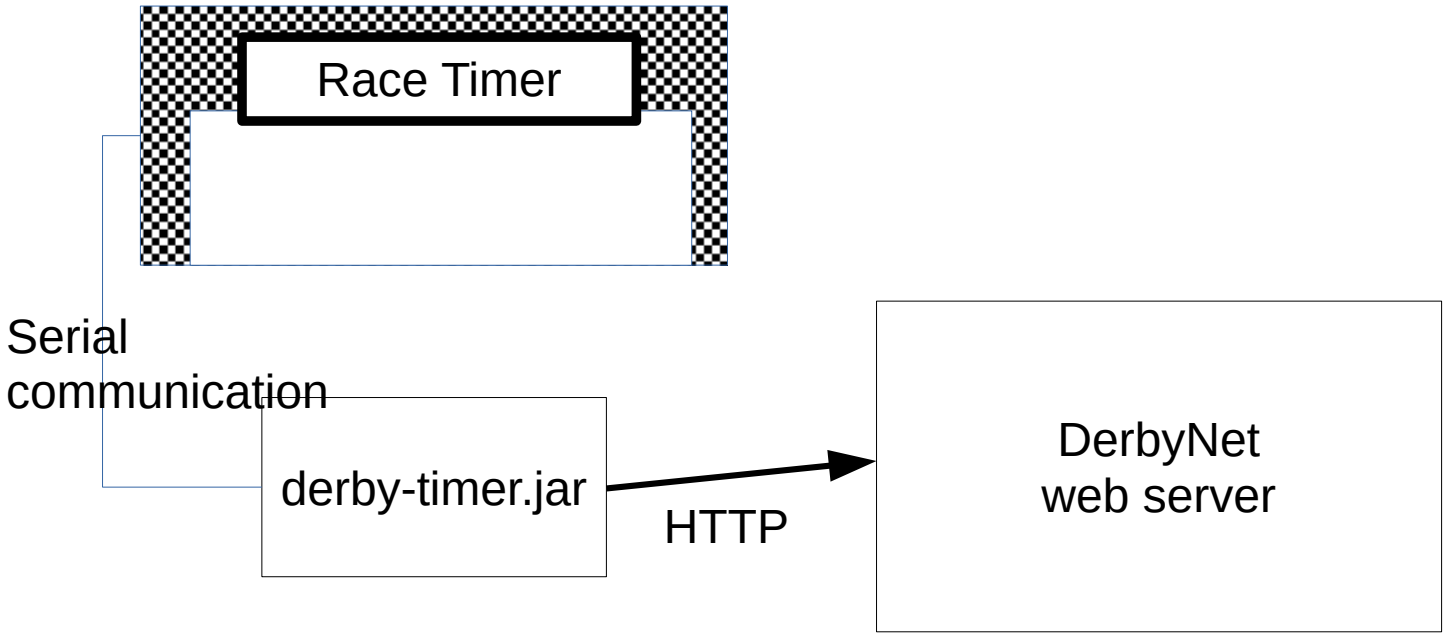
The Timer Protocol for DerbyNet



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[1]. The body of the POST request is presented as if it were data from an HTTP form (typically bearing a content-type of `application/x-www-form-urlencoded`[2]), 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**=`role.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
```

1 Sending logging data uses a different URL; see Remote Logging, below.

2 The content-type may alternatively be `multipart/form-data` (rfc7578).

```
Connection: keep-alive
Content-type: application/x-www-form-urlencoded
Content-Length: 33

action=role.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="role.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=g952vft95ui0ttpdqk92ocqdvc

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 establish communication with the server and indicate that it 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. Several other optional parameters allow details about the timer to be recorded and presented to the user:

- An optional **timer** parameter may convey an identifier of the type of timer; derby-timer.jar sends the name of the Java class used.
- An optional **human** parameter may convey a human-readable name for the timer type (e.g., manufacturer and/or model).
- An optional **ident** parameter may convey a unique identifier (serial number, e.g.) for the timer.
- An optional **options** parameter may convey details about special features that are present in this particular timer. If present, this is a comma-separated list of key:value pairs.

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. The presence of an optional **unhealthy** parameter signifies that the timer is not presently in an operable state. An optional **confirmed** parameter with value 0 indicates that the identity of the timer has not been conclusively confirmed, because the timer has not yet sent any output. (A timer that does not support two-way communication will normally be considered unconfirmed until something causes it to send data.)

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.

message=FLAGS

Send the complete set of command-line flags, detected serial ports, and available timer classes to the server, encoded as additional parameters.

Command-line flags are sent as two parameters:

- flag-*{flagname}={type}*:*{value}*, e.g., flag-x=bool:false, and
- desc-*{flagname}={description}*, e.g., desc-x=Run headless: without GUI.

Detected serial ports are sent as a comma-separated list in the ports parameter, e.g., ports=COM1,COM2.

Finally, each of the available timer classes is sent as a device-*{classname}={human-friendly string}*, e.g., device-FastTrackDevice=FastTrack K-series.

Here's an abbreviated example of what might be sent over the wire:

```
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=g952vft95ui0ttpdqk92ocqdvc

action=timer-message&message=FLAGS&flag-x=bool:false&desc-
x=Run+headless:+without+GUI&ports=COM1%2CCOM2&device-
FastTrackDevice=FastTrack+K-series
```

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.

<remote-log send=/>
See "Remote Logging," below.

<remote-start/>
See "Remote Starting Gate," below.

<query/>
Requests that derby-timer.jar send a FLAG message to transmit the current complete state of its command-line flags, detected serial ports, and available timer classes.

<assign-flag flag= value=/>

Changes the value of a command-line flag. (Note that some command-line flags, such as `-x`, only affect the start-up behavior of derby-timer.jar; changing their values in this way will have no practical effect.)

<assign-port port=/>

If the port attribute is provided, selects the given port for communication with the timer. If no port attribute is provided, removes any such selection made previously (via a `-n` command line option, GUI gesture, or a previous assign-port element).

<assign-device device=/>

If the device attribute is provided, selects the named device class for communication with the timer. If no device attribute is provided, removes any such selection made previously (via a `-d` command line option, GUI gesture, or previous assign-device element).

Remote Logging

The derby-timer.jar can send logging data to the server, to help observe or diagnose timer behavior.

A response from the server that includes a **<remote-log>** element turns remote logging on or off, depending on the value of the **send** attribute.

If remote logging is active, derby-timer.jar makes POST requests to the post-timer-log.php URL. The POST request body has content-type `text/plain` type. The server appends the request body text to the logged text captured on the server.

Remote Starting Gates

Some timers have an option to operate a solenoid to open the starting gate and begin a race. For those timers, the timer client should advertise this capability by including a **remote-start** parameter, with value **YES**, on every message to the server, in addition to the **action** and **message** parameters described above in the “Messages to the Server” section.

To start a race with a remote starting gate, the server will include a **<remote-start/>** element in a response.

Stand-Alone Remote Starting Gates