Virtual Racing

Building A Formula Student Racing Car Simulator

MECH5080M Team Project – Individual Report
Virtual Racing - Building A Formula Student Racing Car Simulator
Author: Mathew Fuller - 201289335
Supervisor: Andrew Jackson
Industrial Mentor: Krzysztof Kubiak
Examiner: Andrew Jackson
Date: 15/05/2024

UNIVERSITY OF LEEDS

SCHOOL OF MECHANICAL ENGINEERING

MECH5080M TEAM PROJECT 45 credits

TITLE OF PROJECT

Virtual Racing - Building A Formula Student Racing Car Simulator

PRESENTED BY

Alex Bury, Mathew Fuller, Curtis Lovett, Qianli Liang, Jordan Partridge

OBJECTIVES OF PROJECT

To design and manufacture a portable and cost-effective simulator of the Formula Student Race Car for driver training and outreach activities. This included developing an actuated cockpit environment, a virtual environment/game engine to simulate driving conditions, auditory and haptic feedback, and force feedback steering systems.

IF THE PROJECT IS INDUSTRIALLY LINKED TICK THIS BOX AND PROVIDE DETAILS BELOW

COMPANY NAME AND ADDRESS:

Formula Student Team, University of Leeds

INDUSTRIAL MENTOR: Krzysztof Kubiak

THIS PROJECT REPORT PRESENTS OUR OWN WORK AND DOES NOT CONTAIN ANY UNACKNOWLEDGED WORK FROM ANY OTHER SOURCES.

SIGNED: Mathew Fuller -  DATE: 14/05/2024

Page | ii

ACKNOWLEDGEMENT

I wanted to thank my friends, family, and all those you supported me through my university career amid all the turmoil, sometimes mounting on Shakespearean tragedy levels. To my parents, thanks you for the unwavering support, I hope I've made you proud. A special thanks to my supervisor, Andrew Jackson, for aiding myself and my team in accomplishing this feat, and a massive thanks towards the rest of Group 135, who I was privileged to work with.

P a g e | **iii**

CONTENTS

P a g e | iv

ABSTRACT

The Formula Student racing simulator project aims to develop a comprehensive simulation environment for educational and research purposes, leveraging state-of-the-art simulation technology, custom telemetry plugins, and real-time actuation control systems. The project utilizes rFactor 2 as the simulation platform, chosen for its advanced physics engine, extensive content flexibility, and multiplayer capabilities. Custom telemetry plugins are developed to extract, process, and transmit real-time telemetry data from rFactor 2 to the actuation control system, enabling responsive and accurate control of the virtual simulator based on telemetry data feedback. The integration of virtual reality (VR) technology is explored to enhance immersion and realism, while the implementation of Reliable User Datagram Protocol (RUDP) ensures robust and reliable data transmission. Future work includes incorporating additional plugin features such as custom messages and data formats, further enhancing the simulation environment's functionality and usability. Through the Formula Student racing simulator project, we aim to provide a dynamic and immersive platform for driver training, vehicle performance analysis, and academic research in automotive engineering and motorsport.

Page | i

1 INTRODUCTION

Leeds Gryphon Racing (LGR) are the Formula Student (FS) entrant from the University of Leeds. LGR takes part in FS, which is an international competition organised by the Institution of Mechanical Engineers (IMechE), where single-seat race cars, designed and manufactured by student groups taking part such as LGR, compete against each other in multiple dynamics and static events at the annual Silverstone competition [1].

This report outlines the design and creation of a portable, low-cost simulator student racing simulator devised to fulfil two needs that LGR identified. These needs are to combat the lack of driver experience, brought about due to current training having a high running cost while also low-availability due to quick development cycles, and also aid to attract new members and sponsors at outreach events. LGR deemed that a racing simulator would be a suitable solution for both.

The gamification of professional activities, such as training and education [2], showed that the use of simulation games beyond the initial use to train military strategy skills [3]. Driving simulators follow this trend, with initial use mostly within the automotive sector, but with the rise of commercial racing simulation rigs and the emergence of eMotorsport competitions, these simulators increasingly are used within the entertainment sector [4]. They serve as a safer alternative to real-world driving, facilitated via simulating realistic, immersive, and controlled environments. They may also be situated upon an actuated platform, using one to six degrees-of-freedom (DoF) to simulate the movement and forces acting on the car during movement.

Simulators require a suitable simulation environment, composed of the simulation software with accompanying software components to enable the transfer of relevant data from the simulation software used and the physical components that comprise the simulator. The simulation environment often is restricted by the hardware components, with the simulation software and data transfer method needing to be selected with limitations of the hardware in mind in order to function and maintain simulation stability.

1.1 PROJECT AND INDIVIDUAL REPORT OVERVIEW

The project's overall aim was to design and manufacture a portable and cost-effective simulator of the Formula Student Race Car for driver training and outreach activities, with several objectives devised to meet this aim, outlined in the Contract Performance Plan (CPP) [5]. Five components that comprise the project were identified, which were the simulation environment, actuation control,

This report covers the simulation environment component, and details the simulation software, integrated data plugin for said software, and immersion modifications to the simulation environment.

P a g e | **ii**

The aim of the simulation environment component was the creation of an optimal simulation environment for the projects needs through optimal simulation software selection and the creation, modification, and optimisation of any additional required software to ensure an efficient and immersive user experience.

Requirements were created to ensure that the overall project objectives, related to the simulation environment, were met. These requirements were:

Selection of suitable racing simulation software for project needs.	Real-time extraction and transfer of relevant simulation data.	Minimal data loss during extraction, transmission & recording.	Minimise impact on simulation performance

The overall project objectives related to the simulation environment component that this report outlines, were objectives:

1) Determine a suitable approach for the simulation encompassing both hardware and software aspects.
    a) Literature Review of existing software to determine a suitable
4) Implement a virtual environment containing an immersive graphical representation of the formula student car and external surroundings.
    a) An in-simulation representation of the formula student car within the simulation software that can be tweaked through the use of software modification.
    b) A custom racetrack that can be loaded into the simulation software.
    c) A stream of data output from the game engine to the simulator hardware to enable seamless communication between the hardware and software components.
7) Minimise simulator discomfort and ensure ergonomic experience for a wide range of users.
    a) A suitable choice of hardware and software to reduce motion sickness.

The above objectives were carried out in written order, with objective seven being considered throughout the project, and all the components of objective four being developed in tandem.

1.2 Report Structure

- **Chapter 1 – Introduction**: Introduces the project and individual component this report covers. Provides report overview, aims, Objectives, and the structure.
- **Chapter 2 – Literature Review**: Provides insight and foundational information on Simulation software options, data transfer methods, and simulator immersion.
- **Chapter 3 – Simulation Environment**: Details the completed simulation environment, why each component was created, and why they were required.

P a g e | **iii**

- **Chapter 4 – Validation**: Provides validation through testing and results that justifying the choices made for the simulation environment and validate their success.
- **Chapter 5 – Conclusion**: Account of final remarks of the report, discussing the achievements, and conclusions drawn from the projects experience. Highlights improvements and additions that can be carried out for future work.

2 LITERATURE REVIEW

2.1 SIMULATION SOFTWARE

Simulation software plays a pivotal role across various domains, ranging from engineering and healthcare to social sciences and entertainment. It offers a virtual environment where complex systems or phenomena can be modelled, analysed, and manipulated to gain insights, test hypotheses, and make informed decisions. This subsection provides an overview of simulation software, highlighting its applications, characteristics, and challenges.

For simulation software emulating realistic driving experiences a plethora of differing options exists, with each catering to various levels of realism, customisation, and graphical fidelity. It was decided to use a sim racing game as the cost of commercial research simulation software exceeds that of the project's entire budgets, as well as the physics engine realism, graphical quality, and scenario realism on average exceeding the capabilities of non-specific, commercial simulation software. [6]

For the low-cost and immersive nature of the project, a focus on the following was required:

Graphics Engine: Graphics of each simulator game, responsible for screen image generation, with higher fidelity improving visual immersion [6].

Physics Engine: Assesses the realism and accuracy of the physics engine used in the simulator game, crucial for realistic vehicle dynamics simulation [6].

Modifiability: This category considers the extent to which the simulator game can be modified or customised, allowing for adjustments to accurately represent specific vehicle behaviours.

User Immersion: This category evaluates overall user immersion, including factors such as graphics, sound, realism, and customisation. [7]

Telemetry Data Extraction: This category assesses the capability of the simulator game to extract telemetry data, which is essential for interfacing with hardware and actuation platforms.

Community Support: This category highlights the presence and activity of a supportive community around each simulator game, which can provide valuable resources, mods, and user-created content for enhancing the simulator experience.

Page | iv

According to Higuera de Frutos et al [6], the top choices for a low-cost simulator are rFactor 2 and Assetto Corza. rFactor 2, developed by Studio 397, is renowned for its advanced physics engine and modifiability [8]. Its state-of-the-art tyre model [9] and dynamic track surface simulation provide an authentic driving experience, making it a popular choice among sim racing enthusiasts and professional drivers alike shown through its adoption as the sim racing platform of Formula E [10]. The platform's open architecture allows for extensive customisation, enabling users to tailor vehicle dynamics to mimic the behaviour of formula student cars accurately. Furthermore, its active modding community ensures a continuous stream of updates and enhancements, offering ample resources for integrating the simulator into research and development projects [8].

Assetto Corsa, developed by Kunos Simulazioni, is acclaimed for its realism and attention to detail. Featuring high-fidelity car models and laser-scanned tracks, it delivers immersive driving experiences. Its physics engine is quite realistic, accurately simulating vehicle dynamics, tyre behaviour, and aerodynamics. While Assetto Corsa lacks the modifiability of rFactor 2, its out-of-the-box fidelity and user-friendly interface make it an attractive option for prototyping and testing formula student car simulations. Additionally, its support for virtual reality (VR) enhances immersion, offering researchers an alternative perspective for evaluating simulator performance and user experience [11].

In recent years, there has been several new sim racing games released, as well as improvements to existing examples, and so it was deemed pertinent to carry out a small comparison between rFactor 2, Assetto Corsa and four other racing simulators. This comparison confirmed that rFactor 2 and Assetto Corsa are the leading choices for a low-cost racing simulator’s simulation software. The result of this comparison is

Simulator software	Graphics & Sound	Physics Engine	Modifiability	Usable with Project Computer	Telemetry Data Extraction	Community Support
rFactor 2 [6], [8]	High	Advanced	Extensive	Yes	Extensive	Active modding community, official Modding tools
Assetto Corsa [6], [11]	High	Realistic	Limited	Partially	Limited	Modding community, user-created content
iRacing [6], [20]	High	Realistic	Limited	Yes – But requires online connection to function	Comprehensive	Professional racing integration, online community

Page | v

Simulator software	Graphics & Sound	Physics Engine	Modifiability	Usable with Project Computer	Telemetry Data Extraction	Community Support
Project CARS 2 [6]	High	Dynamic	Comprehensive	No	Comprehensive	Active modding community, user-created content
Automobilista 2 [6], [21]	High	Realistic	Extensive	No	Extensive	Diverse motorsport community
Assetto Corsa Competizione [19]	High	High-fidelity	Limited	No	Limited	Endurance racing community

shown below in Table 1:

Table 1 - Simulation Software Comparison [6], [8], [11], [19]

2.2 IMMERSION

Immersion refers to the extent to which users feel mentally and emotionally absorbed in a simulated environment, experiencing a sense of presence and engagement. Achieving high levels of immersion is crucial for effective training, learning, and decision-making within simulated environments.

Immersion plays a vital role in simulation-based training and education across various domains, including healthcare, aerospace, Defense, and entertainment [6]. Research has shown that immersive simulations improve learning outcomes, retention of knowledge, and transfer of skills to real-world contexts [12]. Similarly, in military training, immersive simulations offer soldiers realistic combat scenarios, allowing them to develop their military strategy skills in a safe, controlled environment [6].

Several factors contribute to the level of immersion experienced by users in simulated environments:

- **Graphics and Audio Quality:** High-quality graphics and realistic sound effects enhance the visual and auditory immersion, creating a sense of presence and realism within the virtual environment [7].
- **Interactivity and Feedback:** Interactive simulations that allow users to manipulate objects, engage with virtual characters, and receive immediate feedback on their actions increase immersion and engagement [7].
- **User Interface Design:** Intuitive user interfaces, seamless navigation controls, and natural interactions with virtual elements contribute to a smoother and more immersive user experience [7].
- **Realism and Fidelity:** The degree of realism, such as realistic environmental behaviour, and fidelity in simulating real-world scenarios, physics, and

P a g e | vi

behaviours significantly impacts immersion. Accurate representation of environmental factors, dynamic interactions, and lifelike characters enhances immersion [7].

Various techniques and technologies can be employed to enhance immersion in simulation-based environments:

- **Virtual Reality (VR) and Augmented Reality (AR):** VR and AR technologies offer immersive experiences by placing users in fully or partially virtual environments, enhancing their sense of presence and interaction with simulated elements [7].
- **Haptic Feedback:** Haptic feedback devices provide tactile sensations to users, such as vibrations or force feedback, enhancing the sense of touch and realism in simulated interactions [7].
- **Multi-sensory Integration:** Combining visual, auditory, tactile, and olfactory stimuli in simulations improves immersion by engaging multiple senses and creating a more holistic and immersive experience [7].
- **Real-Time Feedback:** Instantaneous feedback

Immersion is a critical component of simulation-based environments, influencing users' engagement, learning, and decision-making. By understanding the factors influencing immersion and employing techniques to enhance immersion, simulation developers can create more effective and impactful training and educational experiences across various domains [7].

2.3 DATA TRANSMISSION METHODS

Data transmission is a critical component in the design and development of Formula Student racing simulators, enabling seamless communication between various subsystems such as the game engine, telemetry extraction module, data processing software, and actuation control system.

User Datagram Protocol (UDP) is a connectionless protocol that offers fast transmission of data packets with minimal overhead [13], [14]. It is well-suited for real-time applications requiring low latency and rapid updates, making it a popular choice for streaming, gaming, and telemetry data transmission [13], [14]. However, UDP does not guarantee delivery of packets or ensure their order, which may result in occasional packet loss or out-of-order delivery [13], [14].

Transmission Control Protocol (TCP) is a reliable, connection-oriented protocol that ensures the delivery of data packets in the correct order with error detection and recovery mechanisms [13], [14]. While TCP offers higher reliability compared to UDP, it introduces additional overhead and latency due to connection establishment, flow control, and congestion control mechanisms [13], [14]. This makes TCP less suitable for real-time applications where low latency is crucial [13], [14].

Shared memory enables fast communication between processes running on the same machine by allowing them to share a common region of memory [15]. It offers low-latency data exchange with minimal overhead, making it suitable for real-time

P a g e | **vii**

applications and inter-process communication [15], [16]. However, shared memory is limited to communication between local processes and requires careful synchronization to avoid race conditions and ensure data consistency [15], [16].

Message queues, provide asynchronous communication between processes or threads within the same system, allowing them to exchange messages without direct interaction [17]. Message queues offer reliability and scalability, making them suitable for inter-process communication and asynchronous data transfer [17], [18]. However, they may introduce latency due to message queuing and processing overhead [17], [18].

These data transmission methods were compared, as shown in Table 2 below.

Method	Speed	Latency	Reliability	Hardware Impact	Suitability
UDP	Fast	Very Low	Low	Low	Real-time applications, streaming, gaming
TCP	Moderate	Moderate	High	Moderate	Reliable data transfer, error recovery
Shared Memory	Very Fast	Very Low	Very High	Low	Real-time applications, inter-process communication
Message Queues	Fast	Low	High	High	Asynchronous communication, inter-process communication

Table 2 - Comparison of Data Transmission Protocols

Based on the project's requirements, almost real-time and extremely low-latency data extraction and transmission, the most suitable method is either Shared Memory or UDP over WLAN [14]. However, Shared Memory was not applicable as the Actuation Control System made use of shared memory. UDP offers fast transmission of data packets with minimal overhead, low latency, and high scalability, making it ideal for real-time applications such as streaming, gaming, and telemetry data transmission [13]. While UDP does not guarantee delivery of packets, its speed and efficiency outweigh the occasional packet loss or out-of-order delivery, especially in scenarios where responsiveness is prioritised over absolute data integrity [13], [14].

This literature review section provides an overview of areas to focus on to maximise immersion, as well as different simulation software and data transmission methods by comparing their characteristics, and identifying those best suited for creating a suitable simulation environment for a Formula Student racing simulator. It serves as a foundation for guiding the design and implementation of the data transmission component within the simulator, ensuring optimal performance and responsiveness in real-world racing scenarios.

Page | viii

3 SIMULATION ENVIRONMENT

The simulation environment is comprised of the simulation software, a plugin interfacing with said software that handles data acquisition, saving, and transmission, and a data processing software that is a part of the actuation control system. These are all confined within a single computer, running windows 10, and adhere to criteria to ensure maximal user immersion.

Additionally, modifications for the simulation software in the form of a car model and playable racing track were crafted in order to increase the user experience and visual immersion. The High-level design of the simulation environment is shown below in Figure 1.

