Verily Framework Documentation

Release 1.1.2

John L. Singleton

September 01, 2015

Contents

Contents:

CHAPTER 1

So What is Verily and How Can It Help Me?

Web applications are an increasingly important type of application. We do our banking online, manage portfolios, and in the US, we now even manage out health care online.

However, web application development is also a very trend driven domain. Part of the way in which these trends manifest themselves is the tools and technologies used to construct them. Web application frameworks, for example. While much work has been done that focuses on issues of performance and productivity (eg: DRY, scaffolding, convention over configuration) very little has been in the interest of making our web applications more *reliable*. This is what Verily is all about.

Verily combines application construction recipes with static analysis to help you build more reliable web applications. If this is the sort of thing that you are building, then Verily is for you.

CHAPTER 2

Installation

The Verily installer comes with everything you need to start writing applications in Verily right away. To start, download the latest installer from the releases page. Verily requires that you have a Java version 1.7+ and a recent version of Maven 3.

On Windows platforms, you can install Verily simply by running the downloaded JAR file. On other platforms (Linux and Mac) you will have to start the Verily installer via the command line as follows:

```
~ » sudo java -jar verily-<release>.jar
```

Where `release` is the release version of Verily that you downloaded, above.

Once Verily is installed, you can interact with it in a number of ways. The first (and perhaps most simple) is to interact with Verily on the command line. After installing Verily, the `verily` executable will be available on your system's `PATH`. The command options of Verily are summed up in the listing below:

```
~ » verily -help
usage: verily
 -contracts           enable checking of contracts
 -d                   run this application in the background
 -fast                do not recalculate dependencies before running
 -help                display this help
 -init <dir>          create a new Verily application in the specified
                      directory
 -jml <path-to-jml>   the path to the OpenJML installation directory.
 -n <threads>         the number of threads to create for handling
                      requests.
 -new <newclass>      create a new Verily Method+Router pair
 -nocompile           do not do internal recompile (used for development
                      only)
 -nostatic            disables extended static checking
 -port <portnumber>   port number to bind to (default 8000)
 -run                 run the application
 -test                run the unit tests for this application
 -w                   try to dynamically reload classes and templates (not
                      for production use)
 -z3 <path-to-z3>     the path to the Z3 installation directory.
```

While an IDE is not strictly necessary to work with Verily, if you are an IntelliJ user, you can use our simple VerilyIdea Plugin for IntelliJ. You can also get the plugin from the [main page](/).

CHAPTER 3

Headless Installation

The Verily installer by default requires a graphical environment. If you wish to install Verily without a graphical environment (perhaps on a server) you can use the procedure, below.

Put the following in a file called `install-verily.xml`

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<AutomatedInstallation langpack="eng">
  <com.izforge.izpack.panels.HelloPanel id="UNKNOWN (com.izforge.izpack.panels.HelloPanel)"/>
  <com.izforge.izpack.panels.LicencePanel id="UNKNOWN (com.izforge.izpack.panels.LicencePanel)"/>
  <com.izforge.izpack.panels.TargetPanel id="UNKNOWN (com.izforge.izpack.panels.TargetPanel)">
    <installpath>/usr/local/Verily</installpath>
  </com.izforge.izpack.panels.TargetPanel>
  <com.izforge.izpack.panels.InstallPanel id="UNKNOWN (com.izforge.izpack.panels.InstallPanel)"/>
  <com.izforge.izpack.panels.FinishPanel id="UNKNOWN (com.izforge.izpack.panels.FinishPanel)"/>
</AutomatedInstallation>
```

Download a release from the releases page and execute the following command as root:

```
» java -jar verily-installer-<version>.jar install-verily.xml
```

This will start non-interactive installation of Verily.

CHAPTER 4

Using Verily in Vagrant

If you are using Verily within Vagrant the setup is quite straightforward, but will require the use of the headless technique specified, above. To make this easier, users wishing to use Verily within Vagrant can use the following Vagrantfile. To use it, put the listing below into a file called `Vagrantfile` and execute the `vagrant up` command.

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure(2) do |config|
  config.vm.box = "hashicorp/precise32"
  config.vm.network :forwarded_port, host: 4000, guest: 4000
  config.vm.provision "shell", inline: <<-SHELL
     sudo apt-get update
     sudo apt-get -y install openjdk-7-jdk
     sudo apt-get -y install maven

     cat >/tmp/install-verily.xml <<EOL
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<AutomatedInstallation langpack="eng">
  <com.izforge.izpack.panels.HelloPanel id="UNKNOWN (com.izforge.izpack.panels.HelloPanel)"/>
  <com.izforge.izpack.panels.LicencePanel id="UNKNOWN (com.izforge.izpack.panels.LicencePanel)"/>
  <com.izforge.izpack.panels.TargetPanel id="UNKNOWN (com.izforge.izpack.panels.TargetPanel)">
    <installpath>/usr/local/Verily</installpath>
  </com.izforge.izpack.panels.TargetPanel>
  <com.izforge.izpack.panels.InstallPanel id="UNKNOWN (com.izforge.izpack.panels.InstallPanel)"/>
  <com.izforge.izpack.panels.FinishPanel id="UNKNOWN (com.izforge.izpack.panels.FinishPanel)"/>
</AutomatedInstallation>
EOL

     sudo update-alternatives --set java /usr/lib/jvm/java-7-openjdk-i386/jre/bin/java
     sudo wget https://github.com/jsinglet/Verily/releases/download/v0.1.2/verily-installer-0.1.2.jar
     sudo java -jar verily-installer-0.1.2.jar /tmp/install-verily.xml

  SHELL
end
```

