Two Groovy scripts are provided to simulate the sending and receiving of package information. The scripts were tested with various Groovy versions from 1.8.6 to 2.4.5. Please get in touch with us on the forum if you have any questions or run into any trouble. (If you use groovy locally stale information in your .gradle or .m2 directories might block the grabbing of dependencies.. the fix is to delete those directories as outlined here http://stackoverflow.com/questions/16871792/groovy-grab-download-failed)

PackageTrackingJettyWebServer.groovy:

This script starts a Jetty HTTP server on port 8080 and implements the minimal functionality required to support to the package tracking protocol specified in the requirements.

Feel free to change the port in the script if 8080 is not available, just be sure to update the PackageEventsSimulator.groovy to send to the new port and document in your delivery to us that you are using a different port.

Groovy's built in @Grab feature will download the needed dependencies to run the Jetty web server automatically. The first execution requires a internet connection to get these dependencies and it may be a bit slow to start up.

This script is only meant to be an example of how the reception of REST style HTTP/JSON can be done.. since HTTP and JSON are standards you are free to implement the HTTP server in any language or with any technology that you see fit (provided it can be delivered to and used at IDT at no cost).

PackageEventsSimulator.groovy

This script uses Groovy's HTTP Builder to communicate with an HTTP server on the localhost on port 8080 and sends simulated package events using the data in GPX files.

Groovy's built in @Grab feature will download the needed dependencies to use the HTTP Builder automatically. The first execution requires a internet connection to get these dependencies and it may be a bit slow to start up.

```
usage: groovy PackageEventsSimulator.groovy [options] [gpxFiles]
Options:
 -c,--compressTo <minutes>     Compress all of the times so that the first
```

```
                        start time and last end time fits within
                        the specified number of minutes.
 -help,--help                Print this message.
 -m,--matchStarts            Shift all of the start times to match the
                        first start time.
 -n,--now                    Shift all of the start times to now.
                        (Incompatible with the -s --shiftToEndIn
                        option)
 -s,--shiftToEndIn <minutes>   Shift all of the times so that the last end
                        time is the specified minutes from sending
                        start. (Incompatible with the -n --now
                        option)
------------------------------
```

As an example, " groovy PackageEventsSimulator.groovy -n *.gpx " will load all of the gpx files in the current directory and start playing them as if they were happening now. This usage is likely the most useful for display purposes. The other options may help in testing load or other features of your solution.