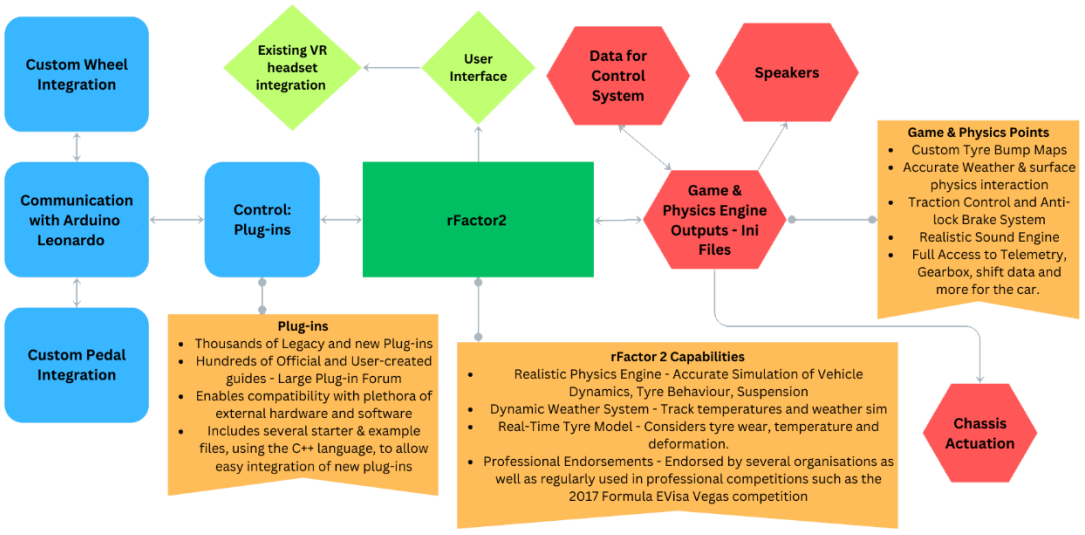


Figure 1 – High-level Simulation Environment Design

3.1 SIMULATION SOFTWARE

The choice of simulation software plays a crucial role in the development of a Formula Student racing simulator, as it directly impacts the realism, accuracy, and performance of the simulator. rFactor 2 was chosen as the simulation software for our project, as its features closely matched the requirements. While Assetto Corsa was similar in several regards, its focus on graphics compared to rFactor 2's focus on physical realism hindered its performance [6].

rFactor 2 is renowned for its advanced physics engine and realistic driving dynamics, which accurately simulate vehicle behaviour and handling characteristics. The software incorporates sophisticated tire and suspension models, aerodynamics simulations, and dynamic track conditions, providing an immersive and authentic racing experience. These realistic physics are essential for accurately replicating the performance and behaviour of Formula Student race cars on the virtual track. These

P a g e | ix

reasons are several why it was determined as a suitable simulation software for the project.

3.2 TELEMETRY PLUGIN

A telemetry plugin is a software module or add-on that integrates with a simulation environment, such as rFactor 2, to capture and extract real-time telemetry data generated during gameplay. The plugin interacts with the simulation engine to access vehicle telemetry parameters, including vehicle position, speed, acceleration, steering angle, and tire temperature, among others. This data is essential for monitoring and analysing the performance of the virtual vehicle and informing real-time control decisions.

In the context of our Formula Student racing simulator project, the telemetry plugin serves as a critical component for extracting real-time telemetry data from the rFactor 2 simulation environment, processing it, and transmitting it to the actuation control system for informing real-time actuation of the virtual simulator. A high-level design of the plugin is shown below in Figure 2.

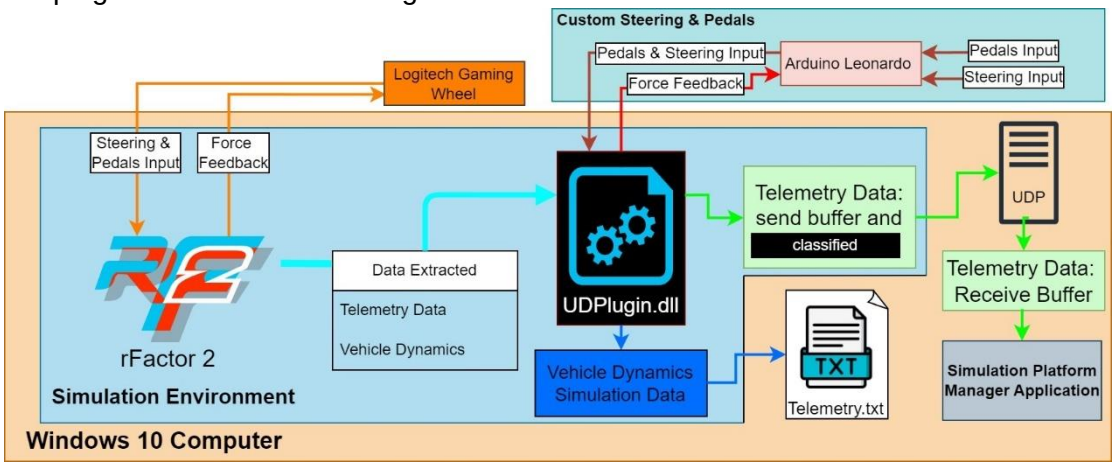


Figure 2 - High-Level design of Telemetry Plugin

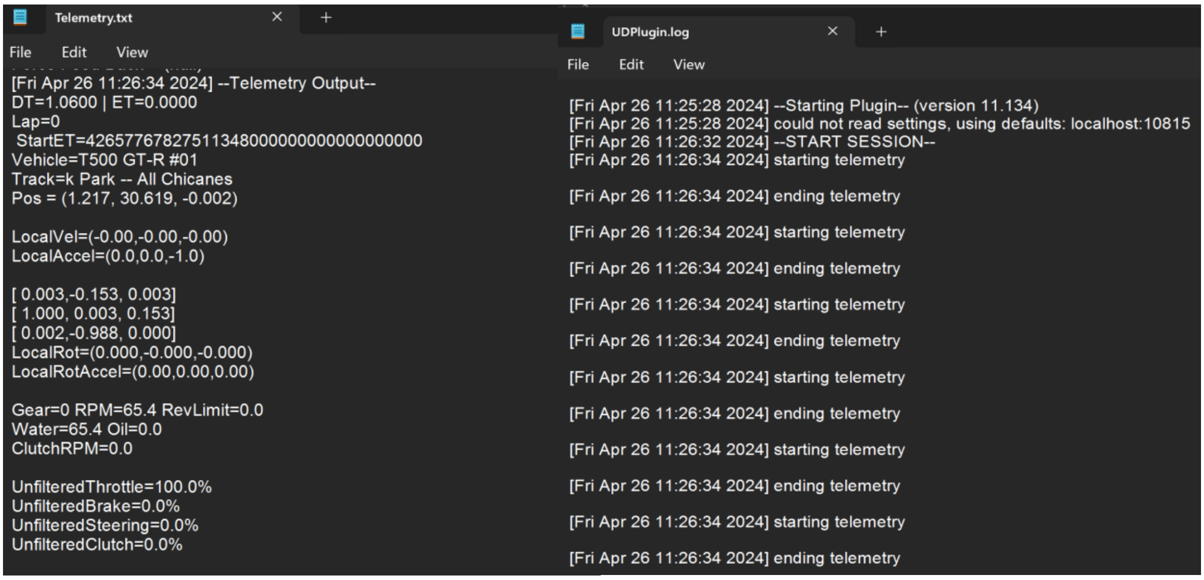
The telemetry plugin is integrated into the rFactor 2 simulation environment using the software's plugin architecture. This is done by inserting the plugin into rFactor 2's game files, in a dedicated plugin folder. Through this integration, the plugin hooks into the simulation engine's telemetry data output, allowing it to access and extract relevant data at high frequencies, with the higher frequency range reaching through testing reaching 162 Hz. The plugin utilizes rFactor 2's telemetry API to retrieve vehicle telemetry parameters and sensor readings, ensuring accurate and timely data capture during gameplay [22].

Once the telemetry data is extracted from rFactor 2, it is processed and prepared for transmission to the actuation control system. The plugin aggregates the telemetry data into a buffer, where it is stored temporarily to minimise data loss and ensure smooth transmission. Custom error checking mechanisms are implemented to validate the integrity of the transmitted data, detecting and correcting errors or anomalies that may occur during transmission.

Page | x

The telemetry data, stored in the buffer, is then transmitted over UDP to the data processing application within the actuation control system. UDP is chosen for its low latency and minimal overhead, making it suitable for real-time data transmission applications when paired with buffers to reduce data loss. The use of UDP ensures timely delivery of telemetry data to the actuation control system, facilitating responsive and accurate control of the virtual simulator.

In addition to real-time transmission, the telemetry plugin also facilitates logging and data storage for error tracing and analysis purposes. The plugin creates and writes to a log file, recording any errors, warnings, or debugging information encountered during operation. Furthermore, the plugin writes telemetry data to a text file, providing a comprehensive record of vehicle telemetry parameters and sensor readings for vehicle dynamics equations, modelling, and analysis. As displayed below in Figure 3.



Telemetry.txt

File Edit View

```
[Fri Apr 26 11:26:34 2024] --Telemetry Output--
DT=1.0600 | ET=0.0000
Lap=0
 StartET=4265776782751134800000000000000000000
Vehicle=T500 GT-R #01
Track=k Park -- All Chicanes
Pos = (1.217, 30.619, -0.002)

LocalVel=(-0.00,-0.00,-0.00)
LocalAccel=(0.0,0.0,-1.0)

[ 0.003,-0.153, 0.003]
[ 1.000, 0.003, 0.153]
[ 0.002,-0.988, 0.000]
LocalRot=(0.000,-0.000,-0.000)
LocalRotAccel=(0.00,0.00,0.00)

Gear=0 RPM=65.4 RevLimit=0.0
Water=65.4 Oil=0.0
ClutchRPM=0.0

UnfilteredThrottle=100.0%
UnfilteredBrake=0.0%
UnfilteredSteering=0.0%
UnfilteredClutch=0.0%
```

UDPlugin.log

File Edit View

```
[Fri Apr 26 11:25:28 2024] --Starting Plugin-- (version 11.134)
[Fri Apr 26 11:25:28 2024] could not read settings, using defaults: localhost:10815
[Fri Apr 26 11:26:32 2024] --START SESSION--
[Fri Apr 26 11:26:34 2024] starting telemetry

[Fri Apr 26 11:26:34 2024] ending telemetry

[Fri Apr 26 11:26:34 2024] starting telemetry

[Fri Apr 26 11:26:34 2024] ending telemetry

[Fri Apr 26 11:26:34 2024] starting telemetry

[Fri Apr 26 11:26:34 2024] ending telemetry

[Fri Apr 26 11:26:34 2024] starting telemetry

[Fri Apr 26 11:26:34 2024] ending telemetry

[Fri Apr 26 11:26:34 2024] starting telemetry

[Fri Apr 26 11:26:34 2024] ending telemetry

[Fri Apr 26 11:26:34 2024] starting telemetry

[Fri Apr 26 11:26:34 2024] ending telemetry
```

Figure 3 - Example of Telemetry.txt and plugin error log file

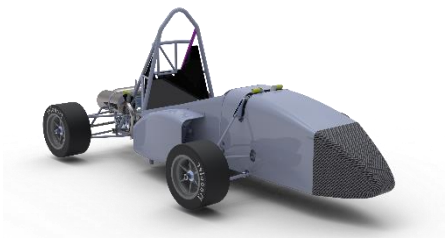
The telemetry plugin plays a pivotal role in the simulation environment, enabling the extraction, processing, and transmission of real-time telemetry data from rFactor 2 to the actuation control system. Through its integration with rFactor 2, custom error checking, UDP-based transmission, and logging capabilities, the telemetry plugin ensures reliable and efficient communication between the simulation environment and the actuation control system, contributing to the overall realism and performance of the Formula Student racing simulator.

3.3 Immersive Additions/Modifications

Continuing the simulation environment development, the next step was to create an accurate in-game car model that matched the physical simulator. The car model was essential for providing a realistic and immersive driving experience, as it would be the primary visual focus for the drivers.

P a g e | **xi**

3ds Max was used in both the creation of the car model, and the track. The process for the car and map was similar in ways, with both making use of 3ds Max to create the base models, texturization, and materials. The car model was based on a 2023 LGR formula car 3D model shown below in Figure 4, while the track was created by hand while using a google maps street view of the track as reference.



Optimising the car model for real-time performance was also undertaken, with polygon count reduction, Level of Detail implementation, and texture optimisation being implemented. The high polygon count of the detailed car model could impact the game engine's rendering performance, potentially causing framerate drops and visual lag. To address this, several optimization techniques were employed:

After each component was created in 3ds max, these immersive modifications were implemented by exporting the car and track models from 3ds Max as OBJ files, and import them into rFactor 2. Next, the materials and shaders were configured for the important car and truck models by assigning textures, adjusting material properties, and fine-tuning shader parameters to achieve the desired visual appearance.

Finally, the physics of the car and track were implemented. Road roughness bump maps and wear values configured for the track, while the cars physical values were assigned as close to matching vehicle data provided by the LGR team. The finalised car design is shown below in Figure 5 and 6, with the track layout and a snapshot shown in Figure 7 and 8.

Figure 4 - LGR Formula student car 2023

P a g e | xii

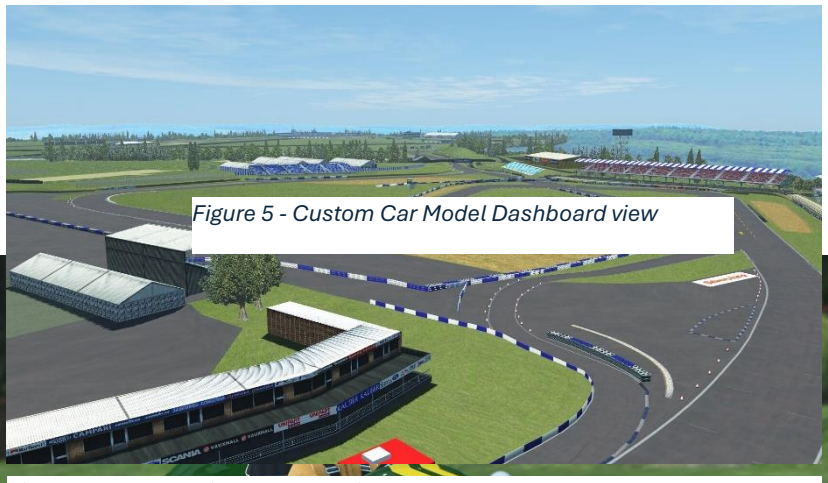


Figure 5 - Custom Car Model Dashboard view

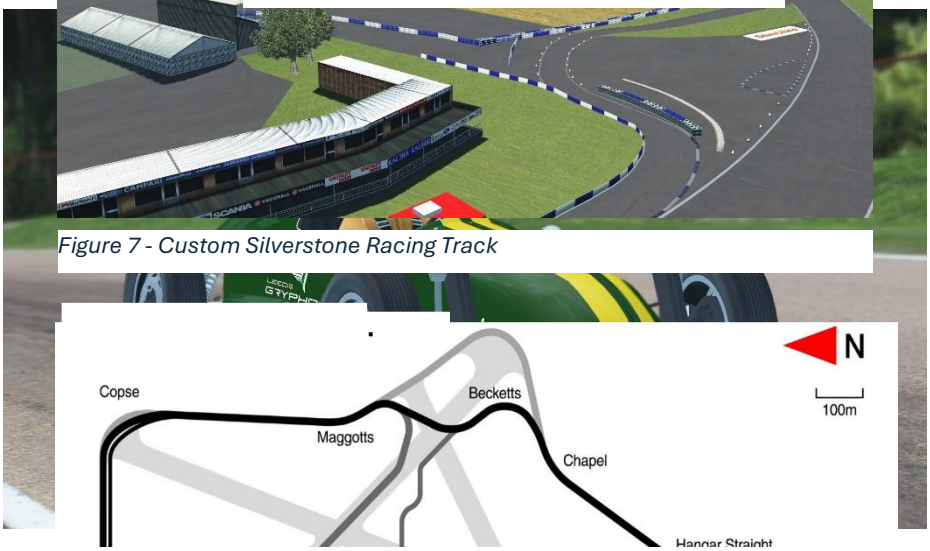


Figure 7 - Custom Silverstone Racing Track

Figure 6 - in-game model of custom car

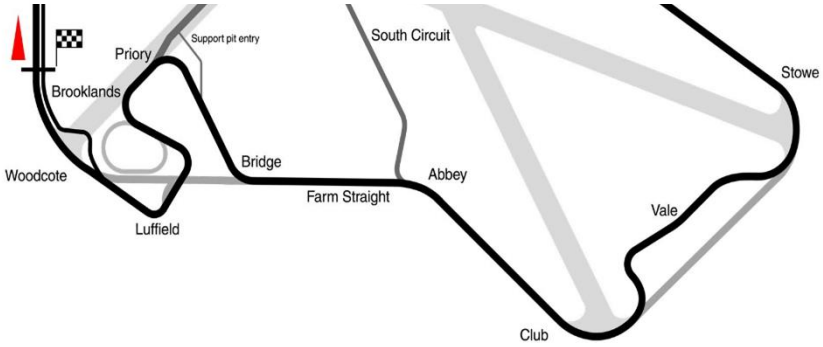


Figure 8 - Custom Track Map

By successfully creating the virtual racetrack and car model, and integrating them into the rFactor 2 game engine, the simulation environment development laid the groundwork for an immersive and realistic driving experience. The accurate representation of the real-world competition venue and the physical simulator enhanced driver familiarity and training effectiveness. The optimization techniques employed ensured smooth performance and visual fidelity, while the game engine's advanced physics simulation provided a realistic and responsive driving experience.

Page | xiii

P a g e | xiv

4 VALIDATION

Numerous tests were conducted to test the efficacy and functionality of the plugin's features. Several of these tests were quite straight forward, with differing project component validations themselves serving to validate the plugins capabilities.

