Automobilista 2 - Dedicated Server

User Guide

The AMS2 dedicated server is currently in beta the information in this guide is considered WIP and subject to change. Only the Windows version of the dedicated server is available at this time

Contents

Downloading and Updating the Server

The dedicated server is available on Steam to all users, not just owners of the game. So you can download and run the server anywhere. Windows and Linux (64-bit only) versions are available, Steam will download the appropriate build for your current platform automatically.

Only a command-line version of the server is available, with simple a text file used to configure it. The executable is called DedicatedServerCmd.exe (on Windows) or DedicatedServerCmd.elf (on Linux). Downloading from the Steam Client You can download the server from the Steam's library. Go to your Library, then switch it from Games to Tools and search for Automobilista 2 - Dedicated Server. Then install it. Since the server is a command-line application, while you can run it from the library it won't show any nice configuration window. So first I recommend configuring it. Also it's strongly recommended to review the configuration whenever the server updates.

When downloaded from Steam Client, the client will automatically keep the server up to date as long as you did not disable automatic updates in the Steam UI. It works the same way as any other Steam library applications.

Downloading via SteamCMD

You don't need to have the Steam Client installed to be able to download and run the server. The alternative to full-blown Steam Client is the SteamCMD utility. Full SteamCMD documentation and download links are available here:
https://developer.valvesoftware.com/wiki/SteamCMD

The installation via SteamCMD will differ slightly depending on your platform and the exact version of the OS. Right now SteamCMD is 32-bit only, so on 64-bit Linux platforms you might have to perform extra steps. For example when I was installing it on our 64-bit Ubuntu-based testing server, the steps were:

- Install 32-bit gcc support: "apt-get install lib32gcc1"
- Create Steam user: "adduser steam". You will also probably want to edit /etc/sudoers to let select user switch to this account, and remove password from this user to prevent direct remote login: "passwd -l steam".
- Switch to the Steam account: "sudo su - steam"
- And install SteamCMD:
    - mkdir steamcmd
    - cd steamcmd
    - wget http://media.steampowered.com/installer/steamcmd_linux.tar.gz
    - tar -xvzf steamcmd_linux.tar.gz

Then to install the dedicated server via SteamCMD while running as the steam user:

- cd steamcmd

- ./steamcmd.sh
- login STEAM_USERNAME
- force_install_dir ./ams2
- app_update 413770 validate
- quit

Or you can just run it all on with a single command

- ./steamcmd.sh +login STEAM_USERNAME +force_install_dir ./ams2 +app_update 413770 validate +quit Of course the "force_install_dir" directory is entirely up to you.

Run the same command to update the server, it won't update automatically on its own.

Note that "login" step requires valid steam account info entitled to Project CARS 2 game license. It will ask for a password and Steam Guard code. Only 64-bit Linux binary is provided

Server Configuration

After you install the server, before running it, it's strongly recommended to customize the server's config. The configuration is read from file "server.cfg" from the current working directory when the server starts. You can specify an alternative location of the config file on the command line, using the "-c" (or "--conf" or "--config") option. For more information about the command line arguments, run the server with "-h" or "--help" argument.

Sample Configuration Files

By default there is no configuration file in that location. There is a subdirectory "config_sample" with two examples. It's strongly recommended to start with those. So usually you would start by copying "config_sample/server.cfg" to "./server.cfg" and then editing it.

There is also more advanced config sample available in "config_sample/server_with_lists.cfg", "config_sample/blacklist.cfg" and "config_sample/whitelist.cfg". Use those files if you want to use blacklists or whitelists on your server. Don't forget to rename "server_with_lists.cfg" to "server.cfg". Also make sure you edit the whitelist in that config - if there is any whitelist set, only users in that list will be able to connect to the server, and the sample config has just my ID in the list.

The options in the sample will usually match the default options baked into the executable. So if you don't want to customize some of them, feel free to completely

remove them from the config. The only exception are the sample blacklist and whitelist, those are empty by default but set to non-empty lists to provide examples.

Configuration File Format

The config's format is loosely based on JSON, but it's a bit more free-form and allows you to omit commas between individual options, and it's not required to have the whole config enclosed in top-level { } block. You can also use comments in the file, any text following "//" will be ignored.

