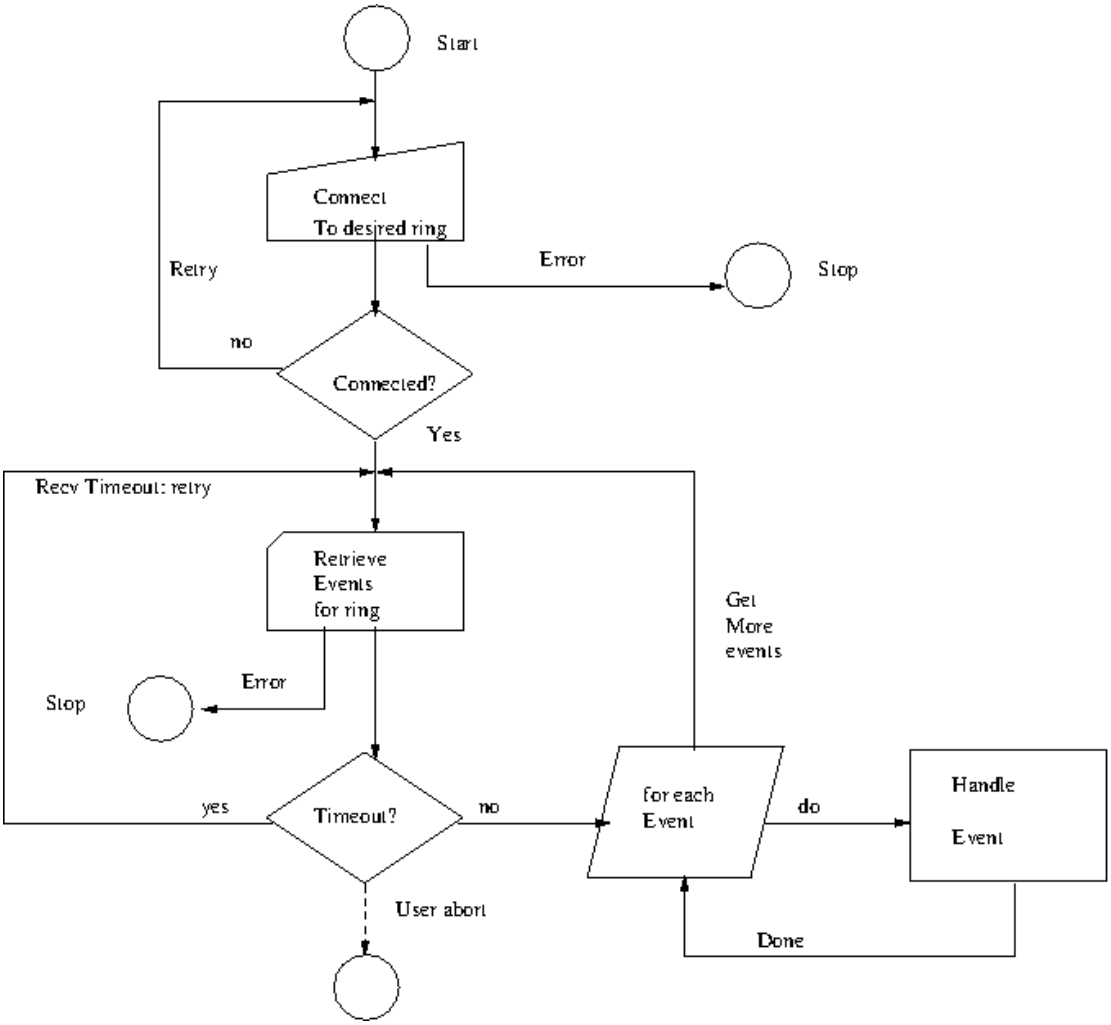
AgilityContest Event Protocol Reference Manual

Author: Juan Antonio Martínez < jonsito at gmail dot com >
Version: 1.0 May 2016



This document describes the internal messaging protocol used to allow communication between every AgiltyContest subsystem. Samples and Reference implementations are provided to allow developers integrate their own systems with AgilityContest.