CHAPTER 5

Hello World in Verily

In this section we are going to construct the most minimal version of a Verily application possible: the so-called "Hello World" application. To begin, make sure you have already installed Verily and run the following command on the command prompt from the directory in which you'd like to create your project:

```
~/Projects » verily -init HelloWorld
[INFO] Creating directory hierarchy...
[INFO] Done.
[INFO] Initializing Maven POM...
[INFO] Done. Execute "verily -run" from inside your new project directory to run this project.
```

After this command completes, you will have a new directory called `HelloWorld` in your current working directory.

Next, change to the newly-created directory and create a new Verily Method with the `-new` command:

```
~/Projects » cd HelloWorld
~/Projects/HelloWorld » verily -new Hello
[INFO] Creating a new Method/Router pair...
[INFO] Method/Router Pair Created. You can find the files created in the following locations:
[INFO] M: src/main/java/methods/Hello.java
[INFO] R: src/main/java/routers/Hello.java
[INFO] T: src/test/java/HelloTest.java
```

Note that in addition to a Verily Method, a corresponding router and unit test is also created for you. We'll get to that in a moment.

5.1 Writing Your Method

After creating your new method/router pair, you should see the following in the `src/main/java/methods/Hello.java` file:

```
package methods;

import verily.lang.*;

public class Hello {

     public static final void myFunction(ReadableValue<String> message){
          // TODO - Write your application
     }
}
```

This class corresponds to a Verily method class. There are several ways to make our example say "Hello World," and as you learn more about Verily you will find other methods, but for the moment we will do this by transforming the class in the following way:

```
package methods;

import verily.lang.*;

public class Hello {

     public static final String sayHello(){
               return "Hello World";
     }
}
```

The thing to note here is the return type of the method `sayHello`. You'll notice that it's a return type of type `String`. This value will then be passed as a formal parameter to your router.

5.2 Writing Your Router

To write the corresponding router you will want to replace the generated router in your `src/main/java/routers/Hello.java` with the code in the following listing:

```
package routers;

import verily.lang.*;

public class Hello {


    public static final Content sayHello(String result) {
              return new TextContent(result);
    }


}
```

In the router, above, we have created the sayHello function. After the method class (`methods.Hello.sayHello`) executes, control will be passed to the `routers.Hello.sayHello` function. Note that the actual parameter value of the router method will be the return value of the `methods.Hello.sayHello`.

The control flow of a Verily application looks like the application flow given in the following diagram.

5.3 Running Your Application

Once you have at least one method/router pair set up, you are ready to run your web application. To do this, use the `-run` option of Verily. The output below has been somewhat elided in order to highlight some of the important startup messages Verily will create:

```
~/Projects/HelloWorld » verily -run
[INFO] Scanning for projects...
[INFO] Bootstrapping Verily on port 8000...
[INFO] Constructed new Verily container @ Sun Jun 08 11:44:24 EDT 2014
[INFO] Created new thread pool with [10] threads.
[INFO] Starting Verily container...
```

```
[INFO] The Following MRR Endpoints Are Available in Your Application:
[INFO] +----------------------+---------+-----------------+
[INFO] | ENDPOINT             | METHOD SPEC | VERBS           |
[INFO] +----------------------+---------+-----------------+
[INFO] | /Hello/sayHello      | ()      | [POST, GET]     |
[INFO] +----------------------+---------+-----------------+
[INFO] [verily] Reloading project...
[INFO] Starting services...
[INFO] --------------------------------------------------------------------------
[INFO] Verily STARTUP COMPLETE
[INFO] --------------------------------------------------------------------------
[INFO] Bootstrapping complete in 4.134 seconds. Verily ready to serve requests at http://localhost:80
```

