Project CARS Telemetry Logging and Analysis

v0.14

User Manual

This document is a work in progress, and is to be considered a partial draft. Last updated 1/1-2016.

Index

1.1 - Overview

Purpose

“Project CARS Telemetry logging and Analysis Application”, or pCARS Telemetry for short, is a companion application to Project CARS. It provides the following functions :

- Recording log files of Project CARS session telemetry data
- Visualtisation and analysis of real time and logged data
- Real time or log driven instrumentation
- Scripting for extension and customisation

The app is run either on the same computer which runs Project CARS, or it can be run on a remote computer using the provided network client/server functionality. When running on the same PC as the sim, using a second screen is handy, but you can also Alt+Tab between the sim and the app. A convenient setup may be running the telemetry app on a laptop, connecting to the PC running the sim over the network.

Manual organisation

This manual is organized in 3 sections decribing the main areas of the application:

1 - Introduction
2 - The various functional modules
3 - Scripting

Basic concepts

The main function of the app is to record and display sessions from pCARS. A "session" corresponds to an in game session, e.g. one Time Trial, one Qualification, one Race etc.

The app uses the concept of “A” and “B” sessions. The A session is the reference, and the B session is for comparison of laps from another session or the same session as A.

A and B sessions can be displayed separately, but the usefulness of the app is in the display of an A session only, or both A and B sessions, where the B session is displayed as an overlay on the A session for comparison of the data.

Data visualisation

The session data display is mainly provided by the Plotter(s) where the telemetry data is shown as graphs. Any number of plotters with any number and styles of graphs can be displayed. Each plotter displays its graphs on a common horisontal scale, by time or distance.

Instrumentation

Real time or logger data driven instruments can display any telemetry parameter in a number of ways. Full customisation is provided via scripting, and some example “dahsboards” are provided.

Scripting

The application can be extended and customised by scripting. The scripts are tiny programs which can analyse telemetry data, plot graphs, display instruments, create applets with their own GUIs etc.. The scripting engine can be used for anything from a calculator to complete applets that integrate with the main applications GUI.

1.1 - Overview

Main window

The main window of the application contains the various functional modules. Each module is a dockable window which can be moved, resized, closed/opened and dragged outside the main window. The window layouts can be saved and loaded to help keep the work area tidy and flexible.

Here is an example window layout with brief descriptions of the different parts:

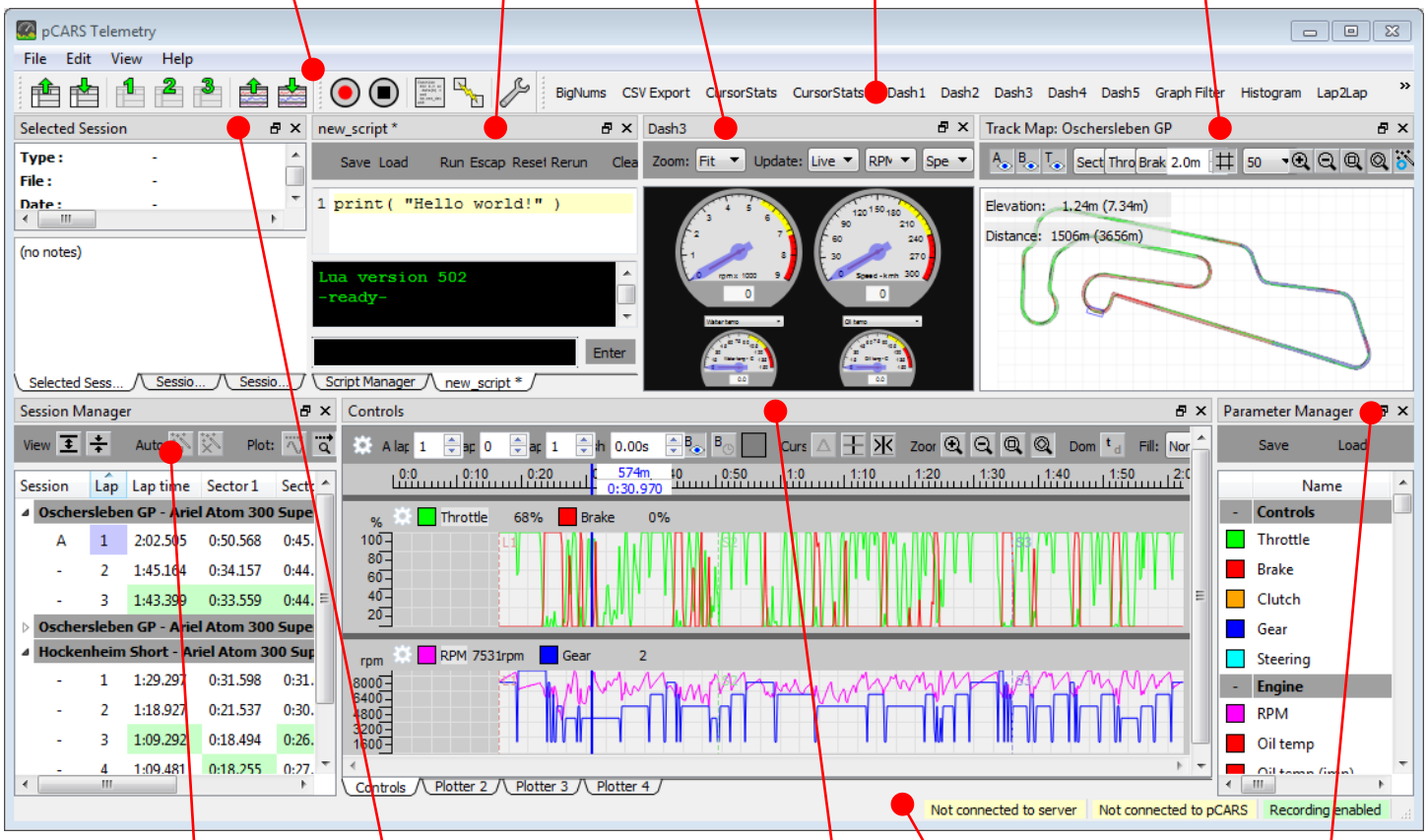
Main toolbar and menubar
Contains various buttons and menus for top level control.

Script console(s)
Used to manage and edit scripts.

Script toolbar
Contains buttons belonging to scripts.

Track map
Displays the track map and the driven lines.

Script (example)
An example of scripted instruments.



Status bar
Shows the connection and logger status.

Session Manager
The recorded session logs are managed here.

Session Info
Basic session info and notepad.

Plotter(s)
Displays the session data as graphs.

Parameter Manager
The telemetry parameters are managed here.

1.2 - Quick Start

Starting the application

When the application is launched for the first time, using the default configuration, it is ready to record Project CARS sessions running on the same PC. It will automatically attempt to connect to a local instance of Project CARS. For this to be possible you must make sure Shared Memory is enabled in Project CARS - you will find that on the “Visuals -> Hardware” menu page.

When the sim is connected, a green field in the status bar at the bottom of the main window will show “Connected to pCARS”. If recording is enabled another green fill will show “Recording enabled”. This means you are ready to go.

Recording a session

You can now drive a session and it will be recorded. You can see it in the session manager as soon as the session timing starts. As you drive the Session Manager will display the running sector and lap times as you progress, and it will highlight the fastest times. A session which is currently being recorded has a red heading.

When the session is completed, either by ending normally or by exiting or restarting, the heading will go yellow to indicate that the session is not yet saved. Any last incomplete lap will be removed auomatically - only the completed laps are kept.

Every session you drive will be recorded, unless you disable recording.

Saivng a recorded session

By opening the context menu for a session and selecting “Save session”, a save dialogue will be opened. It will initially be set to the configured default sessions direcory, but you can of course save sessions wherever you want.

pCARS Telemetry session files use a “.tld” extension by default.

Loading a previously recorded session

By opening the menu anywhere in the session manager, you can click “Load session” which opens the load dialogue. Select the session you want to load.

The session manager can hold as many sessions as you want, including new recordings and those you load in.

Viewing the recorded telemetry data

Double click on any session in the Session Manager to display that session in the plotters. If you expand the sessions so you can see each lap, you can double click on the lap you want to display. When a session is double clicked, it is set as the ”A” session, or “reference”, session. This will be indicated by an “A” to the left of trhe selected lap.