Introduction:

AgilityContest Event Protocol is the way used for each subsistem to tell what's going on:

- When configuration is changed at console
- When camara source changes for embbedded LiveStream
- When an operator connects a tablet to a ring session
- Round selection, Competitor entering ring
- Electronic chronometer event notifications (coursewalk, 15s countdown, start/stop…)
- Competitor result notification: faults, refusals, eliminated and so
….

Event data contains everything needed to keep track of what's going on:

- Event source and type
- Associated sesion/ring
- Timestamp
- Textual information to be shown at OSD screens
- Competitor results

Events are ALLWAYS stored in database. Additionally (configurable option) you can keep textual track of them in a disk file, and in thermal printer (to make a real time hard copy of competitor stored results)

Event protocol is based on asynchronous AJAX calls and JSON responses, but development is not restricted to javascrip: this document provides sample reference implementations either in JavaScript or Python3 languages

There are to kind of actors in this scenario

Senders: system that generates events:
- Console
- Assistant tablet(s)
- Electronic chronometer(s)

Receivers: systems that tracks events
- VideoWall screens
- LiveStream system
- Senders (also can act as receivers)

The basic flowchart is described in above page diagram:

- Connect to a ring. If the system is going to act as a sender, it must authenticate and provide a name
- Enter in an infinite loop to receive events for requested ring
- if no new events (timeout) call server again
- else iterate on received events and repeat loop

Notice due to overload risk (too many clients) PublicWeb access subsystem is not event driven: it uses their internal timers to refresh information. Due to this fact cannot show pure-realtime events (as tablet/crono faults, timing/scoress and so)

Event Summary:

General events:

```
  0  => 'null',              // null event: no action taken
  1  => 'init',              // start assistant console operation
  2  => 'login',             // user login into server subsystem
  3  => 'open',              // round selection in assistant console
  4  => 'close',             // end of round handling
```

Manual Chronometer events

```
  5  => 'salida',            // 15seconds countdown start
  6  => 'start',             // Manual chronometer start
  7  => 'stop',              // Manual chronometer stop
```

Electronic Chrono events

```
  8  => 'crono_start',       // start run
  9  => 'crono_restart',     // manual-to-automatic chrono transition
  10 => 'crono_int',         // intermediate time
  11 => 'crono_stop',        // stop run
  12 => 'crono_rec',         // 7 minutes course walk start
  13 => 'crono_dat',         // competitor data from chronometer (when available)
  14 => 'crono_reset',       // reset manual/auto/countdown/coursewalk clocks
  15 => 'crono_error',       // sensor error detected
```

Competitor data events

```
  17 => 'datos',             // manual introduction of data from assistant console
```

Action events

```
  16 => 'llamada',           // call competitor to enter ring
  18 => 'aceptar',           // save competitor scores
  19 => 'cancelar',          // exit competitor without saving
```

Configuration events

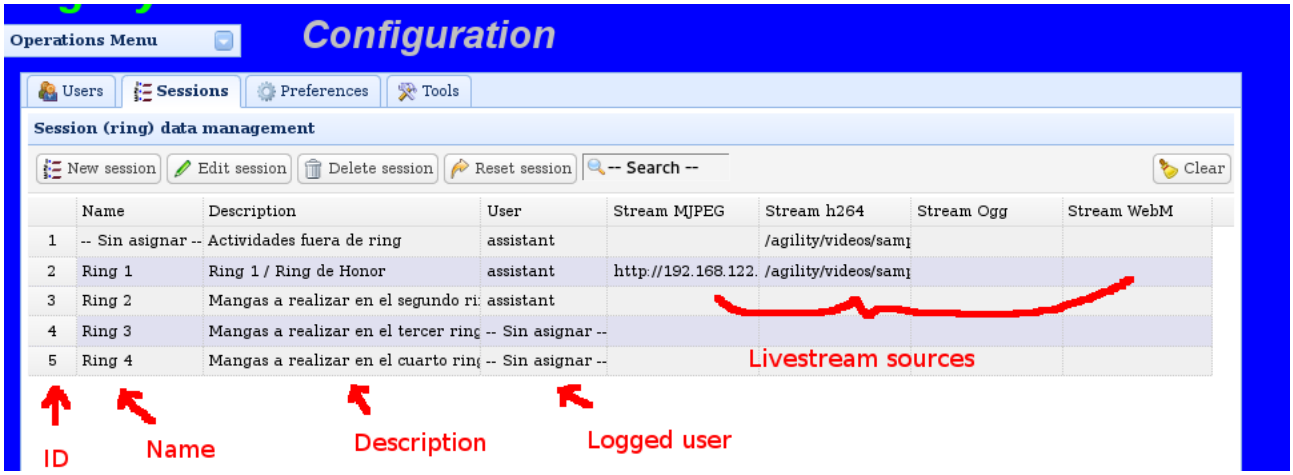
```
  20  => 'info',             // round information
  21 => 'camera',            // embedded livestream video source changed
  22 => 'reconfig'           // system configuration changed
```