The first test was rather simple and was conducted with regards to the Telemetry data recording. This involved conducting Driver-in-Loop testing, recording the data to the Telemetry.txt file, shown previously in Figure 3. Once recorded, this was further validated by inputting the recorded value into the vehicle dynamic models [Albert 2024], which produced results closely resembling the simulate data, with an acceptable error margin, shown below in figure 9.

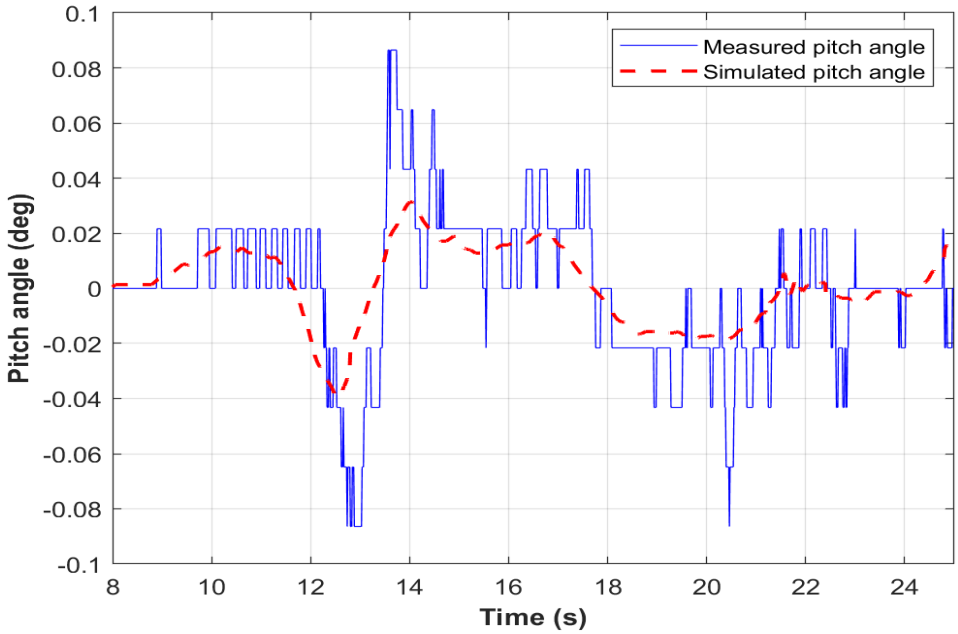


Figure 9 - Vehicle Dynamics Validation showing measured and simulated pitch angle [Qianli, 2024]

Following this test, which resulted in a successful validation of the telemetry recording feature of the plugin, a second validation test was carried out using the validated telemetry recorded data to aid in validation.

Conducted simultaneously with the Vehicle Dynamics and Actuation validation tests, these tests involved parsing through the output text file to ensure the data located within it matched the data transmitted. The first step was matching the data sent via the UDP and displayed on a UDP client created for testing purposes, while monitoring the transmission using Wireshark, as shown below in figure 10.

P a g e | xv

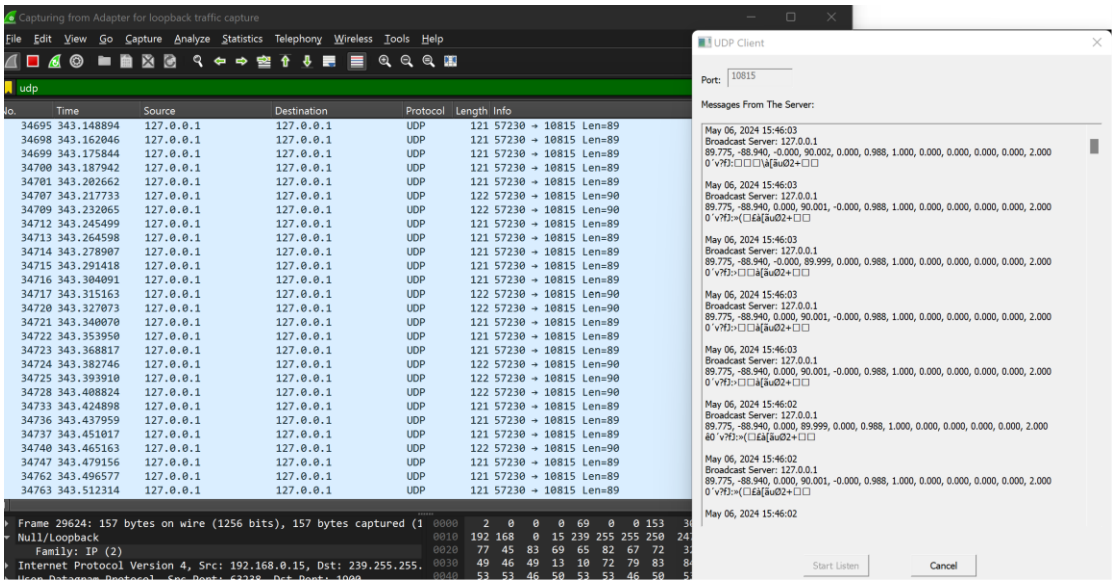


Figure 10 - UDP validation test using Wireshark and custom-made UDP Client

By cross-checking the received data with its corresponding entries within the text file, the data received was found to match that recorded, validating the efficacy of the data transmission method. After this was done, the plugin was testing during Driver-in-Loop testing to validate the actuation control system, which itself was successful in validation supporting the successful validation assertion made above.

During the test, using Wireshark, it was found that throughout several tests there was at most a change of the buffer length by one, shown above in figure 10 with len alternating between 89 and 90, indicating that while data loss has occurred it is minimal and well within acceptable margins for such high-frequency data transmission, which ranged between 16, set intentionally to ease the parsing of vehicle dynamics telemetry data saved in file, and 162 Hz. Overall, the validation tests carried out indicated success in all areas of the simulation environments.

P a g e | **xvi**

5 CONCLUSION

In conclusion, the Formula Student racing simulator project represents a significant achievement in the advancement of low-cost, portable racing simulator research and development. By ensuring the optimal simulation environment was designed and implemented, the project has successfully created a dynamic and immersive simulation environment that serves as a valuable tool for driver training, and recruitment during outreach events.

5.1 ACHIEVEMENTS

- Selection of rFactor 2 as the simulation platform for its advanced physics engine, extensive content flexibility, and multiplayer capabilities.
- Development of custom telemetry plugins enabling extraction, processing, and transmission of real-time telemetry data with robust error checking and logging capabilities.
- Seamless integration of the simulation environment with the actuation control system for real-time actuation based on telemetry data feedback.
- Rigorous validation and testing procedures ensuring reliability, accuracy, and performance of the simulation environment and its components.

5.2 FUTURE WORK

- Incorporation of VR to increase immersion and make full use of custom Model & Car
    - Explore the integration of virtual reality (VR) technology into the simulator to enhance immersion and realism for users.
    - Develop custom VR environments and scenarios tailored to the Formula Student racing experience, leveraging the simulator's custom vehicle models and tracks.
- Implementation of Reliable User Datagram Protocol (RUDP)
    - Investigate the implementation of (RUDP) as an alternative to UDP for telemetry data transmission, providing enhanced reliability and error recovery mechanisms.
    - Develop and integrate custom RUDP protocols or libraries into the telemetry plugins and actuation control system to ensure reliable and robust communication between simulation components.

These future work bullet points outline potential areas for enhancement and expansion of the Formula Student racing simulator project, including the integration of VR technology, implementation of reliable data transmission protocols, and incorporation of additional plugin features to further enrich the simulation experience and functionality.

Page | xvii

6 REFERENCES

1. Formula Student. [Online]. Available: https://www.imeche.org/events/formula-student.
2. D. R. Michael and S. L. Chen, "Serious games: Games that educate train and inform," Muska Lipman/Premier-Trade, 2005.
3. D. Djaouti, J. Alvarez, and J.-P. Jessel, "Classifying serious games: the g/p/s model," in Handbook of research on improving learning and motivation through educational games: Multidisciplinary approaches, 2011, pp. 118-136.
4. Scacchi, W. (2018). Autonomous eMotorsports racing games: Emerging practices as speculative fictions. Journal of Gaming & Virtual Worlds, 10(3), 261-285. https://doi.org/10.1386/jgvw.10.3.261_1.
5. **GROUP 135 – CONTRACT PERFORMANCE PLAN**
6. "Assessing sim racing software for low-cost driving simulator to road geometric research," https://doi.org/10.1016/j.trpro.2021.11.076.
7. M. Pinto, V. Cavallo, and T. Ohlmann, "The development of driving simulators: toward a multisensory solution.," in Le travail humain, 2008, vol. 71(1), pp. 62-95.
8. Studio 397, "About," [Online]. Available: https://www.studio-397.com/about-2/.
9. Polimi, "Assetto Corsa Competizione," [Online]. Available: https://www.politesi.polimi.it/handle/10589/153184.
10. Motorsport Games, "rFactor 2 becomes the official sim racing platform of Formula E," [Online]. Available: https://motorsportgames.com/motorsport-games-rfactor-2-becomes-the-official-sim-racing-platform-of-formula-e/.
11. Assetto Corsa, "Assetto Corsa," [Online]. Available: https://assettocorsa.gg/assetto-corsa/.
12. Adejumobi, B., Franck, N., Janzen, M. (2014). Designing and Testing a Racing Car Serious Game Module. In: Ma, M., Oliveira, M.F., Baalsrud Hauge, J. (eds) Serious Games Development and Applications. SGDA 2014. Lecture Notes in Computer Science, vol. 8778. Springer, Cham. [Online]. Available: https://doi.org/10.1007/978-3-319-11623-5_16.
13. IEEE. (1999). Conference Proceedings. INFCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. (pp. 1-2). New York, NY, USA: IEEE. DOI: 10.1109/INFCOM.1999.751376. ISBN: 0-7803-5417-6. ISSN: 0743-166X.
14. Wang, A. I., Jarrett, M., & Sorteberg, E. (2009). Experiences from Implementing a Mobile Multiplayer Real-Time Game for Wireless Networks with High Latency. International Journal of Computer Games Technology, 2009(Article ID 530367), 14 pages. Hindawi Publishing Corporation. DOI: 10.1155/2009/530367.
15. IBM. (2023). Shared memory. IBM Informix Servers Documentation, Last Updated: July 31, 2023. [Online]. Available: https://www.ibm.com/docs/en/informix-servers/12.10?topic=memory-shared.
16. Gupta, S. (2007). A Performance Comparison of Windows Communication Foundation (WCF) with Existing Distributed Communication Technologies. Microsoft Corporation. Retrieved April 30, 2007, from https://learn.microsoft.com/en-us/previous-versions/dotnet/articles/bb310550(v=msdn.10)?redirectedfrom=MSDN.
17. MIT. (2015). Reading 22: Queues and Message-Passing. Software in 6.005. Retrieved from https://web.mit.edu/6.005/www/fa15/classes/22-queues/.
18. Klaiber, A. C., & Levy, H. M. (1994). A comparison of message passing and shared memory architectures for data parallel programs. SIGARCH Computer Architecture News, 22(2), 94–105. Association for Computing Machinery. https://doi.org/10.1145/192007.192020.
19. Assetto Corsa Competizione. (2023, January 17). [Webpage]. Retrieved January 7, 2024, from https://assettocorsa.gg/assetto-corsa-competizione/.

P a g e | **xviii**

20. Greenhalgh, G., & Goebert, C. (2023). From Gearshifts to Gigabytes: An Analysis of How NASCAR Used iRacing to Engage Fans During the COVID-19 Shutdown. International Journal of Sport Communication, 17(1), 46–60. https://doi.org/10.1123/ijsc.2023-0145.
21. Reiza Studios Ltda. (2020). Automobilista 2. [Webpage]. Retrieved from https://www.game-automobilista2.com/.
22. Image Space Incorporated. (2004-2006). rFactor Internals Plugin. [PDF document]. Retrieved from https://www.studio-397.com/wp-content/uploads/2016/12/rFactorInternalsPlugin.pdf.
23. Wireshark. (n.d.). About Wireshark. Retrieved from https://www.wireshark.org/about.html.

Page | ii

Appendix

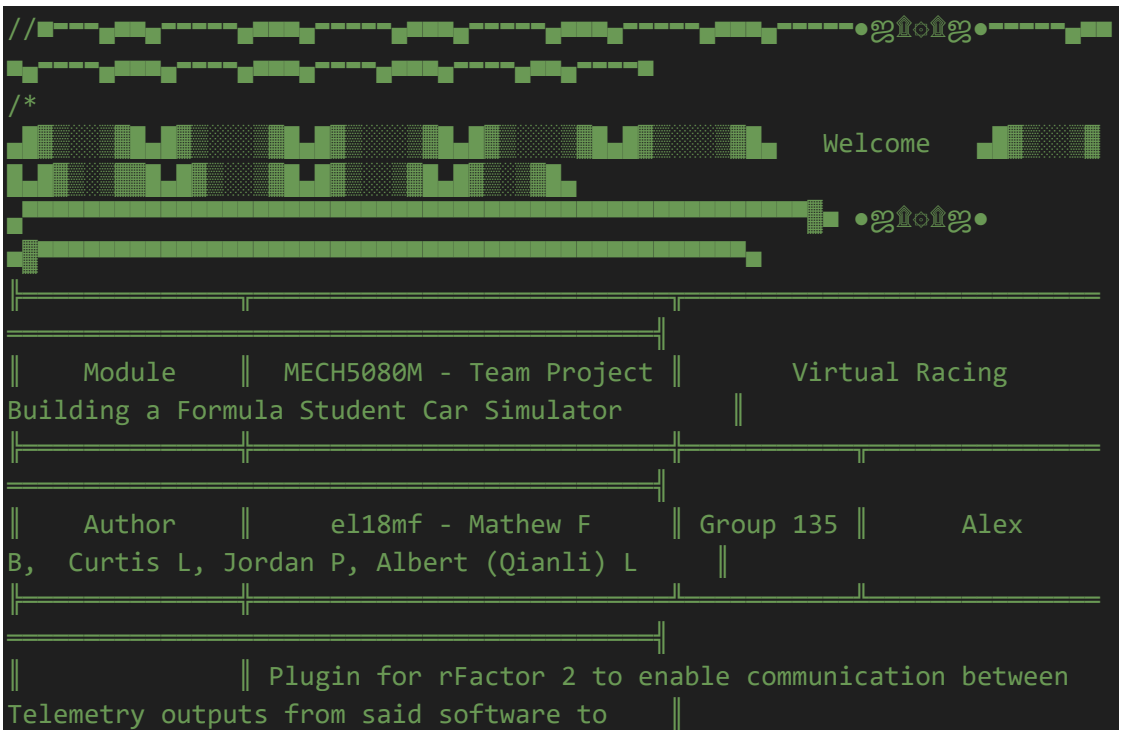
UDP_Listener_Alex.cs

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;

public class UDPListener
{
    private const int listenPort = 10815;

    private static void StartListener()
    {
        UdpClient listener = new UdpClient(listenPort);
        IPEndPoint groupEP = new IPEndPoint(IPAddress.Any, listenPort);

        try
        {
            while (true)
            {
                Console.WriteLine("Waiting for Telemtry...");
                byte[] bytes = listener.Receive(ref groupEP);

                Console.WriteLine($"Received Telemetry from {groupEP} :");
                Console.WriteLine($" {Encoding.ASCII.GetString(bytes, 0,
bytes.Length)}");
            }
        }
        catch (SocketException e)
```

Page | iii

```
        {
            Console.WriteLine(e);
        }
        finally
        {
            listener.Close();
        }
    }

    public static void Main()
    {
        StartListener();
    }
}
```

UDPlugin.cpp



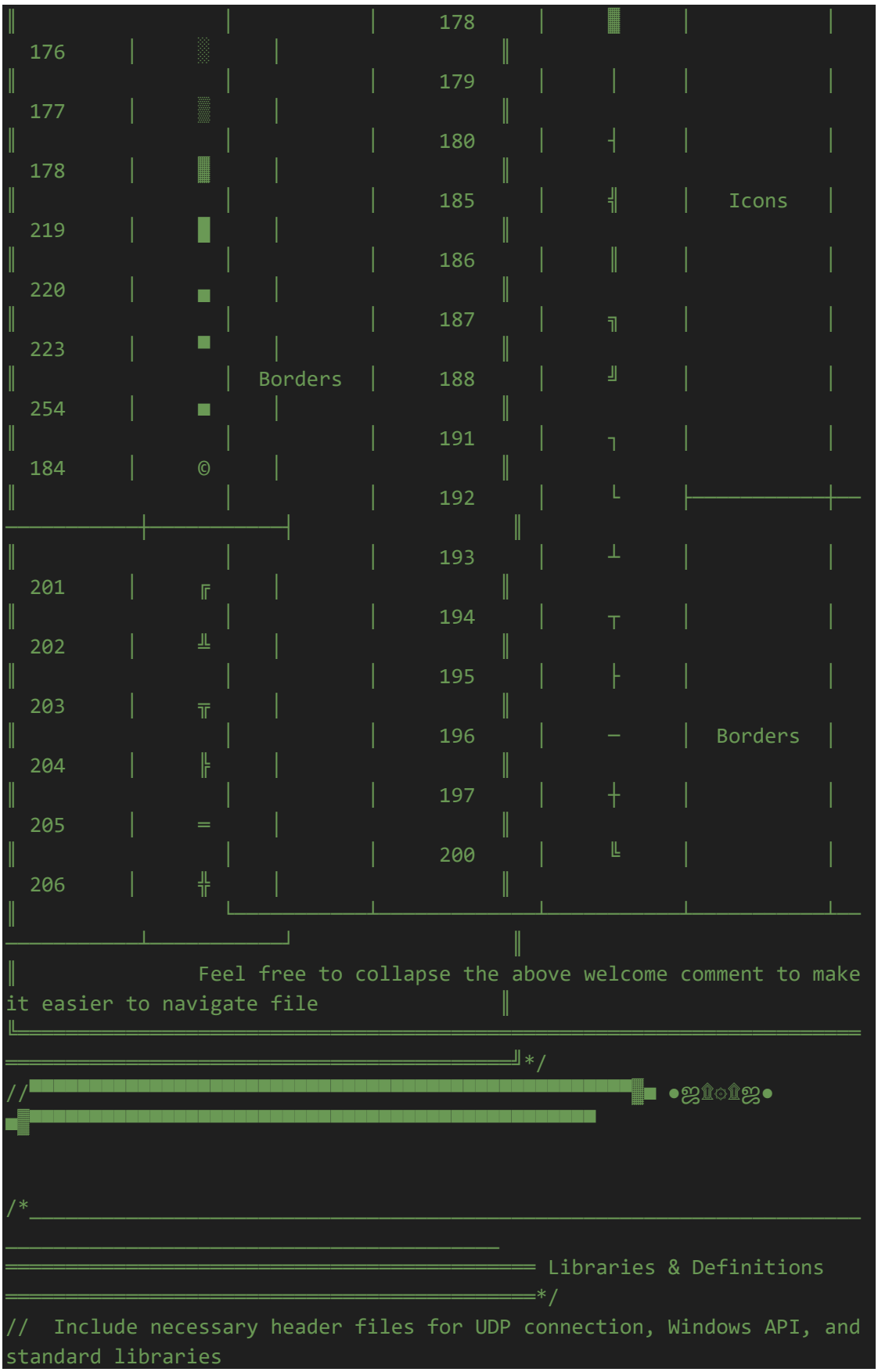
Page | iv

