Revision History

date	revision	details
2009-12-18	7	added connection-id init line for delayed newlocation events
2009-09-03	6	enhancement to protocol for aiding in resume stream operations
2009-03-12	5	clarifications and textual updates
2008-10-16	4	added time-format option
2008-07-25	3	minor additions
2008-04-23	2	added some details on command replies updated the start command updated explanation on newlocation events
2008-04-15	1	added pushmode-ack functionality (by request of Alan Jones)
2008-04-04	0	initial document

Introduction

The following document describes Chronotrack Socket Protocol 1 (from now on referred to as CTP01). This protocol can be implemented by Race Scoring software, Text Messaging Services and other Event software to interface with Chronotrack System Software through TCP socket connections. Currently the Chronotrack Software will act as a TCP client and 3rd party software needs to implement a TCP Socket Server.

The protocol is human readable making it easy to implement and/or debug. All commands and responses will be terminated by Carriage Return + Line Feed (**\r\n**) and uses tilde (**~**) as field separator.

The protocol allows for different methods of dealing with data, making it suitable for a variety of different programs. By sending initialization requests in the greeting phase it is possible to select operating mode and additional features / commands. This way the Chronotrack software knows how to communicate with the other software and not all features of the protocol have to be implemented.

In the following few pages two stream modes are distinguished: pull and push mode. You do not need to implement both pull and push mode, implement the mode that makes the most sense for how your software is developed and deals with data streams.

If your software will be connecting with Chronotrack client software over an unsecured network like the internet it might be a good idea to implement authentication as described below. However, most results scoring software will connect with Chronotrack clients on local networks controlled by the timer and often both software packages even run on the same computer. In these cases authentication does not seem very necessary. It is again up to the developer to decide to implement authentication or not.

ChronoTrack Software defaults to TCP Port 61611 for the CTP01 protocol. It is possible to use another listen port but will be inconvenient to timers expecting default settings. Only divert from this port if absolutely necessary (example: other services already using this port).

For more detailed information on how tag observations can be send over the wire, please consult the text-file format documentation.

Connection Initiation

Upon connection the Chronotrack client will send a greeting message to the server :

Syntax:
<program-name>~<program-version>~<protocol-version>
Example:
`TripleC~0.1.0~CTP01`
Comment:
This protocol version is identified by CTP01

The Server must respond with:

Syntax:
<program-name>~<program-version>~<number of request lines>
Examples:
`RunScore~Version 6.5 Level 2007.09.26~2`
`Race Director~Version 2008 rev 3~0`
Comment:
If <number of request lines> = 0 all default values are assumed as described in the table below

The <number of request lines> informs the Chronotrack System client that it can expect a couple of initialization requests in the form of:

Syntax:
<request-1>=<value-1>
..
<request-n>=<value-n>
Examples:
`location=single`
`guntimes=true`

These must be sent immediately after the Server Response line. Currently available request options:

request option	**values**	**default**	**comment**
location	single, multi	multi	send tag observation data from a single location or from multiple locations
guntimes	true, false	false	send gun-time events if available (affects push mode only)
newlocations	true, false	false	send new location events if available (affects push mode only)
authentication	none plaintext hmac-sha1 hmac-md5 seed-sha1 seed-md5	none	specify authentication type
authentication-seed	<string>		seed used for hmac or seeded-hash based authentication
stream-mode	push, pull	pull	put client in push mode (automatic sending of data) or pull mode (request data from client)
tagevent-format	CT??_??	CT01_33	For available values see text-file format documentation
pushmode-ack	true, false	false	send replies on push mode start and stop commands

request option	values	default	comment
time-format	normal iso unix msecs	normal	time formatting options, only available with variable-width formatting (see tagevent-format) Examples: October 16th 2008, 2:02:15pm and 31 hundreds. normal: 14:02:15.31 iso: 2008-10-16T14:02:15.31 msecs: 43455310 unix: 1224165735.31
connection-id	true, false	false	if set to true and newlocations is set to true, the newlocation events will be delayed until the server has requested the connectionid using the getconnectionid command. This eliminates the need for the server to buffer these events.

