GeoTools

Image Workbook

FOSS4G 2009 Geospatial for Java Tutorials

27 September 2009

Jody Garnett
Michael Bedward

Table of Contents

1 Welcome

Welcome to Geospatial for Java. This workbook is aimed at Java developers who are new to geospatial and would like to get started.

Please ensure you have your IDE set up with access to the GeoTools jars (either through maven or as a directory of Jar files). For those of you using Maven we will start off each section with the dependencies required.

This workbook is once again “code first” giving you a chance to try the concepts out in a Java program and then read on for more information if you have any questions.

This workbook covers the handling of GridCoverages (in the rest of computing these are known as “rasters” or “bitmaps”). The idea is that a coverage completely covers the surface of a map with no gaps forming a surface. A grid coverage is a special case of a coverage in which all the features end up as small rectangles on the surface of the earth.

This idea is so similar to our concept of pixels we end up using a lot of the same file formats to represent a grid coverage in our computing systems.

This workbook is part of the FOSS4G 2009 conference proceedings.

Jody Garnett

Jody Garnett is the lead architect for the uDig project; and on the steering committee for GeoTools; GeoServer and uDig. Taking the roll of geospatial consultant a bit too literally Jody has presented workshops and training courses in every continent (except Antarctica). Jody Garnett is an employee of LISAsoft.

Michael Bedward

Michael Bedward is a researcher with the NSW Department of Environment and Climate Change and an active contributor to the GeoTools users' list. He has a particularly wide grasp of all the possible mistakes one can make using GeoTools.

2 ImageLab

When working with image data you are going to have to treat each image format on a case by case basis; there are examples of how to set up each in the GeoTools User Guide wiki.

1. If you are using Maven, we will start by adding dependencies to our project the GeoTools plugins supporting the image formats that we'll be working with: geotiff and image+world. Please ensure that your pom.xml file includes the following:

```
<dependencies>
    <dependency>
        <groupId>org.geotools</groupId>
        <artifactId>gt-shapefile</artifactId>
        <version>${geotools.version}</version>
    </dependency>
    <dependency>
        <groupId>org.geotools</groupId>
        <artifactId>gt-epsg-hsql</artifactId>
        <version>${geotools.version}</version>
    </dependency>
    <dependency>
        <groupId>org.geotools</groupId>
        <artifactId>gt-geotiff</artifactId>
        <version>${geotools.version}</version>
    </dependency>
    <dependency>
        <groupId>org.geotools</groupId>
        <artifactId>gt-image</artifactId>
        <version>${geotools.version}</version>
    </dependency>
    <dependency>
        <groupId>org.geotools</groupId>
        <artifactId>gt-swing</artifactId>
        <version>${geotools.version}</version>
        <!-- For this module we explicitly exclude some of its own -->
        <!-- dependencies from being downloaded because they are   -->
        <!-- big and we don't need them                            -->
        <exclusions>
          <exclusion>
            <groupId>org.apache.xmlgraphics</groupId>
            <artifactId>batik-transcoder</artifactId>
          </exclusion>
        </exclusions>
    </dependency>
</dependencies>
```