```
║ Description  ║ the physical hardware in order to faciliate actuation.
Created for Master's dissertation.         ║
║              ║ Sends Data Packages with order described in
accompanying excel file via UDP port 10815.           ║
╟──────────────╫──────────────────────────────────────────────────────
─────────────────────────────────────────────╢
║              ║ V1) File: rFactor 2 Output -> .csv file -> Arduino
(Alex Code) -> Actuation (Too high Latency). ║
║    Plugin    ║ V2) Memory Buffer: Stored data in the memory buffer
then accessed by arduino (missing data).    ║
║   Versions   ║ V3) TCP/UDP V1: Send data via TCP/UDP port (Detected
as Intrusive by game/caused crashing).     ║
║              ║ V4) UDP V2: Aided in work-around by <redacted>* -
Current Version.                              ║
║              ║ *Asked to keep said person & work-around anonymous to
prevent proliferation of said work-around.║
╟──────────────╨──────────────────────────────────────────────────────
─────────────────────────────────────────────╢
║ Adapted from source code, information, data, and algorithms belonging
to CUBE technology (tm) which form the      ║
║ the Example Plugin V8 for rFactor2, made available by Studio 397 in
the form of a zip file within their forums.║
║        Download found in modding resources section here:
https://www.studio-397.com/modding-resources/         ║
║            PROPRIETARY AND CONFIDENTIAL -  Copyright (c) 2018 Studio
397 B.V.  All rights reserved.             ║
║                                                    ╔═══════╗
                                        ║
╠════════════════════════════════════════════════════╝ Notes
╚═════════════════════════════════════════════════════╣
║
                                        ║
║   -   Tab Size:   4 | Plugin Language: C++ | Sends Data to C# Program
| Data via UDP: Sent as ascii hexcode       ║
║
                                        ║
║   -   Formatting: | Max Column - 118 | ═════ Title ═════ |
_____Section_Split_____                            ║
║
                                        ║
║   -   Commenting: | If fits in 118 column max // on same line | If
too long, /* across multiple lines before    ║
║
                                        ║
```

Page | v

```
‖   -   "ctrl + /": Visual Studio Code shortcut to comment selected
line/s, eases enabling and disabling of      ‖
‖                   features.
                                        ‖
‖
                                        ‖
‖   -   Compiling:  Use of Visual Studio Express 2012 (free via
microsoft coding insider programmer) used to     ‖
‖                   compile, but compiling possibly through Visual
Studio Code via Bash terminal using (check if ‖
‖                   plugin compiles/error
testing):
‖
‖
                                        ‖
‖       1)  g++ -c -fPIC file.cpp -o file.o         //  Created object
file that can be converted to a .dll      ‖
‖
                                        ‖
‖       2a) g++ -shared -o file.dll file.o          //  Compilation of
Object file into a .dll                   ‖
‖           g++ -shared -o file.dll file.o -lWs2_32 //  Missing
Library: Winsock library Manual link (UDP)       ‖
‖
                                        ‖
‖   -   Compiling done using Visual Studio Express 2012 which is free
via Visual Studio Dev Essentials service:  ‖
‖                                                  ╔═══════╗
                                        ‖
╠══════════════════════════════════════════════════╝ ASCII
╚══════════════════════════════════════════════════╗
‖
                                        ‖
‖                ┌──────────┬──────────┬──────────┬──────────┬──
──────────┬──────────┐                  ‖
‖                │   Type   │  alt +
No.  │  Symbol  │   Type   │  alt +
No.  │  Symbol  │                  ‖
‖                ├──────────┼──────────┼──────────┼──────────┼──
──────────┼──────────┤                  ‖
‖                │          │    176   │    ▒     │          │
 174    │    «     │                  ‖
‖                │          │    177   │    ▒     │          │
 175    │    »     │                  ‖
```

Page | vi


176
177
178
219
220
223
254
184
178
179
180
185
186
187
188
191
192
193
194
195
196
197
200
Icons
Borders
Borders
201
202
203
204
205
206
Feel free to collapse the above welcome comment to make it easier to navigate file
Libraries & Definitions
// Include necessary header files for UDP connection, Windows API, and standard libraries

Page | vii

```
#include <WinSock2.h>           // required library for UDP connection
#include <Windows.h>
#include "UDPlugin.hpp"         // corresponding header file
#include <math.h>               // for mathematical functions like
atan2, sqrt
#include <stdio.h>              // for standard input/output operations
#include <assert.h>             // for assertion macros
#include <io.h>                 // for low-level I/O functions
#include <sys/stat.h>           // for file status functions
#include <string.h>             // for string manipulation functions
#include <sys/types.h>          // for data types used in system calls
#include <time.h>               // for time-related functions, such as
getting current time
#include <cstring>              // For strlen
#include <limits>               // For INT_MAX

#include <WS2tcpip.h>           // additional library for TCP/IP
functionality

#define TIME_LENGTH 26          // Define the length of the time string
//  UDPlugin::UDPlugin(){}
//  UDPlugin::~UDPlugin(){}



/*_____________________________________________________________________
_____________________________________
================================== Plugin Information
==================================*/

extern "C" __declspec( dllexport )  // Sets Plugin Name.
const char * __cdecl GetPluginName()                    {
return("UDPlugin - 2024.04.08"); }

extern "C" __declspec( dllexport )  // Plugin Object Type - See
PluginObjects.hpp lines 30-41 for all types.
PluginObjectType __cdecl GetPluginType()                {
return(PO_INTERNALS); }

extern "C" __declspec( dllexport )  // InternalsPluginV01 functionality
if return value changed, you must derive.
int __cdecl GetPluginVersion()                          { return(6); }
// from the appropriate class (1-7)!

extern "C" __declspec( dllexport )
```

Page | viii

```
PluginObject * __cdecl CreatePluginObject()           { return(
(PluginObject *) new UDPlugin ); }

extern "C" __declspec( dllexport )
void __cdecl DestroyPluginObject( PluginObject *obj )  { delete(
(UDPlugin *) obj ); }

/*____________________________________________________________________
_____________________________________
============================================ File & Logging
============================================*/

/*____________________________________________________________________
_____________________________________
╔═══════════╦═══════════════╗
║  Function ║ WriteToFiles ║
╠═══════════╬═══════════════╩═══════════════════════════════════════
═══════════════════════════════════╗
║ Description ║ Writes timestamped message/data (Telemetry, Graphics,
Scoring) to individual text files.    ║
╠═══════════╬═══════════════════════════════════════════════════════
═══════════════════════════════════╣
║  Parameters ║ openStr:    Mode in which to open files ("w" for write,
"a" for append, etc.).              ║
╟─────────────╢ msg:        Contains the message to be
written.                                          ║
╚═══════════╩═══════════════════════════════════════════════════════
═══════════════════════════════════╝*/
void UDPlugin::WriteToFiles( const char * const openStr, const char *
const msg )
{
FILE *TelemFile;                                          //  Pointer
to FILE object for file access.
time_t
curtime;                                             //  Variable to
store the current time.
struct tm loctime;                                        //  Pointer
to tm structure for converting time.
char thetime[TIME_LENGTH];                                //  Buffer
to hold formatted time string

//  //  Uncomment and change TelemPath to save to a specific location
//  const char* TelemPath =
"C:\\Users\\Mathew\\Desktop\\rF2_data_files\\Telemetry.txt";
```

Page | ix

```
//  int telem = fopen_s(&TelemFile, TelemPath, openStr ); // Open
'Telemetry.txt', mode specified by openStr.

//  Saves to rFactor 2 steam folder under the name Telemetry.txt
int telem = fopen_s(&TelemFile, "Telemetry.txt", openStr ); // Open
Telemetry.txt, mode specified by openStr.

if(telem == 0)  {                                          // Check if
the file was successfully opened.
    curtime = time(NULL);                                  // Get the
current time.
    int telem2 = localtime_s(&loctime, &curtime);          // Convert
current time to local time
    int telem3 = asctime_s(thetime, TIME_LENGTH, &loctime); // Convert
local time to formatted string
    thetime[TIME_LENGTH - 2] = 0;                          // Remove
newline character from time string

    fprintf(TelemFile, "[%s] %s\n", thetime, msg);  // Writes message
to file with corresponding timestamp.
    fclose(TelemFile);                                 // Close file to
flush stream & release resources.
}

// // Uncomment to enable Graphics Output Data being written to a
corresponding text file.
// fo = fopen( "GraphicsOutput.txt", openStr );
// if( fo != NULL )  {                              // Check if the file
was successfully opened.
//   curtime = time(NULL);                         // Get the current
time.
//  loctime = localtime(&curtime);                // Convert current
time to local time.

// fprintf( fo, "[%s] %s\n", asctime (loctime), msg);//Write the
timestamped message to "TelemetryOutput.txt".
// fclose( fo );                                    //Close the file to
flush the stream and release resources.
// }

// // Uncomment to enable Scoring Output Data being written to a
corresponding text file
// fo = fopen( "ScoringOutput.txt", openStr );
// if( fo != NULL )  {                              // Check if the file
was successfully opened.
```

Page | x

```
//   curtime = time(NULL);                          // Get the current
time.
//  loctime = localtime(&curtime);                // Convert current
time to local time.

//   fprintf( fo, "[%s] %s\n", asctime (loctime), msg);// Write the
timestamped message to "TelemetryOutput.txt"
//   fclose( fo );                                     // Close the file
to flush the stream and release resources
// }

}
//  Finished Code & Comments


/*_____________________________________________________________________
_____________________________________
╔═══════════════╦═══╗
║   Function    ║ log ║
╠═══════════════╬═══╩═══════════════════════════════════════════════════
═══════════════════════════════════╗
║ Description ║ Logs timestamped message/data to an individual .log
file.                                  ║
╠═══════════════╬═══════════════════════════════════════════════════
═══════════════════════════════════╣
║  Parameters ║ msg: Contains the message to be
written.                                                  ║
╚═══════════════╩═══════════════════════════════════════════════════
═══════════════════════════════════╝*/
void UDPlugin::log( const char * const openStr, const char *msg) {

// Define the full path for the log file - Change to suit needs
// const char* logPath =
"C:\\Users\\Mathew\\Desktop\\rF2_data_files\\UDPlugin.log";

FILE *logFile;                                          //  Pointer to
FILE object for log file access
time_t curtime;                                         //  Variable to
store the current time
struct tm loctime;                                      //  Pointer to
tm structure for converting time
char thetime[TIME_LENGTH];                              //  Buffer to
hold formatted time string

// Open the log file in append mode ("a"). If the file doesn't exist,
it will be created.
```

P a g e | xi

```
int err = fopen_s(&logFile, "UDPlugin.log", openStr);       //  Attempt
to open or create log file
if (err == 0) {                                          //  Check if
the file was successfully opened
    curtime = time(NULL);                                   //  Get the
current time
    int err2 = localtime_s(&loctime, &curtime);          //  Convert
current time to local time
    int err3 = asctime_s(thetime, TIME_LENGTH, &loctime); //    Convert
local time to formatted string
    thetime[TIME_LENGTH - 2] = 0;                           //  Remove
newline character from time string

fprintf(logFile, "[%s] %s\n", thetime, msg);                //  Writes
message to file with corresponding timestamp
    fclose(logFile);                                        //  Close
file to flush stream & release resources
}
}
//  Finished Code & Comments


/*_____________________________________________________________________
________________________________________

╓────────────╥───────╖
║   Function ║ Error ║
╟────────────╫───────╨─────────────────────────────────────────────────
╟────────────╫
─────────────────────────────────────╢
║ Description ║ Logs any errors that occur into the .log
file                                                 ║
╟────────────╫─────────────────────────────────────────────────────────
─────────────────────────────────────╢
║  Parameters ║ msg: Contains the message to be
written                                                       ║
╙────────────╨─────────────────────────────────────────────────────────
─────────────────────────────────────╜*/
void UDPlugin::Error(const char * const msg)    {log("a",
msg);}    //  adds error message to log file

//════════════════════════════════════════════ Startup & Stages
══════════════════════════════════════════════

/*_____________________________________________________________________
________________________________________

╓────────────╥─────────╖
║   Function ║ Startup ║
```

P a g e | xii

```
╠═════════════╬═════════╩═══════════════════════════════════════════════════════════════
═════════════════════════════════════╣
║ Description ║ Initiates the plugin, acquired server settings file if
applicable, then connects to the     ║
║═════════════║ the socket with either the given settings or default
settings.                              ║
╠═════════════╬═════════════════════════════════════════════════════════════════════════
═════════════════════════════════════╣
║  Parameters ║ version: Contains current version of plugin *
1000                                                 ║
╚═════════════╩═════════════════════════════════════════════════════════════════════════
═════════════════════════════════════╝*/
void UDPlugin::Startup(long version)
{
//  Change directory to relevant location if .ini has been created
const char* SettingsPath =
"C:\\Users\\Mathew\\Desktop\\rF2_data_files\\UDPlugin.ini";
FILE *settings;            // Pointer to FILE object for settings file
access
// struct hostent *ptrh;    // Pointer to hostent structure (commented
out, not used in this version)
data_version = 1;          // Assigning data_version variable a value
of 1
char portstring[10];        // Character array to store port number as
string

ADDRINFO hints = { sizeof(addrinfo) };  // Initialize ADDRINFO
structure with size
hints.ai_flags = AI_ALL;                // Specify AI_ALL flag for
address resolution
hints.ai_family = PF_INET;              // Specify IPv4 address family
hints.ai_protocol = IPPROTO_IPV4;       // Specify IPv4 protocol
ADDRINFO *pResult = NULL;               // Pointer to ADDRINFO
structure for address resolution result

//  Records Startup into the output & Log files with timestamps
char temp[90];                          // Character array to store
formatted startup message
                                        // (below) Format startup
message with version number
sprintf( temp, "--Starting Plugin-- (version %.3f)", (float) version /
1000.0f );
WriteToFiles( "w", temp);               // Write startup message to
output files
```

Page | xiii

```
log("w", temp);                          //  Records Startup into Log
with timestamp

// open socket
s = socket(PF_INET, SOCK_DGRAM, 0);                 // Create datagram
socket
if (s < 0) {
    log("a", "could not create datagram socket");       // Log error
message if socket creation fails
    return;
}

int err = fopen_s(&settings, SettingsPath, "r");    // Open settings
file for reading
if (err == 0) {

        log("a", "reading settings");   // Log message indicating
settings file is being read

        if (fscanf_s(settings, "%[^:]:%i", hostname,
_countof(hostname), &port) != 2)   {
            // Log error message if reading host and port from settings
file fails
            log("a", "could not read host and port");
        }

        //ptrh = gethostbyname(hostname);   //  used with previous
approach, kept for possible future use

        //  Log message indicating settings have been successfully read
from file
        log("a", "settings read from file");

        int errcode = getaddrinfo(hostname, NULL, &hints,
&pResult);    // Resolve host name to IP address

        fclose(settings);       // Close settings file

        log("a", "hostname is:");               // Log hostname 1/2
        log("a", hostname);                     // Log hostname 2/2
        log("a", "port is:");                   // Log port number
        sprintf_s(portstring, "%i", port);  // Convert port number to
string
        log("a", portstring);                   // Log port number
}
```

Page | xiv

```
else    {
        // Log message indicating default settings are being used
        log("a", "could not read settings, using defaults:
localhost:10815");

        //  used with previous approach, kept for possible future use
        //ptrh = gethostbyname("localhost");// Convert host name to
equivalent IP address and copy to sad.

