Developing with Breaking The Multi-Colored Box – 0.5a

Sumit Khanna – http://penguindreams.org/page/see/Bmcb -sumit (a-t) penguindreams.org

The following is an incomplete developers guide for the 0.5 alpha release of *Breaking the Multi Colored Box* (BMCB). It is part of a much larger document that I hope to eventually publish in its entirety.

The engine is function and the developers guide, combined with the installation documentation, is enough for any experienced Java programmer to get started developing with the BMCB framework.

In the upcoming weeks, I hope to enlist the aid of other open source developers in expanding the existing engine to use new analysis techniques. This documentation and the framework are licensed as GNU GPL (currently GNU GPLv3).

Thank you for your interested. Feel free to e-mail me at the above address if you have any questions.

Appendix B – Developing with the Engine

The Bmcb Engine is comprised of several components including the Generators, Segmentators, Image Filters, Analyzers and Utilities. All these components are called through a Workflow. New workflows must be added to the entry point of the program. This guide is intended for developers who want to modify the engine for their own analysis techniques and build new experiments into the engine.

This document follows a bottom up approach focusing on the individual components and building up to incorporating all these components into the full engine. Many useful examples are included with the engine itself and should be read alongside this document to gain a full understanding of how to build new experiments into the existing framework.

Utility Classes

There are several independent utility classes contained within the framework that are used throughout the application. Many of these classes contain static standalone methods for basic tasks such as image conversion, logging, configuration, result graphing and various other common tasks [Figure 27 - Diagram of Utility Classes].

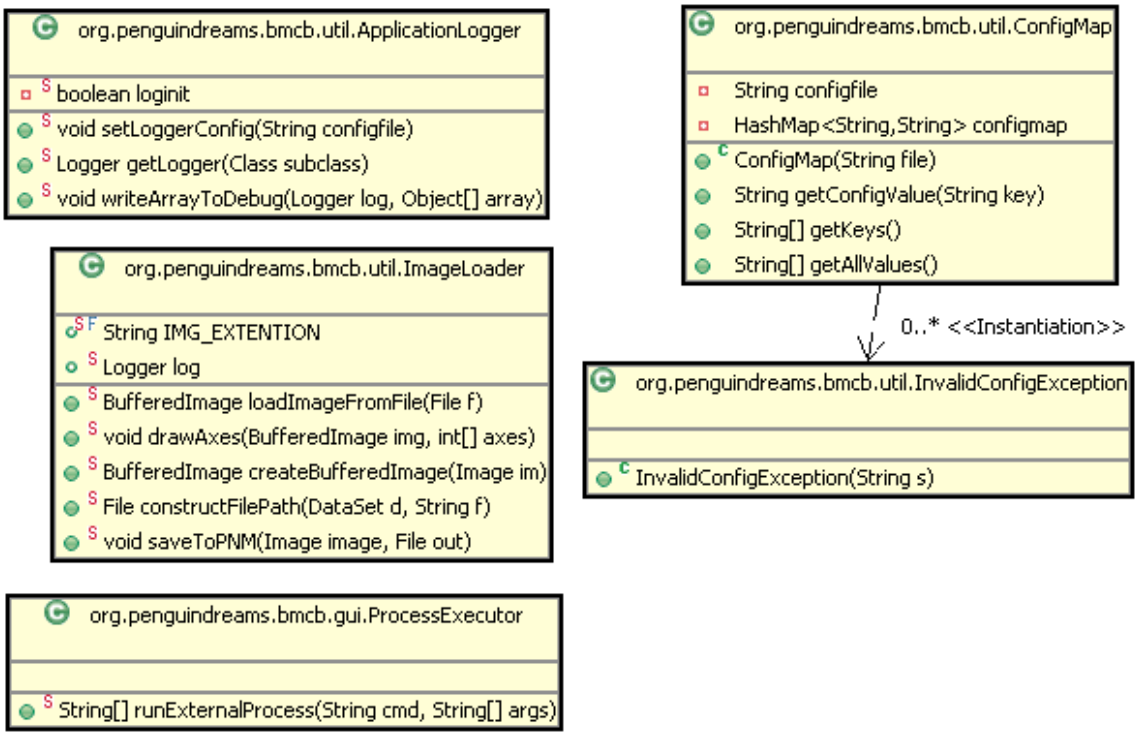


Figure 27 - Diagram of Utility Classes

Some of the more important utilities are as follows:

