Icy Plugin Development
Training - Level 3

http://icy.bioimageanalysis.org
Icy Development Environment

Follow the **Quick Start** tutorial to set up your development environment and get the basics:

http://icy.bioimageanalysis.org/developer/create-a-new-icy-plugin/

Hello World! (in Icy)

Something as simple as...

Your plugin class name

Plugin actionable from a button

```java
public class HelloWorldPlugin extends PluginActionable {

    @Override
    public void run() {
        new AnnounceFrame("Hello, World!");
    }
}
```

Display an announce in Icy

Method called when user click on plugin button
Data structure

5D image data organized by channel

Sequence [XYCZT] (5D)

IcyBufferedImage [XYC] (3D)

Planar Image [XY] (2D)

PIXEL DATA
byte, short, int, float or double

Any data type!
Data structure - view

3D image
Z stack - single channel

4D image
timelapse of Z stack - single channel

5D image
timelapse of Z stack - multi channel

Z slider to navigate through Z slices

T slider to navigate through frames

Play / pause, loop, frame rate control
Working with image

The basics...

Get the current active image

```java
IcyBufferedImage image = getActiveImage();

// check if an image is opened
if (image == null)
{
    MessageDialog.showDialog("This plugin needs an opened image.");
    return;
}
```

Display a message if no image is opened

Retrieve image information

```java
int w = image.getSizeX(); Get the image width
int h = image.getSizeY(); Get the image height
int numChannel = image.getSizeC(); Get the number of channel
DataType type = image.getDataType_(); Get the image data type
double pixel = image.getData(0, 0, 0); Get the pixel value at specified X, Y, C position
```
Modify an image

Goal: Divide intensity by 2 in a single image

Naive way...

```java
for (int x = 0; x < w; x++)
{
    for (int y = 0; y < h; y++)
    {
        image.setData(x, y, 0, image.getData(x, y, 0) / 2);
    }
}
```

Iterate through all pixels of the image

Get pixel value at position X, Y, C

Set pixel value at position X, Y, C
Modify an image

Goal: Divide intensity by 2 in a single image

```java
for (int x = 0; x < w; x++)
{
    for (int y = 0; y < h; y++)
    {
        image.setData(x, y, 0, image.getData(x, y, 0) / 2);
    }
}
```

Slow! Because `image.setData(..)` causes image refresh events and other recalculations.

http://icy.bioimageanalysis.org/doc/icy-dev-plugin/ModifyImagePluginSlow.java
Modify an image

Using cursors

IcyBufferedImageCursor

cursor
x  y  c
0  4  0

double get(x, y, c)
void set(x, y, c, value)

- Handles pixel data type for you. You only use double values
- Readable way of accessing pixel data
- Handles internal temporal variables
- Access pixels in any order
- Avoids constant calls for internal image updates when changing pixel intensities: Using commitChanges()
Modify an image

Goal: Divide intensity by 2 in a single image

A simpler way... Using cursors

Create cursor

```java
IcyBufferedImageCursor cursor = new IcyBufferedImageCursor(image);
try {
    for (int y = 0; y < h; y++) {
        for (int x = 0; x < w; x++) {
            cursor.set(x, y, 0, cursor.get(x, y, 0) / 2);
        }
    }
} finally {
    cursor.commitChanges();
}
```

Use set and get to read/modify pixels

End image modification → image refresh and recalculations
Modify an image

Goal: Divide intensity by 2 in a single image

Create an *image cursor* for *image* using `IcyBufferedImageCursor`

```java
IcyBufferedImageCursor cursor = new IcyBufferedImageCursor(image);
try {
    for (int y = 0; y < h; y++) {
        for (int x = 0; x < w; x++) {
            cursor.set(x, y, 0, cursor.get(x, y, 0) / 2);
        }
    }
} finally {
    cursor.commitChanges();
}
```
Modify an image

Goal: Divide intensity by 2 in a single image

Read pixels using \( \text{get}(x, y, c) \). Returned value is a double.
Update pixels using \( \text{set}(x, y, c, \text{val}) \). \text{val} is a double.
Use \texttt{setSafe} to handle data type value bounds.

```java
IcyBufferedImageCursor cursor = new IcyBufferedImageCursor(image);
try {
    for (int y = 0; y < h; y++) {
        for (int x = 0; x < w; x++) {
            cursor.set(x, y, 0, cursor.get(x, y, 0) / 2);
        }
    }
} finally {
    cursor.commitChanges();
}
```
Modify an image