        // Resolve default host name to IP address
        int errcode = getaddrinfo("localhost", NULL, &hints, &pResult);

        port = 10815;                       // Set default port number
}

memset((char *)&sad, 0, sizeof(sad));       //  clear sockaddr
structure
sad.sin_family = AF_INET;                   //  set family to
Internet
sad.sin_port = htons((u_short)port);        //  originally 6789 but
changed to 10815

//  Assigns the IPv4 address obtained from address resolution to the
sin_addr member of the sockaddr_in structure
sad.sin_addr.S_un.S_addr = *((ULONG*)&(((sockaddr_in*)pResult-
>ai_addr)->sin_addr));

//  Old Code used with alternative method, could still possibly be
useful for future applications.
// memcpy(&sad.sin_addr, ptrh->h_addr, ptrh->h_length);
// mEnabled = true;  // default HW control enabled to true
}
//  Finished Code & Comments


/*_____________________________________________________________________
_____________________________________
╔═══════════╦═══════════╗
║  Function ║ Shutdown  ║
╠═══════════╩═══════════╩═══════════════════════════════════════════
═══════════════════════════════════╗
║ Description ║ Closes the UDP socket and shuts down the
plugin.                                                 ║
╚═════════════╩═══════════════════════════════════════════════════
═══════════════════════════════════╝*/
void UDPlugin::Shutdown() {
```

Page | xv

```
if (s > 0) {                                        //  Checks to see if
socket is active
closesocket(s);                                  //  If active, closes the
socket
s = 0;                                           //  Sets socket to equal 0
}

WriteToFiles( "a", "--Shutting Down--" );        //  Writes Shutdown
message to enabled output files
log("a", "--Shutting Down--" );                      //  Records
Shutdown into Log
}
//  Finished Code & Comments


/*_____________________________________________________________________
_____________________________________
╔═══════════╦═══════════════╗
║   Function  ║ StartSession ║
╠═══════════╬═══════════════╩═══════════════════════════════════════════
═══════════════════════════════════════╗
║ Description ║ Executes code/commands when session starts - Logging
session start in files currently       ║
╚═══════════╩═══════════════════════════════════════════════════════════
═══════════════════════════════════════╝*/
void UDPlugin::StartSession()
{
WriteToFiles( "a", "--START SESSION--" );    //  Writes Session Start
message to enabled output files
log("a", "--START SESSION--" );                  //  Records Session
Start into Log
}
//  Finished Code & Comments


/*_____________________________________________________________________
_____________________________________
╔═══════════╦═══════════╗
║   Function  ║ EndSession ║
╠═══════════╬═══════════╩═══════════════════════════════════════════════
═══════════════════════════════════════╗
║ Description ║ Executes code/commands when session ends - Loggs
session end in files.                  ║
╚═══════════╩═══════════════════════════════════════════════════════════
═══════════════════════════════════════╝*/
void UDPlugin::EndSession()
```

Page | xvi

```
{
WriteToFiles( "a", "--END SESSION--" ); //  Writes Session End message
to enabled output files
log("a", "---END SESSION--" );              //  Records Session End
into Log
}
//  Finished Code & Comments


/*_____________________________________________________________________
_____________________________________
╔═══════════════╦═══════════════╗
║   Function    ║ EnterRealtime ║
╠═══════════════╬═══════════════╩═══════════════════════════════════════
═══════════════════════════════════╗
║ Description ║ Executes code when entering Real-time - Logs real-time
start & resets elapsed time counter. ║
╚═══════════════╩═══════════════════════════════════════════════════════
═══════════════════════════════════╝*/
void UDPlugin::EnterRealtime()
{
// start up timer every time we enter realtime
mET = 0.0f;                              //  Reset elapsed time counter
to 0 when entering real-time session
WriteToFiles( "a", "---ENTER REALTIME---" );//  Writes message to
enabled output files when real-time entered
log("a", "---ENTER REALTIME---" );          //  Records real-time
session entry into Log
}
//  Finished Code & Comments


/*_____________________________________________________________________
_____________________________________
╔═══════════════╦═══════════════╗
║   Function    ║ ExitRealtime  ║
╠═══════════════╬═══════════════╩═══════════════════════════════════════
═══════════════════════════════════╗
║ Description ║ Executes code when exiting Real-time - Logs real-time
start & set elapsed time counter to -1║
╚═══════════════╩═══════════════════════════════════════════════════════
═══════════════════════════════════╝*/
void UDPlugin::ExitRealtime()
{
mET = -1.0f;                                    //  RSet elapsed time
counter to -1 when exiting real-time session
```

Page | xvii

```
WriteToFiles( "a", "---EXIT REALTIME---" );     //  Writes real-time
exit message to enabled output files
log("a", "---EXIT REALTIME---" );                   //  Records real-
time session exit into Log
}
//  Finished Code & Comments

//═══════════════════════════════════════════ Data Output
═══════════════════════════════════════════
/*_____________________________________________________________________
_____________________________________
╔═══════════╦═══════════════════╗
║   Function  ║ UpdateTelemetry ║
╠═══════════╬═══════════════════╩═══════════════════════════════════════
═══════════════════════════════════╗
║ Description ║ Sends Telemetry Data included via UDP server to C#
program. Data sent in the form of ascii  ║
╠═══════════╣ Hexcode. Also records specified Telemetry data to
Telemetry.txt file.                      ║
╠═══════════╬═══════════════════════════════════════════════════════
═══════════════════════════════════╣
║  Parameters ║ TelemInfoV01 &info: Enables fetching of Telemetry
Data                                     ║
╚═══════════╩═══════════════════════════════════════════════════════
═══════════════════════════════════╝*/
void UDPlugin::UpdateTelemetry(const TelemInfoV01 &info) {

log("a", "starting telemetry\n"); // Records Telemetry start into Log

//
=============================================================================
==============================================
// Declare a buffer to store telemetry data to be sent
char buffer [200];

// Get the size of the broadcast address structure
int len = sizeof(sad);

// Our world coordinate system is left-handed, with +y pointing up.
// The local vehicle coordinate system is as follows:
//   +x points out the left side of the car (from the driver's
perspective)
//   +y points out the roof
//   +z points out the back of the car
// Rotations are as follows:
```

Page | xviii

```
//   +x pitches up
//   +y yaws to the right
//   +z rolls to the right
// Note that ISO vehicle coordinates (+x forward, +y right, +z upward)
are
// right-handed.  If you are using that system, be sure to negate any
rotation
// or torque data because things rotate in the opposite direction.  In
other
// words, a -z velocity in rFactor is a +x velocity in ISO, but a -z
rotation
// in rFactor is a -x rotation in ISO!!!

// Compute auxiliary vectors based on the telemetry orientation data
TelemVect3 forwardVector = { -info.mOri[0].z, -info.mOri[1].z, -
info.mOri[2].z };
TelemVect3 upVector =
{  info.mOri[0].y,  info.mOri[1].y,  info.mOri[2].y };
TelemVect3 leftVector =
{  info.mOri[0].x,  info.mOri[1].x,  info.mOri[2].x };

// Calculate pitch, yaw, and roll from orientation data
// These are normalized vectors, and the world Y coordinate is up.
// So, pitch and roll (w.r.t. the world x-z plane) can be determined as
follows:
const double pitch = atan2(forwardVector.y, sqrt((forwardVector.x *
forwardVector.x) + (forwardVector.z * forwardVector.z)));
const double yaw = atan2(info.mOri[0].z, info.mOri[2].z);
const double roll = atan2(leftVector.y, sqrt((leftVector.x *
leftVector.x) + (leftVector.z * leftVector.z)));
const double radsToDeg = 57.296;    // Radians to Degree conversion

/*  Final Data points to output - Remember to prefix info. when adding
to the sprintf:
Albert
    // Driver input
    double mUnfilteredThrottle;     //  ranges  0.0-1.0 (accelerator
pedal)
    double mUnfilteredBrake;        //  ranges  0.0-1.0 (brake pedal)
    double mUnfilteredSteering;     //  ranges -1.0-1.0 (left to right
- Steering Wheel)
    double mUnfilteredClutch;       //  ranges  0.0-1.0 (clutch pedal)
    mUnfilteredBrake mUnfilteredSteering mUnfilteredClutch
    // Misc
    double mSteeringShaftTorque;    //  torque around steering shaft
```

Page | xix

```
    double mRotation;              // radians/sec (Wheel Turning
Angles??)

Alex & Albert
    //  Pitch - Remember to multiply by radsToDeg to change from
radians to degree output
    const double pitch = atan2(forwardVector.y, sqrt((forwardVector.x *
forwardVector.x) + (forwardVector.z * forwardVector.z)));

    //  Yaw - Remember to multiply by radsToDeg to change from radians
to degree output
    const double yaw = atan2(info.mOri[0].z, info.mOri[2].z);

    //  Roll - Remember to multiply by radsToDeg to change from radians
to degree output
    const double roll = atan2(leftVector.y, sqrt((leftVector.x *
leftVector.x) + (leftVector.z * leftVector.z)));

    TelemVect3 mLocalAccel;         // acceleration (meters/sec^2) in
local vehicle coordinates
    mLocalAccel.y;                  //  Heave acceleration - Refers to
vertical acceleration, usually along the z-axis.
    - mLocalAccel.x;                //  Sway acceleration  - Associated
with lateral movement, typically along the y-axis.
    - mLocalAccel.z;                //  Surge acceleration - Relates to
longitudinal movement, generally along the x-axis.

*/
//  Format telemetry data into a string buffer - May give "warning
C4267: 'argument' : conversion from 'size_t' to 'int', possible loss of
data" during compilation, however should only reach the size of 125
which is far below the max int size
sprintf(buffer, "%.3f, %.3f, %.3f, %.3f, %.3f, %.3f, %.3f, %.3f, %.3f,
%.3f, %.3f, %.3f\r\n", roll * radsToDeg,  pitch * radsToDeg,
info.mLocalAccel.y, yaw * radsToDeg, - info.mLocalAccel.x, -
info.mLocalAccel.z, info.mUnfilteredThrottle, info.mUnfilteredBrake,
info.mUnfilteredSteering, info.mUnfilteredClutch,
info.mSteeringShaftTorque,   info.mRotation);
//
                    |     1. Roll     |     2. Pitch      |    3.
HeaveAcc    |      4. Yaw    |       5. SwayAcc    |      6.
SurgeAcc    |    7. Throttle Pedal    |    8. Brake Pedal    |    9.
Steering Wheel    |    10. Clutch Pedal   | 11. Steering Shaft Force |
12.Wheel Rotation |
```

Page | xx

```
// Send telemetry data via UDP broadcast - Remind: Data is transmitted
in ascii (Hex)
    // s:                           Socket descriptor for sending data
    // buffer:                      Buffer containing the telemetry
data
    // strlen(buffer):              Length of the telemetry data in the
buffer
    // 0:                           Flags (here, no special flags are
specified)
    // (sockaddr*)&brdcastaddr:     Pointer to the broadcast address
structure
    // len:                         Size of the broadcast address
structure
int ret = sendto(s, buffer, strlen(buffer), 0, (sockaddr*)&sad, len);

//
================================================================================
================================================
//  Writes the beginning of telemetry output to the output file
WriteToFiles( "a", "--Telemetry Output--");

//  Below records Telemetry data to Telemetry.txt file
//  Open or create a file named "Telemetry.txt" in append mode ("a")
FILE *TelemFile = fopen("Telemetry.txt", "a");
if  (TelemFile != NULL)
{
    for (int i = 0; i < 10; i++) {
        // Delta time is variable, as we send out the info once per
frame
        fprintf(TelemFile, "DT=%.4f | ET=%.4f\n", info.mDeltaTime,
info.mElapsedTime);
        fprintf(TelemFile, "Lap=%d \n StartET=%20.f\n",
info.mLapNumber, info.mLapStartET);
        fprintf(TelemFile, "Vehicle=%s\n", info.mVehicleName);
        fprintf(TelemFile, "Track=%s\n", info.mTrackName);
        fprintf(TelemFile, "Pos = (%.3f, %.3f, %.3f)\n\n", info.mPos.x,
info.mPos.y, info.mPos.z);

    //  Forward is roughly in the -z direction (although current pitch
of car may cause some y-direction velocity)
        fprintf(TelemFile, "LocalVel=(%.2f,%.2f,%.2f)\n",
info.mLocalVel.x, info.mLocalVel.y, info.mLocalVel.z);
        fprintf(TelemFile, "LocalAccel=(%.1f,%.1f,%.1f)\n\n",
info.mLocalAccel.x, info.mLocalAccel.y,
                info.mLocalAccel.z);
```

Page | xxi

```
        // Orientation matrix is left-handed
        fprintf(TelemFile, "[%6.3f,%6.3f,%6.3f]\n", info.mOri[0].x,
info.mOri[0].y, info.mOri[0].z);
        fprintf(TelemFile, "[%6.3f,%6.3f,%6.3f]\n", info.mOri[1].x,
info.mOri[1].y, info.mOri[1].z);
        fprintf(TelemFile, "[%6.3f,%6.3f,%6.3f]\n", info.mOri[2].x,
info.mOri[2].y, info.mOri[2].z);
        fprintf(TelemFile, "LocalRot=(%.3f,%.3f,%.3f)\n",
info.mLocalRot.x, info.mLocalRot.y, info.mLocalRot.z);
        fprintf(TelemFile, "LocalRotAccel=(%.2f,%.2f,%.2f)\n\n",
info.mLocalRotAccel.x,
                info.mLocalRotAccel.y, info.mLocalRotAccel.z);

        // Vehicle status
        fprintf(TelemFile, "Gear=%d RPM=%.1f RevLimit=%.1f\n",
info.mGear, info.mEngineRPM, info.mEngineMaxRPM);
        fprintf(TelemFile, "Water=%.1f Oil=%.1f\n",
info.mEngineWaterTemp, info.mEngineOilTemp);
        fprintf(TelemFile, "ClutchRPM=%.1f\n\n", info.mClutchRPM);

        // Driver input
        fprintf(TelemFile, "UnfilteredThrottle=%.1f%%\n", 100.0 *
info.mUnfilteredThrottle);
        fprintf(TelemFile, "UnfilteredBrake=%.1f%%\n", 100.0 *
info.mUnfilteredBrake);
        fprintf(TelemFile, "UnfilteredSteering=%.1f%%\n", 100.0 *
info.mUnfilteredSteering);
        fprintf(TelemFile, "UnfilteredClutch=%.1f%%\n\n", 100.0 *
info.mUnfilteredClutch);

        // Filtered input
        fprintf(TelemFile, "FilteredThrottle=%.1f%%\n", 100.0 *
info.mFilteredThrottle);
        fprintf(TelemFile, "FilteredBrake=%.1f%%\n", 100.0 *
info.mFilteredBrake);
        fprintf(TelemFile, "FilteredSteering=%.1f%%\n", 100.0 *
info.mFilteredSteering);
        fprintf(TelemFile, "FilteredClutch=%.1f%%\n\n", 100.0 *
info.mFilteredClutch);

        // Misc
        fprintf(TelemFile, "SteeringShaftTorque=%.1f\n",
info.mSteeringShaftTorque);
```

Page | xxii

```
        fprintf(TelemFile, "Front3rdDeflection=%.3f
Rear3rdDeflection=%.3f\n\n",
                info.mFront3rdDeflection, info.mRear3rdDeflection);

        // Aerodynamics
        fprintf(TelemFile, "FrontWingHeight=%.3f FrontRideHeight=%.3f
RearRideHeight=%.3f\n",
                info.mFrontWingHeight, info.mFrontRideHeight,
info.mRearRideHeight);
        fprintf(TelemFile, "Drag=%.1f FrontDownforce=%.1f
RearDownforce=%.1f\n\n", info.mDrag, info.mFrontDownforce,
                info.mRearDownforce);

        // Other
        fprintf(TelemFile, "Fuel=%.1f ScheduledStops=%d Overheating=%d
Detached=%d\n", info.mFuel,
                info.mScheduledStops, info.mOverheating,
info.mDetached);

        fprintf(TelemFile, "Dents=(%d,%d,%d,%d,%d,%d,%d,%d)\n\n",
info.mDentSeverity[0], info.mDentSeverity[1],
                info.mDentSeverity[2], info.mDentSeverity[3],
info.mDentSeverity[4], info.mDentSeverity[5],
                info.mDentSeverity[6], info.mDentSeverity[7]);

        fprintf(TelemFile, "LastImpactET=%.1f Mag=%.1f,
Pos=(%.1f,%.1f,%.1f)\n\n", info.mLastImpactET,
                info.mLastImpactMagnitude,  info.mLastImpactPos.x,
info.mLastImpactPos.y, info.mLastImpactPos.z );

        // Wheels
        for( long i = 0; i < 4; ++i )
        {
            const TelemWheelV01 &wheel = info.mWheel[i];
            fprintf(TelemFile, "Wheel=%s\n",
(i==0)?"FrontLeft":(i==1)?"FrontRight":(i==2)?"RearLeft":"RearRight");

            fprintf(TelemFile, " SuspensionDeflection=%.3f
RideHeight=%.3f\n", wheel.mSuspensionDeflection,
                    wheel.mRideHeight );

            fprintf(TelemFile, " SuspForce=%.1f BrakeTemp=%.1f
BrakePressure=%.3f\n", wheel.mSuspForce,
                    wheel.mBrakeTemp, wheel.mBrakePressure );
```

Page | xxiii

```
            fprintf(TelemFile, " TelemFilerwardRotation=%.1f
Camber=%.3f\n", -wheel.mRotation, wheel.mCamber );

            fprintf(TelemFile, " LateralPatchVel=%.2f
LongitudinalPatchVel=%.2f\n", wheel.mLateralPatchVel,
                    wheel.mLongitudinalPatchVel );

            fprintf(TelemFile, " LateralGroundVel=%.2f
LongitudinalGroundVel=%.2f\n", wheel.mLateralGroundVel,
                    wheel.mLongitudinalGroundVel );

            fprintf(TelemFile, " LateralForce=%.1f
LongitudinalForce=%.1f\n", wheel.mLateralForce,
                    wheel.mLongitudinalForce );

            fprintf(TelemFile, " TireLoad=%.1f GripFract=%.3f
TirePressure=%.1f\n", wheel.mTireLoad,
                    wheel.mGripFract, wheel.mPressure );

            fprintf(TelemFile, " TireTemp(l/c/r)=%.1f/%.1f/%.1f\n",
wheel.mTemperature[0],
                    wheel.mTemperature[1], wheel.mTemperature[2] );

            fprintf(TelemFile, " Wear=%.3f TerrainName=%s
SurfaceType=%d\n", wheel.mWear,
                    wheel.mTerrainName, wheel.mSurfaceType );

            fprintf(TelemFile, " Flat=%d Detached=%d\n\n", wheel.mFlat,
wheel.mDetached );
        }