The syntax to set an option's value is:

```
OPTION_NAME  : OPTION_VALUE
```

The option names are case sensitive.

Strings (so all option names for example) need to be double-quoted, values can be strings, integers or booleans ("true" or "false", without any quotes). In some cases values can also be lists or objects. The syntax for a list option value is:

```
OPTION_NAME  : [ OPTION_VALUE_1  , OPTION_VALUE_2…  ]
```

and the syntax for an object option value is:

```
OPTION_NAME  : { KEY_1  : VALUE_1  , KEY_2  : VALUE_2  … }
```

Useful Configuration Options

Some of the useful options you will want to change are:

- ***"name":*** The server's name. This will appear in session browser and will be also the default name of sessions hosted on the server.
- ***"password":*** The password required to create sessions on the server as well as to join the sessions. Password set in Create options is ignored when using a dedicated server.
- ***"maxPlayerCount":*** Maximum session size that can be created on the server. While the server supports this to be up to 64, the game supports only 32-player sessions maximum (+ 2 additional slots for Race Director and Broadcaster). Also see the information about the ***"GridSize"*** and ***"MaxPlayers"*** session attributes, documented in Session Setup Attributes.
- ***"enableHttpApi":*** Defaults to false in the sample config, and can be used to enable http access to the server.
- **"enableLuaApi":** Defaults to true in the sample config, and can be used to enable server scripting via Lua addons. The addons shipping with the server are documented in Server Addons. You can download more from the community, or write your own addons.

- ***"allowEmptyJoin":*** Defaults to true in the sample config, and this will make the server display in the session browser. You will most likely always want to keep this enabled.
- ***"controlGameSetup":*** Defaults to false in the sample config, which will make the server show as the "server icon without lock" in the server browser. That means that the first player who joins the server will be control of the lobby setup. If set to true, the server will control the setup instead. You should change this to true if you want to use the ***"sessionAttributes"*** config option or the ***"sms_rotate"*** Lua addon to control the server setup, or any other similar addon developed by you or the community.
- ***"sessionAttributes":*** Initial attributes of the session. These are applied to the first user joining the server, and if ***"controlGameSetup"*** is set to true that user will not be able to change those attributes. These can be further modified via HTTP API or Lua API. The individual attributes are documented in Session Setup Attributes.

There are more options available, all of the supported ones are included in the sample configs, including short documentation.

HTTP API Configuration

The server provides a HTTP API which enables third-party tools to access information about the server, the multiplayer session running on the server, details about players and vehicles active on the server, and also control the game's setup.

In the default config the HTTP API is disabled. If you want to use this API, change the value of the ***"enableHttpApi"*** option to true to enable the API. These additional options then control the API's builtin HTTP server:

- ***"httpApiLogLevel":*** The HTTP API logging verbosity - only messages of this level or higher will be logged. The global ***"logLevel"*** still applies, so this can't be more verbose than the server in general. Defaults to ***"warning"***.
- ***"httpApiInterface":*** Interface where the server binds its listening socket. This can be either an IP address in string format, or OS-specific name of the interface. It defaults to ***"127.0.0.01",*** which means that the HTTP API will be accessible only from the computer where the server is running locally, at http://127.0.0.1… If you want to make the API public, this needs to be changed (empty string should bind the API to the public interface).
- ***"httpApiPort":*** Port at which the server listens. Defaults to ***9000***, so by default the server is reachable only locally at http://127.0.0.1:9000. You will have to change this if ***9000*** conflicts with other services running on the server.
- ***"httpApiExtraHeaders":*** Extra headers to emit into the server's responses. Usually you can leave it as is.

Server also implements HTTP API access levels. Each HTTP API endpoint has an access level associated with it, and this access level has filtering rules defined. When a client tries to access a specific HTTP API endpoint, the filters will be processed and if an accepting filter matches the client's credentials, the access to the endpoint will be granted.

Each endpoint has a default access level defined, to one of ***"public"***, ***"private"***, ***"admin"***. Endpoints that return well-known information such as vehicle lists are ***"public"***, endpoints that return potentially more sensitive information about the current session are ***"private"*** and endpoints that modify the server's state are ***"admin"***.