If you hold Shift while double clicking on a session heading or a lap, that session will be selected as the “B”, or “overlay”, session. This will be indicated by an “B” to the left of the selectd lap. When you have selected both an A and a B session the plotters will display overlaid graphs for both sessions. The A and B sessions can be selected from any sessions in the Session Manager.

On the toolbar of each plotter, you can control the lap range to view. Adjust the “A lap” field to select the starting lap for the A session, adjust the “B lap” field to select the starting lap for the B session and adjust the “N laps” field to adjust the number of laps. You can shift the B data in time by adjusting ”B shift” or dragging in the timeline with Alt held down.

By clicking the Time/Distance (t/d) button in a plotter’s toolbar, you can toggle between displaying the data along a time axis or a distance axis.

The lanes containing graphs can be dragged to size and manipulated in a number of useful ways. The plotters can be zoomed horizontally by using the magnifier buttons on the plotter toolbar, or by Ctrl + mouse wheel or by dragging in the timeline with Ctrl held down. Alt + mouse wheel zooms the first lane, and if you also hold down Shift you zoom all lanes.

When a session is set the Track Map will become active. It displays the driven path of the car on a map of the track (currently only a few tracks are mapped - you can map and edit more tracks yourself, but the official collection wil grow). The driven path and the map can be displayed in a number of ways and the map can be zoomed and panned.

There are Session Info modules for each of the set A and B sessions as well as for any session you select in the Session Manager. The Session info modules have a text editor where session notes can be added.

1.2 - Quick Start

Adding and removing telemetry parameters

By dragging one or more parameters from the Parameter Manager to any lane in any plotter, the parameter(s) will appear in the lane's toolbar and their graphs will be displayed. If you drag one or more parameters to the plotter's background, a new lane with those parameters will be added. If you hold down Alt while dragging, you will replace the parameters on the receiving lane. You can open a context menu over parameters on a lane's toolbar to change it or to remove it.

Cursors and the Track Map

In the plotter there are cursors which can be dragged to position. In the lane toolbars you will see the telemetry parameters values at the cursor(s) position. By default only the "main" cursor is enabled. Dragging the main cursor will update the parameter values on the lane toolbars and it will move the car marker on the track map.

There is also a pair of delta cursors for measuring the difference in values at the two points of the cursor pair. The lane toolbars will display the delta values as long as the delta cursors are selected. The delta cursors are also used to select an arbitrary range which scripts analyse.

Scripts

The main window Script Toolbar has buttons for each of the installed, or managed, scripts. You can click these buttons to open or close each script's "applet" window. Most scripts use docking windows which behave just like the built in modules. The managed scripts that are installed by default provide some statistics tools, some instrument panels and some other things.

You can write new scripts yourself, or modify the existing ones. Scripts can be installed so they are loaded on startup, or they can be run at any time in the Script Console(s).Scripts can be anything from a single one-liner ,
e.g. `print("Hello world!"),`
'to a complete applet which extends the app's functionality.

Connection Manager

The Connection Manager allows you to control how the app connects to Project CARS. You can connect locally if pCARS runs on the same PC as the app, or you can connect via the network by using the client/server functionality if the app and Project CARS run on different PCs.

Further exploration

The app has many menus and tooltips, so click the right button "everywhere" to discover the menus, and hover the mouse over the various parts of the app to see some tooltips.

Read more about the various functional modules elsewhere in this manual to get more in depth.

Useful tips

Adjust the configuration to suit you. Anything from colours to advanced setting are found there. Save and load GUI "themes".

Save your window and plotter layouts to keep your work tidy. Use the three window layout quick buttons to save and switch between window layouts on the fly.

Adjust the parameter colours, ranges and ther things to taste. Save and load parameter setups.

You can print out the plotters and lanes - look in the menus. You can also copy them to the clipboard to paste into other programs, e.g. for posting or documenting.

Use the auto save and auto remove features in the Session Manager to record and save multiple sessions while you concentrate on the driving.

Use progressive plotting and/or live data displays applets to act as "pit telemetry engineer" for your team mate.

Most of all, have fun displaying, recording, managing and analysing your telemetry data!

2.1 - Logger, Session Manager and Session Info

Logger

The logger is a central component in the app. It is responsible for gathering data from the sim, recording it and passing data to instruments and scripts. You don't see the logger as such, it is an internal service that is used by the various app modules.

The logger's functionality is represented in the GUI by:

- The record and stop buttons on the main toolbar
- The connection and logging status in the main window's status bar
- The Session Manager
- The Session Info modules
- The Plotters
- The Track Map
- The real time instruments
- Scripts

The logger attempts to maintain a connection with the sim according to the settings in the Connection Manager. The default action is to connect to the sim locally on the same PC as the app runs on. The connection manager can be used to select the means of connecting to the sim, either locally or over the network.

The logger can sample the data at a user selectable rate. This is set in the configuration. The default is a sampling period of 50 milli seconds, or 20Hz.

2.1 - Logger, Session Manager and Session Info

Session Manager

The Session Manager module is where recorded and loaded sessions are handled. It consists of a toolbar and a session list. The toolbar has buttons to control the Session Manager.

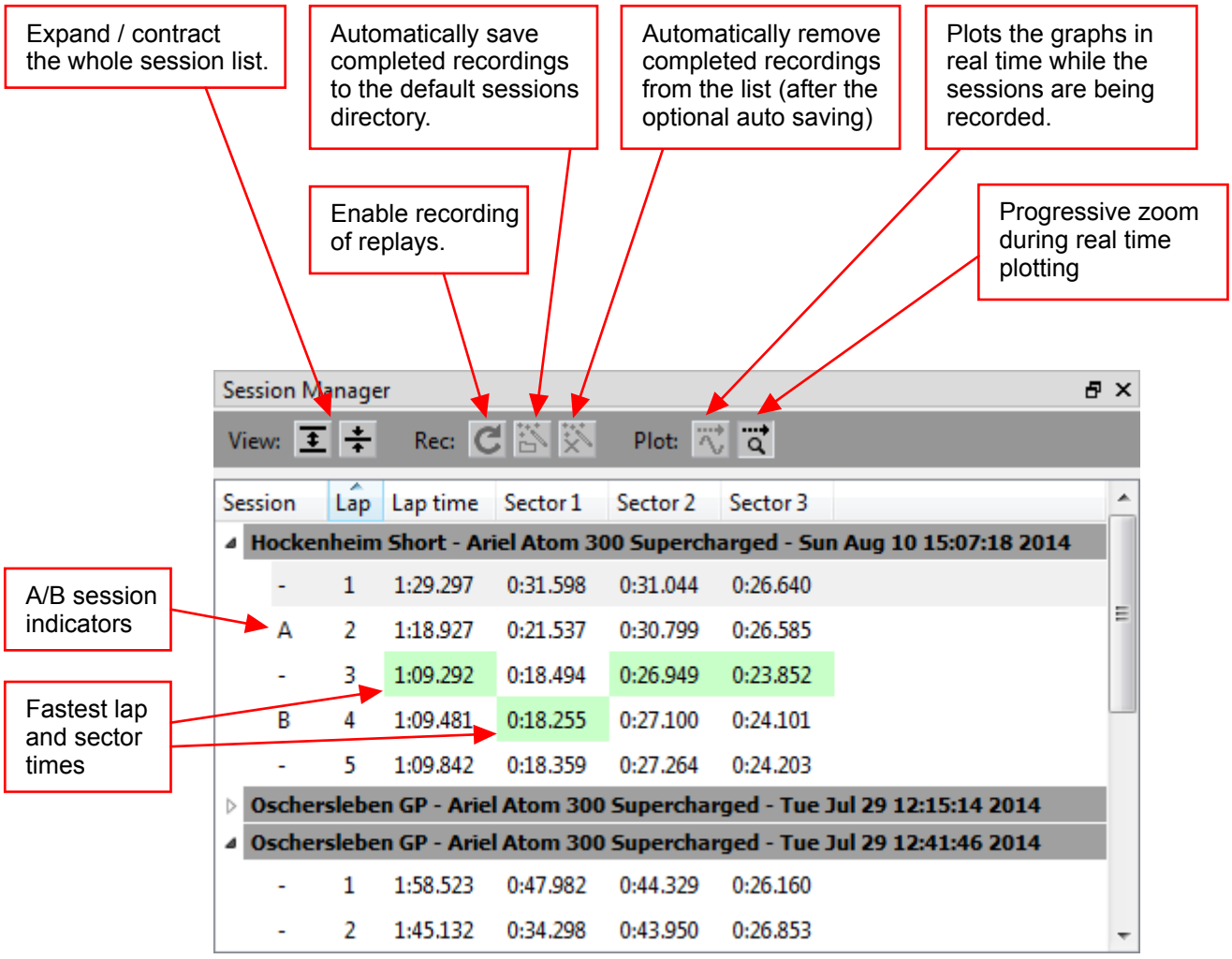
The session list contains a view of each session and each lap and sector time. The fastest lap and sectors in a session is highlighted in green. The leftmost column indicates session selection for analysis.