        // Compute some auxiliary info based on the above
        TelemVect3 forwardVector = { -info.mOri[0].z, -info.mOri[1].z,
-info.mOri[2].z };
        TelemVect3    leftVector =
{  info.mOri[0].x,  info.mOri[1].x,  info.mOri[2].x };

        // These are normalized vectors, and remember that our world Y
coordinate is up.  So you can
        // determine the current pitch and roll (w.r.t. the world x-z
plane) as follows:
        const double pitch = atan2(forwardVector.y,
sqrt((forwardVector.x * forwardVector.x) +
                                   (forwardVector.z *
forwardVector.z)));
```

Page | xxiv

```
        const double  roll = atan2(leftVector.y, sqrt((leftVector.x *
leftVector.x) +
                                    (leftVector.z * leftVector.z)));

        const double radsToDeg = 57.296;
        fprintf(TelemFile, "Pitch = %.1f deg, Roll = %.1f deg\n", pitch
* radsToDeg, roll * radsToDeg);

        const double metersPerSec = sqrt( ( info.mLocalVel.x *
info.mLocalVel.x ) +
                                          ( info.mLocalVel.y *
info.mLocalVel.y ) +
                                          ( info.mLocalVel.z *
info.mLocalVel.z ) );
        fprintf(TelemFile, "Speed = %.1f KPH, %.1f MPH\n\n",
metersPerSec * 3.6, metersPerSec * 2.237 );

        if (info.mElectricBoostMotorState != 0)
        {
            fprintf( TelemFile, "ElectricBoostMotor:");
            char const* const states[] = {"N/A", "Inactive",
"Propulsion", "Regeneration"};
            fprintf( TelemFile, " State = %s\n",
states[info.mElectricBoostMotorState]);
            fprintf( TelemFile, " Torque = %g nm\n",
info.mElectricBoostMotorTorque);
            fprintf( TelemFile, " RPM = %g\n",
info.mElectricBoostMotorRPM);
            fprintf( TelemFile, " Motor Temperature = %g C\n",
info.mElectricBoostMotorTemperature);

            if (info.mElectricBoostMotorTemperature != 0) {
                fprintf( TelemFile, " Water Temperature = %g C\n",
info.mElectricBoostWaterTemperature);
            }
        }

        ForceFeedback;

        // Close file
        fclose( TelemFile );
    }
}
```

Page | xxv

```
log("a", "ending telemetry\n");  //  Records End of Telemetry Data
Stream into Log
}
//  Finished Code & Comments


/*_____________________________________________________________________
_____________________________________
╔═══════════╦═══════════════════╗
║   Function  ║ UpdateTelemetry ║
╠═══════════╬═══════════════════╩═══════════════════════════════════════
═══════════════════════════════════╗
║ Description ║ Old approach to transmitting data via UDP protocal -
Kept for posterity                      ║
╠═══════════╬═══════════════════════════════════════════════════════════
═══════════════════════════════════╣
║  Parameters ║ TelemInfoV01 &info: Enables fetching of Telemetry
Data                                    ║
╚═══════════╩═══════════════════════════════════════════════════════════
═══════════════════════════════════╝*/
/*void UDPlugin::UpdateTelemetry(const TelemInfoV01 &info) {
log("a", "starting telemetry\n"); // Records Telemetry start into Log

StartStream();
StreamData((char *)&type_telemetry, sizeof(char));
StreamData((char *)&info.mGear, sizeof(long));

StreamData((char *)&info.mEngineRPM, sizeof(double));
StreamData((char *)&info.mEngineMaxRPM, sizeof(double));
StreamData((char *)&info.mEngineWaterTemp, sizeof(double));
StreamData((char *)&info.mEngineOilTemp, sizeof(double));
StreamData((char *)&info.mClutchRPM, sizeof(double));
StreamData((char *)&info.mOverheating, sizeof(bool));
StreamData((char *)&info.mFuel, sizeof(double));

StreamData((char *)&info.mPos.x, sizeof(double));
StreamData((char *)&info.mPos.y, sizeof(double));
StreamData((char *)&info.mPos.z, sizeof(double));

//  Data output in order:  pitch, yaw, roll, Sway/lateral acceleration,
Surge/Longitudinal Acceleration, MPH
TelemVect3 forwardVector = { -info.mOri[0].z, -info.mOri[1].z, -
info.mOri[2].z };
TelemVect3 upVector =
{  info.mOri[0].y,  info.mOri[1].y,  info.mOri[2].y };
```

P a g e | **xxvi**

```
TelemVect3 leftVector =
{  info.mOri[0].x,  info.mOri[1].x,  info.mOri[2].x };

const double radsToDeg = 57.296;  // Radians to Degree conversion

//  Calculating and streaming pitch (Order 1)
const double pitch = atan2( forwardVector.y, sqrt( ( forwardVector.x *
forwardVector.x) + ( forwardVector.z * forwardVector.z ) ) );
const double pitch_out = pitch * radsToDeg;
StreamData((char *)&pitch_out, sizeof(double));

//  Calculating and streaming Yaw (Order 2)
const double yaw = atan2(info.mOri[0].z, info.mOri[2].z);
const double yaw_out = yaw * radsToDeg;
StreamData((char *)&yaw_out, sizeof(double));

//  Calculating and streaming Yaw (Order 3)
const double  roll = atan2(leftVector.y, sqrt(
(leftVector.x*leftVector.x) + (leftVector.z * leftVector.z )));
const double roll_out = roll * radsToDeg;
StreamData((char *)&roll_out, sizeof(double));

//  Calculating and streaming Yaw (Order 4)
const double SwayAcc = -info.mLocalAccel.x;
StreamData((char *)&SwayAcc, sizeof(double));

//  Calculating and streaming Yaw (Order 5)
const double SurgeAcc = (info.mLocalAccel.z * -1);
StreamData((char *)&SurgeAcc, sizeof(double));

//  Calculating and streaming Speed (Order 6)
const double metersPerSec = sqrt((info.mLocalVel.x * info.mLocalVel.x)
+    (info.mLocalVel.y * info.mLocalVel.y) + (info.mLocalVel.z *
info.mLocalVel.z));
StreamData((char *)&metersPerSec, sizeof(double));

roll * radsToDeg,  pitch * radsToDeg, info.mLocalAccel.y, yaw *
radsToDeg, - info.mLocalAccel.x, - info.mLocalAccel.z);
//
Roll,               Pitch,              HeaveAcc,           Yaw,
         SwayAcc,            SurgeAcc

StreamData((char *)&info.mLapStartET, sizeof(double));
StreamData((char *)&info.mLapNumber, sizeof(long));
```

Page | xxvii

```
StreamData((char *)&info.mUnfilteredThrottle, sizeof(double));
StreamData((char *)&info.mUnfilteredBrake, sizeof(double));
StreamData((char *)&info.mUnfilteredSteering, sizeof(double));
StreamData((char *)&info.mUnfilteredClutch, sizeof(double));

StreamData((char *)&info.mLastImpactET, sizeof(double));
StreamData((char *)&info.mLastImpactMagnitude, sizeof(double));
StreamData((char *)&info.mLastImpactPos.x, sizeof(double));
StreamData((char *)&info.mLastImpactPos.y, sizeof(double));
StreamData((char *)&info.mLastImpactPos.z, sizeof(double));
for (long i = 0; i < 8; i++) {
    StreamData((char *)&info.mDentSeverity[i], sizeof(byte));
}

for (long i = 0; i < 4; i++) {
    const TelemWheelV01 &wheel = info.mWheel[i];
    StreamData((char *)&wheel.mDetached, sizeof(bool));
    StreamData((char *)&wheel.mFlat, sizeof(bool));
    StreamData((char *)&wheel.mBrakeTemp, sizeof(double));
    StreamData((char *)&wheel.mPressure, sizeof(double));
    StreamData((char *)&wheel.mRideHeight, sizeof(double));
    StreamData((char *)&wheel.mTemperature[0], sizeof(double));
    StreamData((char *)&wheel.mTemperature[1], sizeof(double));
    StreamData((char *)&wheel.mTemperature[2], sizeof(double));
    StreamData((char *)&wheel.mWear, sizeof(double));
}
EndStream();

log("a", "ending telemetry\n");  //  Records End of Telemetry Data
Stream into Log
}*/
//  Alternative Telemetry Code - Kept for Posterity


/*____________________________________________________________________
_________________________________________
╔═════════════╦═══════════════╗
║   Function  ║ UpdateScoring ║
╠═════════════╬═══════════════╩═══════════════════════════════════════
══════════════════════════════════════╗
║ Description ║ Used with old UDP code - Kept for
posterity                                                       ║
╠═════════════╬═══════════════════════════════════════════════════════
══════════════════════════════════════╣
║  Parameters ║ ScoringInfoV01 &info: Enables extraction of Scoring
Data                                        ║
```

Page | xxviii

```
╚═══════════╝╚═══════════════════════════════════════════════════════════════
═══════════════════════════════════════╝*/
void UDPlugin::UpdateScoring(const ScoringInfoV01 &info) {
// //log("a", "starting update");
// StartStream();
// StreamData((char *)&type_scoring, sizeof(char));

// // session data (changes mostly with changing sessions)
// StreamString((char *)&info.mTrackName, 64);
// StreamData((char *)&info.mSession, sizeof(long));

// // event data (changes continuously)
// StreamData((char *)&info.mCurrentET, sizeof(double));
// StreamData((char *)&info.mEndET, sizeof(double));
// StreamData((char *)&info.mLapDist, sizeof(double));
// StreamData((char *)&info.mNumVehicles, sizeof(long));

// StreamData((char *)&info.mGamePhase, sizeof(byte));
// StreamData((char *)&info.mYellowFlagState, sizeof(byte));
// StreamData((char *)&info.mSectorFlag[0], sizeof(byte));
// StreamData((char *)&info.mSectorFlag[1], sizeof(byte));
// StreamData((char *)&info.mSectorFlag[2], sizeof(byte));
// StreamData((char *)&info.mStartLight, sizeof(byte));
// StreamData((char *)&info.mNumRedLights, sizeof(byte));

// // scoring data (changes with new sector times)
// for (long i = 0; i < info.mNumVehicles; i++) {
//  VehicleScoringInfoV01 &vinfo = info.mVehicle[i];
//  StreamData((char *)&vinfo.mPos.x, sizeof(double));
//  StreamData((char *)&vinfo.mPos.z, sizeof(double));
//  StreamData((char *)&vinfo.mPlace, sizeof(char));
//  StreamData((char *)&vinfo.mLapDist, sizeof(double));
//  StreamData((char *)&vinfo.mPathLateral, sizeof(double));
//  const double metersPerSec = sqrt((vinfo.mLocalVel.x *
vinfo.mLocalVel.x) +
//      (vinfo.mLocalVel.y * vinfo.mLocalVel.y) +
//      (vinfo.mLocalVel.z * vinfo.mLocalVel.z));
//  StreamData((char *)&metersPerSec, sizeof(double));
//  StreamString((char *)&vinfo.mVehicleName, 64);
//  StreamString((char *)&vinfo.mDriverName, 32);
//  StreamString((char *)&vinfo.mVehicleClass, 32);
//  StreamData((char *)&vinfo.mTotalLaps, sizeof(short));
//  StreamData((char *)&vinfo.mBestSector1, sizeof(double));
//  StreamData((char *)&vinfo.mBestSector2, sizeof(double));
//  StreamData((char *)&vinfo.mBestLapTime, sizeof(double));
```

Page | xxix

```
//  StreamData((char *)&vinfo.mLastSector1, sizeof(double));
//  StreamData((char *)&vinfo.mLastSector2, sizeof(double));
//  StreamData((char *)&vinfo.mLastLapTime, sizeof(double));
//  StreamData((char *)&vinfo.mCurSector1, sizeof(double));
//  StreamData((char *)&vinfo.mCurSector2, sizeof(double));
//  StreamData((char *)&vinfo.mTimeBehindLeader, sizeof(double));
//  StreamData((char *)&vinfo.mLapsBehindLeader, sizeof(long));
//  StreamData((char *)&vinfo.mTimeBehindNext, sizeof(double));
//  StreamData((char *)&vinfo.mLapsBehindNext, sizeof(long));
//  StreamData((char *)&vinfo.mNumPitstops, sizeof(short));
//  StreamData((char *)&vinfo.mNumPenalties, sizeof(short));
//  StreamData((char *)&vinfo.mInPits, sizeof(bool));
//  StreamData((char *)&vinfo.mSector, sizeof(char));
//  StreamData((char *)&vinfo.mFinishStatus, sizeof(char));
// }
// StreamVarString((char *)info.mResultsStream);
// EndStream();
// //log("a", "ending update\n");
}
//  Alternative Scoring Code - Kept for Posterity

/*____________________________________________________________________
_____________________________________
╔═══════════╦═══════════════════╗
║  Function ║ UpdateGraphics    ║
╠═══════════╬═══════════════════╩═══════════════════════════════════
═══════════════════════════════════╗
║ Description ║ Not Implemented fully - Not needed for
now                                                    ║
╠═══════════╬═══════════════════════════════════════════════════════
═══════════════════════════════════╣
║  Parameters ║ GraphicsInfoV01 &info: Enables extraction of Graphics
Data                                ║
╚═══════════╩═══════════════════════════════════════════════════════
═══════════════════════════════════╝*/
//  void ExampleInternalsPlugin::UpdateGraphics( const GraphicsInfoV01
&info )
//  {
//      FILE *TelemFile = fopen( "GraphicsOutput.txt", "a" );
//      if( TelemFile != NULL )
//      {
//      // Print Graphics Info
//          fprintf(TelemFile, "CamPos=(%.1f,%.1f,%.1f)\n",
info.mCamPos.x, info.mCamPos.y, info.mCamPos.z);
```

Page | xxx

```
//          fprintf(TelemFile, "CamOri[0]=(%.1f,%.1f,%.1f)\n",
info.mCamOri[0].x, info.mCamOri[0].y, info.mCamOri[0].z);
//          fprintf(TelemFile, "CamOri[1]=(%.1f,%.1f,%.1f)\n",
info.mCamOri[1].x, info.mCamOri[1].y, info.mCamOri[1].z);
//          fprintf(TelemFile, "CamOri[2]=(%.1f,%.1f,%.1f)\n",
info.mCamOri[2].x, info.mCamOri[2].y, info.mCamOri[2].z);
//          fprintf(TelemFile, "HWND=%d\n", info.mHWND );
//          fprintf(TelemFile, "Ambient Color=(%.1f,%.1f,%.1f)\n\n",
info.mAmbientRed, info.mAmbientGreen, info.mAmbientBlue);
//          // Close file
//          fclose(TelemFile);
//      }
//  }
//  Finished Code & Comments - Not needed so not implemented fully

//═══════════════════════════════════════ Alt Data Streaming
═══════════════════════════════════════
/*_____________________________________________________________________
_____________________________________
Functions for streaming data
╔═══════════╦═══════════╗
║  Function ║ StartStream ║
╠═══════════╬═══════════╩═══════════════════════════════════════════════
═══════════════════════════════════╗
║ Description ║ Used with old UDP code - Kept for
posterity                                                    ║
╚═══════════╩═══════════════════════════════════════════════════════════
═══════════════════════════════════╝*/
void UDPlugin::StartStream() {

//  Initialize data packet and sequence number
data_packet = 0;
data_sequence++;

//  Populate data array with version, packet number, and sequence
number
data[0] = data_version;                             //  Version of the
data stream
data[1] = data_packet;                              //  Packet number
memcpy(&data[2], &data_sequence, sizeof(short));    //  Sequence number

//  Set data offset for further data population
data_offset = 4;    //  Offset for subsequent data writing
}
```

Page | xxxi

```
//  Finished Code & Comments - Data Streaming for Alternative Data
streaming method


/*_____________________________________________________________________
_____________________________________________
╔═══════════════╦═══════════════╗
║   Function    ║ StreamData    ║
╠═══════════════╬═══════════════╩═══════════════════════════════════════
═══════════════════════════════════════╗
║ Description   ║ Used with old UDP code - Kept for
posterity                                                    ║
╠═══════════════╬═══════════════════════════════════════════════════════
═══════════════════════════════════════╣
║  Parameters   ║ *data_ptr: Data
pointer
  ║
║               ║ length: Length of
data
     ║
╚═══════════════╩═══════════════════════════════════════════════════════
═══════════════════════════════════════╝*/
void UDPlugin::StreamData(char *data_ptr, int length) {
int i;

//  Iterate through the data_ptr and copy it to the data array
for (i = 0; i < length; i++) {
    //  Check if data array is full (reached the maximum packet size)
    if (data_offset + i == 512) {

        //  Send the current data packet
        sendto(s, data, 512, 0, (struct sockaddr *) &sad, sizeof(struct
sockaddr));

        //  Increment packet number and reset data array for the next
packet
        data_packet++;
        data[0] = data_version;
        data[1] = data_packet;
        memcpy(&data[2], &data_sequence, sizeof(short));
        data_offset = 4;

        //  Increment packet number and reset data array for the next
packet
        length = length - i;
        data_ptr += i;
```

Page | xxxii

