Overview

The BikeServer.exe process is an HTTP server that implements an open standard for software and devices to exchange bicycle sensor data.

This is a high level overview:

```
[PowerCurve application] → [BikeServer HTTP server] ← [External application]
```

The PowerCurve application handles the communication with the USB sensors and posts the data to the HTTP server. The External application then obtains the data using the requests defined in this document.

See the “Connection” settings in the PowerCurve application. In the examples below, the “Id prefix” is configured to 'bike' (without the single quotes) in the PowerCurve software.

A DLL based interface is also available to embed directly in an external application. See the PowerCurveDLL.pdf file for documentation.

Data is uploaded to a server using a standard HTTP GET request (http://en.wikipedia.org/wiki/GET_(HTTP)#Request_methods).

Data is retrieved from the server using a standard HTTP GET request and returned in JSON (http://en.wikipedia.org/wiki/Json) format.

Metrics units are used. Weight is in kilograms, speed in meters per second.

Request Format

The easiest way to explain the request format of the bike server is to work by example.

Consider the following request:

http://localhost:7007/?set&id=bike1&name=Mike&weight_kg=62&trainer_type=8&grade=2,3&watts=555.5&headwind_mps=-2

The rider is identified by the parameter **id=bike1**.

So for rider **bike1**, the following attributes will be set:

name = **Mike**
weight_kg = **62**
trainer_type = **8** (see list in the TrainerTypes section of this document)
grade = **2** (in percent)
watts = **555.5** (current power output of rider)
headwind_mps = **-2** (rider is subject to a 2 meters per second tailwind)

This returns a simple JSON object,

{"Status":"OK"}

When a rider does not exist on the server based on the unique id parameter it is created. If the rider already exists, it is updated.

Then we can query the rider with,

http://localhost:7007/?get&id=bike1&name&weight_kg&trainer_type&grade&speed_mps&watts&headwind_mps

and this will return the name, weight_kg, trainer_type, grade, speed_mps, watts, and headwind_mps

in a JSON object like:

```
{
  "Status":"OK",
  "Riders":
  [
    { "weight_kg"    : 62,
      "speed_mps"    : 14.1068192904572,
      "trainer_type" : 8,
      "headwind_mps" : -2,
      "watts"        : 555.5,
      "grade"        : 2,
      "id"           : "bike1",
      "name"         : "Mike"
    }
  ]
}
```

What is interesting here is the speed_mps return value. This value is a realistic speed based on,

1. The power input
2. The weight and corresponding frontal area of the rider
3. The grade
4. The wind

So for example, even with a power input of 0 watts, if the grade is negative, the speed will be greater than 0.

Multiple riders can be queried in one request. For example,

http://localhost:7007/?get&id=bike1&id=bike2&id=bike3&grade&speed_mps&watts&headwind_mps

will return the grade, speed_mps, watts, and headwind_mps for riders identified by id's **bike1**, **bike2**, **bike3**.

What is interesting here is the speed_mps return value. This value is a realistic speed based on,

A few points:

1. The values for the id parameter should only use the characters 0..9, a..z.
2. Decimal values must use the dot, example: watts=235.4.
3. Grade is specified in percent, example: grade=10.3 means 10.3% grade.
4. On a “set” query, any number of attributes may be set. It is acceptable to omit some attributes.
5. On a "set" query, if watts is specified, the speed_mps parameter will be ignored as the speed will be recomputed based on the watts.
6. On a "set" query, if watts is not specified, the speed_mps parameter will be used to estimate the watts based on the trainer types, and a new speed will be recomputed based on the estimated watts.
7. If a rider profile is not specified, the defaults are:

```
Name            = Id
TrainerType     = 5 (Generic fluid trainer)
WeightKg        = 72
Grade           = 0
HeadwindMPS     = 0
```

8. If a "set" query for a rider is not made within 1 minute, the next call to get will return 0 watts and 0 speed for the rider.

Examples

This query,

http://localhost:7007/?get&id=bike1&id=bike1&name&weight_kg&trainer_type&grade&speed_mps&watts&headwind_mps

returns when riders are unknown:

```
{
  "Status":"OK",
  "Riders":[]
}
```

This query,

http://localhost:7007/?set&id=bike1&name=Mike&weight_kg=62&trainer_type=0&grade=2,3&speed_mps=7.345&watts=555.5&headwind_mps=-2

returns:

{"Status":"OK"}

This query,

```
http://localhost:7007/?set&id=bike2&name=Bob&weight_kg=62&trainer_type=0
&grade=2,3&speed_mps=7.345&watts=555.5&headwind_mps=-2
```

returns:

```
{"Status":"OK"}
```

This query,

```
http://localhost:7007/?get&id=bike1&id=bike2&name&weight_kg&trainer_type
&grade&speed_mps&watts&headwind_mps
```

returns when rider's data has been sent to server:

```
{
  "Status":"OK",  "Riders":
  [
      { "weight_kg"    : 62,
        "speed_mps"    : 14.1068192904572,
        "trainer_type" : 0,
        "headwind_mps" : -2,
        "watts"        : 555.5,
        "grade"        : 2,
        "id"           : "bike1",
        "name"         : "Mike"
      },

      { "weight_kg"    : 62,
        "speed_mps"    : 14.1068192904572,
        "trainer_type" : 0,
        "headwind_mps" :-2,
        "watts"        :555.5,
        "grade"        :2,
        "id"           :"bike2",
        "name"         :"Bob"
      }
  ]
}
```

Trainer Types

The list of supported trainers is available online at http://www.powercurvesensor.com/cycling-trainer-power-curves/. The trainer_type value are encoded in the URL. For example, the Kinetic Rock and Roll trainer has a picture URL of, http://www.powercurvesensor.com/files/18.jpg, so the trainer_type is 18.