As of build 72 these are the defaults:

- public: /api/help, /api/version, /api/list*
- private: /status, /api/log*, /api/session/status
- admin: /api/session/kick, /api/session/set_attributes, /api/session/set_next_attributes, /api/session/send_chat

These default access levels can be overridden by the config option ***"httpApiAccessLevels"***. This is a map from wildcard patterns to access level names, the first pattern that matches the endpoint of an query will be used. The pattern can contain the usual "?" and "*" wildcards, matching any character or a sequence of zero or more characters, respectively. There is also a special "%" wildcard which is similar to "*" but does not match a slash character.

These configuration options that control the HTTP API access filter rules:

- ***"httpApiAccessFilters":*** The master map from access level names to lists of rules for the levels. The sample config defines rules for the standard access levels ***"public"***, ***"private"*** and ***"admin"***, but you can add any other access level names and rules for them.

The default ***"public"*** rule just accepts all, the default ***"private"*** rule accepts access from 127.0.0.1 or from any users in the ***"private"*** group, and the default ***"admin"*** rule accepts access from 127.0.0.1 or from any users in the ***"admin"*** group.

- ***"httpApiUsers"***: Defines the HTTP API user lists, a map from user names to their passwords. The passwords are stored in plain text in the server config, so make sure the file itself is properly protected from external access.

- ***"httpApiGroups":*** Defines the HTTP API groups, a map from group names to lists of user names belonging to the group.

The server implements two basic kinds of filters. The IP filter can be used to restrict access to requests made from specific IP addresses or ranges, or reject access from specific IP ranges. This is used by the rule types ***"ip-accept"*** and ***"ip-reject".***

The server also supports the Basic HTTP access authentication. This is used by the rule types ***"user"*** and ***"group"*** - only clients who authenticate as specific users, or users belonging to a specific group, will be granted access. Please refer to the Wikipedia link above to understand how to authenticate as a specific user from the tool of your choice. When using the HTTP API from a browser you will usually be prompted to enter your username and password. Alternatively you can specify the user and password in the URL, for example this address would authenticate as the (commented-out) dave user in the default config:
http://dave:letmein@127.0.0.1:9000/api/session/set_attributes?...

Please be aware that the user name and password are sent in plain text when using the Basic access authentication.

Each filtering rule in the list is a JSON object with these properties:

- ***"type":*** The only required property which defines the type of the rule. It can be one of:
  - ***"accept":*** Accept this request, no additional checks.
  - ***"reject":*** Reject this request, no additional checks.
  - ***"reject-password":*** Reject this request and let the client know that a password is required, no additional checks.
  - ***"ip-accept":*** Accept this request if it matches the "ip" mask in CIDR notation (for example, "192.168.1.0/24")
  - ***"ip-reject":*** Reject this request if it matches the "ip" mask in CIDR notation (for example, "192.168.1.0/24")
  - ***"user":*** Accept this request if it authenticates as given "user".
  - ***"group":*** Accept this request if it authenticates as given "group".
- ***"ip":*** IP address mask for ip-accept and ip-reject rules.
- ***"user":*** User name for user rule.
- ***"group":*** Group name for group rule.

Running the Server

The simplest way to launch the server is to just run the DedicatedServerCmd executable. Note that running from Steam client is not supported due to how dedicated servers applications are set up on Steam internally.

It will probably be desirable to run it in a more controllable way on a dedicated server machine. Some options are documented here.

Running the Server in Background on Linux, Using Screen

There are many ways to run dedicated server in Linux environment. You could write an init.d script and then use the usual server/daemon commands to control it. We don't provide such scripts at the moment, as they tend to slightly differ from OS to OS.

One easy way to run a server via SSH without having to stay remotely logged in forever is using "screen". On Debian/Ubuntu-based OS you would use "apt-get install screen" to install it.

To run the server for the first time:

- SSH to the server
- sudo su - steam
- script /dev/null
- screen (then press enter to confirm the initial screen, read through it if you want to)
- The following commands will be run inside the screen
    - cd
    - cd steamcmd/ams2
    - ./DedicatedServerCmd 2>&1 | tee server.log
- Press ctrl-a followed by d to detach from the screen.
- Press ctrl-d to leave the screen, then few more times to leave the script/sudo/logout.