When a race, time trial, practice or online session is started in Project CARS, the telemetry data for the players car will be recorded (if recording is enabled). Session recordings appear in the session list as soon as the race timer starts. The session heading will be highlighted in red to indicate that recording is in progress. As the recording is in progress you'll see the laps and sectors being added and the live current lap and sector times will be shown.

When a session recording is completed, either by ending normally or by exiting or restarting, the heading will be highlighted in yellow to indicate that there is a new session that has not yet been saved. Any last incomplete lap will be removed auomatically - only the completed laps are kept. When a session is completed, the app is ready to record the next.

By double clicking on a session or a lap, that session and/or lap is set as the A session. If Shift is held when double clicking, the selection becomes the B session. The context sensitive menu can also be used to set the A and B sessions. The set session(s) are displayed in the plotters, the session info module and in the trackmap.

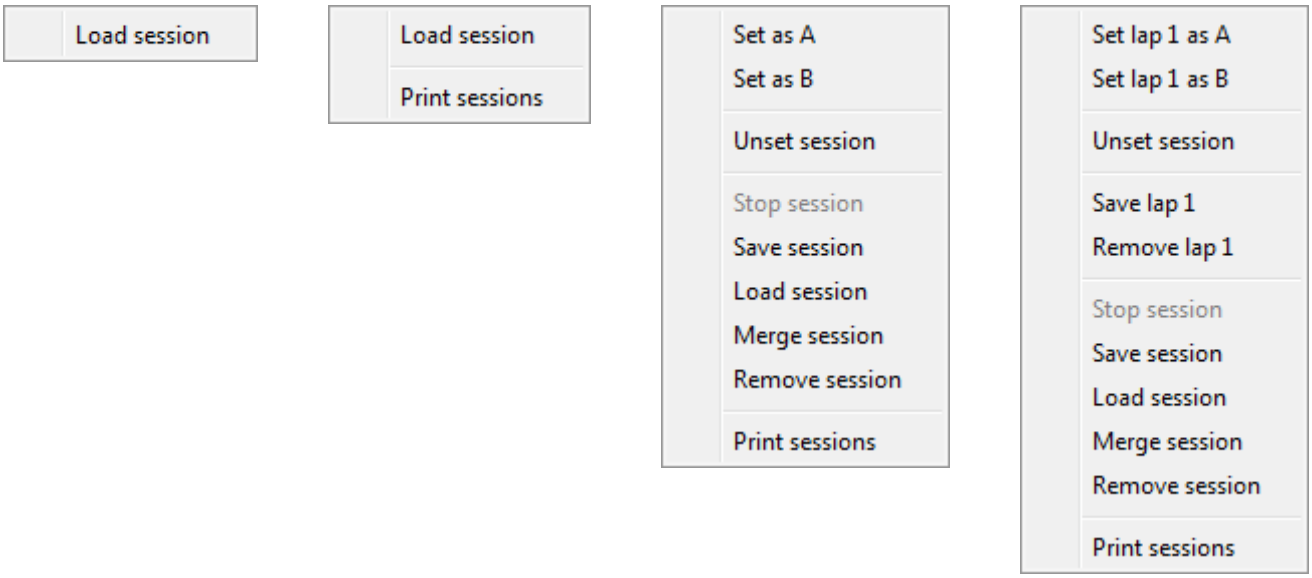
When a race, time trial, practice or online session is started in Project CARS, the telemetry data for the players car will be recorded (if recording is enabled). Recorded sessions wil start at lap 1 when the race timer starts and end with the last completed lap.



2.1 - Logger, Session Manager and Session Info

Session Manager menus

There are four variants of the session manager menu depending on the context. The first is for a completely empty Session manager, the second is for the window background, the third is for the session headers and the fourth is for laps.



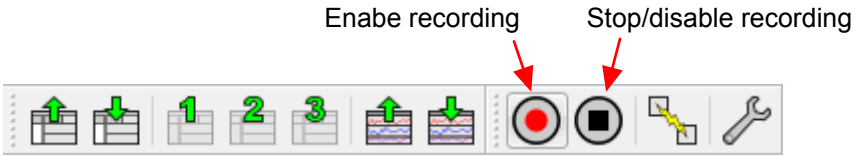
Session Info

There are three session info modules: the A session, the B session and any currently selected session in the Session Manager.

Each Session Info module displays some infromation about the session. There is also a text editor where you can type session notes which will be saved with the session. The session notes will also get annotations appended when sessionas are being edited by removing laps or merging in other sessions.

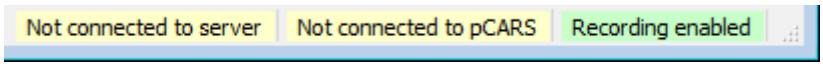
Recording controls

Each Project CARS sessions is recorded as long as recording is enabled. When recording is enabled the record button is red and the status bar shows “Recording enabled”. When a session is being recorded the status bar shows “Recording in progress”. Recording can be stopped and disabled by clicking the stop button. If a recording is in progress when Stop is clicked, it will retain all completed laps.



Status bar

The sim and server connection status and the recording status is displayed in the main window status bar:



2.4 - Trackmap

Overview

The Track Map displays the driven line(s) for the set session(s), and a symbol for the car. It can also display an actual map of the track - currently only a few maps are supplied with the app, but you can generate track maps yourself (ref. “Generating track maps” later in this chapter).

Display modes

You can choose display modes for the track itself from the leftmost combobox in the track map toolbar. These only take effect for tracks that have had an actual map generated. The track display modes are:

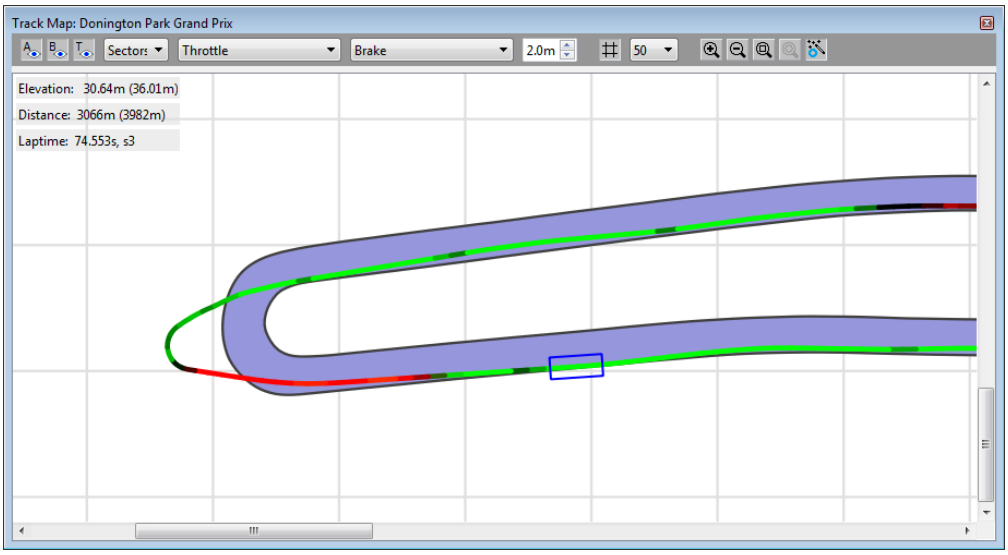
Uniform => flat grey track colour
Sectors => coloured by sectors, red = S1, green = S2 blue = S3 (the colours are configurable)
Height => coloured by height, black is the lowest point and white is the highest point
Banking => coloured by overall track banking, green banks inwards, red banks outwards
Combo => combines Height and Banking

For the driven lines, two parameters can be displayed. Use the 2nd and 3rd combobox in the track map toolbar. The first parameter will be displayed as a green tint, the second as red tint. The base colour is black. When both parameters have high values you will se a yellow tint (red and green mixed).