- Application Logger: This class is called from many of the abstract classes in the framework to initialize a protected log variable. Developers shouldn't need to call this directly unless they create a new class from scratch as the existing log variable is accessible in nearly every abstract class. Examples for creating new instances are located in the abstract classes.

- Image Loader: This class contains several functions to assist with image manipulation including image loading, converting between Images and Buffered Images, determining the path of an image from a dataset, converting images to

PNM files used by OCR programs, and drawing lines on segment boundaries on images to be used with the Segment Viewer debugging tool.

- Process Executor: A simple class to handle the basic task of running an external process.

- Config Map: Used to read in settings from the bmcb.property file

Generators

The engine comes with a command line generation class which passes two arguments to a command line application, the first being the CAPTCHA to be produced and the second being the location to write the file to. This process can be seen in the built-in generation workflow where random challenge/responses are generated, stored in a database and then generated for each CAPTCHA type.

Typically, a CAPTCHA application is not setup to take in challenge, but rather it generates it randomly. Therefore the CAPTCHA script needs to be modified. For the purposes of using this engine, it is best to use open source CAPTCHAs which can easily be modified. The following changes may be necessary before using a CAPTCHA script with this engine:

- Modify the program to take in the CAPTCHA phrase as the first argument

- Write the output to a file given by the second argument

- Adjust the front path to be independent from the scripts location

The following examples detail how these three modifications can be performed in a typical PHP based CAPTCHA script. Each CAPTCHA script a developer wants to incorporate will require different modifications to be compatible with the built in Command Line Generator, or it may require a custom generator specifically for it.

Modifying the CAPTCHA application to take the input from a command line, rather than generating it randomly, can be done several ways. The developer can modify the function that generates the random CAPTCHA, or the point at which the key is saved to the session can be modified [Figure 28].

```
#added by Sumit
#$_SESSION[CAPTCHA_SESSION_KEY] =  $text = substr($t, rand(0, (strlen($t)-6)), rand(3,6));
$text = $argv[1];
```

Figure 28 - Modification to Gotcha to take Challenge Input

Most CAPTCHA scripts have a function used to generate a random phrase or set of letters. This section is what must be modified in order to use the script with the engine. The type of modification will vary depending on the programming language used and may require the creation of a customized generation class.

Typically, most CAPTCHA scripts are designed to output directly to a web

browser. This behavior must also be modified to write the file, with the challenge solution as the filename, to a directory [Figure 29].

```
/*
switch($output)
{
    // add other cases as desired
    case "jpg":
        header("Content-Type: image/jpeg");
        ImageJPEG($pic);
        break;
    case "gif":
        header("Content-Type: image/gif");
        ImageGIF($pic);
        break;
    case "png":
    default:
        header("Content-Type: image/png");
        ImagePNG($pic);
        break;
}*/
//Modified by Sumit to write captcha file out
global $argv;
imagepng($pic,$argv[2]);
```

Figure 29 - Modification of Freecap to Output Challenge to a File

As with the previous modification, this will vary heavily depending on the programming language of the script and the script itself.

Another modification that may or may not be necessary involves modifying the path for included and dependent files. For many of the scripts included with the engine, this involved modifying the statement that declared which font to use to be current directory independent [Figure 30].

```
//ADDED by Sumit -- Adjusting Font Paths
$FONT_DIR = dirname(__FILE__);
$fonts = Array();
$fonts += glob($FONT_DIR."/*.ttf");
foreach($fonts as $f) {
  $t->addFont($f);
}
//$t->addFont('SFTransRoboticsExtended.ttf');
//$t->addFont('arialbd.ttf');
```