To reattach to the server later:

- SSH to the server
- sudo su – steam
- You can review server.log in /home/steam/steamcmd/ams2 ; follow these steps to reattach to the server:
    - script /dev/null
    - screen -d -r

Note that the "script /dev/null" command is a semi-ugly trick to make screen work in sudo sessions.

Server Addons

The server ships with several Lua addons. These server as samples for community developers, but they can also be used for basic server scripting as is. To enable or disable specific addons, just enter their name into the ***"luaApiAddons"*** list in the server configuration file. The sample config comes with ***"sms_motd"*** and ***"sms_rotate"*** addons enabled, and ***"sms_stats"*** addon present but disabled.

The addons are read from directory set by the config option ***"luaAddonRoot"***, which defaults to ***"lua"*** (relative to the current directory when running the server

Each addon stores its configuration file in ***"lua_config/ADDON_NAME_config.json"***. The ***"lua_config"*** directory can be changed to anything else by the config option ***"luaConfigRoot".*** Initially this directory is empty. Each addon ships with a default config file, and when the server starts with a new addon enabled for the first time it will automatically copy the default config file over to this location. So to customize a new addon, first add it to ***"luaApiAddons"***, then run the server once, stop it, and then edit the configuration file.

Each configuration file contains two entries:

- ***"version : 1"*** - never change this line, it allows the server to notice when an addon significantly changes the format of its configuration file. In that case, the server will backup the customized config, and copy the new default config over. It will notify when this happens on startup
- ***"config : …"*** - the actual config, with documentation comments as written by the addon author. Feel free to edit this.

The addons that ship with the game are:

Message of the Day: sms_motd

This addon monitors the session and sends messages to players who join the server, and optionally to all players after they return back to lobby when a race is finished. It's useful to always have this addon (or something similar) enabled so that players joining your server know the basic information about it.

The addon's config is stored in "lua_config/sms_motd_config.json" by default, and can contain these fields:

- ***"motd" :*** The message to print first. It defaults to an empty string, in which case the addon will use "Welcome to 'SERVER NAME'", where the server name is read from the server config.

- ***"send_setup" :*** The addon then sends information about the server's setup. This list controls which parts of the setup will be sent. You can leave it empty to send nothing, or disable/enable individual parts of the config:
    - ***"controls" :*** This will tell players which parts of the session setup are controlled by the server
    - ***"format" :*** Describes the race format (if enabled and how long are the practice/qualify/race rounds)
    - ***"restrictions" :*** Dumps information about various restrictions imposed by the setup (whether ABS is allowed and so on, it's quite long so you might have to keep this one disabled)
    - ***"weather" :*** Information about the weather setup
    - ***"date" :*** Starting date and time progression speed
- ***"send_when_returning_to_lobby":*** If set to true (the default), the addon will also send the above messages to all session members when they finish a race and return back to lobby. If set to false, it will send the information only to new joining players.

Session Setup Control and Setup Rotations: sms_rotate

This addon servers as an improvement on top of the ***"sessionAttributes"*** field from the master server config. It offers more user-friendly way to set the attributes, and also allows you to configure fixed rotation of session setups.

It works as follows:

- **Default Setup:** This tells the addon the basics of ***"sessionAttributes***" setup and servers as the based of all setups in the rotation.
- **Rotation Setups** (optional): When rotation is enabled, the server will start with the first setup. Then whenever a race is finished and everyone returns to the lobby, or when everyone leaves the server, it advances to the following setup. When the server is done with the last setup in the list, it will start from the first setup again. The setup applied to the game is created as a combination of the Default Setup, overridden by the active individual Rotation Setup.

The addon's config is stored in "lua_config/sms_rotate_config.json" by default, and can contain these fields:

- ***"persist_index"* :** If set to true (the default), the server will save the rotation index regularly, and continue from the next setup when the server is shut down - same as when everyone leaves the server and then a new session is created. If set to false, the server will just start from the first setup in the rotation every time it's restarted. You can also reset the server to the first setup by deleting the addon's persistent data, stored in "*lua_config/sms_rotate_data.json*" by default.