Uniform => shows a black line, or if both A and B sessions are selected, A is green and B is red.
Sectors => coloured by sectors, red = S1, green = S2 blue = S3
..... => selected parameter, base colour is black, parameter 1 colour is green and parameter 2 colour is red

You can select the width of the driven line with the spinbox, and you can switch the grid on and off as well as adjusting the distance between the grid lines.

In the example below, the map is shown with sector colours, and the driven line is shown with green for throttle and red for braking. You can also see how late braking before the Melbourne Hairpin on Donington leads to an excursion off track and into the gravel:



You can zoom the map to your liking, using either the buttons on the track map toolbar, or the mouse wheel. When the track is zoomed in, you can click and drag to pan it around. There are +/- buttons to zoom in and out, and a toggle button to automatically keep the map zoomed to the size of the window.

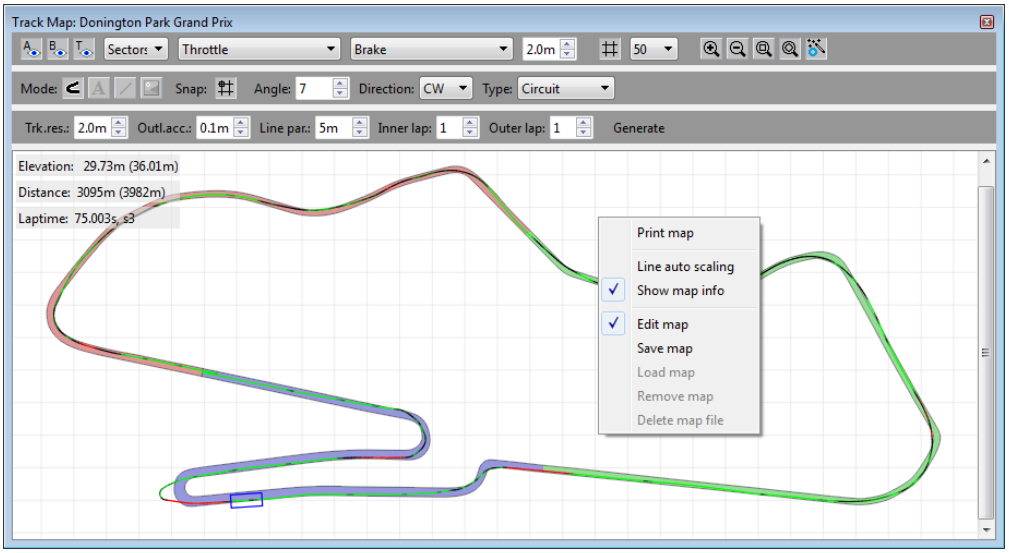
When the map is zoomed in, it can be made to automatically pan so that the car is kept in view when you move the cursors in the plotters. To enable this, click the rightmost button on the toolbar (the one with the magic wand).

In the context menu, you can toggle the map info shown in the upper left corner on and off, and you can print out the current view of the map. There is also an option, "Line auto scaling", to keep the driven line width constant independent of the map zoom level.

2.4 - Trackmap

Generating track maps

To generate a map, you need to drive one complete lap along the inner track boundary (the white line) and one complete lap along the outer track boundary. These drives need to be slow and careful since the map data is taken from your driven path. I like to use the Formula Rookie since it's easy to control and has a nice "iron sight". Since two wheels will be on the grass while driving the white line, lowering the FFB and maybe softening up the car may help with precise placement.
It should be possible to achieve an accuracy of 10 to 30 cm if the drive is very slow and smooth. When both the inner and outer track boundaries have been driven, it's time to save the session, select it as the A session and start the map editor to generate the actual map.



The complete procedure is:

1 - Record a session with one complete lap along the inner track boundary (the white line) and one complete lap along the outer track boundary. There needs to be at least one lap between the two where you cross over to the other boundary. I like to use the Formula rookie with almost no FFB, then I drive the 1st lap along the inner boundary, cross over to the other boundary on the 2nd lap (also an opportunity to relax and loosen up that death grip on the wheel from the deep concentration needed for an accurate boundary capture). Then the 3rd lap is driven along the outer boundary. It doesn't matter which laps you follow which boundary, that can be selected in the map generator. So if you're not satisfied with one or both of your first boundary captures, go ahead and drive them as many times as you like.

2 - Save the session. Perhaps adding a "_MAPRUN" suffix, e.g. "Donington Park Grand Prix - Formula Rookie - Mon Oct 07 23-50-50 2013_MAPRUN.tld". Perhaps even add something like "_MAPRUN_1_3", to indicate that the best inner and outer boundary captures was on laps 1 and 3. The session can be used to (re)generate the map at any time.

3 - Select the session as the A session in the Session Manager by double clicking it or using the context menu to select "Set as A".

4 - Open the context menu in the Track Map window and select "Edit map" to enter the track map editing mode. This brings up a 2nd toolbar for the editor.

5 - Click the leftmost button on the editor toolbar to select the map generator. This brings up a 3rd toolbar for the map generator tool.

6 - Select the desired resolution, this is the smallest segment length in meters, and your preferred inner and outer boundary-capture laps.

7 - Click the Generate button.

8 - Select map rotation in the editor toolbar, if you wish.

9 - Open the context menu in the Track Map window and select "Save map".

10 - Done!