Goal: Divide intensity by 2 in a single image

When all changes have been made, use commitChanges() to make sure all changes are applied to the image.

```java
IcyBufferedImageCursor cursor = new IcyBufferedImageCursor(image);
try {
    for (int y = 0; y < h; y++) {
        for (int x = 0; x < w; x++) {
            cursor.set(x, y, 0, cursor.get(x, y, 0) / 2);
        }
    }
} finally {
    cursor.commitChanges();
}
```
Modify an image

Complete code with cursors ...

```java
IcyBufferedImage image = getActiveImage();

if (image == null) {
    MessageDialog.showMessageDialog("This plugin needs an opened image.");
    return;
}

int w = image.getSizeX(), h = image.getSizeY();
IcyBufferedImageCursor cursor = new IcyBufferedImageCursor(image);
try {
    for (int y = 0; y < h; y++) {
        for (int x = 0; x < w; x++) {
            cursor.set(x, y, 0, cursor.get(x, y, 0) / 2);
        }
    }
} finally {
    cursor.commitChanges();
}

http://icy.bioimageanalysis.org/doc/icy-dev-plugin/ModifyImagePluginCursor.java
```
Modify an image

Goal: Divide intensity by 2 in a single image

Another way… Using update flags

Start image modification

```java
image.beginUpdate();
try {
    for (int x = 0; x < w; x++) {
        for (int y = 0; y < h; y++) {
            image.setData(x, y, 0, image.getData(x, y, 0) / 2);
        }
    }
} finally {
    image.endUpdate();
}
```

End image modification → image refresh and recalculations

http://icy.bioimageanalysis.org/doc/icy-dev-plugin/ModifyImagePluginSimple.java
Modify an image

Goal: Divide intensity by 2 in a single image

An optimized solution… Similar performance as cursors but less readable

Object dataArray = image.getDataXY(0);

double[] doubleDataArray = Array1DUtil.arrayToDoubleArray(dataArray, image.isSignedDataType());

MathUtil.divide(doubleDataArray, 2d);

Array1DUtil.doubleArrayToSafeArray(doubleDataArray, image.getDataXY(0), image.isSignedDataType());

image.dataChanged();

http://icy.bioimageanalysis.org/doc/icy-dev-plugin/ModifyImagePluginFast.java
Modify an image

Complete code: optimized version

```java
IcyBufferedImage image = getActiveImage();

if (image == null)
{
    MessageDialog.showDialog("This plugin needs an opened image.");
    return;
}

Object dataArray = image.getDataXY(0);

double[] doubleDataArray = Array1DUtil.arrayToDoubleArray(dataArray, image.isSignedDataType());

MathUtil.divide(doubleDataArray, 2d);

Array1DUtil.doubleArrayToSafeArray(doubleDataArray, image.getDataXY(0), image.isSignedDataType());

image.dataChanged();
```

http://icy.bioimageanalysis.org/doc/icy-dev-plugin/ModifyImagePluginFast.java
Image utilities...

IcyBufferedImageUtil class

// Add a channel to the image
image = IcyBufferedImageUtil.addChannel(image);

// Change the image data type
image = IcyBufferedImageUtil.convertToType(image, DataType.DOUBLE, true);

// Extract a channel
image = IcyBufferedImageUtil.extractChannel(image, 1);

// Get a region of the image
image = IcyBufferedImageUtil.getSubImage(
    image, new Rectangle(10, 10, 50, 50), 0, 1);

// Resize the image
image = IcyBufferedImageUtil.scale(image, 500, 500);
Sequence processing & creation

Goal: Compute the maximum intensity projection over the Z dimension

Get the current active sequence

```java
Sequence sequence = getActiveSequence();

// check if a sequence is opened
if (sequence == null)
{
    MessageDialog.showMessageDialog("This plugin needs an opened sequence.");
    return;
}
```

Display a message if no sequence is opened
Sequence processing & creation

Goal: Compute the maximum intensity projection over the Z dimension

The idea...

```java
private Sequence sequence;

@Override
public void run()
{
    sequence = getActiveSequence();

    if (sequence == null) // check if a sequence is opened
    {
        MessageDialog.showDialog("This plugin needs an opened sequence.");
        return;
    }

    createResultSequence(); // Create a sequence with black frames
    computeMaxZProjection();
    showResultSequence();
}
```
**Sequence processing & creation**

