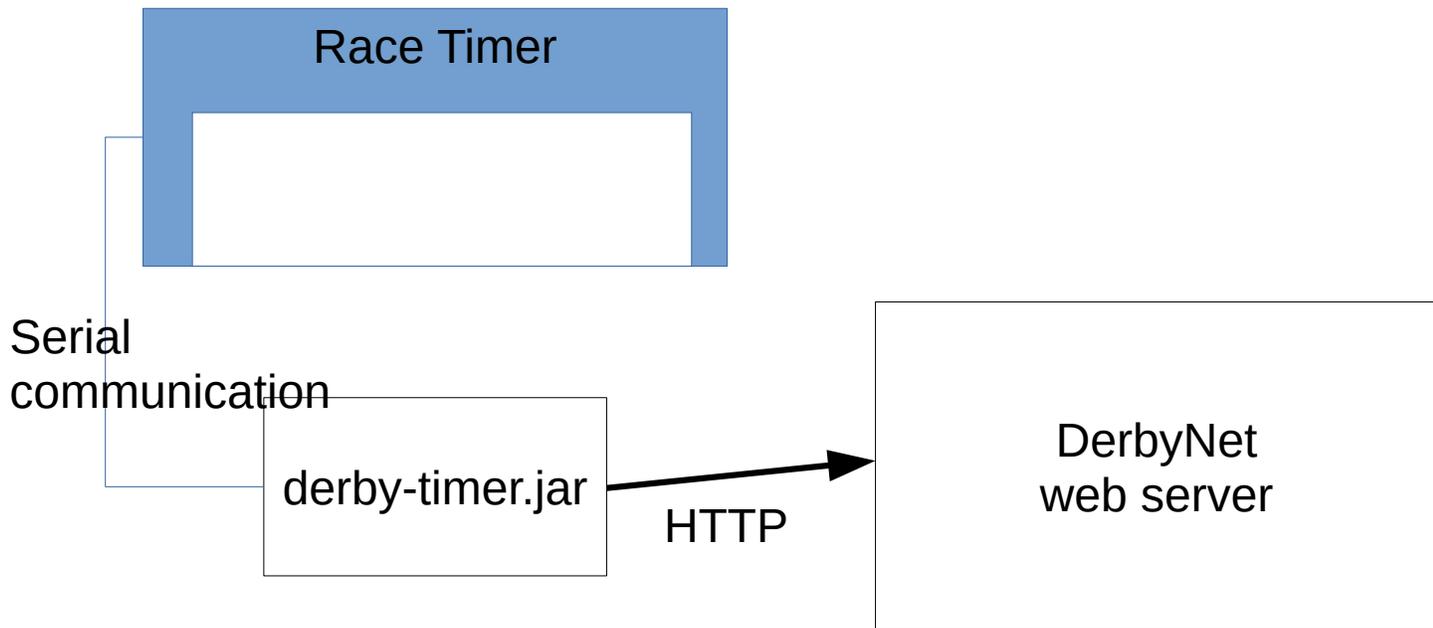


The Timer Protocol for DerbyNet



Most currently-available electronic pinewood derby timers communicate over a serial or USB connection to a host computer of some kind. DerbyNet interfaces with these timers by using a small java program, `derby-timer.jar`, to interact with the timer over the serial data link; `derby-timer.jar` simultaneously interacts with DerbyNet by sending HTTP requests to the DerbyNet web server.

This document describes the HTTP protocol by which `derby-timer.jar` and the DerbyNet web server communicate. A “smart” timer might conceivably implement this protocol directly, in which case there would be no need for a separate `derby-timer.jar`.

`derby-timer.jar` communicates with the server by sending HTTP POST requests the `action.php` URL. The body of the POST request in `application/x-www-form-urlencoded` format, and always includes an **action** parameter that identifies what the server is being requested to do.

Logging In To The Server

Before any data will be accepted by the server, `derby-timer.jar` must log in to the server and obtain a session cookie. This is done by sending a POST request with **action**=login, and additional parameters **name** and **password**. Over the wire, the request might look like this:

```
POST /derbynet/action.php HTTP/1.1
User-Agent: Java/1.8.0_25
Host: localhost
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-type: application/x-www-form-urlencoded
Content-Length: 33
```

action=login&name=Timer&password=

(By default, DerbyNet provides for a "Timer" user with an empty password.)