Authentication

If your software package is running as a service on the internet or another unsecured network you can require the client to use authentication by setting the “authentication” initialization request during the initial connection phase. If authentication is requested through the server greeting, the client sends an authentication request to the server. The authentication string reported back will use the authentication-seed provided by the server to calculate the correct hash (when using hmac or seed authentication). The server should generate a different seed for each new connection to the server.

Syntax:

authorize~<user-id>~<authentication-string>

Example:

```
authorize~TimersRus~b617318655057264e28bc0b6fb378c8ef146be00
```

If plaintext authentication is used the <authentication-string> is the password itself, if an HMAC (hmac-sha1 or hmac-md5) is used it will be the digest where the HMAC message is the <authentication-seed> and the shared secret is the client password.
If a seeded hash is used (seed-sha1 or seed-md5) the <authentication-string> is calculated by hashing the following string: <authentication-seed><password>
By comparing the client digest with your local generated digest you can check the authenticity of the connected client. HMAC authentication “hmac-sha1” is the “safest” method for authentication.

If authorization fails the server should send an error message and disconnect the TCP socket

Syntaxes:

ack~authorize
err~authorize[~<optional error message>]

Examples:

```
ack~authorize
err~authorize~invalid password
err~authorize~your account has been suspended
```

Note: If you want to use a hashing method (hmac-sha1, hmac-md5, seed-sha1, seed-md5), your server must create it's own digest by retrieving the requested password for the specified <client-id> from it's database and compare it's own digest against the digest from the client. This means the password cannot be saved in the database as a hash since you will need to use the real password to calculate the local digest. Either save passwords plaintext in the database or use a 2-way encryption method like Rijndael or TripleDes to store encrypted passwords.

Post authentication

When authentication is successful or no authentication is required, the client will start listening to commands. If a command or it's parameters is invalid, the client will throw an error reply.

Syntax:

err~<command>~<error message>

Examples:

```
err~~unknown command
err~getdata~invalid location
```

Generic Commands

The following commands can be issued regardless of the stream mode.

Receive event name, event id, event description from active Chronotrack System (if available):

Syntax:
geteventinfo
Response Syntax:
ack~geteventinfo~<event name>~<event id>~<event description>
Example:
S: `geteventinfo`
C: `ack~geteventinfo~LA08~12~Los Angeles Marathon 2008`

Check if connection to Chronotrack client is still alive:

Syntax:
ping
Response Syntax:
ack~ping

Retrieve a list of locations that the Chronotrack client has available to the server through this socket link:

Syntax:
getlocations
Response Syntax:
ack~getlocations~<location 1>[~<location ..>[~<location n>]]
Example:
S: `getlocations`
C: `ack~getlocations~start~10k~half~30k~finish`

Retrieve the unique connection id from a client connection. This can be used to identify if the connection is a new connection or a re-established one. Together with the last received sequence numbers per location for a specific connection, the server will be able to resume the stream(s) at the correct offset:

Syntax:
getconnectionid
Response Syntax:
ack~getconnectionid~<connection id>
Example:
S: `getconnectionid`
C: `ack~getconnectionid~b617318655057264e28bc0b6fb378c8ef146be00`
Comment:
this function is only available on SimpleClient version 0.4.0+.

Push Mode Commands

In push mode tag observations (and if requested gun-times) are sent as soon as they become available to the Chronotrack client and the server has started the stream for the location(s). If necessary the server can instruct the Chronotrack client to stop sending data by issuing a stop command.
When using the getconnectionid command after initial connection and authorization, it is possible to keep a status list at the server on sequence numbers per available location for that specific connection. In case a connection drops and is reestablished it is possible to correctly identify the location sequence numbers to use for resuming previous streams. Do note that in between new locations might have become available to the client. These new locations should be started from the beginning.

