

Using the uniformgen software – a tutorial

© Dirk Schönberger 2009

Table of Contents

Using the uniformgen software – a tutorial.....	1
Disclaimer.....	1
Introduction.....	1
Setup.....	1
Creating some images.....	2
Creating some basic images.....	4
Some dynamic images – color replacement.....	4

Disclaimer

This tutorial uses image content taken from the site <http://www.juniorgeneral.org> The copyright of the original authors is unchanged.

Introduction

Uniformgen is a software which can be used to automatically create images from tabular data. The created images consist from couple of distinct layers, which may be partially transparent. Each layer contains a part of the resulting image and can consist of various types of output, including

- static text
- dynamic text
- static images
- dynamic images

Static elements are referenced directly in the file system, while dynamic data is referenced by a column number in a database file (e.g. a CSV file).

The software allows for a couple of advanced features which make image creating easier. These include

- Search and replace of pixels per color by another solid color or a texture
- Image effects

Setup

Before we start, we have to create a basic setup. Uniformgen needs the following things in order to work

1. a control file which defines the actions to be executed

2. a database file which contains information about which are later used
3. some resources, mostly images, which are used to create the images
4. the actual software, „uniformgen.jar“. This is a Java application, so you need a current Java runtime system. If it is not already existant on your computer, you may download it at <http://java.sun.com/javase/downloads/index.jsp>

Each of the following steps assumes that you have created a directory hierarchy on your file system, and you have open a console (DOS command line) in which you are in the root directory of this hierarchy. It is also assumed that you are able to execute the Java software, which is done via typing

```
$(PATH_TO_YOUR_JAVA_RUNTIME) -jar $(name of the executable)
```

e.g

```
java.exe -jar uniformgen.jar
```

The tool uniformgen expects its control file with the name „setup.xml“ in the current directory of your console.

For this tutorial we assume the following file system hierarchy

Starting from our root directory there exist the following two directories:

- out
- templates

The folder „out“ will contain the generated images.

In the „templates“ folder we keep our resources, separated by the categories. For our examples we have the sub-folders „base“ and „colors“.

The archive „tutorial0.zip“ contains our basic setup. The subdirectories in the templates folder contain already some image resources, which we will discuss later. The setup.xml is currently only a dummy file which has no real use.

Creating some images

In this step we will create a base control file and some database and will use this setup to automatically create some images.

The image generation process is controlled by the control file „setup.xml“ in the root directory of your file hierarchy. If you open this file, or create it if it does not yet exist, it should contain the following lines:

```
<PieceSet>  
</PieceSet>
```

The control file consist of some header part which controls the position size etc of the created

images and a zero or more layer definitions. Each layer has a type attribute and more parameters according to its type. A layer may be rendered into the resulting image, or it may be invisible and can be used as reference in other layers.

We will start with the header part.

Please add the following lines after the `<PieceSet>` line:

```
<Data>
  <OutputFolder>./out/</OutputFolder>
  <ImageSizeX>120</ImageSizeX>
  <ImageSizeY>200</ImageSizeY>
  <Data>./data.csv</Data>
  <Naming>1</Naming>
  <NamingColumn>0</NamingColumn>

<Data>
```

`OutputFolder` is used to define where the created image are to be stored. We use the relative path „./out/“ here.

`ImageSizeX` and `ImageSizeY` are used to define the size of the created images.

`Data` is used to specify the location of our database file. We use the file „data.csv“ in the current folder.

`Naming` and `NamingColumn` are used to control the file names of the generated images. You may ignore the values for now. Please consult the user guide for further information.

In order to create images, we also need a database file. This should be a CSV file (comma separated values), and should be positioned according to the `Data` setting (see above)

For our tutorial we create a file called `data.csv`, which should contain the following lines:

```
1,1st Regt,red,white,gold,white,white
2,2nd Regt,red,white,silver,white,white
3,3rd Regt,red,white,gold,blue,white
4,4th Regt,red,white,silver,blue,white
```

In this step we are only interested in the number of lines (4), not yet in the content.

If we execute the `uniformgen` tool in the folder with these data, there should be 4 files (`1.png`, `2.png`, `3.png`, `4.png`) in the subfolder „out“.

The archive „tutorial1.zip“ contains the results of this step.

Creating some basic images

If you open the above mentioned files 1.png etc, you will see that these are fully transparent. In this step we will add some base to our create images. This is an static image, so it may have any content.

Open the file setup.xml and after the line

```
<NamingColumn>0</NamingColumn>
```

add the following lines

```
<Layer>
  <Name>Ground</Name>
  <Type>2</Type>
  <Image>./templates/base/base.png</Image>
  <WriteAtXA>0</WriteAtXA>
  <WriteAtYA>0</WriteAtYA>
</Layer>
```

This will create our first layer. The type 2 signifies a static image layer. The parameter Image defines the image file to use, WriteAtXA and WriteAtYA specify the coordinates where the image will be drawn in the resulting image.

The Parameter Name specifies the name of the layer. For now it is not needed. Later we will reference other layers by their name.

If you run the uniformgen tool with this control file, you should see 4 files in the out folder. If you open the file with an image editor, you should see a white background with a black border.

The archive „tutorial2.zip“ contains the results of this step.