2. Create the file ImageLab.java and copy and paste in the following imports and main method:

```
/*
 *    GeoTools - The Open Source Java GIS Tookit
 *    http://geotools.org
 *
 *    (C) 2006-2008, Open Source Geospatial Foundation (OSGeo)
 *
 *    This file is hereby placed into the Public Domain. This means anyone is
 *    free to do whatever they wish with this file. Use it well and enjoy!
 */
package org.geotools.demo;

import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;

import org.geotools.coverage.GridSampleDimension;
import org.geotools.coverage.grid.GridCoverage2D;
import org.geotools.coverage.grid.io.AbstractGridCoverage2DReader;
import org.geotools.coverage.grid.io.AbstractGridFormat;
import org.geotools.coverage.grid.io.GridFormatFinder;
import org.geotools.data.FeatureSource;
import org.geotools.data.FileDataStore;
import org.geotools.data.FileDataStoreFinder;
import org.geotools.data.Parameter;
import org.geotools.factory.CommonFactoryFinder;
import org.geotools.map.DefaultMapContext;
import org.geotools.map.MapContext;
import org.geotools.styling.ChannelSelection;
import org.geotools.styling.ContrastEnhancement;
import org.geotools.styling.RasterSymbolizer;
import org.geotools.styling.SLD;
import org.geotools.styling.SelectedChannelType;
import org.geotools.styling.Style;
import org.geotools.styling.StyleFactory;
import org.geotools.swing.JMapFrame;
import org.geotools.swing.data.JParameterListWizard;
import org.geotools.swing.wizard.JWizard;
import org.geotools.util.KVP;
import org.opengis.feature.simple.SimpleFeature;
import org.opengis.feature.simple.SimpleFeatureType;
import org.opengis.filter.FilterFactory2;
import org.opengis.style.ContrastMethod;

public class ImageLab {
    private StyleFactory sf = CommonFactoryFinder.getStyleFactory(null);
    private FilterFactory2 ff = CommonFactoryFinder.getFilterFactory2(null);
    private JMapFrame frame;
    private AbstractGridCoverage2DReader reader;
    public static void main(String[] args) throws Exception {
        ImageLab me = new ImageLab();
        me.getLayersAndDisplay();
    }
}
```

3. Next we will add a method to prompt for two files: an image file (either a geotiff or a jpg image+world file) and a shapefile that we will display over the image:

```
private void displayLayers(File rasterFile, File shpFile) throws Exception {
        // AbstractGridFormat format = GridFormatFinder.findFormat( rasterFile );
        reader = format.getReader(rasterFile);

        // Initially display the raster in greyscale using the
        // data from the first image band
        Style rasterStyle = createGreyscaleStyle(1);

        // Connect to the shapefile
        FileDataStore dataStore = FileDataStoreFinder.getDataStore(shpFile);
        FeatureSource<SimpleFeatureType, SimpleFeature> shapefileSource = dataStore
                .getFeatureSource();

        // Create a basic style with yellow lines and no fill
        Style shpStyle = SLD.createPolygonStyle(Color.YELLOW, null, 0.0f);

        // Set up a MapContext with the two layers
        final MapContext map = new DefaultMapContext();
        map.setTitle("ImageLab");
        map.addLayer(reader, rasterStyle);
        map.addLayer(shapefileSource, shpStyle);

        // Create a JMapFrame with a menu to choose the display style for the
        frame = new JMapFrame(map);
        frame.setSize(800, 600);
        frame.enableStatusBar(true);
        //frame.enableTool(JMapFrame.Tool.ZOOM, JMapFrame.Tool.PAN, JMapFrame.Tool.RESET);
        frame.enableToolBar(true);

        JMenuBar menuBar = new JMenuBar();
        frame.setJMenuBar(menuBar);
        JMenu menu = new JMenu("Raster");
        menuBar.add(menu);
        menu.add( new SafeAction("Grayscale display") {
            public void action(ActionEvent e) throws Throwable {
                Style style = createGreyscaleStyle();
                if (style != null) {
                    map.getLayer(0).setStyle(style);
                    frame.repaint();
                }
            }
        });
        menu.add( new SafeAction("RGB display") {
            public void action(ActionEvent e) throws Throwable {
                Style style = createRGBStyle();
                if (style != null) {
                    map.getLayer(0).setStyle(style);
                    frame.repaint();
                }
           }
        });
        // Finally display the map frame.
        // When it is closed the app will exit.
        frame.setVisible(true);
    }
```

Note that we are creating a Style for each of the map layers:

- An initial greyscale Style for the image, created with a method that we'll examine next
- A simple outline style for the shapefile using the **SLD** utility class

4. In general there are two ways to display the image
   - Displaying a single band as a greyscale view
   - Combining three bands to form a RGB view