Note **sequence** is accessible from other methods in the plugin (1, 2, 3)

```java
private Sequence sequence;

@Override
public void run()
{
    sequence = getActiveSequence();

    if (sequence == null) // check if a sequence is opened
    {
        MessageDialog.showDialog("This plugin needs an opened sequence.");
        return;
    }

    createResultSequence(); // Create a sequence with black frames
    computeMaxZProjection();
    showResultSequence();
}
```
Sequence processing & creation

1. Create a new sequence with black frames and using the original name

Note `resultSequence` is accessible from other methods in the plugin

```java
private Sequence resultSequence;

private void createResultSequence()
{
    resultSequence = new Sequence(sequence.getName() + " - Z projection");
    resultSequence.beginUpdate();
    for (int t = 0; t < sequence.getSizeT(); t++) {
        resultSequence.addImage(t,
                                new IcyBufferedImage(sequence.getSizeX(),
                                                      sequence.getSizeY(),
                                                      sequence.getSizeC(),
                                                      sequence.getDataType_()));
    }
    resultSequence.endUpdate();
}
```

Add an planar image at each frame
Sequence processing & creation

2. Compute max projection on sequence

`sequenceCursor` and `resultCursor` are accessible from other methods.

```java
private SequenceCursor sequenceCursor;
private SequenceCursor resultCursor;

private void computeMaxZProjection() {
    sequenceCursor = new SequenceCursor(sequence);
    resultCursor = new SequenceCursor(resultSequence);
    try {
        for (int t = 0; t < sequence.getSizeT(); t++) {
            computeFrame(t);
        }
    } finally {
        resultCursor.commitChanges();
    }
    resultSequence.dataChanged();
}
```

- Instantiate cursors
- Compute projection for each frame
- Commit changes on `resultSequence`
Sequence processing & creation

2.1. Compute max projection on each frame

```java
private void computeFrame(int t) {
    for (int c = 0; c < sequence.getSizeC(); c++) {
        for (int z = 0; z < sequence.getSizeZ(); z++) {
            projectMax(t, c, z);
        }
    }
}
```

1 projection on each channel

```java
private void projectMax(int t, int c, int z) {
    for (int y = 0; y < sequence.getHeight(); y++) {
        for (int x = 0; x < sequence.getWidth(); x++) {
            resultCursor.set(x, y, 0, t, c,
                             Math.max(resultCursor.get(x, y, 0, t, c),
                                       sequenceCursor.get(x, y, z, t, c)));
        }
    }
}
```

Compute projection on the volume

at each pixel of the slice z

Keep the maximum value

http://icy.bioimageanalysis.org/doc/icy-dev-plugin/MaximumZProjectionPlugin.java
Sequence processing & creation

3. Show result

```java
private void showResultSequence()
{
    addSequence(resultSequence);
}
```

Before

![Before image](image1)

After

![After image](image2)

Z max projection

http://icy.bioimageanalysis.org/doc/icy-dev-plugin/MaximumZProjectionPlugin.java
Sequence processing & creation exercice

Try computing a projection like the maximum Z-projection, but on the time axis (maximum T-projection)
Sequence processing & creation exercice - solution

Basically the idea is to exchange the Z and T dimension in the algorithm.

\[
\text{computeMaxZProjection} \rightarrow \text{computeMaxTProjection} \\
\text{computeFrame} \rightarrow \text{computeSlice}
\]

```java
private void computeMaxTProjection() {
    ...
    try {
        for (int z = 0; z < sequence.getSizeZ(); z++)
            computeSlice(z);
    }
    ...
}

private void computeSlice(int z) {
    for (int c = 0; c < sequence.getSizeC(); c++) {
        for (int t = 0; t < sequence.getSizeT(); t++) {
            projectMax(t, c, z);
        }
    }
}
```

[Link to source code](http://icy.bioimageanalysis.org/doc/icy-dev-plugin/MaximumTProjectionPlugin.java)
Sequence utilities...