Some dynamic images

In this step we will create images based on a database file. This database file contains lines, each line contains columns which are separated by commas (','),. You may create such files with a database or spreadsheet application. The data.csv file mentioned earlier has the following columns:

Column 0 – Id: used for naming purposes

Column 1 – Label: used later for labeling the created images

Column 2 – Coat: the name of a color

Column 3 – Trouser: the name of a color

Column 4 – Buttons: the name of a color

Column 5 – Markings: the name of a color

Column 6 - Vest: the name of a color

Id	Name	Coat	Trouser	Buttons	Markings	Vest
1	1st Regt	red	white	gold	white	white

2 2nd Regt	red	white	silver	white	white
3 3rd Regt	red	white	gold	blue	white
4 4th Regt	red	white	silver	blue	white

In this step we only want to use the values in the column 4 (Button) to render a dynamic image. This is temporary only, the steps after this start from start 2 of the tutorial.

In order to create an dynamic image layer, we add the following lines to our setup xml (after the line

```
</Layer>
```

```
<Layer>
  <Name>TestImage</Name>
  <Column>4</Column>
  <Type>5</Type>
  <Folder>./templates/colors</Folder>
  <Extension>png</Extension>
  <WriteAtXA>20</WriteAtXA>
  <WriteAtYA>20</WriteAtYA>
</Layer>
```

The parameters Name, WriteXA and WriteYA have the same meaning than in the last step- Type 5 signifies a dictionary based dynamic image. The parameter column defines the column in the database which is to be used. The file to be used is calculated from the content of the given column and the parameter settings of Folder and Extension.

If you execute this project, there should be created 4 images, all of the have a white background bordered black. The images 1.png and 3.png contain a small yellow/golden rectangle, the images 2.png and 4.png should contain a small grey/silver rectangle.

The archive „tutorial3.zip“ contains the results of this step.

Dynamic texts – labels

In this step we will create a layer which contains the text taken from a column from the database file. This e.g. allows to label images. We start with the results of step tutorial2 and add the following lines after the line

```
</Layer>
```

```
<Layer>
  <Name>Label</Name>
  <Type>3</Type>
  <Column>1</Column>
  <WriteAtXA>4</WriteAtXA>
  <WriteAtYA>1</WriteAtYA>
  <FontSize>11</FontSize>
  <FontType>Times</FontType>
  <Color>Black</Color>
</Layer>
```

Type 3 specifies a dictionary based dynamic text layer. Name, WriteAtXA, WriteAtXB are already

known. The Parameter Column specifies the column number where the value of the text to draw should be taken from. FontSize, FontType and Color are used to select the typeface and text color to use.

If you execute the uniformgen tool using this settings, there should be created 4 files, white background bordered black. The image 1.png should contain a text „1st Regt“, 2.png should contain a text „2nd Regt“ etc.

The archive „tutorial4.zip“ contains the results of this step.

Color replacements – Texturing

In this step a rather powerful feature of the uniformgen tool is demonstrated. You may replace each pixel according to their colour by another pixel value. There are several possible replacements:

- a pixel may be replaced by a transparent pixel
- a pixel may be replaced with a solid colour
- a pixel may be replaced as part of a texture image, i.e. the actual pixel value is the color at the same position in the image used as a texture
- if not specified otherwise, the pixel value is not replaced

In order to do this, we have to create a couple of hidden layers. Hidden layers are not rendered to the resulting image, but they are kept internal in order to be referenced from other layers.

So at first we defined the needed hidden layers:

```
<Layer>
  <Name>Coat</Name>
  <Column>2</Column>
  <Type>7</Type>
  <Folder>./templates/colors</Folder>
  <Extension>png</Extension>
</Layer>

<Layer>
  <Name>Trouser</Name>
  <Column>3</Column>
  <Type>7</Type>
  <Folder>./templates/colors</Folder>
  <Extension>png</Extension>
</Layer>

<Layer>
  <Name>Buttons</Name>
  <Column>4</Column>
  <Type>7</Type>
  <Folder>./templates/colors</Folder>
  <Extension>png</Extension>
</Layer>

<Layer>
```

```

<Name>Markings</Name>
<Column>5</Column>
<Type>7</Type>
<Folder>./templates/colors</Folder>
<Extension>png</Extension>
</Layer>

<Layer>
<Name>Vest</Name>
<Column>6</Column>
<Type>7</Type>
<Folder>./templates/colors</Folder>
<Extension>png</Extension>
</Layer>

```

Type 7 specifies a dictionary based dynamic hidden image layer. The rest of the parameters should already be known.

In order to do color replacements, we create a image layer (this time we need a static image) and define a color (mapping) model. A color model maps a color value (name or RGB value) to a fill strategy. We want to make all white pixels transparent, all red pixels should be filled with the texture from „Coat“, all blue pixel should be textured using the „Trouser“ layer, all magenta pixels should be textured using the „Button“ layer, all yellow pixels should be rendered using the „Markings“ layer and all cyan pixels should be textured using the „Vest“ layer. This gives the following layer definition

```

<Layer>
<Name>Soldier</Name>
<Column>7</Column>
<Type>8</Type>
<Folder>./templates/soldiers</Folder>
<Extension>png</Extension>
<WriteAtXA>0</WriteAtXA>

<WriteAtYA>0</WriteAtYA>
<ColourModel>
<Entry key="white" mode="1" />
<Entry key="red" mode="2" layer="Coat" />
<Entry key="blue" mode="2" layer="Trouser" />
<Entry key="magenta" mode="2" layer="Button" />
<Entry key="yellow" mode="2" layer="Markings" />
<Entry key="cyan" mode="2" layer="Vest" />
</ColourModel>
</Layer>

```

Note: In order to make this work, you need an adapted database file. Currently color models are only supported for dynamic layers.

The archive „tutorial5.zip“ contains the results of this step.