   We are going to start with a method that prompts the user to choose a single band, and then creates a greyscale Scale for that band:

```
private Style createGreyscaleStyle() {
    GridCoverage2D cov = null;
    try {
        cov = reader.read(null);
    } catch (IOException giveUp) {
        throw new RuntimeException(giveUp);
    }
    int numBands = cov.getNumSampleDimensions();
    Integer[] bandNumbers = new Integer[numBands];
    for (int i = 0; i < numBands; i++) { bandNumbers[i] = i+1; }
    Object selection = JOptionPane.showInputDialog(
            frame,
            "Band to use for greyscale display",
            "Select an image band",
            JOptionPane.QUESTION_MESSAGE,
            null,
            bandNumbers,
            1);
    if (selection != null) {
        int band = ((Number)selection).intValue();
        return createGreyscaleStyle(band);
    }
    return null;
}
```

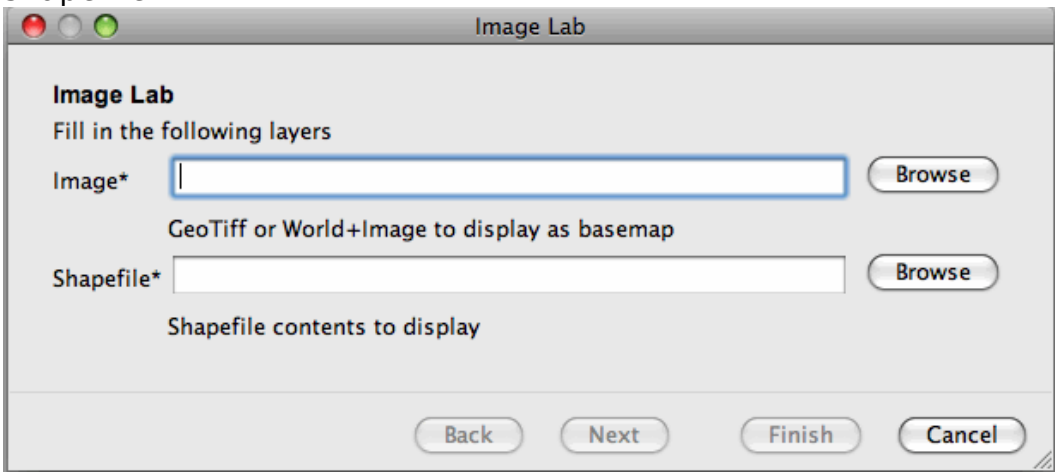
5. We can now create a Symbolizer to display the selected band.

```
private Style createGreyscaleStyle(int band) {
    ContrastEnhancement ce = sf.contrastEnhancement(ff.literal(1.0), ContrastMethod.NORMALIZE);
    SelectedChannelType sct = sf.createSelectedChannelType(String.valueOf(band), ce);
    RasterSymbolizer sym = sf.getDefaultRasterSymbolizer();
    ChannelSelection sel = sf.channelSelection(sct);
    sym.setChannelSelection(sel);
    return SLD.wrapSymbolizers(sym);
}
```