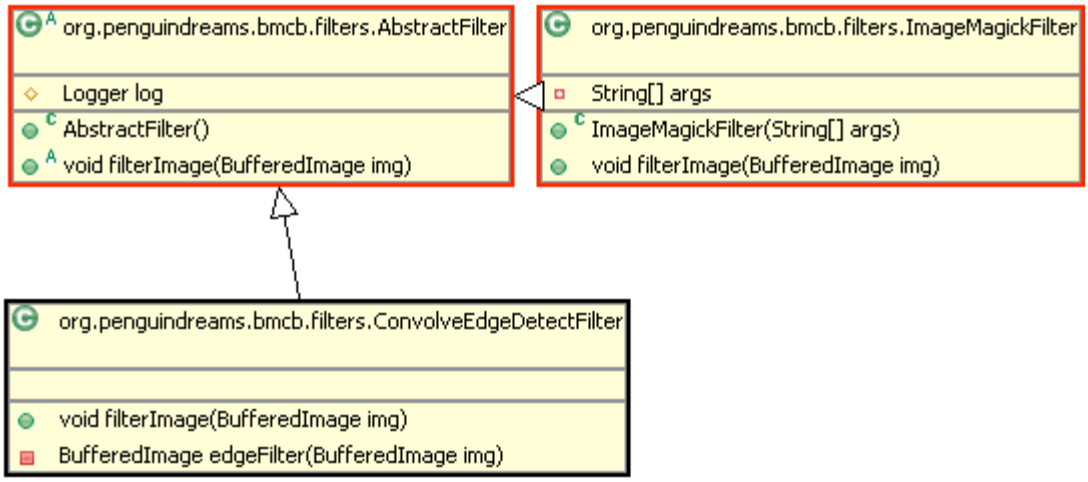
Figure 30 - Modification of Gotcha for Font Path

The above modification may or may not be necessary depending on the way the script loads its fonts and dependencies.

The above modifications are only some of the chances that may need to be made to a CAPTCHA application in order to get it to work with the engine. Developers may run into other challenges, however most CAPTCHA should be adaptable, either by extending the Abstract Command Line Generator or by creating a custom generator class, so long as the CAPTCHA application provides some means for manually inputing the challenge response.

Image Filters

Image Filters extend the Abstract Filter class. They must modify a Buffered Image that is passed to the filter by reference. If the calling class requires an unaltered version of the Buffered Image, it must clone a copy before passing it to the filter.



In addition, if a new Buffered Image is created during the filtering process, it can be copied into the passed in argument. An example is the function for edge filtering which returns a new Buffered Image object[Figure 31].

```
public void filterImage(BufferedImage img) {

    BufferedImage t = edgeFilter(img);
    img.getGraphics().drawImage(t,0,0,null);

}
```

Figure 31 - Copying one Buffered Image to Anohter

Segmentators

Segmentators are based on the Abstract Segmentator class. An instance of all the segmentator objects is created specifically for an image. Derived classes must implement the abstract getSegmentAxes function which returns the x coordinate where the image is split into separate vertical segments. Various other function in the base class can be

called to split the image into individual arrays of Buffered Image to be used within the workflows[Figure 32].