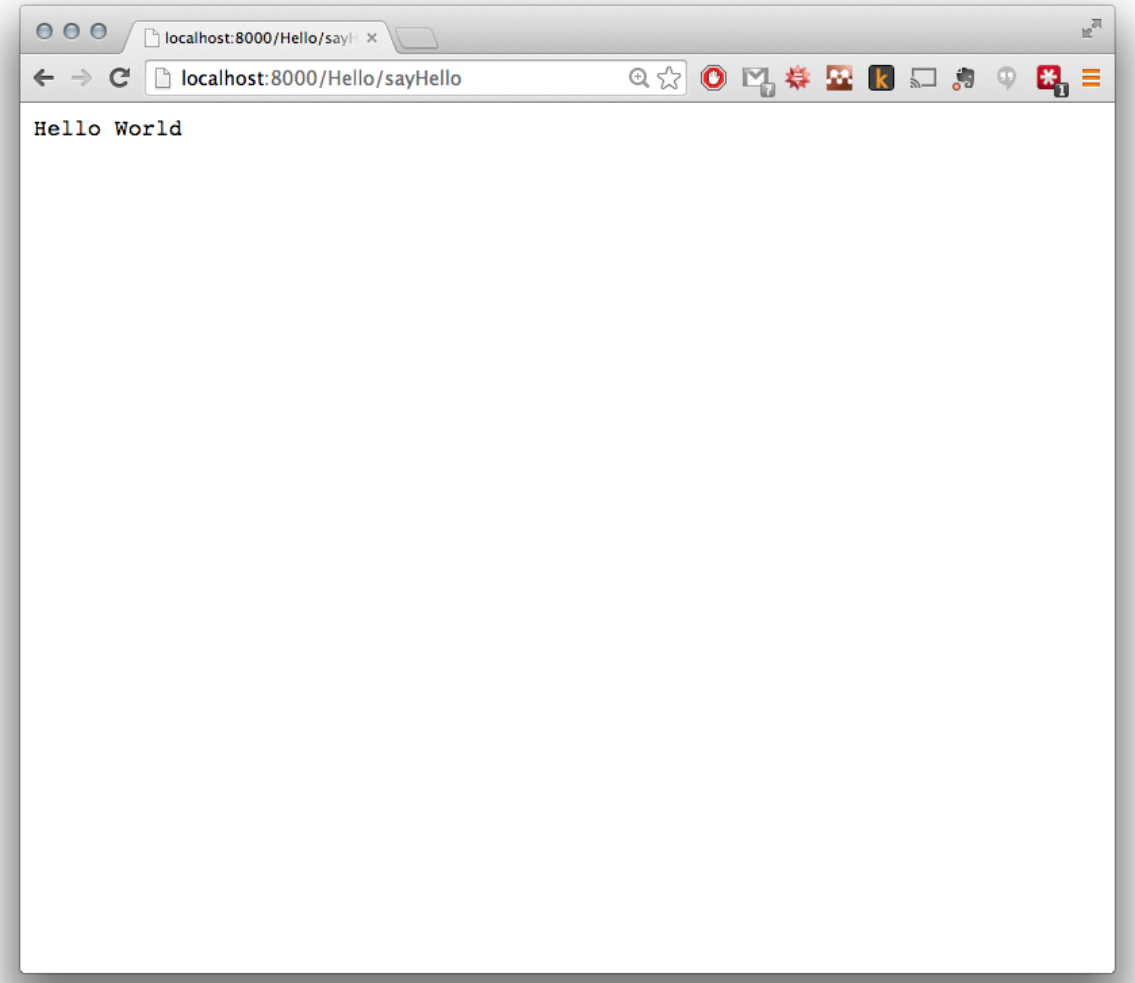
Perhaps the most conceptually most important aspect of the above output is the MRR table, which has been excerpted, below:

```
[INFO] The Following MRR Endpoints Are Available in Your Application:
[INFO] +----------------------+-------------+-------------+
[INFO] | ENDPOINT             | METHOD SPEC | VERBS       |
[INFO] +----------------------+-------------+-------------+
[INFO] | /Hello/sayHello      | ()          | [POST, GET] |
[INFO] +----------------------+-------------+-------------+
```

The table printed above gives us several pieces of information about our small application:

- First, we know that there is exactly one application endpoint available.
- The endpoint that is available maps to our `sayHello` method at the URL `/Hello/sayHello`.
- The `sayHello` method has no formal parameters, thus we should not expect to supply any in the request URI.
- The `sayHello` method is available for either `POST` or `GET` requests.

To execute this method, point your web browser at: `http://localhost:8000/Hello/sayHello`. Your web browser should render something similar to the figure, below:


localhost:8000/Hello/sayH
localhost:8000/Hello/sayHello
Hello World

CHAPTER 6

Next Steps

In this quick start we've only just scratched the surface of Verily. If you'd like to start using the more advanced facilities of Verily to be more reliable web applications, please take a look at the rest of the documentation.

CHAPTER 7

Creating Applications in Verily

CHAPTER 8

How Verily Applications are Structured

8.1 Methods

8.2 How Request Parameters are Decoded

8.3 Routers

CHAPTER 9

Verily Templating Guide

CHAPTER 10

The Global Mutable State Contract

10.1 Using Sessions

CHAPTER 11

Exposing your Methods via AJAX

CHAPTER 12

Adding Libraries to Your Project

CHAPTER 13

Testing Verily Applications

CHAPTER 14

Deploying Verily Applications

CHAPTER 15

A Very Opinionated Guide To Deploying Verily Applications

15.1 Security Considerations

15.2 NGINX

15.3 Supervisor

CHAPTER 16

Why Verify a Web Application?

CHAPTER 17

Program Specification 101

CHAPTER 18

Supported Program Verification Techniques

18.1 Static vs Runtime Checking

CHAPTER 19

Writing Specifications

19.1 Enable Static Checking

19.2 Enable Runtime Checking

CHAPTER 20

Indices and tables

- genindex
- modindex
- search