For LinkedIn

I created the GherkinExecutor as a "let's try it out project." It reads Gherkin files and converts them to unit tests. You can download and try it out at the link in a comment (or just look for atdd-bdd/GherkinExecutorKotlin on Github.

The first version was written in Kotlin. I'm subsequently porting it to Java, C++, C#, and Python.

**Why a new program?**

I've used Cucumber for a number of years. In the beginning, it was simple to have a step data table be passed to a step definition. However it has become a bit more complicated in subsequent versions, especially with classes that have custom data types for the parameters. Google for DataTables and you will find few examples of how to turn them into a List<Class>. . The Cucumber documentation doesn't appear to give much information as how to create a List<Class> from a table. You need to create a TableTransformet and call configureTypeRegistry.

In my work with customers, I've found they understand tables, since that is the language of Excel. But the difficulty of using them in Gherkin makes some developers shy away from them

GherkinExecutor only accepts data in table (or multiline string) format. This greatly simplied its development. No external libraries are used. The entire program is in one source file, so that it easy to install in a project. If a developer wishes to change how it operates, the source code is there for them to do that.

One other advantage is that the scenarios run as native unit tests. So they will be faster than any interpretive Gherkin framework.

**An Example**

Gherkin Executor uses a few extra statements in a Gherkin feature file to simplify reading a table as List<Class>. These statements also document the attributes. Here's an example:



The Data Forecast statement introduces a class / collection of attributes. When this statement is translated, two classes are created. One has attributes of the String type. The other has attributes with the Datatypes listed. A conversion method from one to the other is also created.

The Forecast class is used in the Given statement to create a list of Forecasts. These values could be read from a csv file with an Include statement (e.g. Include '"forecasts.csv").

The GherkinExecutor translates this feature file into three files, named Feature\_Forecast\_Searching.kt, Feature\_Forecast\_Searching\_glue.tmpl, and Feature\_Forecast\_Searching\_data.tmpl. The first time a feature is translated, the latter two files should be renamed to the .kt suffix. You will be adding code in the glue file to call the production code. You may be adding imports of other classes to the data file (e.g. classes used to call the production code).

The Feature\_Forecast\_Searching.kt contains unit tests for each Scenario. It calls a glue function in the glue file and passes it the data in the table. For this Given, it will be a type of List<Forecast>

The When step has two criteria listed. They are passed as a List<ForecastSearchCriteria> to the glue code.

The Then step is passed a List<Forecast> . If a field is not represented by a column, then its value is set to the default in the corresponding Data statement.

**Summary**

You can check out running examples of GherkinExecutor