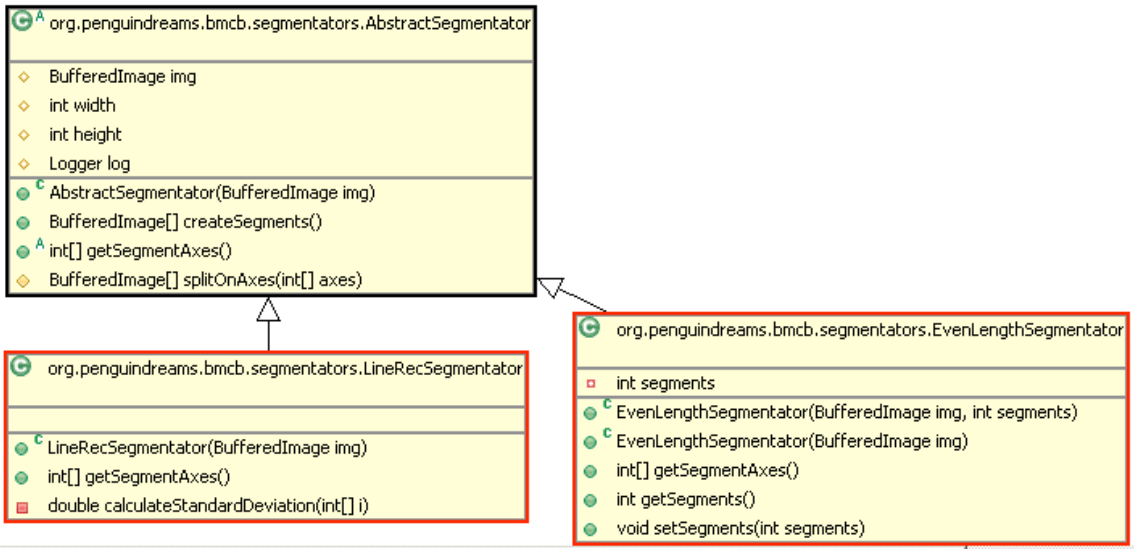


Figure 32 - Class Diagram for Segmentator Classes

Analyzers

Analyzers are what actually try to solve the CAPTCHA challenge after appropriate filters and segmentation has been applied. The Abstract Analyzer has been designed to contain several functions in the case of learning and non-learning algorithms and well as different functions for analyzing segments as opposed to full images. Analyzers which don't support a given abstract function can choose to throw an Analysis Not Supported Exception. An abstract Command Line Analyzer has been included to assist in the process of calling an external application for analysis [Figure 33].

