Scripting

Application Value

- Applications specialize in producing, manipulating specific data types
- An application's value can thus be seen in two lights
  - The data it produces/manages
  - The functionality it provides

Application Value

- But…
  - No one application do everything
  - No one application can include support for every possible use
  - Our data may not always be in the application's native format
  - The application's data may not be in the final format we require
- How can we increase the chance the application can bend to unforeseen, real-world needs?

Planning for Flexibility

- An application gains value if it can:
  - Import/export data
  - Be internally scripted
  - Be externally controlled/scripted
- Gains further value if the source code is freely available…
- These facilities extend the range of possibilities beyond the delivered capabilities

Scripting Options

- Recording Events
- Incorporating Scripting Engine
  - Several Java options
- External scripting / control

Recording Events

- Sometimes done by the app itself
  - Most basic level of scripting possible
- Can also be done by another application
  - Keystroke/mouse logger which re-injects events
  - Slightly more complex to code, but easily do-able
- Uses:
  - Preparing a demo
  - Testing a user interface
  - When more robust scripting isn't available
  - Keyboard logging (for good or ill)

Recording Events

```
// Event stream for recording events
private Vector<Serializable> eventStream = null;
private void handleMousePress(java.awt.event.MouseEvent e) {
   ... recordEvent(e); ... }


private void recordEvent(MouseEvent event) {
            if (interactionMode.equals(InteractionMode.RECORDING)) {
                        long thisTime = System.currentTimeMillis();
                        eventStream.add(thisTime - lastTime);
                        eventStream.add(event);
                        lastTime = thisTime;
            }
}
```

Playback Events

```
// Create and start a thread to play back the events
Thread playbackThread = new Thread(new Runnable() {
        public void run() {
                Iterator<Serializable> iter = playbackStream.iterator();
                while (iter.hasNext() &&
                interactionMode.equals(InteractionMode.PLAYBACK)) {
                        Long waitTime = (Long) iter.next();
                        MouseEvent event = (MouseEvent) iter.next();
                        Thread.sleep(waitTime);
                        switch (event.getID()) {
                        case MouseEvent.MOUSE_PRESSED:
                                handleMousePress(event);        break;
                        case MouseEvent.MOUSE_DRAGGED:
                                handleMouseDragged(event);    break;
                        case MouseEvent.MOUSE_RELEASED:
                                handleMouseReleased(event);   break;
                        }}}});
playbackThread.start();
```

Scripting Requirements

- External “language”
- Internal support

Scripting Engine Option 1

- Build on the undo/redo command objects
- Recall:
  - Each action in the interface corresponds to a command object implementing “undo” and “redo” (and perhaps “do” for clarity).
    - void insertText(doc, “I have a dream”);
    - void moveSelection(doc, Direction.RIGHT, 1, Unit.PARAGRAPH);
    - void extendSelection(doc, Direction.RIGHT, 1, Unit.SENTENCE);
    - void boldSelection(doc);
- Design a language interpreter that calls these methods

```
public class TriangleBaseUndoableEdit extends AbstractUndoableEdit {
  private TriangleModel model;
protected double oldBase;
  protected double newBase;

  private TriangleBaseUndoableEdit(TriangleModel model,
                double oldBase, double newBase) {
    this.model = model;
    this.oldBase = oldBase;
    this.newBase = newBase;
  }
 public void undo() {
      this.model.setBase(this.oldBase);
  }
   public void execute() {
       this.model.setBase(this.newBase);
   }
  public void redo() {
      this.execute();
  }
}
```

Issues

- Consider the following scripting commands:
  - void insertText(doc, “I have a dream”);
  - void moveSelection(doc, Direction.RIGHT, 1, Unit.PARAGRAPH);
  - void extendSelection(doc, Direction.RIGHT, 1, Unit.SENTENCE);
  - void boldSelection(doc);
- Additional capabilities needed?
- Exposure of appropriate data structures?
  - Need to understand data types … or crash program.

Scripting Engine Option 2

- Include a full interpreter library
- Uses reflection to give access to public members of program's data structures
  - Implies that you need a language supporting reflection
  - Otherwise need to include some kind of adapter library
- Options...

Interpreter Options

- Java
  - Jython
    - www.jython.org/
    - Python implemented as a Java library
    - provides dead-simple scripting engine for Java
  - BeanShell
    - www.beanshell.org/
    - Very Java-like syntax
  - Groovy
    - groovy.codehaus.org/
    - Python, Ruby, Smalltalk influences
  - Rhino
    - www.mozilla.org/rhino/)
    - Java-based JavaScript
  - JRuby
    - jruby.sourceforge.net/
    - Ruby implemented in Java
- .NET languages
  - IronPython (by same guy who did Jython)
  - Probably others...

Jython Demo: Scripting Triangles

Demo 1:

```
model.setBase(30)
model.setBase(40)
```

Demo 2:

```
import time
for i in range(10, 101, 5):
  model.setValues(i, i)
  time.sleep(0.25)
```

Jython Example (1/2)

```
// Import the python interpreter
import org.python.util.PythonInterpreter;
...
public class ScriptingView extends JPanel {
...
   private JTextArea script;
   private JButton executeButton;
   private PythonInterpreter pyInterp = null;
...
            private void initInterpreter() {
                        pyInterp = new PythonInterpreter();

                        // Make the these objects available to scripts
                        pyInterp.set("model", ScriptingView.this.model);
                        pyInterp.set("app", Application.getInstance());
                        pyInterp.set("frame", Application.getInstance()
                                                .getActiveFrame());
            }
```

Jython Example (2/2)

```
private void registerListeners() {
        this.executeButton.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                        if (pyInterp == null) {
                                ScriptingView.this.initInterpreter();
                        }
                        // Get the script we are to run.
                        final String s = ScriptingView.this.script.getText();
                        new Thread() {
                                // Execute the script
                                public void run() {   pyInterp.exec(s);    }
                        }.start();
                }
        });
}
```

Initialization Scripting

```
public void newDocument() {
                final model.TriangleModel model = new …
...
                if (new File("tri.ini").canRead()) {
                        PythonInterpreter pyInterp = new  PythonInterpreter();
                        pyInterp.set("app", Application.getInstance());
                        pyInterp.set("model", model);

                        pyInterp.set("frame", frame);
                        pyInterp.execfile("tri.ini");
                }
        }
...
```

Sample .ini file

```
print "Initializing triangle model..."
model.base = 60
model.height = 80

frame.setBounds(30*app.currentDocumentIndex(),
    30*app.currentDocumentIndex(),
    750, 500)
```

External Scripting / Control

- More complex than internal scripting
- Why?

External Scripting / Control

- Must expose scriptable portions of application in a standardized way to “outside world”
- How can such functionality be exposed?

Approaches to Exposing Functionality

- Initialization via command-line switches; reading from pipes
  - Run once
  - Not suitable for scripting a running, interactive application
- Stream-based protocol
  - Develop a protocol for communicating to application
  - Develop data formats
  - Create a network server/client paradigm
  - Named pipes
- Other possibilities
  - Shared memory, blackboards

Specific Implementations

- Various scripting options, depending on OS and exposed functionality:
  - Shell scripts on linux systems
  - Applescript for OSX
  - Powershell for Windows 7/8
  - Etc.

Summary

- Scripting significantly enhances the value of an application...
  - ... to those who know how to use scripting
  - ... and have more advanced needs
- Options include:
  - recording and playing back events
  - writing an interpreter that creates the command objects used by undo/redo
  - integrating a full-fledged interpreter using reflection
  - supporting an existing external scripting system