- ***"default" :*** The default server setup. The format of the setup is similar to ***"sessionAttributes"*** in "*server.cfg*", documented below.
- ***"rotation" :*** The array of rotation setups. If non-empty, the server will rotate these setups, each determined by taking the ***"default"*** setup, and then applying the individual rotation setup on top of it. Only attributes set in the ***"default"*** setup can be also used in rotation setups, with the exception of ***TrackId***, ***VehicleModelId*** and ***VehicleClassId*** (i.e. each setup in the rotation can control different parts of the setup)

Setup Format

The format of the default and rotation setups is very similar to ***"sessionAttributes"***, extended as documented in Extended Session Attributes. To be specific, ***TrackId***, ***VehicleModelId*** and ***VehicleClassId*** attributes can use track/vehicle/class names instead of the numeric identifiers, and the same applies to any "enum" or "flag" attributes.

In addition to that:

- If ***TrackId*** is specified: The track to enforce. Automatically sets the ***ServerControlsTrack*** to 1, otherwise it will be set to 0
- If ***VehicleModelId*** is specified: The vehicle to enforce. Automatically sets the ***ServerControlsVehicle*** attribute to 1 and sets ***FORCE_IDENTICAL_VEHICLES*** to ***Flags***. Otherwise the addon will set ***ServerControlsVehicle*** to 0 and clear the ***FORCE_IDENTICAL_VEHICLES*** flag.
- If ***VehicleClassId*** is specified: The class to enforce. Automatically sets ***ServerControlsVehicleClass*** to 1 and sets ***FORCE_SAME_VEHICLE_CLASS*** to ***Flags***, otherwise clears those. Or in other words, setting attributes ***TrackId***, ***VehicleModelId*** or ***VehicleClassId*** attributes will automatically enforce that track/model/class in the server setup. Never set the corresponding ***"ServerControlsXXX"*** attributes or the relevant "Flags" directly.

This might sound a bit confusing at first, but it should all be pretty straightforward once you review the default configuration and experiment with it a bit. In short:

- Take the ***"default"*** setup from the default configuration, and tweak it to match your preferred default server setup.
- Use the individual ***"rotation"*** setups to set the tracks that should rotate - in the most simple setup, it would really be just an array of setups with just the ***"TrackId"*** attribute, and the server would then rotate those tracks, using the ***"default"*** setup for all sessions. Then you can start experimenting a bit more, and add for example weather overrides for some tracks in the rotation, or add class or specific vehicle restrictions to some of them.

So the basic setup you should start with, rotating four tracks all with the same default setup, might look like this:

```
"persist_index" : true,
"default" : { … leave the default as is … },
"rotation" : [
      { "TrackId" : "Brands Hatch Indy" },
      { "TrackId" : "Interlagos" },
      { "TrackId" : "Londrina Long" },
      { "TrackId" : "Cadwell Park" },
]
```

Setup Handling Library: lib_rotate

Just as a side note for community addon developers, the logic of combining session setups and automatically enabling track/vehicle/class restrictions, is available in addon called ***"lib_rotate"***, which ***"sms_rotate"*** uses automatically. Feel free to use this class LibRotate provided by this addon in your own addons, or read through it to understand how working with attributes can be done from Lua. The ***"sms_rotate"*** addon then adds the rotation logic, handling appropriate server events to know when the next setup should be applied.

Persistent Statistics: sms_stats

This addon tracks and stores server and session statistics. The stats are saved in the addon's persistent storage regularly and can be accessed at *<lua_config>/sms_stats_data.json*. In the future the addon will prove a HTTP API endpoint to access the stats, but for now reading the JSON data is the only way to access the stats.

This addon has a simple configuration with three fields:

- ***"history_length":*** Length limit for the history. Oldest sessions will be removed from the history. Use 0 to completely disable session history tracking. Use negative number for infinite history. Defaults to 50.
- ***"track_events":*** Should the history track session events? The event list can be quite long so it's disabled by default. Enable this for extra stats tracking details.
- ***"track_results":*** Should the history track session results? Enabled by default.

The data is separated into four sections:

- ***"server":*** Information about the server itself.
- ***"session":*** Various session-related counters.
- ***"players":*** Stats tracked for individual players who played on the server.
- ***"history":*** History of all sessions played on the server.