Events have associated data:

- Every event have ID, date&time, source and session ring
- Chronometer events also include timestamp marks (milliseconds)
- Data and Chrono_dat events include information on faults, refusals, eliminated, and so
- Open/Info and Llamada events include information on Textual names (contest, journey, round and competitor's info)
- Also events include numerical ID's to allow AgilityContest subsystem make additional server calls to retrieve extra information (get results, SCT and MCT, and so). These server calls are internals to AgilityContest and not included in this document.

Session handling

At AgilityContest console, user can configure and declare/remove sessions. It's recommended not to change nor delete pre-defined one (except for setting livestream associated data)



ID: is the session Identifier: user must provide this value when requesting events. Notice that AgilityContest support several simultaneous sessios, so need to use ID to evaluate which events should be reported

Session 1 is “broadcast”: Every event sent to session ID:1 will be reported to every event receivers

Name and Description are usefull to allow receiver retrieve correct session ID that he is looking for (see server/session discovery section)

User tracks the last user logged in this session.

LiveStream sources are used for embedded video at livestream subsystem. It tells system where to get background embedded video in proper displays

Server/Session discovery

You can connect to event system in two ways:

a)- With previous knowledge of Server IP Address and Session ID
b)- Trying to locate server and evaluate Session ID based on their name/description

In automated systems there is no easy way to provide a) way so AgilityContest provides a discovery protocol. You can check it in Raspberry PI python reference code for chronometer at function “lookForServer()”

To summarize:

- Retrieve from DHCP Server IP address and NetMask for client
- Iterate over every available host addres in the network. There only can be 1 AgilityContest server in same network/netmask
- From response extract session list and retrieve their ID's. Notice that at least knowledge of Session Name is required to identify ID

You can see here a sample implementation for the discovery protocol written in nodejs-v4. The complete code can be located at AgilityContest github site

```
/**
 * This function explores network trying to locate an AgilityContest server
 * When found, deduce desired sesion ID and set up working data parameters
 *
 * Notice that due to async nature of http requests, a dirty trick is needed
 * to wait for host polling to finish. this should be revisited in a later revision
 *
 * @param {int} ring Ring number (1..4) to search their sessionID for
 * @returns {boolean} true when server and session found. otherwise false
 */
function findServer(ring) {
   var addresses = []; // to be evaluated
   // locate any non local interface ipaddress/mask
   // take care on multiple interfaced hosts
   function getInterfaces() {
      var os = require('os');
      var interfaces = os.networkInterfaces();
      for (var k in interfaces) {
         for (var k2 in interfaces[k]) {
            var address = interfaces[k][k2];
            if (address.family === 'IPv4' && !address.internal) {
               addresses.push(address);
            }
         }
      }
   }
   /**
    * Try to connect AgilityContest server.
    * on success retrieve Session ID and update working Data
    * @param hostaddr IP address of host to check
    * @return true on success; otherwise false
     */
   function connectServer(hostaddr){
      var url="http://"+hostaddr+"/agility/server/database/sessionFunctions.php?Operation=selectring";
      var request = require('sync-request');
      try {
         var res = request('GET', url, {
            timeout: 250,
            headers: { // this is for some routers that return "Auth required" to fail and send proper error code
               authorization: 'Basic ' + new Buffer('AgilityContest' + ':' + 'AgilityContest',
'ascii').toString('base64')
            }
         });
         if (res.statusCode!=200) return false; // http request failed
         console.log("Found AgilityContest server at: "+hostaddr);
         var data = JSON.parse(res.getBody('utf8'));
         // this code assumes that first returned row matches ring 1, second ring 2 and so
         workingData.hostname=hostaddr;
         workingData.sessionID=parseInt(data['rows'][ring-1]['ID']);
         console.log("SessionID for ring:"+ring+" is:"+workingData.sessionID);
         return true;
      } catch (err) {
         console.log("Host: "+hostname+" Error: "+err);
         return false;
      }
   }
   getInterfaces(); // retrieve network interface information
   var ip=require('ip');
   for (var item in addresses) { // iterate on every ip/mask found
      var ipaddr=addresses[item].address;
      var mask=addresses[item].netmask;
      var start=ip.toLong(ip.subnet(ipaddr,mask).firstAddress);
      var stop=ip.toLong(ip.subnet(ipaddr,mask).lastAddress);
      for (var n=start;n<=stop;n++) {
         var hostname=ip.fromLong(n).toString();
         if (connectServer(hostname) ) return true;
      }
   }
   // arriving here means server not found. Notify and return
   console.log("Cannot locate server. exiting");
   return false;
}
```