**SequenceUtil class**

```java
// Get a copy of specified sequence
result = SequenceUtil.getCopy(sequence);

// Add one Z slice at position 5
SequenceUtil.addZ(sequence, 5, 1);

// Convert the specified sequence (Z-stack format) to timelapse
SequenceUtil.convertToTime(sequence);

// Remove the frame 12
SequenceUtil.removeTAndShift(sequence, 12);

// Get a sub-region of the sequence
result = SequenceUtil.getSubSequence(sequence,
            new Rectangle5D.Integer(
            //x   y   z  t  c
            50, 50, 0, 0, 0,
            //sx   sy   sz
            100, 100, sequence.getSizeZ(), sequence.getSizeT(), 1));
```
The **run** method of the plugin is always called on the *Event Dispatch Thread* (graphic thread) as you need it to create your graphical interface. But if you do a long process in this thread the application won’t respond for sometime.

```java
public class BadPlugin extends PluginActionable {
    @Override
    public void run() {
        // We are in the Event Dispatch Thread

        // this is a really bad idea! It blocks the UI
        ThreadUtil.sleep(10000);
    }
}
```
Thread & good practices

ThreadUtil class

```java
public class GoodPlugin extends PluginActionable {

    @Override
    public void run() {
        // WE ARE IN THE EVENT DISPATCH THREAD (GRAPHICS THREAD)

        // request the following Runnable to execute in background thread
        ThreadUtil.bgRun((Runnable) {
            @Override
            public void run() {
                // WE ARE NOT ANYMORE IN THE EVENT DISPATCH THREAD
                ThreadUtil.sleep(10000);
            }
        });
    }
}
```
ROI (Region Of Interest)

ROI are very important in Icy. They can both be used as input information (to know where to apply a process for instance) but they are also the way to provide results. Almost all plugins generate ROIs as results from which you can easily extract quantitative information.
ROI in Icy

Quantitative information extracted from ROIs

ROI manipulation tools
All ROI have 5D information (XYCZT size/pos.) But specialized versions provide more functionalities & operations.
ROI - basics

Generic 5D contains method

```java
boolean contained = roi.contains(x, y, z, t, c);
```

```java
if (roi instanceof ROI2D)
{
    contained = ((ROI2D) roi).contains(x, y);
}
```

2D specialized contains method

Get all 2D roi from this sequence

```java
List<ROI2D> roi2ds = sequence.getROI2Ds();
```

Get all selected roi from this sequence

```java
List<ROI> rois = sequence.getSelectedROIs();
```

ROI `roi` = sequence.getSelectedROI();

Get first selected ROI from this sequence

boolean contained = `roi`.contains(x, y, z, t, c);

if (`roi` instanceof `ROI2D`)
{
    contained = `((ROI2D) roi)`.contains(x, y);
}
ROI - mean intensity

Simple method using `roi.contains(...)`:

```java
// consider first image only here
IcyBufferedImage image = sequence.getFirstImage();
double mean = 0;
double sample = 0;

for (int x = 0; x < sequence.getSizeX(); x++)
{
    for (int y = 0; y < sequence.getSizeY(); y++)
    {
        if (roi.contains(x, y))
        {
            mean += image.getData(x, y, 0);
            sample++;
        }
    }
}

System.out.println("mean intensity over ROI: " + (mean / sample));
```

Roi ‘contains’ test (can be slow)

http://icy.bioimageanalysis.org/doc/icy-dev-plugin/MeanIntensityPluginROI.java
ROI - mean intensity

Problem:

The roi `contains()` method can be slow depending on the ROI implementation.

→ slow processing on large images

Solution:

Use `BooleanMask2D` object for fast `contains()` method.
ROI - mean intensity

Faster method using `BooleanMask2D`:

```java
BooleanMask2D mask = roi.getBooleanMask(true);
// consider first image only here
IcyBufferedImage image = sequence.getFirstImage();
double mean = 0;
double sample = 0;

for (int x = 0; x < sequence.getSizeX(); x++)
{
    for (int y = 0; y < sequence.getSizeY(); y++)
    {
        if (mask.contains(x, y))
        {
            mean += image.getData(x, y, 0);
            sample++;
        }
    }
}
```

Mask ‘contains’ test (fast)

[http://icy.bioimageanalysis.org/doc/icy-dev-plugin/MeanIntensityPluginBooleanMask.java](http://icy.bioimageanalysis.org/doc/icy-dev-plugin/MeanIntensityPluginBooleanMask.java)
ROI - mean intensity

Alternative method using DataIterator:

Create a new Sequence data iterator that iterates through all pixels contained in a ROI

```java
SequenceDataIterator iterator = new SequenceDataIterator(sequence, roi);
double mean = 0;
double sample = 0;

while (!iterator.done())
{
    mean += iterator.get();  // Get the current pixel value
    sample++;
    iterator.next();  // Move to the next pixel
}
```

While we still have pixels to process

http://icy.bioimageanalysis.org/doc/icy-dev-plugin/MeanIntensityPluginIterator.java
The idea: Create a ROI which contains all pixel value $\geq$ threshold value

$\rightarrow$ many different ways to do that!