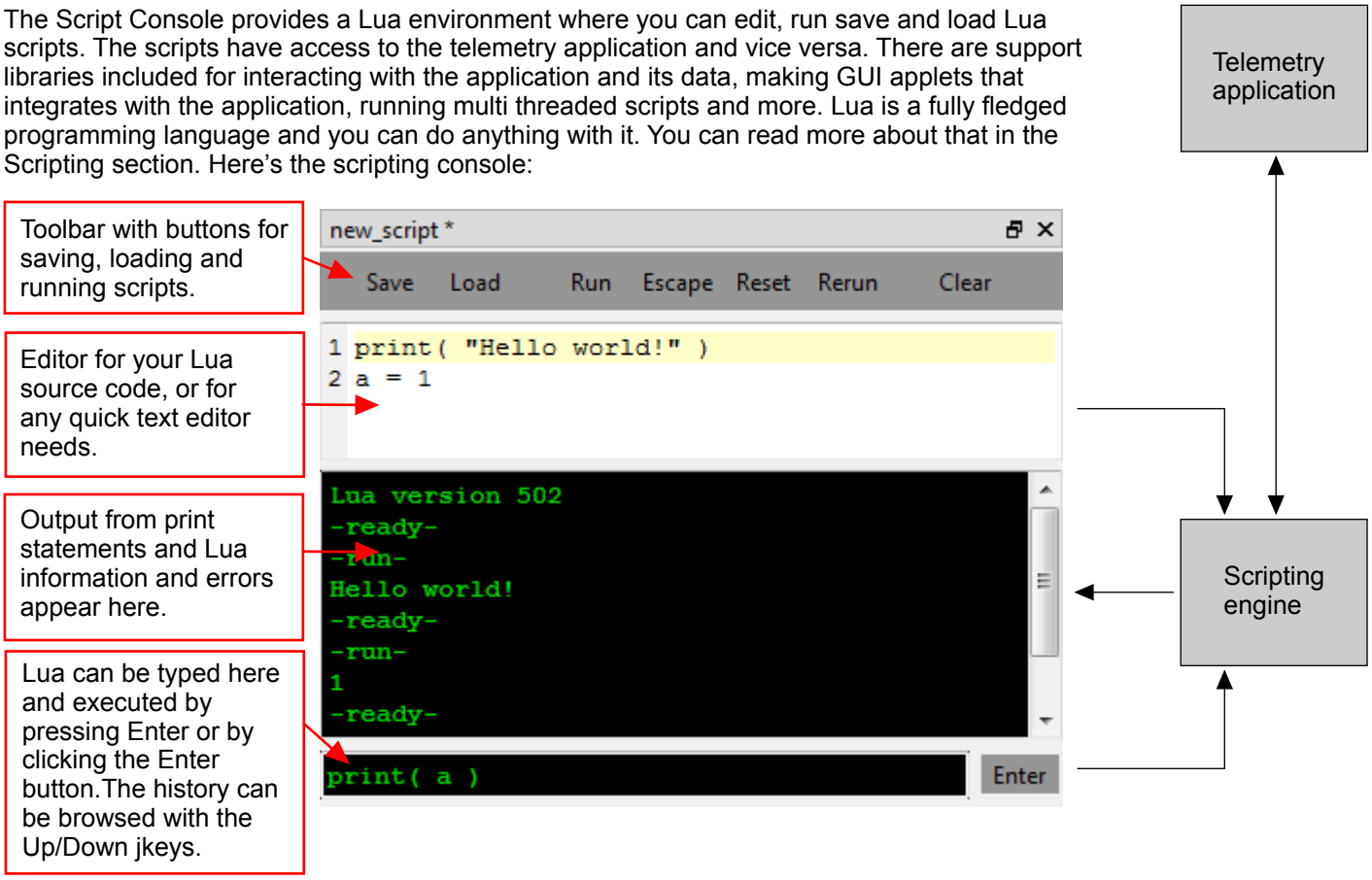
2.4 - Trackmap

You can disable the editor mode by opening the context menu and clicking "Edit map" again. The map can be regenerated and saved again at any time. Later versions may offer more editing facilities, such as placing text and lines on the map.

The map files live in the "Documents/pCARS Telemetry/maps" folder. If you want to remove a map, you can go there and delete it.

2.5 - Script Console

The Script Console provides a Lua environment where you can edit, run save and load Lua scripts. The scripts have access to the telemetry application and vice versa. There are support libraries included for interacting with the application and its data, making GUI applets that integrates with the application, running multi threaded scripts and more. Lua is a fully fledged programming language and you can do anything with it. You can read more about that in the Scripting section. Here's the scripting console:



The buttons in the Script Console Toolbar do the following:

Save: opens a save dialogue to pick a file where the editor's contents can be saved (shortcut Ctrl+S)

Load: opens a file dialogue for loading a new script, replacing the current one

Run: runs the script in the editor in the current Lua environment (shortcut Ctrl+R)

Escape: stops any running script (shortcut Escape)

Reset: stops any running scripts and resets the Lua environment (shortcut Ctrl+T)

Rerun: stops any running script, resets the Lua environment and runs the script in the editor (shortcut Alt+R)

Clear: clears the output console (shortcut Ctrl+L)

Enter: runs the Lua typed in the command line field, at the bottom of the console, in the current Lua environment

There can be any numbers of Script Consoles. This is configurable. Each is a completely independent Lua environment with its own instance of the scripting engine.

Keep in mind that if your script contains an infinite loop, or runs for very long, it will time out. The main thread, which is the global scope of the editor contents or command line contents, is run to completion. That is, it is not timesliced, so while it runs the application is blocked - better keep it short then, and do any length work in a timesliced thread instead.

New timesliced threads can be created simply with the thread.new() function. It's very easy!

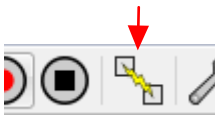
The timeout period is configurable and it can be set from the scripts on a per thread basis using the thread.set_timeout() function.

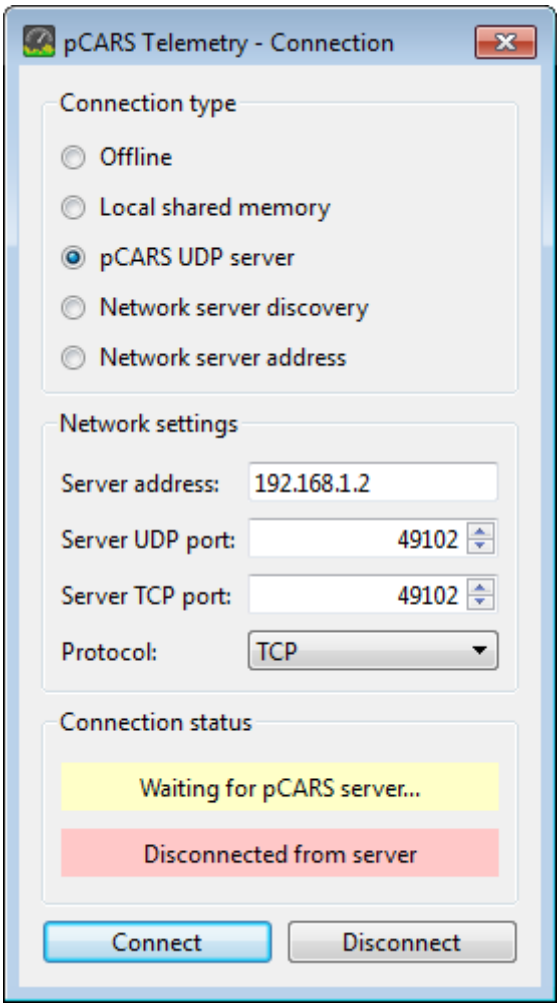
Read more about this in the Scripting section.

2.7 - Connection Manager and Server

Connection Manager

The Connection Manager lets you control how the connection to Project CARS is made. You can connect locally, to pCARS' built in UDP server, or by running the pCARS Telemetry Server you can connect over the LAN or the Internet. It can be opened by clicking the Connection Manager button in the main toolbar:





By default it is set up to connect to an instance of Project CARS that runs on the same PC as the app. I.e. a local connection. The connection type can be any of:

Offline	=> no connection is made
Local shared memory	=> a local connection is made
pCARS UDP server	=> a connection to a pCARS' built in UDP server on the LAN is made
Network server discovery	=> a connection to a server on the LAN is made
Network server address	=> a connection to a specific server on the LAN or the Internet is made

If "Local shared memory" is chosen, a connection to the a local instance of Project CARS will be made. Shared Memory must be enabled in the game options. The local connection will be automatically maintained.

If "pCARS UDP server" is chosen, a connection to an instance of Project CARS on the LAN will be made. The UDP server must be enabled in the game options.The LAN connection will be automatically maintained.

If "Network server discovery" is chosen, the Connection Manager will automatically search for, and connect with, the first server that is found on the LAN.

If "Network server address" is chosen, the IP address of the server must be typed into the "Server address" field. The ports can normally be left at their default values, just ensure that they match the ones set up in the Server.

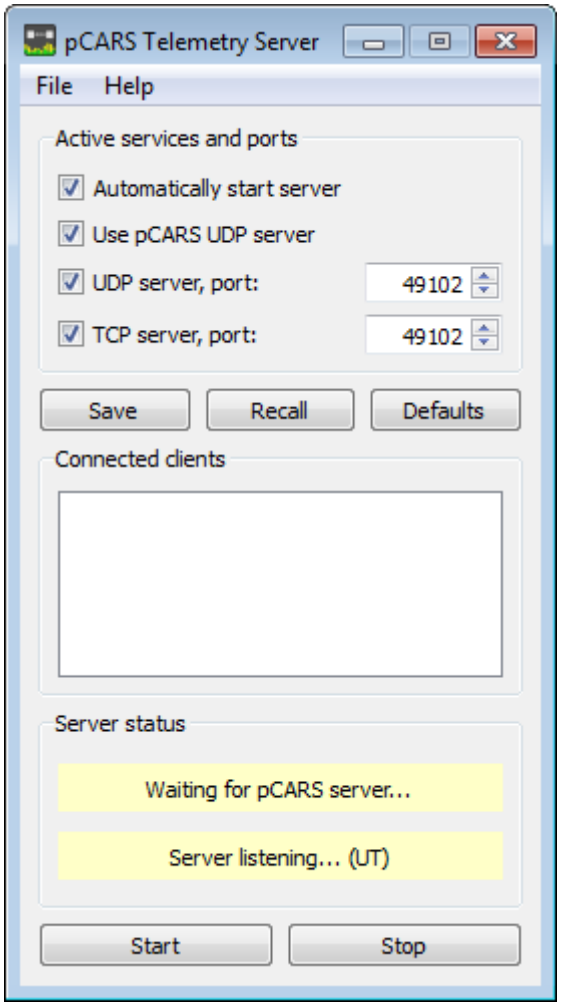
The "Connection status" shows the progress and status of the connection - green fields indicate the connected state.