Server stats

The server stats sub-table contains these fields:

- ***"name":*** Current server name.
- ***"uptime":*** Current server uptime, in seconds. Resets to zero when the server restarts.
- ***"total_uptime":*** Total server uptime, in seconds.
- ***"steam_disconnects":*** Number of time the server got disconnected from Steam.
- ***"steam_downtime":*** Number of seconds while disconnected from Steam. Resets to zero when the server restarts.
- ***"total_steam_downtime":*** Total number of seconds while disconnected from Steam.

Session stats

The session sub-table contains another sub-table counts and nothing else. The counts sub-table tracks various aggregate counters related to sessions that ran on the server:

- ***"sessions":*** Number of sessions started on the server.
- ***"lobbies":*** Number of lobbies started on the server. Unlike sessions, returning back after race counts as another lobby.
- ***"stage_counts":*** Number of times each stage has been started on the server. This is a sub-table with one numeric field for each stage type: ***"practice1"***, ***"practice2"***, ***"qualifying"***, ***"warmup"***, ***"race1"***.
- ***"stage_durations":*** Similar to the stage counts table, but stores total playtime durations in seconds of individual stages. This is actual time spent playing on the stage, so if a race is setup with stages but the session quits before the stage is reached, that stage will not be counted here. If a stage end early, only the time spent on it will count.
- ***"race_loads":*** Number of Lobby->Loading transitions.
- ***"race_loads_done":*** Number of Loading->Race transitions.
- ***"race_finishes":*** Number of fully finished race. If all session member leaves before the race completely ends, that session will not count here.
- ***"player_loads":*** Number of players during Lobby->Loading transitions.
- ***"player_loads_done":*** Number of players during Loading->Race transitions.
- ***"player_finishes":*** Number of players when the race fully finishes.
- ***"tracks":*** Map from track id to the number of times the track was active while loading.
- ***"track_distances":*** Map from track id to the number of meters travelled on the (aggregate over all player participants).
- ***"vehicles":*** Map from vehicle id to the number of times the vehicle was used while loading. Note: does not count players who JIP into the race yet.
- ***"vehicle_distances":*** Map from vehicle id to the number of meters travelled in the vehicle (updated each lap)

Player stats

The ***"players"*** sub-table is a map from player Steam IDs to structures with information about the players. Each player structure consists of:

- ***"name":*** Last known Steam profile name of the player.
- ***"last_joined":*** Unix UTC time in seconds of the last time the player joined the server.
- ***"counts":*** Sub-table with aggregate counter stats.

The counts table contains:

- "race_joins": Number of times the player joined a race on the server.
- "race_loads": Number of Lobby->Loading transitions.
- "race_loads_done": Number of Loading->Race transitions.
- "race_finishes": Number of fully finished races.
- "tracks": Map from track id to the number of times the player loaded onto this track. Note: does not count JIP yet.
- "track_distances": Map from track id to the number of meters the player travelled on this track.
- "vehicles": Map from vehicle id to the number of times the player loaded in this vehicle. Note: does not count JIP yet.
- "vehicle_distances": Map from vehicle id to the number of meters the player travelled in the vehicle.
- "qualify": Sub-table with counters for qualification results of the player:
    - "states": Map from state name at the end of qualification to counter for that state. So this sub-table says how many times the player properly finished a qualification stage vs how many times the player was disqualified etc.
    - "positions": Map from qualifying positions to counter for that position, i.e. qualification finishing position histogram.
    - "positions_per_size": Map from the number of players at the end of a qualification, to map from qualifying positions to counters. I.e. several qualification finishing position histograms, one for each session size the player joined.
- "race": Sub-table with counters for race results of the player. Same layout as the qualify table, but with stats for the race stage instead of the qualify stage.

History

The history sub-table is an array, with one entry for each race played on the server, from lobby to loading to several stages to loading back. The first entry is for the oldest race. The entries are structures with these contents:

- ***"index":*** Unique index assigned to the race. The first race to happen on the server will be assigned index of 1, increasing with each successive race. If history cleanup is enabled in the config and old races are removed from the history, the

races will not be reindexed, i.e. the oldest race will no longer have index equal to one.