Let's see some of them…
Naive way.. Create a new empty ROI (type: mask)

ROI2DArea roi = new ROI2DArea();
// consider first image only here
IcyBufferedImage image = sequence.getFirstImage();

for (int x = 0; x < sequence.getSizeX(); x++)
{
    for (int y = 0; y < sequence.getSizeY(); y++)
    {
        if (image.getData(x, y, 0) >= threshold)
        {
            roi.addPoint(x, y);
        }
    }
}

sequence.addROI(roi);

Attach the result ROI to the sequence (make it visible)

http://icy.bioimageanalysis.org/doc/icy-dev-plugin/ThresholdPluginSimple.java
Faster method...

Get sequence data in double array format

// consider first image and first channel only here
double[] doubleArray = Array1DUtil.arrayToDoubleArray(sequence.getDataXY(0, 0, 0), sequence.isSignedDataType());
boolean[] mask = new boolean[doubleArray.length];

Create a boolean array with same size as the data array

Set mask to true where pixels are accepted

for (int i = 0; i < doubleArray.length; i++)
    mask[i] = (doubleArray[i] >= threshold);

BooleanMask2D mask2d = new BooleanMask2D(sequence.getBounds2D(), mask);
ROI2DArea roi = new ROI2DArea(mask2d);

Create a BooleanMask2D object and then a ROI from this mask

http://icy.bioimageanalysis.org/doc/icy-dev-plugin/ThresholdPluginFast.java
ROI utilities...

ROIUtil class

```java
// Get the area of the specified roi and return result with correct units.
double area = ROIAreaDescriptor.computeArea(roi, sequence);

// Return the mean intensity of sequence pixels contained in the ROI.
double meanIntensity = ROIMeanIntensityDescriptor.computeMeanIntensity(roi, sequence);

// Process the union of specified rois
ROI unionRoi = ROIUtil.getUnion(rois);

// Process the subtraction of roi2 to roi1
roi = ROIUtil.subtract(roi1, roi2);
```
Overlay - concept

Image + ROI + Tooltip = View
Overlay in Icy

Overlay allow to enrich image display with various informations.

Layer panel displaying all layers visible on this image

Some layers can display extra settings when you select them
So plugins can use Overlays to display rich information over images...

...but it also allows user interaction by receiving directly mouse and key events.
public class SimpleCrossOverlay extends Overlay {

    public SimpleCrossOverlay() {
        super("Simple cross");
    }

    @Override
    public void paint(Graphics2D g, Sequence sequence, IcyCanvas canvas) {
        if (g != null) {
            // paint a yellow cross all over the image
            g.setColor(Color.YELLOW);
            g.setStroke(new BasicStroke(5));
            g.drawLine(0, 0, sequence.getWidth(), sequence.getHeight());
            g.drawLine(0, sequence.getHeight(), sequence.getWidth(), 0);
        }
    }
}

Create a new Overlay class

Name (appears in the ‘Layers’ tab)

Method called by Icy to draw the overlay

Drawing is relative to the image coordinate system
Overlay - user interaction

Mouse click event method

```java
@Override
public void mouseClicked(MouseEvent e, Point2D point, IcyCanvas canvas)
{
    // remove the overlay when the user clicks on the image
    remove();
}
```

Remove the Overlay from sequences where it is attached

Almost done...

```java
sequence.addOverlay(new SimpleCrossOverlay());
```

Add a new `SimpleCrossOverlay` to the sequence
Overlay

Before

After

http://icy.bioimageanalysis.org/doc/icy-dev-plugin/SimpleCrossOverlayPlugin.java
Create an overlay displaying the overlapped ROI name at the mouse cursor position.
public class ROINameOverlay extends Overlay {