6. This completes the ImageLab code.
   Build the application and check that there are no errors.

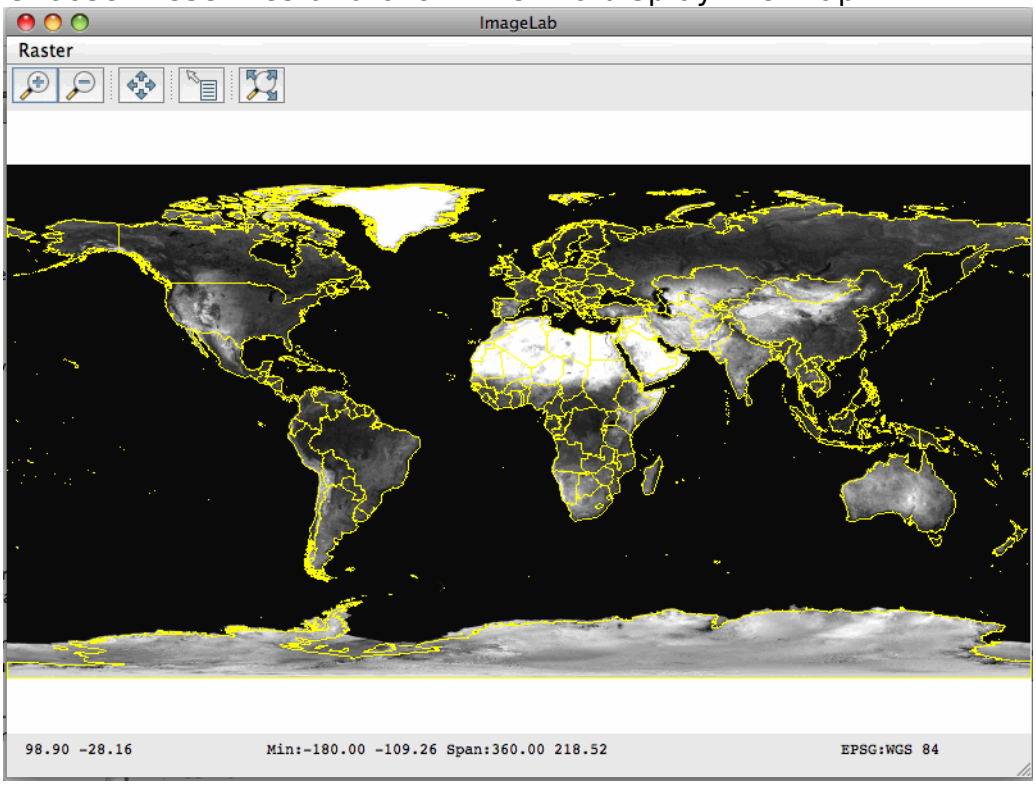
2.1 Running ImageLab

1. When you run the application you are prompted for the image and shapefile.



2. In the uDig sample dataset (http://udig.refractions.net/docs/data-v1_2.zip) there is a geotiff global image, **bluemarble.tif** and a shapefile of country borders, **countries.shp**

3. Choose these files and click finish to display the map.



4. Experiment with switching between different bands of the image.

5. You can also try re-running the application with an image+world format file in the sample dataset: **clouds.jpg** (another global image)

3 Going Color

You may wonder why the application is only showing gray scale right now? The answer is that to display color we need to use a slightly more complex Style that specifies which bands in the grid coverage map to the R, G and B colors.

While this will be very easy for a simple color image; it can be harder for things like satellite images where none of the bands quite line up with what human eyes see.

1. Start by adding a new menu item.

```
        menu.add( new SafeAction("RGB display") {
            public void action(ActionEvent e) throws Throwable {
                Style style = createRGBStyle();
                if (style != null) {
                    map.getLayer(0).setStyle(style);
                    frame.repaint();
                }
            }
        });
```

2. Next we will add a method that creates the RGB Style. The method below first examines the image to see if its bands (known as sample dimensions in GeoTools-speak) are labelled to indicate which to use.

```
private Style createRGBStyle() {
        GridCoverage2D cov = null;
        try {
            cov = reader.read(null);
        } catch (IOException giveUp) {
            throw new RuntimeException(giveUp);
        }
        // We need at least three bands to create an RGB style
        int numBands = cov.getNumSampleDimensions();
        if (numBands < 3) {
            return null;
        }
        // Get the names of the bands
        String[] sampleDimensionNames = new String[numBands];
        for (int i = 0; i < numBands; i++) {
            GridSampleDimension dim = cov.getSampleDimension(i);
            sampleDimensionNames[i] = dim.getDescription().toString();
        }
        final int RED = 0, GREEN = 1, BLUE = 2;
        int[] channelNum = { -1, -1, -1 };
        // We examine the band names looking for "red...", "green...", "blue...".
        // Note that the channel numbers we record are indexed from 1, not 0.
        for (int i = 0; i < numBands; i++) {
            String name = sampleDimensionNames[i].toLowerCase();
            if (name != null) {
                if (name.matches("red.*")) {
                    channelNum[RED] = i + 1;
                } else if (name.matches("green.*")) {
                    channelNum[GREEN] = i + 1;
                } else if (name.matches("blue.*")) {
                    channelNum[BLUE] = i + 1;
                }
            }
        }
```