Start streaming tag observations (and gun-times):

Syntax:
start[~<location name>[~<location sequence number>]]
Examples:
`start`
`start~10k`
`start~finish~4530`
Comment:
If <location name> is omitted, start all locations available at that time (new locations will never automatically start). If <location sequence number> is omitted, start from the first observation.

Stop streaming tag observations (and gun-times):

Syntax:
stop[~<location name>]
Example:
`stop`
`stop~start`
`stop~10k`
Comment:
if <location name> is omitted, all locations connected through this link will be stopped.

When pushmode-ack=true, the start and stop commands will issue replies:

Syntax:
ack~start<~location name>
err~stop~error message details

When streams are active the following type of messages can be expected:

Tag Observation message:

Syntax:
Depends on tagevent-format setting, for details consult the text-file format documentation.
Example:
`CT01_33~1~start~9478~08:29:10.29~0~0F2A38~1`

Gun-time message:

Syntax:
Depends on tagevent-format setting, for details consult the text-file format documentation.
Example:
`CT01_33~4~start~guntime~07:45:01.01~0~DF239A~0`
Comment:
The syntax of a guntime message is exactly the same as for a tag observation. Instead of the tagcode the string “guntime” is placed in the tagcode field and the reader id field contains the controller id. Gator number and lap count are always 0.

If the Chronotrack client receives a new timing location and the protocol is in multi-location mode and newlocations are allowed to be send, the server will be informed of newly available locations. This will be done by a message similar to the gun-time message. If the client has locations available and the server is connected later on, the server can expect newlocation messages to arrive after the authentication phase or after the greeting phase if authentication was not required.

New location available message:

Syntax:
Depends on tagevent-format setting, for details consult the text-file format documentation.
Example:
`CT01_33~2038~30k~newlocation~09:15:21.00~0~ED932A~0`
Comment:
The syntax of a new location message is the same as for a tag observation. The location field contains the label of the new location. Instead of the tagcode the string “newlocation” is placed in the tagcode field and the reader id field contains the controller id. The time field shows the timestamp of when the connection with the Chronotrack client was achieved. Gator number and lap count are always 0.

If you have not allowed for new location event messages but are in multi location mode, you will need to regularly request a locations listing through the “getlocations” command, to check if a new location has become available.

Note on line termination and field separators:
Even though the protocol itself uses **~** (tilde) as field separator and **<Cr><Lf>** as line terminator the tag observation data stream can use other characters as defined by the **tagevent-format** option. Default setting is CT01_33 which means the tag observations use the same characters for field separation and line termination. Be careful when altering line termination as it complicates parsing the data stream when both commands and tag observations are sent.

Pull mode Commands

In pull mode tag observations must be requested by the server, which can be done with the getdata command:

Syntax:

getdata~<maxitems>[~optional location filter[~optional sequence number]]

Examples:

```
getdata~50
getdata~50~finish
getdata~50~finish~209
```

Comment:

The optional location filter can be useful when multiple location streams are connected to this TCP socket (location=multi). If a certain location has priority or processing of single location data / data-burst is preferred the filter can be set. If an optional location filter is set, it is also possible to request data starting from a specific sequence number. This is useful in case a connection was dropped but you do not wish to start again from the first observation for that location. To identify that a previously dropped connection has indeed reconnected (thus it's not a new connection) use the getconnectionid command described earlier.

The Chronotrack client will send available tag observations to the server limited by <maxitems>. The remaining tag observations must be requested again by issuing another getdata command. If less than maxitems is available this will be reflected by <itemcount> and the amount of following tag observation lines. The client sends tag observations as follows:

Response Syntax:

ack~getdata~<itemcount>
<tag observation line as described in CSV format and set to tagevent-format>

Example #1:

```
ack~getdata~3
CT01_33~1~start~guntime~07:45:01.01~0~DF239A~0
CT01_33~2~finish~12~08:38:58.97~0~0FA22E~4
CT01_33~3~finish~2948~08:39:01.62~0~04BE82~2
```

Example #2:

```
ack~getdata~0
```