- ***"start_time":*** Unix UTC timestamp when the lobby for the race was created.
- ***"end_time":*** Unix UTC timestamp when the game finished.
- ***"finished":*** True if the game fully finished and loaded back to lobby (starting a next race), false if the game finished prematurely because all players left the server before.
- ***"setup":*** Setup attributes of the game, recorded when it starts loading. All writable attributes are stored here.
- ***"members":*** Members of the session, map from ***refid*** to:
    - ***"index"***, ***"steamid"***, ***"name":*** Basic member details
    - ***"join_time":*** Time when the member joined the session.
    - ***"leave_time":*** Time when the member left the session.
    - ***"participantid"***: Id of the member's player participant.
    - ***"setup":*** Member's attributes ***VehicleId***, ***LiveryId***, ***RaceStatFlags***.
- ***"participants":*** Participants who were present during this session, map from participant id to structure with these attributes: ***RefId***, ***Name***, ***IsPlayer***, ***VehicleId***, ***LiveryId***.
- ***"stages":*** Map from stage name to structure with the stage details: ○
    - ***"start_time", "end_time":*** Start and end time of this stage.
    - ***"events":*** Array of participant events related to the stage. Each event has fields ***event_name***, ***time***, ***participantid*** and attributes with the event details, and fields ***name***, ***refid*** and ***is_player*** with useful participant info. Events that are recorded here are: ***Lap***, ***State***, ***Impact***, ***CutTrackStart***, ***CutTrackEnd***. The default config does not enable events tracking, so this array will always be empty unless the ***"track_events"*** boolean config flag is changed to true.
    - ***"results":*** Similar to the ***"events"*** table, with only the Results events stored. Each event has these fields: ***time***, ***participantid***, ***attributes*** with the event details, and ***name***, ***refid*** and ***is_player*** with additional participant info. The default config enables the results tracking, if you are not interested in this info you can change the ***"track_results***" boolean config flag to false to save some memory.

Session Setup Attributes

In several places in the server configuration, we mentioned "session attributes". To be specific, they are included in the server configuration file (the ***"sessionAttributes"*** entry), and they are used in the configuration of the ***"sms_rotate"*** addon.

Session attributes in general are bunch of data with information about the session setup, and state. There are also player attributes and participant (vehicle) attributes.

Here we will document only the session attributes relevant for the server setup - those control the game's settings. These attributes closely mirror the settings available in the game's lobby UI to the host player in standard peer to peer multiplayer sessions.

The available session setup attributes and their default values used in the sample config are :

- ***"ServerControlsTrack" :*** The host player can control track selection if set to 0. Set to 1 to disable track selection in the game, server then controls the track selection. See also ***"TrackId".***
- **"*ServerControlsVehicleClass*"**: The host player can change the vehicle class by going through the lobby vehicle selection screen if set to 0. Set to 1 to disallow players changing the class. See also ***"VehicleClassId"*** and ***"Flags"*** (***FORCE_SAME_VEHICLE_CLASS,1024***)
- **"*ServerControlsVehicle*"**: Players can change their vehicle if set to 0. Set to 1 to disallow players changing the vehicle. See also ***"VehicleModelId"*** and ***"Flags"*** (***FORCE_IDENTICAL_VEHICLES,2***)
- ***"GridSize"***: The size of the grid specifies the maximum number of vehicles in the session. The game will not allow players to select tracks that do not have enough grid positions. The behavior is undefined if you enforce track with not enough grid positions on the server, so don't do that.
- ***"MaxPlayers"*** : Maximum number of players that can join the session. It needs to be less than or equal to ***"GridSize"*** (which limits the number of vehicles) and the main server configuration field ***"maxPlayerCount"***. If this number is lower than ***"GridSize"***, the additional grid positions might be filled by AI vehicles depending on the ***"Flags"***.
- **"*PracticeLength*", "*QualifyLength*","*RaceLength*"** : The lengths of the individual race sessions. Practice and Qualifying are in minutes, while ***"RaceLengths"*** is in laps.
- **"*Flags*"** : This is the trickiest setting and it's strongly recommended to use the "sms_rotate" addon to configure the flags, unless you want to do some math. The flags number is computed by adding together values corresponding to various flags controlling the game's setup. These flags allow custom vehicle setups, allow any realistic assists, force same vehicle class if ***"ServerControlsVehicleClass"*** is also set to 1, fill session with AI cars if ***"GridSize"*** is larger than ***"MaxPlayers"***, and allow automatic engine starts. The list of all valid flag values is via the api route */api/list/flags*.
- **"*DamageType*"** , **"*TireWearType*"**, **"*FuelUsageType*"**, **"*PenaltiesType*"**, **"*AllowedViews*"**: The "enum" attributes. The numeric values correspond the various meanings, all documented in the */api/list/enums/* endpoint. In this case, the values mean - visual only damage, no tire wear, fuel usage off, penalties on, any camera views allowed
- **"*TrackId*"** : The numeric ID of the track set by the server. If ***"ServerControlsTrack"*** is set to 1, players will not be able to change the track away from this value. The list