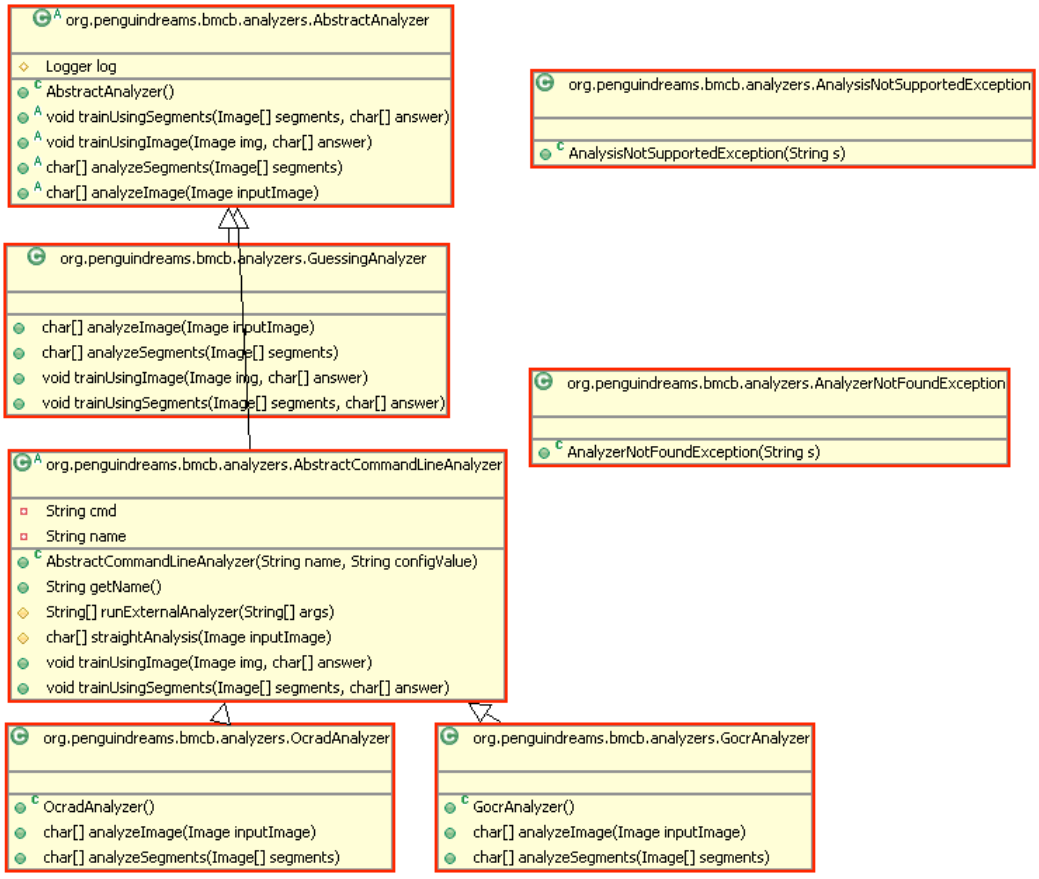


Figure 33 - Class Diagram for Analyzers

Workflows

The piece that ties all the individual components together is the workflow. The workflow calls all the individual pieces listed so far and can be used for generation of set data, analysis, or any various other tasks. The workflows included with the engine are used for data generation, analysis and testing. Developers will want to either modify existing workflows or create new workflows for whatever experiments they may wish to perform [Figure 34].