Event Messages Data Format

1- From client to server:

Send data to event system is done by mean of an AJAX call. Notice that client-only systems should not send events, as previous authentication is required and messages will be rejected

```
...
var startDate=Date.now(); // global variable initialized when client starts
...

/**
 * send events
 * @param {string} Subsystem sender's name
 * @param {integer} Session ID to send event to
 * @param {string} type Event Type
 * @param {object} data Event { key:value [, ... ] } data
 */
function client_putEvent(senderName,sessionID,eventType,eventData){
   var common= {   // setup default elements for this event
         'Operation':'putEvent',
         'Type':    eventType,
         'TimeStamp': Date.now() - startDate,
         'Source':  senderName,
         'Session': sesionID,
         'Value':   0 // may be overriden with 'data' contents
   };
   // send "putEvent" command to server
   $.ajax({
      type:'GET',
      url:"/agility/server/database/eventFunctions.php",
      dataType:'json',
      data: $.extend({},common,eventData)
   });
}

...
```

example: send “NoPresentado” to server. Notice that real code, automagically inserts additional information on dog, contest, journey and so id's.

```
...
function notPresent() {
      tablet_putEvent(
         'tablet0',
         2,
         'datos',
         {
            'NoPresentado':  1,
            'Faltas'      :  0,
            'Tocados'     :  0,
            'Rehuses'     :  0,
            'Tiempo'      :  0,
            'TIntermedio' :  0,
            'Eliminado'   :  0
         }

      );
}
```

2- From server to client:

As above use an ajax call to request events from server. Notice that you can receive more than 1 event, so need to iterate over all of events received

Notice that every received event has an string 'Data' that contains JSON representation of event content. So need to be json parsed.

Also Ajax call must include as parameter last EventID and last timestamp values received from server, to allow receive only new events

```
/**
* wait for events to be received
* On timeout or error trigger this function again
*@param {integer} evtID last received event ID
*@param {integer} timestamp of last received event, as received from server.
*       In first call use value of 0
*@param {function} function(id,type,data) to call on each event received
*/
function waitForEvents(sesID,evtID,timestamp,callback){
   function handleSuccess(data,status,jqXHR){
      var timestamp= data.TimeStamp;
      var lastID=evtID;
      for (var n=0;n<parseInt(data.total);n++) {
         var row=data.rows[n];
         lastID=row['ID']; // store last evt id
         callback(lastID, row['Type'], JSON.parse(row['Data']) );
      }
      // re-queue event
      setTimeout(
         function(){waitForEvents(sesID,lastID,timestamp,callback);}
         ,1000
      );
   }
   function handleError(data,status,jqXHR) {
      // notify error and retray
      console.log(status);
      setTimeout(
        function(){ waitForEvents(sesID,evtID,timestamp,callback);}
        ,5000
      );
   }
   $.ajax({
      type: "GET",
      url: "/agility/server/database/eventFunctions.php",
      data: {
         'Operation' : 'getEvents',
         'ID'      : evtID,
         'Session'  : sesID,
         'TimeStamp' : timestamp
      },
      dataType: 'json',
      async: true,
      cache: false,
      success: handleSuccess,
      error: handleError
   });
}
```

Additional parameters for specific events

Every received event have a 'Data' section that includes some specific information. Due to historical reasons and backward compatibility, field names may differ when sending event data to server and when event is received in the client. When this is the case, this document state it.