    String roiName; // stored ROI name
    Point2D mousePos; // stored mouse position

    public ROINameOverlay() {
        super("ROI name display"); // initialize vars
        roiName = null;
        mousePos = null;
    }

    @Override
    public void paint(Graphics2D g, Sequence sequence, IcyCanvas canvas) {
        if (g != null) {
            // paint ROI name in white
            g.setColor(Color.WHITE);
            if (roiName != null) {
                g.drawString(roiName, (float) mousePos.getX(), (float) mousePos.getY());
            }
        }
    }
}
public void mouseMove(MouseEvent e, Point2D imagePoint, IcyCanvas canvas) {
    updateROIName(canvas.getSequence(), imagePoint);
}

public void mouseDrag(MouseEvent e, Point2D imagePoint, IcyCanvas canvas) {
    updateROIName(canvas.getSequence(), imagePoint);
}

private void updateROIName(Sequence sequence, Point2D imagePoint) {
    // find overlapped ROI
    ROI2D roi = getOverlappedROI(sequence, imagePoint);
    // store its name
    if (roi != null) {
        roiName = roi.getName();
    } else {
        roiName = null;
    }
    // store mouse position
    mousePos = (Point2D) imagePoint.clone();
    // notify that we need to be redraw
    painterChanged();
}
Overlay - Anchor2D & tools

Anchor2D class

Offer simple object mouse manipulation as selection and drag operation

ImageOverlay class

Simple overlay to display an image

VtkPainter interface

Easier VTK overlay implementation
Creating a user interface can be a pain (especially in Java)

But in 90% of cases they look the same:
- Parameters (name, value)
- Buttons here, labels there, etc.

Our goal: provide an API that is
- Simple (your plugin in less than 5 min.)
- Standardized (users don’t get lost)
- Powerful (bells & whistles included!)
public class MyNiceLookingPlugin extends EzPlug {
    @Override
    public void initialize() {
        // create the interface here
    }

    @Override
    public void execute() {
        // do some “real” work
    }

    @Override
    public void clean() {
        // clean things when closing
    }
}

The interface is generated automatically

*No more hardcore Swing!*

- Load/save parameters
- Online documentation
- Progress bar
public class MyNiceLookingPlugin extends EzPlug {

    EzVarInteger age = new EzVarInteger("your age", 30, 10, 100, 1);

    EzVarBoolean isYummy = new EzVarBoolean("like chocolate?", true);

    @Override
    public void initialize() {
        addEzComponent(isYummy);
        addEzComponent(age);
    }

    ...
}
public class MyNiceLookingPlugin extends EzPlug {

    enum Choice {
        Yes, No, Perhaps
    }

    EzVarEnum<Choice> choice = new EzVarEnum<Choice>(
            "choice", name
            Choice.values(), valid values
            Choice.Perhaps); default

    @Override
    public void initialize()
    {
        addEzComponent(choice);
    }
    ...
}
### EzPlug - GUI - parameters

<table>
<thead>
<tr>
<th>Java File</th>
<th>Parameter Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>EzVarBoolean.java</td>
<td>Check box</td>
</tr>
<tr>
<td>EzVarDimensionPicker.java</td>
<td>Spinner</td>
</tr>
<tr>
<td>EzVarDouble.java</td>
<td>Spinner, Combo box</td>
</tr>
<tr>
<td>EzVarDoubleArrayNative.java</td>
<td>Free text (spaced values), Combo box</td>
</tr>
<tr>
<td>EzVarEnum.java</td>
<td>Combo box</td>
</tr>
<tr>
<td>EzVarFile.java</td>
<td>Button (File chooser)</td>
</tr>
<tr>
<td>EzVarFileArray.java</td>
<td>Button (Multi-File chooser)</td>
</tr>
<tr>
<td>EzVarFloat.java</td>
<td>Spinner, Combo box</td>
</tr>
<tr>
<td>EzVarFloatArrayNative.java</td>
<td>Free text (spaced values), Combo box</td>
</tr>
<tr>
<td>EzVarFolder.java</td>
<td>Button (opens a folder chooser)</td>
</tr>
<tr>
<td>EzVarInteger.java</td>
<td>Spinner, Combo box</td>
</tr>
<tr>
<td>EzVarIntegerArrayNative.java</td>
<td>Free text (spaced values), Combo box</td>
</tr>
<tr>
<td>EzVarPlugin.java</td>
<td>Combo box</td>
</tr>
<tr>
<td>EzVarSequence.java</td>
<td>Combo box</td>
</tr>
<tr>
<td>EzVarSwimmingObject.java</td>
<td>Combo box</td>
</tr>
</tbody>
</table>
public class MyNiceLookingPlugin extends EzPlug
{
    ActionListener action = new ActionListener()
    {
        @Override
        public void actionPerformed(ActionEvent e)
        {
            new AnnounceFrame("ouch!");
        }
    };