```
        i = 0;  //  Increment packet number and reset data array for
the next packet
    }
    data[data_offset + i] = data_ptr[i];    // Copy data from data_ptr
to data array
}
data_offset = data_offset + length;         // Update data_offset for
the next data population
}
//  Finished Code & Comments - Data Streaming for Alternative Data
streaming method


/*_____________________________________________________________________
_____________________________________
╔═══════════════╦═══════════════════╗
║   Function    ║ StreamVarString ║
╠═══════════════╬═══════════════════╩═══════════════════════════════════
═══════════════════════════════════╗
║ Description   ║ Used with old UDP code - Kept for
posterity                                                  ║
╠═══════════════╬═══════════════════════════════════════════════════════
═══════════════════════════════════╣
║  Parameters   ║ *data_ptr: Data
Pointer
    ║
╚═══════════════╩═══════════════════════════════════════════════════════
═══════════════════════════════════╝*/
void UDPlugin::StreamVarString(char *data_ptr) {
int i = 0;
while (data_ptr[i] != 0) {              //  Find the length of the
variable-length string
    i++;
}
//  Stream the length of the string followed by the string data
StreamData((char *)&i, sizeof(int));    // Stream the length of the
string
StreamString(data_ptr, i);              // Stream the string data
}
//  Finished Code & Comments - Data Streaming for Alternative Data
streaming method


/*_____________________________________________________________________
_____________________________________
╔═══════════════╦═══════════════════╗
```

P a g e | xxxiii

```
║   Function    ║ StreamString ║
╠═══════════════╬══════════════╩══════════════════════════════════════════════════════
══════════════════════════════════════╣
║ Description   ║ Used with old UDP code - Kept for
posterity                                                                ║
╠═══════════════╬═════════════════════════════════════════════════════════════════════
══════════════════════════════════════╣
║  Parameters   ║ *data_ptr: Data
Pointer
    ║
║               ║ length: Length of
Data
   ║
╚═══════════════╩═════════════════════════════════════════════════════════════════════
══════════════════════════════════════╝*/
void UDPlugin::StreamString(char *data_ptr, int length) {
int i;
for (i = 0; i < length; i++) {      //  Iterate through the string
characters
    if (data_offset + i == 512) {   //  Check if data array is full
(reached the maximum packet size)

        //  Send the current data packet
        sendto(s, data, 512, 0, (struct sockaddr *) &sad, sizeof(struct
sockaddr));

        //  Increment packet number and reset data array for the next
packet
        data_packet++;
        data[0] = data_version;
        data[1] = data_packet;
        memcpy(&data[2], &data_sequence, sizeof(short));
        data_offset = 4;

        //  Update remaining length and data pointer
        length = length - i;
        data_ptr += i;
        i = 0;  //  Reset i to 0 for the next iteration
    }
    //  Copy character from data_ptr to data array
    data[data_offset + i] = data_ptr[i];
    if (data_ptr[i] == 0) { //  Check for end of string

        //  Move data_offset to the end of the string and return
        data_offset = data_offset + i + 1;
```

Page | xxxiv

```
        return;
    }
}
data_offset = data_offset + length; //  Update data_offset for the next
data population
}
//  Finished Code & Comments - Data Streaming for Alternative Data
streaming method


/*_____________________________________________________________________
_____________________________________________
╔═══════════════╦═══════════╗
║   Function    ║ EndStream ║
╠═══════════════╬═══════════╩═══════════════════════════════════════════
═══════════════════════════════════════╗
║ Description   ║ Used with old UDP code - Kept for
posterity                                                    ║
╚═══════════════╩═══════════════════════════════════════════════════════
═══════════════════════════════════════╝*/
void UDPlugin::EndStream() {
//      Check if there is any data in the data array to be sent
if (data_offset > 4) {
    //  Send the remaining data as a packet
    sendto(s, data, data_offset, 0, (struct sockaddr *) &sad,
sizeof(struct sockaddr));
}
}
//  Finished Code & Comments - Data Streaming for Alternative Data
streaming method

//_____________________________________________________________________
_____________________________________________
//═══════════════════════════════════════ Extra Features
═══════════════════════════════════════════════
/*_____________________________________________________________________
_____________________________________________
╔═══════════════╦═══════════════╗
║   Function    ║ CheckHWControl ║
╠═══════════════╬═══════════════╩═══════════════════════════════════════
═══════════════════════════════════════╗
║ Description ║ Checks if Hardware control is enabled. Enabled: return
= True / Disabled: return = False    ║
╠═══════════════╬═══════════════════════════════════════════════════════
═══════════════════════════════════════╣
```

Page | xxxv

```
╟  Parameters ╟ controlName: Name of hardware being
controlled                                                    ╟
╟             ╟ &fRetVal: return
value
   ╟
╚═══════════╩═══════════════════════════════════════════════════════════
═══════════════════════════════════╝*/
bool UDPlugin::CheckHWControl( const char * const controlName, double
&fRetVal ) {

// // only if enabled, of course
// if( !mEnabled )
//  return( false );

// // Note that incoming value is the game's computation, in case
you're interested.

// // No control allowed over actual vehicle inputs - Due to cheating
possibility
// // However, you can still look at the values.

// // Note: since the game calls this function every frame for every
available control, you might consider
// // doing a binary search if you are checking more than 7 or 8
strings, just to keep the speed up.
// if( _stricmp( controlName, "LookLeft" ) == 0 )
// {
//  const double headSwitcheroo = fmod( mET, 2.0 );
//  if( headSwitcheroo < 0.5 )
//      fRetVal = 1.0;
//  else
//      fRetVal = 0.0;
//  return( true );
// }
// else if( _stricmp( controlName, "LookRight" ) == 0 )
// {
//  const double headSwitcheroo = fmod( mET, 2.0 );
//  if((headSwitcheroo > 1.0) && (headSwitcheroo < 1.5 )) {
//      fRetVal = 1.0;
//  }
//  else {
//      fRetVal = 0.0;
//  return( true );
//  }
// }
```

Page | xxxvi

```
return( false );
}
//  Finished Code & Comments

/*_____________________________________________________________________
_____________________________________
╔═══════════════╦═══════════════════╗
║   Function    ║ ForceFeedback     ║
╠═══════════════╬═══════════════════╩═══════════════════════════════════
═══════════════════════════════════╗
║ Description   ║ Enables reading & manipulation of Force
Feedback                                                 ║
╠═══════════════╬═══════════════════════════════════════════════════════
═══════════════════════════════════╣
║  Parameters   ║ &forceValue: Value of FFB torque force
value                                                    ║
╚═══════════════╩═══════════════════════════════════════════════════════
═══════════════════════════════════╝*/
bool UDPlugin::ForceFeedback( double &forceValue )
{
// // CHANGE COMMENTS TO ENABLE FORCE EXAMPLE
// return( false );

// bounds are -11500 to 11500 ...
// forceValue = 11500.0 * sinf( mET );


WriteToFiles( "a", "--FFB Output--\n");

//  Below records Telemetry data to Telemetry.txt file
//  Open or create a file named "Telemetry.txt" in append mode ("a")
FILE *TelemFile = fopen("Telemetry.txt", "a");
if  (TelemFile != NULL)
{


        fprintf(TelemFile, "Force Feed Back = %s \n", forceValue);
        // Close file


    fclose(TelemFile);
}
```

Page | xxxvii

```
return( false );
}
//  Finished Code & Comments

/*_____________________________________________________________________
_____________________________________
╔═══════════════╦═══════════════════════╗
║   Function    ║ RequestCommentary     ║
╠═══════════════╬═══════════════════════╩═══════════════════════════════
═══════════════════════════════════════╗
║ Description   ║ Enables manually triggering game
commentary                                                    ║
╠═══════════════╬═══════════════════════════════════════════════════════
═══════════════════════════════════════╣
║  Parameters   ║ CommentaryRequestInfoV01 &info: Enables use and data
acess of commentary related variables   ║
╚═══════════════╩═══════════════════════════════════════════════════════
═══════════════════════════════════════╝*/
bool UDPlugin::RequestCommentary( CommentaryRequestInfoV01 &info )
{
// This function requests commentary information to be provided to the
plugin.

// COMMENT OUT TO ENABLE EXAMPLE
return (false); // Disable this function and return false by default.

// Check if the plugin is enabled
if( !mEnabled )
    return( false ); // If not enabled, return false.

// Note: function is called twice per second

// Trigger a green flag event every 20 seconds
const double timeMod20 = fmod( mET, 20.0 ); // Calculate the remainder
of mET divided by 20
if( timeMod20 > 19.0 ) // If the remainder is greater than 19, it's
almost 20 seconds
{
    // Populate the CommentaryRequestInfoV01 structure with green flag
event data
    strcpy( info.mName, "GreenFlag" ); // Set the event name to
"GreenFlag"
    info.mInput1 = 0.0; // Set input 1 value to 0.0
    info.mInput2 = 0.0; // Set input 2 value to 0.0
    info.mInput3 = 0.0; // Set input 3 value to 0.0
```

Page | xxxviii

```
    info.mSkipChecks = true; // Skip checks for this event
    return true; // Return true to indicate that commentary information
is provided
}

// If no event triggered, return false
return false;
}

/*_____________________________________________________________________
_____________________________________________
╔═══════════════╦═══════════════════════════╗
║   Function    ║ WantsToDisplayMessage     ║
╠═══════════════╬═══════════════════════════╩═══════════════════════════
═══════════════════════════════════════════╗
║ Description   ║ Enables custom message display in-
game.                                                                 ║
╠═══════════════╬═══════════════════════════════════════════════════════
═══════════════════════════════════════════╣
║  Parameters   ║ MessageInfoV01 &msgInfo: Enables access to variables
and data regarding in-game messaging.   ║
╚═══════════════╩═══════════════════════════════════════════════════════
═══════════════════════════════════════════╝*/
bool UDPlugin::WantsToDisplayMessage(MessageInfoV01 &msgInfo) {return
false;}
//  Finished Code & Comments

/*_____________________________________________________________________
_____________________________________________
╔═══════════════╦═══════════════════════════╗
║   Function    ║ WantsToViewVehicle        ║
╠═══════════════╬═══════════════════════════╩═══════════════════════════
═══════════════════════════════════════════╗
║ Description   ║ Allows control of the in-game player
camera.                                                 ║
╠═══════════════╬═══════════════════════════════════════════════════════
═══════════════════════════════════════════╣
║  Parameters   ║ CameraControlInfoV01 &camControl: Enables access to
variables and data related to the camera║
╚═══════════════╩═══════════════════════════════════════════════════════
═══════════════════════════════════════════╝*/
unsigned char UDPlugin::WantsToViewVehicle(CameraControlInfoV01
&camControl) {return 0;}
//  Finished Code & Comments
```

Page | xxxix

```
//=============================================== Unused Functions
================================================
void UDPlugin::Load(){}   //  Not Currently Used - Kept for posterity
void UDPlugin::Unload(){} //  Not Currently Used - Kept for posterity
```

UDPlugin.hpp

```
#pragma once

// #ifndef _INTERNALS_EXAMPLE_H
// #define _INTERNALS_EXAMPLE_H

#include "InternalsPlugin.hpp"

// This is used for the app to use the plugin for its intended purpose
class UDPlugin : public InternalsPluginV06  // REMINDER: exported
function GetPluginVersion() should return 1 if you are deriving from
this InternalsPluginV01, 2 for InternalsPluginV02, etc.
{
 public:
  UDPlugin() {}     // Constructor
  ~UDPlugin() {}    // Destructor

  // These are the functions derived from base class InternalsPlugin
  // that can be implemented.
  void Startup(long version);  // game startup
  void Shutdown();               // game shutdown

  void Load();                   // scene/track load
  void Unload();                 // scene/track unload

  void EnterRealtime();          // entering realtime
  void ExitRealtime();           // exiting realtime

  void StartSession();           // session has started
  void EndSession();             // session has ended

  // GAME OUTPUT
  long WantsTelemetryUpdates() {return(1); }            // 1 = ENABLE
TELEMETRY EXAMPLE!
  void UpdateTelemetry(const TelemInfoV01 &info);

  bool WantsGraphicsUpdates() {return( false ); }       // TRUE =
ENABLE GRAPHICS EXAMPLE!
  //  Extended Game Output
```

Page | xl

```
  // void UpdateGraphics( const GraphicsInfoV01 &info );
  virtual void UpdateGraphics(const GraphicsInfoV02 &info)          {}
// update plugin with extended graphics info

  // GAME INPUT
  bool HasHardwareInputs() { return( false ); } // CHANGE TO TRUE TO
ENABLE HARDWARE EXAMPLE!
  void UpdateHardware( const double fDT ) { mET += fDT; } // update the
hardware with the time between frames
  void EnableHardware() { mEnabled = true; }             // message
from game to enable hardware
  void DisableHardware() { mEnabled = false; }           // message
from game to disable hardware

  // See if the plugin wants to take over a hardware control.  If the
plugin takes over the
  // control, this method returns true and sets the value of the double
pointed to by the
  // second arg.  Otherwise, it returns false and leaves the double
unmodified.
  bool CheckHWControl( const char * const controlName, double &fRetVal
);

  bool ForceFeedback( double &forceValue );  // SEE FUNCTION BODY TO
ENABLE FORCE EXAMPLE

  // SCORING OUTPUT
  bool WantsScoringUpdates() { return(true); } // TRUE = ENABLE
SCORING!
  void UpdateScoring( const ScoringInfoV01 &info );

  // COMMENTARY INPUT
  bool RequestCommentary( CommentaryRequestInfoV01 &info );  // SEE
FUNCTION BODY TO ENABLE COMMENTARY EXAMPLE

  // MESSAGE BOX INPUT
  bool WantsToDisplayMessage(MessageInfoV01 &msgInfo);

  // CAMERA CONTROL
  unsigned char WantsToViewVehicle(CameraControlInfoV01 &camControl);

  // ERROR FEEDBACK
  virtual void Error(const char * const msg); // Called with
explanation message if there was some sort of error in a plugin
callback
```

Page | xli

```
  // VIDEO EXPORT (sorry, no example code at this time)
  virtual bool WantsVideoOutput() { return(false);
}                              // whether we want to export video
  virtual bool VideoOpen(const char * const szFilename, float fQuality,
unsigned short usFPS, unsigned long fBPS,
    unsigned short usWidth, unsigned short usHeight, char *cpCodec =
NULL) {
    return(false);
  }
        // open video output file
  virtual void VideoClose()
{}                                                  // close video
output file
  virtual void VideoWriteAudio(const short *pAudio, unsigned int
uNumFrames) {} // write some audio info
  virtual void VideoWriteImage(const unsigned char *pImage)
{}                  // write video image

 private:

  void WriteToFiles( const char * const openStr, const char * const msg
);
  double mET;  // Event Time | needed for the hardware example
  bool mEnabled; // needed for the hardware example

  // constant types
  static const char type_telemetry = 1;
  static const char type_scoring = 2;

  void StartStream();
  void StreamData(char *data_ptr, int length);
  void StreamString(char *data_ptr, int length);
  void StreamVarString(char *data_ptr);
  void EndStream();
  void log(const char * const openStr, const char *msg);

  SOCKET s; // socket to send data to
  struct sockaddr_in sad;
  char data[512];
  int data_offset;
  byte data_version;
  byte data_packet;
  short data_sequence;
  char hostname[256];
```

Page | xlii

```
  int port;
};


// #endif // _INTERNALS_EXAMPLE_H
```

InternalsPlugin.hpp

```
//############################################################################
######
//#
    #
//# Module: Header file for internals
plugin                                  #
//#
    #
//# Description: Interface declarations for internals
plugin                #
//#
    #
//# This source code module, and all information, data, and
algorithms      #
//# associated with it, are part of isiMotor Technology
(tm).               #
//#                 PROPRIETARY AND
CONFIDENTIAL                            #
//# Copyright (c) 2017 Studio 397 B.V.  All rights
reserved.               #
//#
    #
//# Change
history:                                                            #
//#   tag.2005.11.29:
created                                             #
//#
    #
//############################################################################
######

#ifndef _INTERNALS_PLUGIN_HPP_
#define _INTERNALS_PLUGIN_HPP_

#include "PluginObjects.hpp"     // base class for plugin objects to
derive from
```

Page | xliii

```
#include <cmath>                // for sqrt()
#include <windows.h>            // for HWND


// rF and plugins must agree on structure packing, so set it explicitly
here ... whatever the current
// packing is will be restored at the end of this include with another
#pragma.
#pragma pack( push, 4 )


//##########################################################################
########
//# Version01 Structures - Structs to retrieve internal info (e.g.
Telemetry) #
//##########################################################################
########

struct TelemVect3
{
  double x, y, z;

  void Set( const double a, const double b, const double c )  { x = a;
y = b; z = c; }

  // Allowed to reference as [0], [1], or [2], instead of .x, .y, or
.z, respectively
        double &operator[]( long i )              { return( ( &x )[ i
] ); }
  const double &operator[]( long i ) const        { return( ( &x )[ i
] ); }
};


struct TelemQuat
{
  double w, x, y, z;

  // Convert this quaternion to a matrix
  void ConvertQuatToMat( TelemVect3 ori[3] ) const
  {
    const double x2 = x + x;
    const double xx = x * x2;
    const double y2 = y + y;
    const double yy = y * y2;
    const double z2 = z + z;
```

Page | xliv

```
    const double zz = z * z2;
    const double xz = x * z2;
    const double xy = x * y2;
    const double wy = w * y2;
    const double wx = w * x2;
    const double wz = w * z2;
    const double yz = y * z2;
    ori[0][0] = (double) 1.0 - ( yy + zz );
    ori[0][1] = xy - wz;
    ori[0][2] = xz + wy;
    ori[1][0] = xy + wz;
    ori[1][1] = (double) 1.0 - ( xx + zz );
    ori[1][2] = yz - wx;
    ori[2][0] = xz - wy;
    ori[2][1] = yz + wx;
    ori[2][2] = (double) 1.0 - ( xx + yy );
  }

