will be included in the classpath. Starting the plugin must be done by passing the correct parameters to your plugin class from the script interface. A couple of test scripts are included in the main J-Express installation.

A plugin becomes available from the J-Express framework when an XML file with the below parameters are found either in the plugins folder or within a jar file located in the plugins folder.

Example:

```

<?xml version="1.0" encoding="UTF-8"?>
<java version="1.5.0_06" class="java.beans.XMLDecoder">
  <object class="expresscomponents.plugins.Settings">
    <void property="buttonImage">
      <string>/plugins/stat1.gif</string>
    </void>
    <void property="launchScript">
      <string>

from javax.swing import JOptionPane
from plugins import *

if data==None or data.getDataLength()==0:
    JOptionPane.showMessageDialog(main.MW, "No Data Set
Selected", "Missing Data", JOptionPane.ERROR_MESSAGE)
else:
    st = Stat1(main,data)

</string>
</void>
    <void property="pluginName">
      <string>Statistics tutorial Plugin</string>
    </void>
    <void property="Description">
      <string>A Statistics tutorial Plugin for calculating t-score and
regularized t-score</string>
    </void>
  </object>
</java>

```

The script is launched whenever a plugin is started by clicking the plugin button or menu in J-Express. “/plugins/stat1.gif” is the icon of the plugin and is in this case located in a jar-file. Stat1 is the plugin class and is in this case started with Stat1(main,data). If this plugin was an internal frame it could be added to the main J-Express frame with for instance:

```
main.MW.JDesktopPane1.add(st)
st.setVisible(1)
```

For more examples, see the plugins folder.

Join the J-Express forum at [www.molmine.com/forum](http://www.molmine.com/forum) to share your scripts or plugins with the J-Express community or ask questions.