2.7 - Connection Manager and Server

Server

The server allows you to make Project CARS telemetry data available over the LAN or the Internet. The server is a stand alone program. You can start it by double clicking its icon on the desktop, or by selecting it from the Windows programs menu.

The server can connect to pCARS, that is source it's telemetry data, via either the Shared Memory interface or the game's built in UDP broadcast server. To use the Shared Memory interface, the server must run on the same PC that pCARS runs on. To use the game's built in server, the game must run on a PC or console on the same LAN as the server, or on the same PC as the server. The server can be used to relay the game's built in server's data over a non-broadcast connection to one or more clients, including over the internet. This enables transmitting telemetry for consoles over the internet.



By default the server listens for incoming connection requests when it is started. In most scenarios you can leave all its settings at the deafults. You can alter the settings, save them and recall them, or recall the defaults.

If you want to use the game's built in UDP broadcast server as the data source rather that Shared Memory, tick the "Use pCARS UDP server" check box.

To allow incoming connections over the Internet, the TCP and/or UDP server ports must be opened in your router (using "port mapping" or similar), and possibly your firewall. You must refer to your router and/or firewall documentation for this.

It is recommended that Internet connections are made using TCP. This is the default setting in the Connection Manager.

The "Connection status" shows the progress and status of the server - green fields indicate the connected state.

The server supports multiple connections, so you can e.g. run pCARS Telemetry on your laptop over the LAN, while your race engineer also connects, either over the LAN or over the internet. Other apps can also potentially use the server if they're made compatible with it, but as of now, only pCARS Telemetry is compatible with this server.

3.1 - Scripting Engine

Overview:

The scripting engine uses the Lua programming language both for internal purposes and to provide the user with a programming environment which can interact with the application.

Scripts can be used for anything from a simple calculator, examining and manipulating telemetry data and writing applets that integrates seamlessly with the main application. Lua libraries are provided to cover the interaction with the application. The application itself uses Lua for the configuration files and the parameter definition files.

The application provides two means of executing scripts. The first is running scripts in the Script Console(s). There you can edit, save, load, and run scripts in an interactive environment. The second is the Script Manager. It loads and executes scripts at the application startup, or when commanded to later. This is used to provide pre installed scripts of the users choice.

The Lua language:

Lua is a very easy to learn and use language, while still being quite powerful. It is often used as an embedded language in engineering, industry, science, gaming etc..

Lua uses a Virtual Machine, a bit like Java does. When you run source code it is automatically translated to the internal bytecode and executed. Lua can also execute pre translated code, but there is little advantage in that for our application, where emphasis is on quick interactivity.

Start reading about Lua here:
http://www.lua.org
http://en.wikipedia.org/wiki/Lua_(programming_language)

There are plenty of tutorials on the web - here is the previous edition of "Programming in Lua" as a free e-book:
http://www.lua.org/pil/contents.html

A nice "What does it look like?" quick overview :
http://www.londonlua.org/wikimedia/

The application uses Lua version 5.2.2. It is a full version with all standard Lua libraries included.

Running scripts in the application:

The scripts are run in a "sandboxed" environment in the host application's GUI thread. When a script is run, it will first be compiled to bytecode by Lua (unless it has been pre-compiled) then it runs to completion in a non timesliced thread. We call this the **main** script thread. Since it is not timesliced, it will block the host applications GUI until it has run to its end. However, a thread library is provided so that timesliced threads may be run. The timeslicing of threads is preemptive from the scripts perspective, although operations in the non script domain will block the host application's GUI thread, e.g. an IO operation using the underlying OS.

The main thread itself should therefore not run for long, and certainly not contain any infinite loops which would render the host application's GUI unresponsive. A configurable timeout exists which will stop the execution of long running treads. Any long running processes should be started as new threads, and if required the timeout for the thread can be set.

All the standard Lua libraries are present, to handle maths, strings, io etc.. The scripts have further access to a set of application specific libraries to allow integration with the host application:

thread	=>	create and manage threads
lock	=>	create and manage thread synchronisation objects
tm	=>	telemetry API, for data retrieval and submission
gui	=>	GUI api, for easy integration with the host application

The scripts are event driven in the sense that they can define functions to handle various kinds of events. When combined with the thread facility and the supplied libraries, it is possible to write fully integrated GUI applets to extend the host application. It is of course also possible to run any kind of Lua programs, using the full language, the standard libraries and imported 3rd party Lua modules.

The script files are plain 8 bit text with an extension of “lua” for script sources and “luc” for precompiled scripts.

3.1 - Scripting Engine

Here's an example of a "real life" script which can be run in the script console. It will display some statistics for the session A telemetry data. Copy the scipt to the script editor and click run. Type `stat("Speed",1,2)` in the command line and hit Enter to run the statistics function starting with lap 1 and computing the statistics for 2 laps.

```
--
--  Example script for the pCARS Telemetry application.
--  This script computes the average, standard deviation,
--  minium and maximum for a telemetry parameter over a range of laps.
--

print("*** Parameter statistics ***")

-- The on_run() function is called when the main thread has completed.

function on_run()
  param_stats( "Speed" )
end

-- Convenience function for calling as a thread from the command line.
-- E.g. stat( "Lateral g", 1, -1 ) to display lateral g stats for all laps

function stat( name, lap_no, n_laps )
  thread.new( "param_stats", name, lap_no, n_laps )
end

-- Parameter statistics function.
-- The lap_no and n_laps arguments are optional.
--   lap_no => lap 1 if nil, else starting lap
--   n_laps => 1 lap if nil, number of laps if +ve, or all remaning laps if -1

function param_stats( name, lap_no, n_laps )
  local data, v_avg, v_dev, v_min, v_max, units, scaling
  v_min   = 999999
  v_max   = 0
  units   = tm.param_get_units( name )
  scaling = tm.param_get_scaling( name )
  data    = tm.get_a_data( name, lap_no, n_laps )

  v_avg = 0
  for k,v in ipairs( data ) do
    v = v * scaling
    -- print( v ) -- Uncomment to print all the values.
    v_avg = v_avg + v
    v_max = math.max( v_max, v )
    v_min = math.min( v_min, v )
  end
  v_avg = v_avg / #data

  v_dev = 0;
  for k,v in ipairs( data ) do
    v = v * scaling
    v_dev = v_dev + (v - v_avg) * (v - v_avg)
  end
  v_dev = math.sqrt( v_dev  / #data )

  print( string.format( "Statistics for %s:", name ) )
  print( string.format( "Number of samples  = %6d", #data ) )
  print( string.format( "Average            = %9.2f %s", v_avg, units ) )
  print( string.format( "Standard deviation = %9.2f %s", v_dev, units ) )
  print( string.format( "Minimum            = %9.2f %s", v_min, units ) )
  print( string.format( "Maximum            = %9.2f %s", v_max, units ) )
end
```

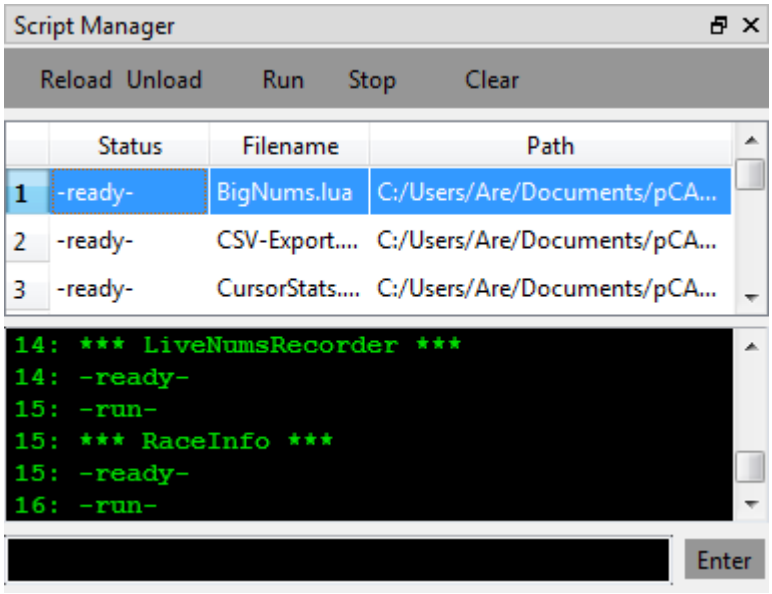
3.2 - Managed Scripts

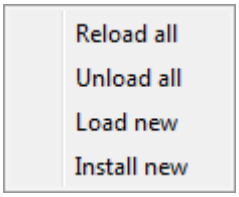
Overview:

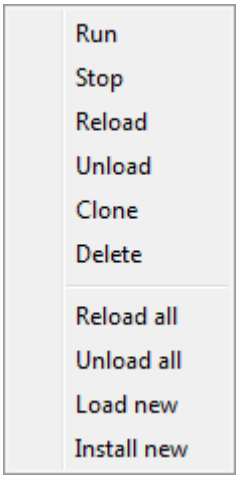
“Managed scripts” are normal Lua scripts which are loaded and run atuomatically when the application starts. They can be further managed at run time using the Script Manager module. Installation of a managed script simply requires that it’s copied into the managed script directory. By default this directory is:
“C:\Users\<username>\My Documents\pCARS Telemetry\managed_scripts”
The location of the managed script firectory is configurable.