Common numeric information. When not needed set to 0

Client-to-server	*Server-to-client*	*Meaning*
'Prueba':	'Pru'	ID of current contest,
'Jornada':	'Jor'	ID of current jornada,
'Manga':	'Mng'	ID of current Round,
'Tanda':	'Tnd'	ID of current Series,
'Perro':	'Dog'	ID of current Dog,
'Dorsal':	'Drs'	Dorsal number of current dog,
'Equipo':	'Eqp'	ID of current team,
'Celo':	'Hot'	0 or 1 depending of Dog Heat status (on female dogs),
'Value':	'Value'	E*xtra value, usage depends of event type*

Open and Info Events specific data:

These parameter offer string to identify contest, journey and rond. Data can be used for external OSD systems

'NombrePrueba':	**Current contest's name**
'NombreJornada':	**Name of running journey**
'NombreManga':	**Name of active round**
'NombreRing':	**Name of selected ring/session**

'llamada' (Call To ring) specific data:

These parameters offers textual information on current dog in ring. Data can be used in external OSD (On Screen Display) system

'Numero':	Order number in call to ring queue (first:1)
'Nombre':	Dog name
'NombreLargo':	Dog Long name (Pedigree name)
'NombreGuia':	Handler's name
'NombreClub':	Club/Country name
'NombreEquipo':	Team name
'Categoria':	Dog category
'Grado':	Dog grade

Timing (manual and electronic chrono) events

Every timing event, either from electronic or manual chronometer includes a timestamp milliseconds mark in field **'Value'**
To evaluate final resulting course time, user must substract “stop” - “start” values and trunc (not round) to cents if needed. This rule also aplies to intermediate time.
In AgilityContest, every “start” event fires up a local timer, that finishes when “stop” event is received; then timer info is replaced by result of stop-start value

CourseWalk event:

'crono_rec' event includes two fields: **'Value'** with current time mark in miliseconds and **'start',**

that contains the number of seconds of course walk (i.e: 420 for 7 minutes)

Manual-to-Auto transition ('crono_resync') event

This event provides 'start' with timestamp for starting automatic chrono; and 'stop' with tiomestamp of when manual chrono stops. *Usage is discouraged* due to problems with synchronization betwen local and remote timers, and variable time response to events on each subsystem. It's only held for historic reasons

Data ('datos' and 'chrono_dat') events

This events provide information on faults, refusals, touchs, eliminated, notpresented information on current dog at ring.
If a variable is not defined, their value must remain unchanged
If a variable is defined, and set to -1 their value must remain also unchanged
Otherwise, their value is the one to be set in client subsystem
In 'Eliminado' and 'NoPresentado' fields, 0 means 'false' and nonzero means 'true'

Client-to-Server	*Serve-to-Client Data*	*meaning*
'NoPresentado':	'Npr'	0: dog IS present at call to ring; 1: dog IS NOT present
'Eliminado':	'Eli'	0: dog IS NOT eliminated at course run; 1 dog IS eliminated
'Faltas':	'Flt'	Number of faults; -1 means no change
'Rehuses':	'Reh'	Number of refusals; -1 means no change
'Tocados':	'Toc'	Number of handler touch: -1 means no change
'Tiempo':	'Tim'	Final course time as to be stored into results data base
'Tintermedio':	'Int'	Intermediate time as to be stored into database

Notice that *'Tiempo'* and *'Tintermedio'* are independent of chrono timing events: these values are intended to be used when timers are not running (i.e. to introduce time manually via the assistant console, or at call to ring to show previous stored values) So when timers are running, programmer should ignore these fields

Course Walk data and timmings ('crono_rec') event

As usual in chrono events, Value contains a timestamp mark. There is an additional “start” field that indicates de duration (in seconds) for the course walk. A zero value for 'start' field means course walk end.

As other chrono events, user can evauate course walk duration (in msecs) by substracting end to start 'Value' fields

Sensor alignment error ('crono_err') event

Whenever the electronic chronometer detects sensor alignment error, it sends (only once) this message to server. When alignment is fixed, an event is sent again to indicate problem solved

The error status is provided by mean 'Value' field: nonzero value means sensor failure; zero means problem solved