Tines Offline Coding Exercise

An Introduction to Tines

This coding exercise is based around implementing a very simple version of Tines. To get to grips with it, it will be very useful to first set up your own free, Community Edition of Tines and see how the real thing works. You'll only need to fill in your name and email address to sign up - there are no credit card numbers involved!

To get started go to https://www.tines.io/ and click "Get Community Edition" and complete the sign up process. You can enter "Candidate" when prompted for your company name.

Once you're up and running you can go to the "Your drafts / Stories" page, click on "Import" and upload the attached `sunset.json` file.

This will add a new Story called "Sunset". This is an extremely simple example of a Tines Story. Here's how it works:

- The "Location" HTTP Request Action calls a web service for translating IP addresses into a location (including longitude and latitude) and emits an Event containing that location data.
- That Event is passed to the "Sunset" HTTP Request Action, which takes the longitude and latitude and sends them to a web service that returns the sunset time at that location. The Sunset Action then emits an Event containing all of the location data from the first Action along with the sunset information it has retrieved.
- That Event is passed to the "Email" Email Action, which formats and sends an email with pieces of both the location data and the sunset information.

To give it a go, you can click on the "Email" Action and edit the recipient to change it your email address, then select the "Location" Action and click run. All going well, within a few seconds you should receive an email with the sunset time… wherever ipwhois.io thinks our servers are running.

As mentioned, for this coding exercise we're going to implement a very simple version of Tines, which we'll call Tiny Tines, and try it out by executing a version of the Sunset Story.

Tiny Tines by Example

Tiny Tines is a command line program. It takes the path to a JSON file describing a Tiny Tines Story as its only command line argument and then runs that Story.

To understand how it should work, we'll step through how the attached `tiny-tines-sunset.json` Story should execute with Tiny Tines.

Much like our example full Tines Sunset Story above, the `tiny-tines-sunset.json` Story defines three Actions, two of which are HTTP Request Actions. The first should fetch the longitude and latitude of the location corresponding to the requestor's IP address. The second should use that longitude and latitude to lookup the sunset time at that location. However, then, instead of sending an email with the sunset time as we did when using Tines proper, we'll simply print it to the console.

(If you're curious, you can compare `sunset.json` and `tiny-tines-sunset.json` to get a sense of how we've stripped things back for Tiny Tines.)

The first Action defined in `tiny-tines-sunset.json` is:

```
  {
    "type": "HTTPRequestAction",
    "name": "location",
    "options": {
      "url": "http://free.ipwhois.io/json/"
    }
  }
```

When it runs, it should make a HTTP GET request against http://free.ipwhois.io/json/ and output an Event that corresponds to the JSON response it receives. The output Event should look something like this (we're just showing a small selection of the fields in this document for clarity):

```
{
  "location": {
    // ...
    "country": "Ireland",
    // ...
    "city": "Dublin",
    "latitude": "53.3165322",
    "longitude": "-6.3425318",
    // ...
  }
}
```

Note that this output Event is just an internal construct. It isn't, e.g., printed to the console. It's just passed to the second Action as that Action's input Event.

So the second Action receives that Event as its input and is defined as follows:

```
  {
    "type": "HTTPRequestAction",
    "name": "sunset",
    "options": {
      "url": "https://api.sunrise-sunset.org/json?lat={{location.latitude}}&lng=
{{location.longitude}}"
    }
  }
```

When it runs, it should interpolate the correct values into the URL from its input Event, so that the URL it requests looks something like `https://api.sunrise-sunset.org/json?lat=53.3338&lng=-6.2488`. Its output Event should then look something like this:

```
{
  "location": {
    // ...
    "country": "Ireland",
    // ...
    "city": "Dublin",
    // ...
  },
  "sunset": {
    "results": {
      "sunrise": "5:04:59 AM",
      "sunset": "7:41:16 PM",
      // ...
    },
    "status": "OK"
  }
}
```

The final Action, in turn, receives that Event as its input. It is defined as follows:

```
  {
    "type": "PrintAction",
    "name": "print",
    "options": {
      "message": "Sunset in {{location.city}}, {{location.country}} is at
{{sunset.results.sunset}}."
    }
  }
```

When it runs, it should interpolate the correct values into its message so that it prints the following to STDOUT:

```
Sunset in Dublin, Ireland is at 7:41:16 PM.
```

The Exercise

The exercise is to implement Tiny Tines in the programming language that you're most comfortable in.