    EzButton tap = new EzButton("hit me", action);

    @Override
    public void initialize()
    {
        addEzComponent(new EzLabel("Boring text"));
        addEzComponent(tap);
    }
    ...
}
public class MyNiceLookingPlugin extends EzPlug {
    EzVarBoolean b1 = new EzVarBoolean("test #1", false);
    EzVarBoolean b2 = new EzVarBoolean("test #2", true);
    EzVarBoolean b3 = new EzVarBoolean("some other option", false);
    EzGroup grp = new EzGroup("My group", b1, b2);
    
    @Override
    public void initialize()
    {
        addEzComponent(grp);
        addEzComponent(b3);
    }
    ...
}
public class MyNiceLookingPlugin extends EzPlug {
    EzVarBoolean b1 = new EzVarBoolean("test #1", false);
    EzVarBoolean b2 = new EzVarBoolean("test #2", true);
    EzVarBoolean b3 = new EzVarBoolean("some other option", false);
    EzGroup grp = new EzGroup("My group", b1, b2);

    @Override
    public void initialize() {
        addEzComponent(grp);
        addEzComponent(b3);
        b3.addVisibilityTriggerTo(grp, false);
    }
    ...
    "show {grp} only if {b3} has value {false}"
public class MyNiceLookingPlugin extends EzPlug implements EzStoppable
{
  ...
  @Override
  public void execute()
  {
    // who am i?!
    Thread t = Thread.currentThread();
    for (int i = 1; i <= 100; i++)
    {
      // do stuff
      if (t.isInterrupted())
        break;
    }
    ...
  }
}
public class MyNiceLookingPlugin extends EzPlug implements EzStoppable {
    boolean stopFlag;

    @Override
    public void stopExecution() {
        stopFlag = true;
    }

    @Override
    public void execute() {
        stopFlag = false; // don’t, stop me noooow!
        for (int i = 1; i <= 100; i++) {
            // do stuff
            if (stopFlag)
                break;
        }
    }
}

For more control...

EzPlug - stop a running process
EzPlug - going further

- Progress bar / main buttons can be hidden
- Supports “regular” AWT/Swing components
- Available as modal dialogs (use EzDialog)
- Open to new parameter types (extend EzVar)
EzPlug - going beyond

Make your plugin compatible with block

```java
public class MyNiceLookingPlugin extends EzPlug
implements Block {
    EzVarInteger age = ...
    EzVarText text = ...
    ...
    // no GUI for this one
    VarInteger out = new VarInteger("out", 0);
    ...
    // fill the other methods
    public void declareInput(VarList inputMap) {
        inputMap.add(age.getVariable());
        inputMap.add(text.getVariable());
    }

    public void declareOutput(VarList outputMap) {
        outputMap.add(out);
    }
```
EzPlug - exercise

Create a GUI from the *ROI - Threshold* example using *EzPlug*. It should be compatible with Block programming / Protocols.

**Input**
- Input image (Sequence)
- Threshold value (int)

**Output**
- ROI
public class MyThresholdPlugin extends EzPlug implements Block {
    EzVarSequence sequence = new EzVarSequence("Sequence");
    EzVarInteger threshold = new EzVarInteger("Threshold", 128, 1, 255, 1);
    VarROIArray rois = new VarROIArray("ROI");

    public void initialize() {
        addEzComponent(sequence);
        addEzComponent(threshold);
    }

    public void declareInput(VarList inputMap) {
        inputMap.add("sequence", sequence.getVariable());
        inputMap.add("threshold", threshold.getVariable());
    }

    public void declareOutput(VarList outputMap) {
        outputMap.add("rois", rois);
    }

    public void execute() {
        ROI roi = doThreshold(sequence.getValue(), threshold.getValue());
        if ((roi != null) && !isHeadLess())
            sequence.getValue().addROI(roi);
        rois.setValue(new ROI[] {roi});
    }