As a simple means of uninstalling scripts temporarily, a “managed_scripts_x” directory is also provided. Move scripts between managed_scripts and managed_scripts_x to install and unistall them.

In order to integrate with the app, the scripts can provide event handlers so they can get notified of things happening in the app. By using the gui library and perhaps multi threading if needed, complete extensions of the app can be written.







3.3 - Event Handlers

Event handlers

Scripts can define event handlers, which are just normal functions, to be informed of events occuring in the app. Some events must be explicitly subscribed to by registering a handler function, others are using reserved names and only have to be defined in the script.

There are three classes of events being generated by the host application:

system events => uses predefined handler function names, called on a set of 'system' events
registered events => uses user specified handler function names, called on a set of general application events
script gui events => uses user specified handler function names, called on events from the scripts GUI objcts

Most of the event handlers take one argument, a Lua table. The event table represents name/value pairs and the exact contents depend on the type of event. All events have one field in common, a reason code, simply named "reason". Event handlers are not expected to return anything (any return vaues will be discarded).

A simple event handler which takes an event object might look like:

```
function my_event_handler( event )
  print( event.reason )
end
```

The system event handlers must follow the defined naming as follows (more details later):

on_run() => called after the script main thread has run
on_startup() => called immediately after application startup
on_shutdown() => called immediately before application shutdown
on_connection(event) => called when the logger connection state changes
on_session_changed(event) => called when the user has set or unset the A and B sessions
on_session_loaded(event) => called when a session has been loaded by the session manager
on_session_started(event) => called when a new session has started
on_session_completed(event) => called when a running session has completed
on_session_removed(event) => called when a session is about to be removed by the session manager

Of these, the on_run() handler is run as a timesliced thread. It can be viewed almost as a “main” function. All other handlers are run to completion without timeslicing. If any lengthy processing is to be done in an event handler, a new thread should be started to the work without blocking the app.

3.5.1 - Script library: thread

Overview

The thread library provides multithreading for the scripts. New threads can be created at any time and they will start running concurrently with any other threads. Threads can managed and synchronised using the thread and lock library functions provided.

At any time, a new thread can be created with the thread.new() function :

```
my_thread_id = thread.new( function [, arg1 [, arg2 [, ... ]]] )
```

`function` is the name of a function, and it must be visible from the global scope.
`arg1, arg2` etc. is an optional list of arguments that will be passed to `function`.
The thread starts executing by calling `function` with any given arguments.
The returned thread id can be used for thread management and synchronisation.

The following script creates a thread which gets passed three arguments:

```
function my_function( a, b, c )
  print( a, b, c )
end
thread.new( "my_function", 1, 2, 3 )
```

As mentioned in the scripting overview, a script's main thread runs to completion without timeslicing. We can start preemptive timesliced threads from the main thread though. They will start executing as soon as the main thread has run to completion. Other than this case, threads will start running once they have been created.

When a script's main thread is run, it creates a persistent global environment. This environment holds anything the script has defined, e.g. the my_counter() and on_run() functions in the examples below. New threads run in the context of this global environment, and anything they create which is not declared as 'local' will remain in the global environment until the script is stopped and the global environment is deleted.

Consider this example script:

```
function my_counter( count )
  for i = 1,count do
    print( i )
  end
end
thread.new( "my_counter", 1000 )
print("Hello!")
```

When this script is run, in the main thread, it defines the function "my_counter" and creates a new thread which will execute that function. As soon as the main thread has completed, right after the last line, the newly created thread will start to run concurrently with any other created threads. The output of the example is: “Hello!”, 1, 2, 3, ...

As a special feature of the thread system, a on_run() function can be defined, and this will be called as a timesliced thread as soon as the main thread has finished. You can think of this as similar to a main() function used in many languages, i.e. as an antry point for starting your program. The use of the on_run() function is idiomatic, it’s etirely optional and the same result can be acheived by explicitly starting a thread as shown in the example above.

So, here we achive the same result as in the example above, by calling my_counter() from on_run():

```
function my_counter( count )
  for i = 1,count do
    print( i )
  end
end
function on_run()
  my_counter( 1000 )
end
```

3.5.1 - Script library: thread

Function reference

thread.new(function [, arg1 [, arg2 [, ...]]])

Entry: function => name of function to run as a thread
arg1, ... => optional argument list
Exit: returns a thread id or -1 if the thread could not be created

A new thread is created and its id is returned. The thread starts executing immediately, unless created from the main thread, in which case it will start as soon as the main thread has run to completion.

Threads get garbage collected after they have run to completion or been killed explicitly or by an error.

Note that multiple threads may use the same function entry point - each instance will run independently of the others.

thread.id()

Entry: -
Exit: returns the calling thread's id

thread.kill([thread_id])

Entry: thread_id => id of thread to kill, or nil for the calling thread
Exit: retuirns 0 = killed OK
-1 = error, no such thread, no action taken

Kills the specified thread if it exists, or if no thread is specified the calling thread gets killed.

thread.kill_all()

Entry: -
Exit: -

Kills all threads including the calling one.

thread.unblock([thread_id])

Entry: thread_id => id of thread to unblock, or nil for the calling thread
Exit: returns 0 = killed OK
-1 = error, no such thread, no action taken

Unblocks all threads waiting for the specified thread.

thread.set_timeout(timeout [, thread_id])

Entry: timeout => timeout in milli seconds
thread_id => id of thread to set priority for, or nil for the calling thread
Exit: returns 0 = thread sleep OK
-1 = error, no such thread, no action taken

A thread's execution will be interrupted and stopped after the set timeout.
If timeout = 0, then the thread will not time out.
The default timeout is configurable.

3.5.1 - Script library: thread

thread.set_priority(proprity [, thread_id])

Entry: priority => CPU priority for the given thread
thread_id => id of thread to set priority for, or nil for the calling thread
Exit: returns 0 = thread sleep OK
-1 = error, no such thread, no action taken

The default prority is 1.0. Setting priority to 0.5 halves the allocated CPU time, setting priority to 2.0 doubles the allocated CPU time etc.. Overly high priorities may affect the responsivness of the main application.

thread.sleep(time [, thread_id])

Entry: time => time in milliseconds
thread_id => id of thread to sleep, or nil for calling thread
Exit: returns 0 = thread sleep OK
-1 = error, no such thread, no action taken

The execution of the calling thread is suspended for the specified time.
If time = 0, the thread will be suspended for one VM timeslice

thread.wait(thread_id)

Entry: thread_id => id of thread to wait for
Exit: returns 0 = thread waited OK
1 = nothing to wait for
-1 = error, no such thread or not applicable thread, no action taken

The calling thread waits for the specified thread to finish before resuming again. If the specified thread doesn't exist, the calling thread continues to run.

thread.suspend([thread_id])

Entry: thread_id => id of thread to suspend, or nil for the calling thread
Exit: returns 0 = thread suspended OK
1 = thread is alreday be suspended
-1 = error, no such thread, no action taken

The execution of the given thread is suspended until resume() is called.
A thread can suspend itself, but it can not resume itself.

thread.resume(thread_id)

Entry: thread_id => id of thread to resume
Exit: returns 0 = thread resumed OK
1 = thread was not suspended,
-1 = error, no such thread, no action taken

The execution of the given thread is resumed.
A thread can not resume itself.