When you're done, you should be able to run your program roughly like this (using Ruby as an example):

```
$ ruby tiny-tines.rb tiny-tines-sunset.json
Sunset in Dublin, Ireland is at 7:41:16 PM.
```

Your program should be able to run any Tiny Tines Story JSON file and not just `tiny-tines-sunset.json`. You can try the other example story that's attached, `tiny-tines-today.json`, which has a couple more Actions to test things out. It should run something like this:

```
$ ruby tiny-tines.rb tiny-tines-today.json
Current time:
    2020-05-05T12:20:23.377146+01:00
Fact for today:
    May 5th is the day in 1762 that Russia and Prussia sign the Treaty of St.
Petersburg.
```

You should write your code as if you're going to commit it to a production code base (including tests that use an automated testing framework).

Your implementation must:

- **include tests using an automated testing framework**
- **run any valid Tiny Tines Story JSON file as described by the specification below**
  - Don't hardcode it so that it will only solve the attached `tiny-tines-sunset.json` and `tiny-tines-today.json` files.

When you're ready to submit your solution there are three steps:

1. Edit the first Action in the `tiny-tines-submission.json` Story file to replace `jane@example.com` with your email address.
2. Use your solution to run the updated `tiny-tines-submission.json` Story file. This will post some information to our servers, which will check that your solution is correct. If it is correct, you'll receive an email with the subject line "Tiny Tines Checker - Success!"
3. Email us your source code along with any instructions necessary to run it (and the tests).

If you have any questions, please don't hesitate to ask!

Appendix - Tiny Tines Specification

Tiny Tines is a command line program that takes the path to a Tiny Tines Story JSON file as its only command line argument and executes that Story.

A Story file contains a single JSON object. That object has a single key, `actions`, mapping to an array of Actions. An Action is a building block of a Story that can be configured to perform a task.

An Action is described in the Story file by an object with the keys `type`, `name` and `options`.

An Action's `name` is a string (that must itself be a valid JSON key) and its `options` are a collection of key/value pairs that depend on the Action `type` (a string).

There are just two valid types of Action:

- `HTTPRequestAction` - make a simple HTTP GET request against a given URL.
  - Actions of this type take a single option, `url`, the value of which must be a string.
  - Any non-2xx HTTP response or network failure while making a HTTP request should result in immediate termination of the program.
  - Only requests against URLs that return JSON bodies are supported. The body is parsed and set as the Action's output.
- `PrintAction` - print a given message to STDOUT.
  - Actions of this type takes a single option, `message`, the value of which must be a string.

A Story is executed by running its Actions in the order that they appear in the Story file. When run, each Action produces an output Event, which is then passed to the next Action as its input Event when it is run, and so on.

An Event is a simple dictionary/hash (i.e. something that may be easily serialized to JSON). Its keys are the names of Actions that have run so far and its values are the outputs of those Actions. When producing its output Event, an Action includes in that output Event all of the data from its input Event (so that the output from an Action early on in the Story is passed all the way through to the Actions at the end of the Story).

Values from the Event passed to an Action are interpolated into the options of that Action by placing the Javascript object dot-notation for accessing a value from the Event in between two matching pairs of curly braces. For example, given an input Event `{ "foo": { "bar": "World" } }`, an Action option value of `"Hello {{foo.bar}}!"` should resolve to `Hello World!` when the Action is run.

When the value being referenced within a pair of curly braces doesn't exist in the Event, or is in any way invalid, an empty string should be inserted. For example, given an input Event `{ "foo": { "bar": "World" } }`, an Action option value of `"Hello {{foo.qux}}!"` should resolve to `Hello !` when the Action is run.

When there are mismatched sets of curly braces, they should be left in place and no interpolation should occur. For example, an Action option value of `"Hello {{foo.bar}!"` should resolve to `Hello {{foo.bar}!` when the Action is run. Similarly, given an input Event `{ "foo": { "bar": "World" } }`, an Action option value of `"Hello {{foo.bar}} }}!"` should resolve to `Hello World }}!` when the Action is run.