    //...
Extra - Native library support

Easy integration of native library directly in the plugin jar file

```
//META-INF
  MANIFEST.MF
/plugins
  myname
    myPlugin.class
/lib
  unix32
    myLib.so
  unix64
    myLib.so
  mac32
    myLib.dylib
  mac64
    myLib.dylib
  win32
    myLib.dll
  win64
    myLib.dll
```

JAR file structure

- java classes file(s)
- native library files for each operating system
Extra - Native library support

Load the native library through the plugin

```java
public class NativeLibraryTestPlugin extends PluginActionable {
    public NativeLibraryTestPlugin() {
        super();
        loadLibrary("myLib");
    }

    @Override
    public void run() {
        myLib.executeNativeMethod(..);
    }
}
```

Load the native library from the plugin JAR file

Executive the library native method
Extra - Special behavior plugin

Icy define specific plugin base classes and interfaces with special behaviours to extend freedom of plugin interactions with Icy.

- **PluginActionable**: Icy will automatically add a button in the ribbon menu allowing to execute your plugin (*EzPlug* extends it)
- **PluginDaemon**: Icy will automatically starts your plugin on application start (ideal for permanent enhancer)
- **PluginBundled**: Use it when you can to package several plugin into a single JAR file
- **PluginThreaded**: Icy will execute the plugin in a separate thread
- **PluginLibrary**: This plugin is just used a base library for other plugin
Extra - Enhancing Icy features

In the same idea, Icy also define specific plugin base classes and interfaces to extend default Icy features as visualization mode, ROI type, statistics..

- **PluginImporter**: add new importer.  
  An importer can import/load any kind of resource (image, rois, protocols..) from any source (file, database..)
- **PluginSequenceFileImporter**: add new Sequence importer. This importer specifically takes a File as input and produces a Sequence (image) as result. For instance this is how we add MP4 file loading support.
- **PluginCanvas**: add new visualization mode (as channel montage view)
- **PluginROI**: add new ROI type
- **PluginROIDescriptor**: add new ROI descriptor(s) (statistic)
Extra - Enhancing Icy features - example

Adding a new ROI descriptor..

```java
public class ROISizeDescriptor extendsROIDescriptor {
    public static final String ID = "Size";

    public ROISizeDescriptor() {
        super(ID, "Size", Number.class);
    }

    public String getDescription() {
        return "Size (from 2D bounding box)";
    }

    public Object compute(ROI roi, Sequence sequence) throws UnsupportedOperationException {
        // get 2D bounds of ROI
        Rectangle2D bound2D = roi.getBounds5D().toRectangle2D();
        // return max size
        return Double.valueOf(Math.max(bound2D.getWidth(), bound2D.getHeight()));
    }
}
```
Extra - Enhancing Icy features - example

Adding a new ROI descriptor..

```java
public class MySizeROIDescriptorPlugin extends Plugin implements PluginROIDescriptor {
    public static final String ID_SIZE = ROISizeDescriptor.ID;

    // create the ROI Descriptor only once
    public static final ROISizeDescriptor sizeDescriptor = new ROISizeDescriptor();

    public List<ROIDescriptor> getDescriptors() {
        // return available descriptors in this plugin
        return CollectionUtil.createArrayList(sizeDescriptor);
    }

    public Map<ROIDescriptor, Object> compute(ROI roi, Sequence sequence) throws UnsupportedOperationException {
        Map<ROIDescriptor, Object> result = new HashMap<>();
        // compute descriptor
        result.put(sizeDescriptor, sizeDescriptor.compute(roi, sequence));
        // return result
        return result;
    }
}
```

http://icy.bioimageanalysis.org/doc/icy-dev-plugin/ROISizeDescriptorPlugin.java
What you need next??

When your plugin is done you may want to distribute it, for that you can check out this article:

How to publish a plugin

It’s really important to provide good documentation to your plugin. The plugin can be the best in its application, if it doesn’t have (or not enough) documentation it won’t be used...

If you are stuck and need help, you can contact us through this forum:

https://forum.image.sc/tag/icy

Don’t forget to use the correct category/tags so we can quickly find it (“development” and use the “icy” tag)
Keep in touch!

Icy

http://icy.bioimageanalysis.org

Support forum

https://forum.image.sc/tag/icy

@Icy_BioImaging