3.5.2 - Script library: lock

Overview

The lock library provides a simple mutual exclusion mechanism for use in thread synchronisation. This kind of synchronisation is often used to protect access to data which is shared between threads.

A lock can be acquired by only one thread at a time. The acquire() function makes sure that of all threads calling it, only one gets past it at a time. That thread then owns the lock, and all other threads calling acquire() with the same lock will be kept waiting. When our thread calls release(), the lock becomes free and the next thread in line trying to acquire it will now succeed.

Here are some examples. First creating a new lock object:

```
my_lock = lock.new()

-- my_lock is now available for use
```

Acquiring a lock:

```
lock.acquire( my_lock )

-- my_lock is now acquired, either immediately if the lock was free,
-- or after waiting for it to become released by another thread first
-- or it could have be a non exisiting lock
-- ...now do stuff protected by the lock acquisition...
```

Releasing a previoulsy acquired lock:

```
lock.release( my_lock )

-- my_lock is now released and available for acquisition by other threads
```

Deleting a lock - locks can be deleted at any time, unblocking all threads that are waiting for it:

```
lock.delete( my_lock )

-- my_lock is deleted and will no longer block on calls to acquire()
```

If the lock library functions are called with a non exisiting lock id, no action will be taken. In this case the return code is -1, signifying that the lock id did not exist. A return code of 0 means that the operation completed successfully.

Calling acquire() and release() from the main thread or from coroutines will not work, but new() and delete() can be called.

3.5.2 - Script library: lock

Function reference

lock.new()

Entry: -
Exit: returns a new lock id or -1 if the lock could not be created

A new lock object is created and its id is returned.

lock.delete(lock_id)

Entry: lock_id => id of lock object to delete
Exitt: returns 0 = deleted OK
-1 = error, no such lock, no action taken

The lock object is deleted and any waiting threads will be unblocked.

lock.acquire(lock_id)

Entry: lock_id => id of lock object to acquire
Exit: returns 0 = lock acquired OK
-1 = error, no such lock id, no action taken

The lock object is attempted acquired if its is free.
If not, the thread will be suspended until the lock is released or deleted.
If the lock was successfully acquired, other threads will be held waiting
if they attempt to acquire this lock, until the lock is released or deleted.

lock.release(lock_id)

Entry: lock_id => id of lock object to relase
Exit: returns 0 = lock released OK
1 = lock was not acquired by the calling thread, not released
-1 = error, no such lock id, no action taken

The lock object is released if it has previously been acquired by the calling thread.
The lock is free to acquired by other waiting threads after sucessful release.

4.1 - Acknowledgements

I would like to thank the following people for testing, proof reading and good advice:

Casey Ringley
Thomas Sikora
Jussi Karjalainen
Felix Schmidberger
Espen S. Andresen

The following software components are being used:

Qt, under LGPL license:

Lua, under MIT License:

Stackwalker, under BSD license:

4.2 - End user license agreement

pCARS Telemetry

Copyright (c) 2016 Are Leistad

*** END USER LICENCE AGREEMENT ***

IMPORTANT: PLEASE READ THIS LICENCE CAREFULLY BEFORE USING THIS SOFTWARE.

1. LICENCE
By receiving, opening the file package, and/or using pCARS Telemetry ("Software") containing this software, you agree that this End User User Licence Agreement (EULA) is a legally binding and valid contract and agree to be bound by it. You agree to abide by the intellectual property laws and all of the terms and conditions of this Agreement. Unless you have a different licence agreement signed by Are Leistad your use of pCARS Telemetry indicates your acceptance of this licence agreement and warranty. Subject to the terms of this Agreement, Are Leistad grants to you a limited, non-exclusive, non-transferable licence, without right to sub-licence, to use pCARS Telemetry in accordance with this Agreement and any other written agreement with Are Leistad. Are Leistad does not transfer the title of pCARS Telemetry to you; the licence granted to you is not a sale. This agreement is a binding legal agreement between Are Leistad and the purchasers or users of pCARS Telemetry. If you do not agree to be bound by this agreement, remove pCARS Telemetry from your computer now and, if applicable, promptly return to Are Leistad by mail any copies of pCARS Telemetry and related documentation and packaging in your possession.

2. DISTRIBUTION
pCARS Telemetry and the licence herein granted shall not be copied, shared, distributed, re-sold, offered for re-sale, transferred or sub-licensed in whole or in part except that you may make copies for your personal use. For information about redistribution of pCARS Telemetry contact Are Leistad.

3. USER AGREEMENT

3.1 Use
Your licence to use pCARS Telemetry allow you to use the software for personal purposes only. For information about Public and commercial use contact Are Leistad.

3.2 Use Restrictions
You shall use pCARS Telemetry in compliance with all applicable laws and not for any unlawful purpose. Without limiting the foregoing, use, display or distribution of pCARS Telemetry together with material that is pornographic, racist, vulgar, obscene, defamatory, libelous, abusive, promoting hatred, discriminating or displaying prejudice based on religion, ethnic heritage, race, sexual orientation or age is strictly prohibited. You may make one copies of pCARS Telemetry for personal or backup purposes. The assignment, sublicensing, networking, sale, or distribution of copies of pCARS Telemetry are strictly forbidden without the prior written consent of Are Leistad. It is a violation of this agreement to assign, sell, share, loan, rent or lease the use of pCARS Telemetry.

3.3 Copyright Restriction
This Software contains copyrighted material, trade secrets and other proprietary material. You shall not, and shall not attempt to, modify, reverse engineer, disassemble or decompile pCARS Telemetry. Nor can you create any derivative works or other works that are based upon or derived from pCARS Telemetry in whole or in part. Are Leistad's name, logo and graphics file that represents pCARS Telemetry shall not be used in any way to promote products developed with pCARS Telemetry. Are Leistad retains sole and exclusive ownership of all right, title and interest in and to pCARS Telemetry and all Intellectual Property rights relating thereto.Copyright law and international copyright treaty provisions protect all parts of pCARS Telemetry, products and services. No program, code, part, image, audio sample, or text may be copied or used in any way by the user except as intended within the bounds of the single user program. All rights not expressly granted hereunder are reserved for Are Leistad.

3.4 Limitation of Responsibility
You will indemnify, hold harmless, and defend Are Leistad, his employees, agents and distributors against any and all claims, proceedings, demand and costs resulting from or in any way connected with your use of Are Leistad's Software. In no event (including, without limitation, in the event of negligence) will Are Leistad, his employees, agents or distributors be liable for any consequential, incidental, indirect, special or punitive damages whatsoever (including, without limitation, damages for loss of profits, loss of use, business interruption, loss of information or data, or pecuniary loss), in connection with or arising out of or related to this Agreement, pCARS Telemetry or the use or inability to use pCARS Telemetry or the furnishing, performance or use of any other matters hereunder whether based upon contract, tort or any other theory including negligence.

3.5 Warranties
Except as expressly stated in writing, Are Leistad makes no representation or warranties in respect of this Software and expressly excludes all other warranties, expressed or implied, oral or written, including, without limitation, any implied warranties of merchantable quality or fitness for a particular purpose.

3.6 Governing Law
This Agreement shall be governed by the law of the Norway applicable therein. You hereby irrevocably attorn and submit to the non-exclusive jurisdiction of the courts of Norway therefrom. If any provision shall be considered unlawful, void or otherwise unenforceable, then that provision shall be deemed severable from this Licence and not affect the validity and enforceability of any other provisions.

3.7 Termination
Any failure to comply with the terms and conditions of this Agreement will result in automatic and immediate termination of this licence. Upon termination of this licence granted herein for any reason, you agree to immediately cease use of pCARS Telemetry and destroy all copies of pCARS Telemetry supplied under this Agreement. The financial obligations incurred by you shall survive the expiration or termination of this licence.

4. DISCLAIMER OF WARRANTY
THIS SOFTWARE AND THE ACCOMPANYING FILES ARE PROVIDED "AS IS" AND WITHOUT WARRANTIES AS TO PERFORMANCE OR MERCHANTABILITY OR ANY OTHER WARRANTIES WHETHER EXPRESSED OR IMPLIED. THIS DISCLAIMER CONCERNS ALL FILES GENERATED AND EDITED BY pCARS Telemetry AS WELL.