of all tracks is available in the */api/list/tracks* endpoint, eg *-572148012* means *"Brands Hatch Indy"*

- **"*VehicleClassId*"** : The numeric ID of the vehicle class set by the server. If ***"ServerControlsVehicleClass"*** is set to 1 and the ***FORCE_SAME_VEHICLE_CLASS*** flag is set, players will be able to choose only vehicles from this class. The list of all classes is available at */api/list/vehicle_classes/*.
- **"*VehicleModelId*"** : The numeric ID of the vehicle set by the server. If ***"ServerControlsVehicle"*** is set to 1 and the ***FORCE_IDENTICAL_VEHICLES*** flag is set, players will not be able to choose different car. The list of all vehicles is available at the */api/list/vehicles/* endpoint.
- **"*RaceDateHour*"** : Starting time of the event sessions
- **"*RaceWeatherSlots*"** : **"*RaceWeatherSlot1*"**, **"*RaceWeatherSlot2*"**, **"*RaceWeatherSlot3*"**, **"*RaceWeatherSlot4*":** Up to four weather slots, and weather speed progression multiplier. The list of all weather types is available at */api/list/enums/weather*.

Note that any of the ***"ServerControls…"*** attributes require the global server setting ***"controlGameSetup"*** to be set to true, otherwise the server won't control the game at all, and those attributes won't be relevant.

Extended Session Attributes

When used in the context of Lua addons, the setup structure can use extended format. This is relevant in the ***"sms_rotate"*** configuration as well. The extended format allows you to use names instead of numeric identifiers for these attributes:

- ***TrackId***
- ***VehicleClassId***
- ***VehicleModelId***
- All "enum" attributes - ***DamageType***, ***TireWearType***, ***FuelUsageType***, ***PenaltiesType***, ***AllowedViews***, ***WeatherSlot1***...***WeatherSlot4***
- All "flag" attributes - ***Flags***

So for example in the ***"sms_rotate"*** configuration, you can set the track to Brands Hatch Indy like this:

```
{ "TrackId" : "Brands Hatch Indy" }
```

Instead of having to use the numeric value, which is required in the server config file:

```
{ "TrackId" : 1988984740 }
```

The class, model and enum attributes work in a similar way - you can just use the name instead of the id/value. The ***Flags*** attribute is a bit more complicated, as the string can combine many different flags into one value. The string to represent this is simply the names of the individual flags, separated by commas. No spaces are allowed between the flag names and commas. So for example:

```
{ "Flags" : FILL_SESSION_WITH_AI,ABS_ALLOWED,SC_ALLOWED,TCS_ALLOWED" }
```

You can even combine numeric values and string flag names by commas, and the number can combine individual flags same as when using the numeric value alone.

Session Attribute Lists

Many attributes use weirdly looking numbers for their values. Those numbers represent individual tracks, vehicles, and so on, depending on the context. As described above, in the extended attribute format you can use the corresponding names instead of the values.

You can query a running server for these lists. First enable the HTTP API in the config, and then if you leave the address and port to the default values:

- Start the server.
- On the same computer, open any web browser.
- Navigate to this address: http://127.0.0.1:9000/api/list/ ; this address will be different if you changed ***"httpApiInterface"*** or ***"httpApiPort"*** in the server config.