If the provided name and password are recognized, the server should respond with a session cookie (in a Set-Cookie header) and, in the response body, a "success" response in XML:

```
HTTP/1.1 200 OK
Date: Tue, 18 Dec 2018 23:25:45 GMT
Server: Apache/2.4.34 (Unix) LibreSSL/2.5.5 PHP/7.1.19
X-Powered-By: PHP/7.1.19
Set-Cookie: PHPSESSID=00gkj5pi4t1f6v67kfh7te9pkk;
path=/Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Length: 129
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/xml; charset=utf-8

<?xml version="1.0" encoding="UTF-8"?>
<action-response action="login" name="Timer" password="...">
  <success/>
</action-response>
```

Messages to the Server

Once derby-timer.jar has logged in, it uses the session cookie in all subsequent POST requests, and generally specifies a value of **timer-message** for the **action** parameter, along with a **message** parameter that says more specifically what the timer is trying to convey.

```
POST /derbynet/action.php HTTP/1.1
User-Agent: Java/1.8.0_25
Host: localhost
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-type: application/x-www-form-urlencoded
Content-Length: 38
Cookie: PHPSESSID=g952vft95ui0ttpdqk92ocqdv
```

action=timer-message&message=HEARTBEAT

The individual timer messages are described below:

message=HELLO

After logging in, derby-timer.jar normally sends a HELLO message to indicate that it has established communication with the server and is searching to identify the track timer.

message=IDENTIFIED

Once communication with the track timer has been established, derby-timer.jar sends an IDENTIFIED message to the server. An optional **lane_count** parameter may be supplied if the timer has been identified as having a certain number of lanes.

message=STARTED

If derby-timer.jar is able to detect when the start gate has opened, it sends a STARTED message to the server.

message=FINISHED

When the timer reports results to derby-timer.jar, derby-timer.jar sends a FINISHED message to the server. The FINISHED message is accompanied by additional parameters indicating the time and place of each lane that had a reported result. E.g., the message body might be,

```
action=timer-message&message=FINISHED&roundid=7&heat=4
&lane1=3.21&place1=2&lane2=3.33&place2=3&lane3=3.14&place3=1
```

Here derby-timer.jar is confirming that the results it's reporting are for the fourth heat of the racing round whose identifier is 7, that the car in lane 1 has a time of 3.21 seconds and came in second in this heat, etc.

The server may reject the reported results if e.g. they don't correspond to the currently-running heat (i.e., if the heat and roundid values aren't what were expected).

message=HEARTBEAT

This is just a simple keep-alive message that confirms that communication between derby-timer.jar and the server is still in place. It also acts as a polling message, allowing the server to provide details about the current heat in the response.

message=MALFUNCTION

If derby-timer.jar detects that it has lost communication with the timer (e.g., because of an unplugged cable), or if some other problem is detected, derby-timer.jar sends a MALFUNCTION message to the server. The two additional parameters for a MALFUNCTION message are **detectable** (values are 0 or 1) and **error** (an error message, if there is one).

If derby-timer.jar reports that a malfunction is “detectable,” that means that derby-timer.jar will be able to report when the problem has been resolved (e.g., cable plugged back in). If the malfunction is not “detectable,” the server should instead prompt the user to indicate when the problem has been corrected, as derby-timer.jar will not be able to report the change.

Responses from the Server

The server's HTTP response to the POSTed action is in XML, and may contain message elements unrelated to the POST. The response will generally comprise some combination of the following XML elements:

<success/>

The server will normally include a success element to confirm that the POSTed message was received and processed.

<failure code= />

If the response does not have a success element, it will have one or more failure elements indicating why the POSTed message was not processed as expected. For example, if a FINISHED message is received by the server but no heat is current, the server will respond with a failure element that says there's no racing presently underway. The failure element will have a short "code" attribute, along with a human-readable text body.

<heat-ready lane-mask= class= round= roundid= heat= />

When a new heat is ready for racing, the server's response will include a heat-ready element providing derby-timer.jar details about that heat. The heat-ready element has no body, but includes the following attributes:

- **lane-mask:** The attribute value is a decimal representation of a bit mask showing which lanes will be occupied for the heat. E.g., lane-mask="14" means lane 1 (2^0) will be empty but cars will be in lanes 2 (2^1), 3 (2^2), and 4 (2^3).
- **class:** This attribute contains a human-readable name for the current racing class (typically a "den" or other group).
- **round:** The ordinal round number for the class (e.g., 1 for the first round, 2 for the second, etc.).
- **roundid:** The internal integer identifier of the round. Roundids are unique across all rounds in the database. (Thus, two different classes may each have a first round, but those rounds will have different roundids.)
- **heat:** The number of the current heat within the current round, with 1 being the first heat.

<abort/>

If a schedule change occurs after a heat-ready response, the next message to the server will include an abort element, canceling the effect of the heat-ready element.