# Chalkbox

***Release 3.0.1***

**Brae Webb, Emily Bennett, Ella de Lore, Max Miller, Evan Hughes,**

**Apr 13, 2021**

# CONTENTS:

CHAPTER

ONE

# GUIDES

Guides for course coordinators using Chalkbox.

## 1.1 Creating an Engine

Engines in ChalkBox are responsible for taking a single student's submission, running tests on it, and returning a JSON file containing the test results in Gradescope format.

Currently, engines exist for both Java and Python, though engines do not necessarily need to be based on a single language.

This guide will explain how to create a ChalkBox engine, utilising the existing support framework in ChalkBox to make engine development easier.

### 1.1.1 Main Engine Class

Each ChalkBox engine has a main class that extends the abstract class `Engine`. These classes reside in the `chalkbox.engines` package.

A minimal example of an engine, DemoEngine, is as follows.

```
package chalkbox.engines;

public class DemoEngine extends Engine {
    @Override
    public void run() {
        System.out.println("Running DemoEngine");

        Collection submission = super.collect();

        // Operate on the submission here

        super.output(submission);
    }
}
```

The `Engine` abstract class provides a method to collect the student's submission, `collect()`. Calling this method will return a `Collection` which represents a single submission, including all the files submitted and a set of metadata relevant to the submission.

A collection's metadata is stored internally as a JSON object. Initially, the metadata includes the following keys:

- `root` : `String` path to the directory containing the submission

- `json` : `String` path to the JSON file where results will ultimately be written

The `output()` method will write the contents of the metadata object, including any test results added by the engine, to the output JSON file which will be read by Gradescope.

A `Collection` also provides a `Bundle` to represent the submission directory, and a `Bundle` to serve as a temporary directory during the engine's operation. A `Bundle` is essentially a wrapper for a directory, providing helpful methods to retrieve files in that directory.

The `run()` method should contain the core functionality of the engine, and is called once by ChalkBox after loading the engine configuration.

### 1.1.2 Configuration and Loading

Configuration options for engines are specified in a YAML file. The path to this file is passed as a command line argument when running the ChalkBox JAR.

The format for an engine's configuration file is as follows.

```
engine: chalkbox.engines.DemoEngine # fully-qualified class name of engine
---
courseCode: ABCD1234 # course code identifier
assignment: assignment_1 # assessment identifier
submission: /path/to/submission/dir/ # directory containing submission
outputFile: /path/to/results.json # path to output JSON file for Gradescope

# engine-specific options...
```

When ChalkBox is run with a given configuration file, the `EngineLoader` class reads the first document in the file (lines before the `---`) to determine which engine to invoke on the submission.

The engine loader then instantiates the specified engine class.

#### 1.1.2.1 Common Configuration Options

After loading an engine, the values of the common configuration options (courseCode, assignment, submission and outputFile) can be accessed by accessing the appropriate member variables of the `Engine` class.

#### 1.1.2.2 Engine-specific Configuration Options

To add engine-specific configuration options, simply create member variables in the engine's main class with the same name as the desired configuration option keys in the YAML file. A basic getter and setter method will need to be implemented for each configuration option in the class, as per the JavaBeans approach. This allows the YAML parser to set these fields when parsing the configuration file.

#### 1.1.2.3 Configuration Validation

To allow configuration validation, an engine's main class should implement the `Configuration` interface and provide a `validateConfig()` method. This method should first call `Engine`'s implementation, which ensures all the common configuration options are set, as follows.

```
package chalkbox.engines;

public class DemoEngine extends Engine implements Configuration {

    @Override
    public void validateConfig() throws ConfigFormatException {
        super.validateConfig();

        /*
         * Check if engine-specific configuration options are invalid and
         * throw ConfigFormatExceptions as necessary
         */
    }

    // ...
}
```

An engine's configuration is validated before running the engine. If the validation fails, i.e. a ConfigFormatException is thrown, then ChalkBox will print the stack trace of the exception and terminate immediately.

CHAPTER

TWO

# INDICES AND TABLES

- genindex
- modindex
- search