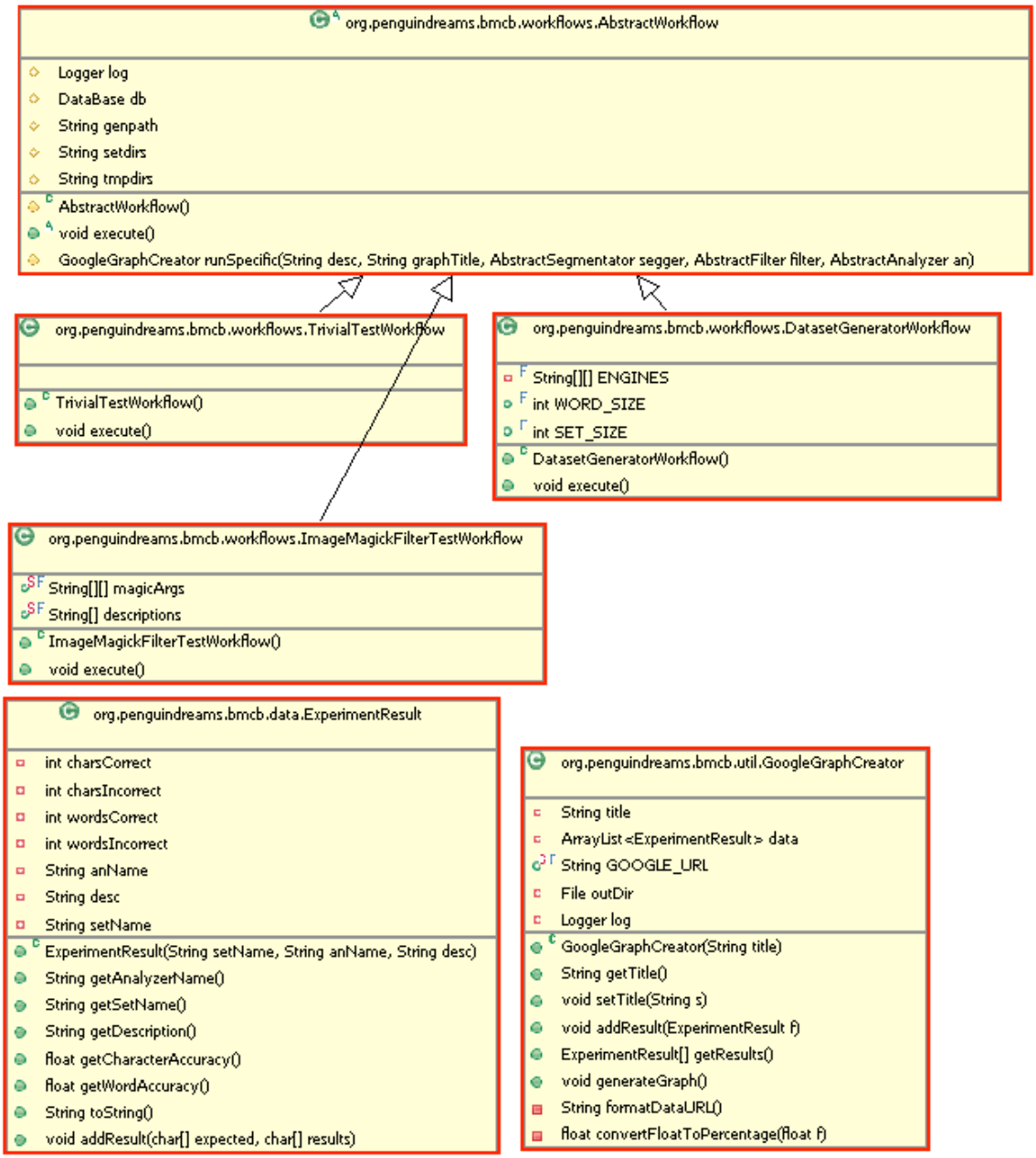


Figure 34 - Class Diagram for Workflow Classes

The Abstract Workflow class contains a protected runSpecifc() function which takes in all the individual components including a Segmentator, Filter and Analyzer, along with a description for results and logging purposes, and runs them against all the

currently available datasets.

Entry Point

The BMCBMain class provides the primary entry point for the application. After creating additional workflows, an appropiate command line argument or set of command line arguments will be need to be added to the main function to kick off the workflow.

Visual Debugging Tools

There is also a Segment Viewer packaged with the engine. It is a graphical tool to view data sets as well as the results from segmentators, image filters and analyzers[Figure 35].

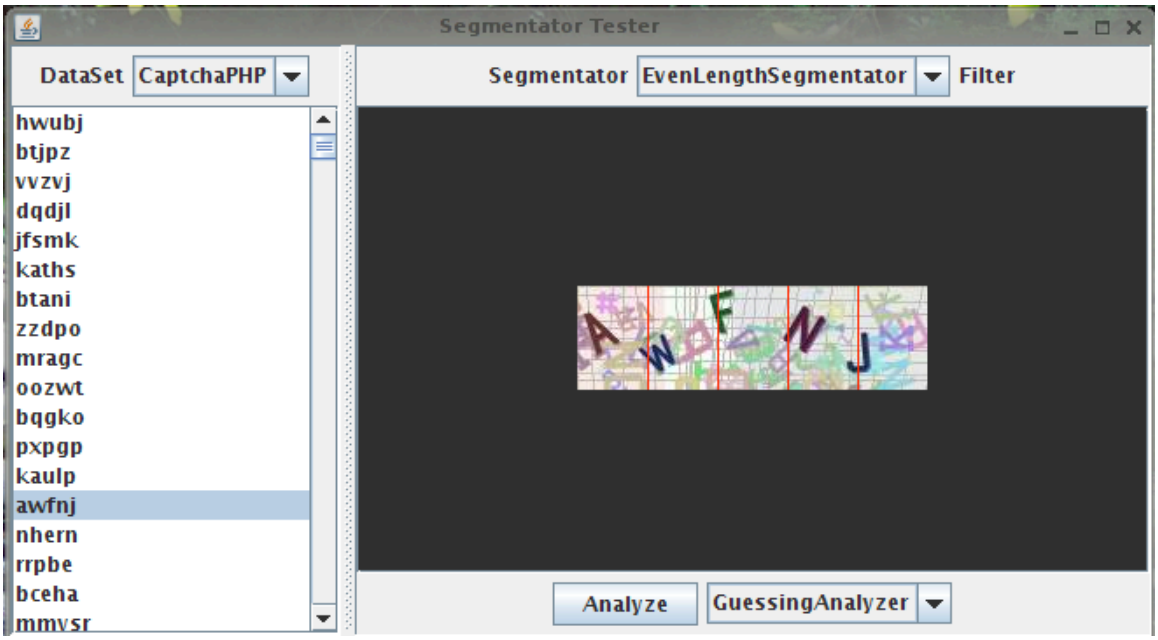


Figure 35 - Visual Debugger

Newly created Segmentators, Analyzers and Image Filters can easily be added to the Segment Viewer.