3. We can now create the Symbolizer (actually a RasterSymbolizer) using our best guess (or if we give up we will just take the first three bands).

```
        // If we didn't find named bands "red...", "green...", "blue..."
        // we fall back to using the first three bands in order
        if (channelNum[RED] < 0 || channelNum[GREEN] < 0 || channelNum[BLUE] < 0) {
            channelNum[RED] = 1;
            channelNum[GREEN] = 2;
            channelNum[BLUE] = 3;
        }
        // Now we create a RasterSymbolizer using the selected channels
        SelectedChannelType[] sct = new SelectedChannelType[cov.getNumSampleDimensions()];
        ContrastEnhancement ce = sf.contrastEnhancement(ff.literal(1.0), ContrastMethod.NORMALIZE);
        for (int i = 0; i < 3; i++) {
            sct[i] = sf.createSelectedChannelType(String.valueOf(channelNum[i]), ce);
        }
        RasterSymbolizer sym = sf.getDefaultRasterSymbolizer();
        ChannelSelection sel = sf.channelSelection(sct[RED], sct[GREEN], sct[BLUE]);
        sym.setChannelSelection(sel);

        return SLD.wrapSymbolizers(sym);
    }
```

4 Rasters and GridCoverages

Support for raster data is provided by the concept of a GridCoverage. As programmers we are used to working with raster data in the form of bitmapped graphics such as JPEG, GIF and PNG files.

On the geospatial side of things there is the concept of a *Coverage.* A coverage is a collection of spatially located features. Informally, we equate a coverage with a map (in the geographic rather than the programming sense).

A *GridCoverage* is a special case of Coverage where the features are rectangles forming a grid that fills the area of the coverage. In our Java code we can use a bitmapped graphic as the backing data structure for a GridCoverage together with additional elements to record spatial bounds in a specific coordinate reference system.

There are many kinds of grid coverage file formats. Some of the most common are:

- **world plus image** – this is a normal image format like jpeg or png that has a side-car file describing where it is located as well as a prj sidecar file defining the map projection just like a shapefile uses.
- **geotiff** – this is a normal tiff image that has geospatial information stored in the image metadata fields.
- **Jpeg2000** – is the sequel to jpeg that uses wavelet compression to handle massive images. The file format also supports metadata fields that can be used to store geospatial information.
- There are also more exotic formats such as ECW and MRSID that can be supported if you have installed the imageio-ext project into your JRE.

Like a bitmapped graphics file, a single GridCoverage can have one or more bands or *sample dimensions* in GeoTools-speak.

5 Web Map Server

Another source of imagery is a Web Map Server (WMS). The Web Map Server specification is defined by the Open Geospatial Consortium – an industry body set up to encourage collaboration on this sort of thing.

At a basic level we can fetch information from a WMS using a GetMap operation.

```
http://localhost:8080/geoserver/wms?bbox=-130,24,-66,50&styles=population&Format=image/png&
request=GetMap&layers=topp:states&width=550&height=250&srs=EPSG:4326
```

The trick is knowing what parameters to fill in for “layer” and “style” when making one of these requests.

The WMS Service offers a GetCapabilities document that describes what layers are available and what other operations like GetMap are available to work on those layers.

GeoTools has a great implementation to help out here – it can parse that capabilities document for a list of layers, the supported image formats and so forth.

```
URL url =  url = new URL("http://www2.dmsolutions.ca/cgi-bin/mswms_gmap?
VERSION=1.1.0&REQUEST=GetCapabilities");

WebMapServer wms = new WebMapServer(url);
WMSCapabilities capabilities = wms.getCapabilities();

// gets all the layers in a flat list, in the order they appear in
// the capabilities document (so the rootLayer is at index 0)
List layers = capabilities.getLayerList();
```

WebMapServer class also knows how to set up a GetMap request for several different version of the WMS standard.

```
GetMapRequest request = wms.createGetMapRequest();
request.setFormat("image/png");
request.setDimensions("583", "420"); //sets the dimensions of the image to be returned from the server
request.setTransparent(true);
request.setSRS("EPSG:4326");
request.setBBox("-131.13151509433965,46.60532747661736,-117.61620566037737,56.34191403281659");

GetMapResponse response = (GetMapResponse) wms.issueRequest(request);
BufferedImage image = ImageIO.read(response.getInputStream());
```