

Progzoo Case Study

We consider:

Progzoo XML format

Progzoo AJAX interaction

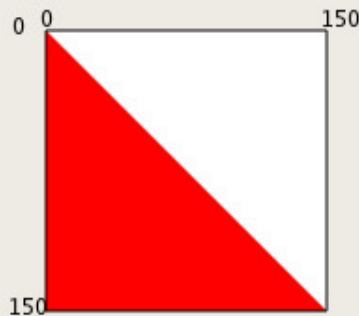
Retrieving Data using XML

```
<question height="300" width="500" imgOut="flag.png"
  className="Raster" title="Scotland">
  <blurb>
  The cross of St. Andrew shows diagonal lines on a blue
  background.<br/>
  <ul>
  <li>Complete the flag by drawing the other diagonal
  line.</li>
  <li>The white lines should be slightly thicker - try
  60.</li>
  </ul>
  </blurb>
  <shell lang="java" className="Flag"
    import="shells.xml#raster"/>
  <prog lang="java">
  static void drawFlag(Graphics2D g){
    g.setColor(Color.blue);
    g.fillRect(0,0,500,300);
    g.setStroke(new BasicStroke(20));
    g.setColor(Color.white);
    g.drawLine(0,0,500,300);
  }
  </prog>
  <answer lang="java">
  static void drawFlag(Graphics2D g){
    g.setColor(Color.blue);
    g.fillRect(0,0,500,300);
    g.setStroke(new BasicStroke(60));
    g.setColor(Color.white);
    g.drawLine(0,0,500,300);
    g.drawLine(500,0,0,300);
  }
  </answer>
</question>
```

We mix HTML and custom tags.

1. Napier

The Napier flag is a red triangle on a white square. It suggests the letter N.



```
static void drawFlag(Graphics2D g) {  
    g.setColor(Color.white);  
    g.fillRect(0,0,150,150);  
  
    Polygon tri = new Polygon();  
    tri.addPoint(0,0);  
    tri.addPoint(150,0);  
    tri.addPoint(0,150);  
    g.setColor(Color.red);  
    g.fillPolygon(tri);  
}
```

[Run](#)[Big](#)[Default](#)☐ Show all

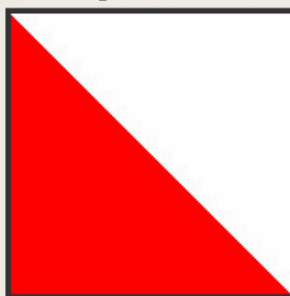
We create a polygon in order to fill it. The [Polygon](#) class is part of the standard awt package. We create a polygon then add points to it as required.

Your score is 50%

Your answer.



The right answer.



2. Bahamas

Done



Generating the Pages

The custom XML is converted to regular HTML using a perl script (or PHP in some cases). This script mostly simply calls an XSLT processor and converts the target XML against the XSLT sheet. Here is a snippet from that:

```
<xsl:template match="question">
<div class="q" id="q{$qnum}">
  <div class="fright">
    <table width='100%'><tr>
      <td class="qfleft" width="40%">
        <xsl:apply-templates select='blurb/*|blurb/text()'/>
      </td>
      <td>
        <form method="post"
          action="/r.cgi" enctype="multipart/form-data" name="fjava{$qnum}">
          <input type='hidden' name='shell' value='{shell[@lang="java"]}'/>
          <textarea name="c" rows="{ $rows}" cols="{ $cols}">
            <xsl:value-of select='prog[@lang="java"]'/>&#10;</textarea>
          <input type='hidden' name='answer' value='{answer[@lang="java"]}'/>
        </form>
      </td>
      <td>
        <form method="post"
          action="/r.cgi"
          enctype="multipart/form-data"
          name="gjava{$qnum}">

          <input type='hidden' name='prog' value=''/>
          <input type='hidden' name='shell' value=''/>
          <input type='hidden' name='imgOut' value='{@imgout}'/>
          <input type='hidden' name='width' value='{@width}'/>
          <input type='hidden' name='height' value='{@height}'/>
          <input type='hidden' name='className' value='{@classname}' t='x'/>
          <input type='hidden' name='copyFile' value='{@copyfile}'/>
          <input type='hidden' name='reference' value=''/>
        </form>
      </td>
    </tr>
  </table>
  </div>
</div>
```

The output of this process is HTML.

The translation process is light-weight and flexible.

Most of the "clever-stuff" is left to JavaScript.

All of the parameters are available to the JavaScript.

Request Response in Progzo

The JavaScript program packages up all of the parameters required by the execution engine. It sends the users attempt plus the correct answer plus information useful for logging activities.

answer	import java.awt.image.BufferedImage; import java.awt.*;--snip-- System.err.println(e); } } }
file	Raster
frag	static void drawFlag(--snip-- g.fillPolygon(tri); }
imgOut	flag.png
lang	java
prog	import java.awt.image.BufferedImage; import java.awt.*; --snip-- System.err.println(e); } } }
quest	1
rnd	0.13577332175402856
tut	00Flags/200FlagPoly.xml
uid	cs66
virgin	1

If the program compiles and executes without error then the answer is sent back together with the "score".

Where the output is an image then the execution engine simply sends a reference to the output.

```
{ "usrImg": "/scratch/105.png", "score": 50,  
  "ansImg": "/cache/00Flags.200FlagPoly.xml.java.1.flag.png" }
```

Where the output is text the actual output is sent.

```
{ "usrTxt": "Samoa\n", "score": 0,  
  "ansTxt": "Afghanistan\nKazakhstan\nPakistan\nTurkmenistan\nUzbekistan\n" }
```

When an error is generated then details of the error message are returned:

```
{ "errorLine": [11],  
  "errorMsg": "Cia.java:11: ';' expected\r\n  }\r\n  ^\r\n1 error\r\n",  
  "error": "compile time",  
  "dir": "/home/andrew/sandbox/home/weak/scratch/740" }
```

Other features

The execution engines are protected by a queuing mechanism.

When there is high traffic we ensure that only one or two processes are permitted simultaneously. If we provide instant accurate feedback to users regarding the size of the queue and their position in the queue this reduces frustration, reduces the "perceived" wait time.