  // Convert a matrix to this quaternion
  void ConvertMatToQuat( const TelemVect3 ori[3] )
  {
    const double trace = ori[0][0] + ori[1][1] + ori[2][2] + (double)
1.0;
    if( trace > 0.0625f )
    {
      const double sqrtTrace = sqrt( trace );
      const double s = (double) 0.5 / sqrtTrace;
      w = (double) 0.5 * sqrtTrace;
      x = ( ori[2][1] - ori[1][2] ) * s;
      y = ( ori[0][2] - ori[2][0] ) * s;
      z = ( ori[1][0] - ori[0][1] ) * s;
    }
    else if( ( ori[0][0] > ori[1][1] ) && ( ori[0][0] > ori[2][2] ) )
    {
      const double sqrtTrace = sqrt( (double) 1.0 + ori[0][0] -
ori[1][1] - ori[2][2] );
      const double s = (double) 0.5 / sqrtTrace;
      w = ( ori[2][1] - ori[1][2] ) * s;
      x = (double) 0.5 * sqrtTrace;
      y = ( ori[0][1] + ori[1][0] ) * s;
      z = ( ori[0][2] + ori[2][0] ) * s;
    }
    else if( ori[1][1] > ori[2][2] )
    {
```

Page | xlv

```
      const double sqrtTrace = sqrt( (double) 1.0 + ori[1][1] -
ori[0][0] - ori[2][2] );
      const double s = (double) 0.5 / sqrtTrace;
      w = ( ori[0][2] - ori[2][0] ) * s;
      x = ( ori[0][1] + ori[1][0] ) * s;
      y = (double) 0.5 * sqrtTrace;
      z = ( ori[1][2] + ori[2][1] ) * s;
    }
    else
    {
      const double sqrtTrace = sqrt( (double) 1.0 + ori[2][2] -
ori[0][0] - ori[1][1] );
      const double s = (double) 0.5 / sqrtTrace;
      w = ( ori[1][0] - ori[0][1] ) * s;
      x = ( ori[0][2] + ori[2][0] ) * s;
      y = ( ori[1][2] + ori[2][1] ) * s;
      z = (double) 0.5 * sqrtTrace;
    }
  }
};


struct TelemWheelV01
{
  double mSuspensionDeflection;  // meters
  double mRideHeight;            // meters
  double mSuspForce;             // pushrod load in Newtons
  double mBrakeTemp;             // Celsius
  double mBrakePressure;         // currently 0.0-1.0, depending on
driver input and brake balance; will convert to true brake pressure
(kPa) in future

  double mRotation;              // radians/sec
  double mLateralPatchVel;       // lateral velocity at contact patch
  double mLongitudinalPatchVel;  // longitudinal velocity at contact
patch
  double mLateralGroundVel;      // lateral velocity at contact patch
  double mLongitudinalGroundVel; // longitudinal velocity at contact
patch
  double mCamber;                // radians (positive is left for left-
side wheels, right for right-side wheels)
  double mLateralForce;          // Newtons
  double mLongitudinalForce;     // Newtons
  double mTireLoad;              // Newtons
```

Page | xlvi

```
  double mGripFract;              // an approximation of what fraction
of the contact patch is sliding
  double mPressure;               // kPa (tire pressure)
  double mTemperature[3];         // Kelvin (subtract 273.15 to get
Celsius), left/center/right (not to be confused with
inside/center/outside!)
  double mWear;                   // wear (0.0-1.0, fraction of maximum)
... this is not necessarily proportional with grip loss
  char mTerrainName[16];          // the material prefixes from the TDF
file
  unsigned char mSurfaceType;     // 0=dry, 1=wet, 2=grass, 3=dirt,
4=gravel, 5=rumblestrip, 6=special
  bool mFlat;                     // whether tire is flat
  bool mDetached;                 // whether wheel is detached
  unsigned char mStaticUndeflectedRadius; // tire radius in centimeters

  double mVerticalTireDeflection;// how much is tire deflected from its
(speed-sensitive) radius
  double mWheelYLocation;         // wheel's y location relative to
vehicle y location
  double mToe;                    // current toe angle w.r.t. the
vehicle

  double mTireCarcassTemperature;       // rough average of temperature
samples from carcass (Kelvin)
  double mTireInnerLayerTemperature[3]; // rough average of temperature
samples from innermost layer of rubber (before carcass) (Kelvin)

  unsigned char mExpansion[ 24 ];// for future use
};

// Our world coordinate system is left-handed, with +y pointing up.
// The local vehicle coordinate system is as follows:
//   +x points out the left side of the car (from the driver's
perspective)
//   +y points out the roof
//   +z points out the back of the car
// Rotations are as follows:
//   +x pitches up
//   +y yaws to the right
//   +z rolls to the right
// Note that ISO vehicle coordinates (+x forward, +y right, +z upward)
are
// right-handed.  If you are using that system, be sure to negate any
rotation
```

Page | xlvii

```
// or torque data because things rotate in the opposite direction.  In
other
// words, a -z velocity in rFactor is a +x velocity in ISO, but a -z
rotation
// in rFactor is a -x rotation in ISO!!!

struct TelemInfoV01
{
  //  Wheel Info brought over from TelemWheel
  double mRotation;              // radians/sec

  // Time
  long mID;                      // slot ID (note that it can be re-
used in multiplayer after someone leaves)
  double mDeltaTime;             // time since last update (seconds)
  double mElapsedTime;           // game session time
  long mLapNumber;               // current lap number
  double mLapStartET;            // time this lap was started
  char mVehicleName[64];         // current vehicle name
  char mTrackName[64];           // current track name

  // Position and derivatives
  TelemVect3 mPos;               // world position in meters
  TelemVect3 mLocalVel;          // velocity (meters/sec) in local
vehicle coordinates
  TelemVect3 mLocalAccel;        // acceleration (meters/sec^2) in
local vehicle coordinates

  // Orientation and derivatives
  TelemVect3 mOri[3];            // rows of orientation matrix (use
TelemQuat conversions if desired), also converts local
                                 // vehicle vectors into world X, Y, or
Z using dot product of rows 0, 1, or 2 respectively
  TelemVect3 mLocalRot;          // rotation (radians/sec) in local
vehicle coordinates
  TelemVect3 mLocalRotAccel;     // rotational acceleration
(radians/sec^2) in local vehicle coordinates

  // Vehicle status
  long mGear;                    // -1=reverse, 0=neutral, 1+=forward
gears
  double mEngineRPM;             // engine RPM
  double mEngineWaterTemp;       // Celsius
  double mEngineOilTemp;         // Celsius
  double mClutchRPM;             // clutch RPM
```

Page | xlviii

```
  // Driver input
  double mUnfilteredThrottle;    // ranges  0.0-1.0
  double mUnfilteredBrake;       // ranges  0.0-1.0
  double mUnfilteredSteering;    // ranges -1.0-1.0 (left to right)
  double mUnfilteredClutch;      // ranges  0.0-1.0

  // Filtered input (various adjustments for rev or speed limiting, TC,
ABS?, speed sensitive steering, clutch work for semi-automatic
shifting, etc.)
  double mFilteredThrottle;      // ranges  0.0-1.0
  double mFilteredBrake;         // ranges  0.0-1.0
  double mFilteredSteering;      // ranges -1.0-1.0 (left to right)
  double mFilteredClutch;        // ranges  0.0-1.0

// Misc
  double mSteeringShaftTorque;   // torque around steering shaft (used
to be mSteeringArmForce, but that is not necessarily accurate for
feedback purposes)
  double mFront3rdDeflection;    // deflection at front 3rd spring
  double mRear3rdDeflection;     // deflection at rear 3rd spring

  // Aerodynamics
  double mFrontWingHeight;       // front wing height
  double mFrontRideHeight;       // front ride height
  double mRearRideHeight;        // rear ride height
  double mDrag;                  // drag
  double mFrontDownforce;        // front downforce
  double mRearDownforce;         // rear downforce

  // State/damage info
  double mFuel;                  // amount of fuel (liters)
  double mEngineMaxRPM;          // rev limit
  unsigned char mScheduledStops; // number of scheduled pitstops
  bool  mOverheating;            // whether overheating icon is shown
  bool  mDetached;               // whether any parts (besides wheels)
have been detached
  bool  mHeadlights;             // whether headlights are on
  unsigned char mDentSeverity[8];// dent severity at 8 locations around
the car (0=none, 1=some, 2=more)
  double mLastImpactET;          // time of last impact
  double mLastImpactMagnitude;   // magnitude of last impact
  TelemVect3 mLastImpactPos;     // location of last impact

  // Expanded
```

Page | xlix

```
  double mEngineTorque;          // current engine torque (including
additive torque) (used to be mEngineTq, but there's little reason to
abbreviate it)
  long mCurrentSector;           // the current sector (zero-based)
with the pitlane stored in the sign bit (example: entering pits from
third sector gives 0x80000002)
  unsigned char mSpeedLimiter;   // whether speed limiter is on
  unsigned char mMaxGears;       // maximum forward gears
  unsigned char mFrontTireCompoundIndex;   // index within brand
  unsigned char mRearTireCompoundIndex;    // index within brand
  double mFuelCapacity;          // capacity in liters
  unsigned char mFrontFlapActivated;       // whether front flap is
activated
  unsigned char mRearFlapActivated;        // whether rear flap is
activated
  unsigned char mRearFlapLegalStatus;      // 0=disallowed, 1=criteria
detected but not allowed quite yet, 2=allowed
  unsigned char mIgnitionStarter;          // 0=off 1=ignition
2=ignition+starter

  char mFrontTireCompoundName[18];         // name of front tire
compound
  char mRearTireCompoundName[18];          // name of rear tire
compound

  unsigned char mSpeedLimiterAvailable;    // whether speed limiter is
available
  unsigned char mAntiStallActivated;       // whether (hard) anti-stall
is activated
  unsigned char mUnused[2];                //
  float mVisualSteeringWheelRange;         // the *visual* steering
wheel range

  double mRearBrakeBias;                   // fraction of brakes on
rear
  double mTurboBoostPressure;              // current turbo boost
pressure if available
  float mPhysicsToGraphicsOffset[3];       // offset from static CG to
graphical center
  float mPhysicalSteeringWheelRange;       // the *physical* steering
wheel range

  double mBatteryChargeFraction; // Battery charge as fraction [0.0-
1.0]
```

Page | l

```
  // electric boost motor
  double mElectricBoostMotorTorque; // current torque of boost motor
(can be negative when in regenerating mode)
  double mElectricBoostMotorRPM; // current rpm of boost motor
  double mElectricBoostMotorTemperature; // current temperature of
boost motor
  double mElectricBoostWaterTemperature; // current water temperature
of boost motor cooler if present (0 otherwise)
  unsigned char mElectricBoostMotorState; // 0=unavailable 1=inactive,
2=propulsion, 3=regeneration

  // Future use
  unsigned char mExpansion[111]; // for future use (note that the slot
ID has been moved to mID above)

  // keeping this at the end of the structure to make it easier to
replace in future versions
  TelemWheelV01 mWheel[4];       // wheel info (front left, front
right, rear left, rear right)
};


struct GraphicsInfoV01
{
  TelemVect3 mCamPos;            // camera position
  TelemVect3 mCamOri[3];         // rows of orientation matrix (use
TelemQuat conversions if desired), also converts local
  HWND mHWND;                    // app handle

  double mAmbientRed;
  double mAmbientGreen;
  double mAmbientBlue;
};


struct GraphicsInfoV02 : public GraphicsInfoV01
{
  long mID;                      // slot ID being viewed (-1 if
invalid)

  // Camera types (some of these may only be used for *setting* the
camera type in WantsToViewVehicle())
  //    0  = TV cockpit
  //    1  = cockpit
  //    2  = nosecam
  //    3  = swingman
```

Page | li

```
  //    4  = trackside (nearest)
  //    5  = onboard000
  //       :
  //       :
  // 1004  = onboard999
  // 1005+ = (currently unsupported, in the future may be able to
set/get specific trackside camera)
  long mCameraType;              // see above comments for possible
values

  unsigned char mExpansion[128]; // for future use (possibly camera
name)
};


struct CameraControlInfoV01
{
  // Cameras
  long mID;                      // slot ID to view
  long mCameraType;              // see GraphicsInfoV02 comments for
values

  // Replays (note that these are asynchronous)
  bool mReplayActive;            // This variable is an *input* filled
with whether the replay is currently active (as opposed to realtime).
  bool mReplayUnused;            //
  unsigned char mReplayCommand;  // 0=do nothing, 1=begin, 2=end,
3=rewind, 4=fast backwards, 5=backwards, 6=slow backwards, 7=stop,
8=slow play, 9=play, 10=fast play, 11=fast forward

  bool mReplaySetTime;           // Whether to skip to the following
replay time:
  float mReplaySeconds;          // The replay time in seconds to skip
to (note: the current replay maximum ET is passed into this variable in
case you need it)

  //
  unsigned char mExpansion[120]; // for future use (possibly camera
name & positions/orientations)
};


struct MessageInfoV01
{
  char mText[128];               // message to display
```

Page | lii

```
  unsigned char mDestination;    // 0 = message center, 1 = chat (can
be used for multiplayer chat commands)
  unsigned char mTranslate;      // 0 = do not attempt to translate, 1
= attempt to translate

  unsigned char mExpansion[126]; // for future use (possibly what
color, what font, and seconds to display)
};


struct VehicleScoringInfoV01
{
  long mID;                      // slot ID (note that it can be re-
used in multiplayer after someone leaves)
  char mDriverName[32];          // driver name
  char mVehicleName[64];         // vehicle name
  short mTotalLaps;              // laps completed
  signed char mSector;           // 0=sector3, 1=sector1, 2=sector2
(don't ask why)
  signed char mFinishStatus;     // 0=none, 1=finished, 2=dnf, 3=dq
  double mLapDist;               // current distance around track
  double mPathLateral;           // lateral position with respect to
*very approximate* "center" path
  double mTrackEdge;             // track edge (w.r.t. "center" path)
on same side of track as vehicle

  double mBestSector1;           // best sector 1
  double mBestSector2;           // best sector 2 (plus sector 1)
  double mBestLapTime;           // best lap time
  double mLastSector1;           // last sector 1
  double mLastSector2;           // last sector 2 (plus sector 1)
  double mLastLapTime;           // last lap time
  double mCurSector1;            // current sector 1 if valid
  double mCurSector2;            // current sector 2 (plus sector 1) if
valid
  // no current laptime because it instantly becomes "last"

  short mNumPitstops;            // number of pitstops made
  short mNumPenalties;           // number of outstanding penalties
  bool mIsPlayer;                // is this the player's vehicle

  signed char mControl;          // who's in control: -1=nobody
(shouldn't get this), 0=local player, 1=local AI, 2=remote, 3=replay
(shouldn't get this)
  bool mInPits;                  // between pit entrance and pit exit
(not always accurate for remote vehicles)
```

Page | liii

```
  unsigned char mPlace;          // 1-based position
  char mVehicleClass[32];        // vehicle class

  // Dash Indicators
  double mTimeBehindNext;        // time behind vehicle in next higher
place
  long mLapsBehindNext;          // laps behind vehicle in next higher
place
  double mTimeBehindLeader;      // time behind leader
  long mLapsBehindLeader;        // laps behind leader
  double mLapStartET;            // time this lap was started

  // Position and derivatives
  TelemVect3 mPos;               // world position in meters
  TelemVect3 mLocalVel;          // velocity (meters/sec) in local
vehicle coordinates
  TelemVect3 mLocalAccel;        // acceleration (meters/sec^2) in
local vehicle coordinates

  // Orientation and derivatives
  TelemVect3 mOri[3];            // rows of orientation matrix (use
TelemQuat conversions if desired), also converts local
                                 // vehicle vectors into world X, Y, or
Z using dot product of rows 0, 1, or 2 respectively
  TelemVect3 mLocalRot;          // rotation (radians/sec) in local
vehicle coordinates
  TelemVect3 mLocalRotAccel;     // rotational acceleration
(radians/sec^2) in local vehicle coordinates

  // tag.2012.03.01 - stopped casting some of these so variables now
have names and mExpansion has shrunk, overall size and old data
locations should be same
  unsigned char mHeadlights;     // status of headlights
  unsigned char mPitState;       // 0=none, 1=request, 2=entering,
3=stopped, 4=exiting
  unsigned char mServerScored;   // whether this vehicle is being
scored by server (could be off in qualifying or racing heats)
  unsigned char mIndividualPhase;// game phases (described below) plus
9=after formation, 10=under yellow, 11=under blue (not used)

  long mQualification;           // 1-based, can be -1 when invalid

  double mTimeIntoLap;           // estimated time into lap
```

Page | liv

```
  double mEstimatedLapTime;      // estimated laptime used for 'time
behind' and 'time into lap' (note: this may changed based on vehicle
and setup!?)

  char mPitGroup[24];            // pit group (same as team name unless
pit is shared)
  unsigned char mFlag;           // primary flag being shown to vehicle
(currently only 0=green or 6=blue)
  bool mUnderYellow;             // whether this car has taken a full-
course caution flag at the start/finish line
  unsigned char mCountLapFlag;   // 0 = do not count lap or time, 1 =
count lap but not time, 2 = count lap and time
  bool mInGarageStall;           // appears to be within the correct
garage stall

  unsigned char mUpgradePack[16];  // Coded upgrades
  float mPitLapDist;             // location of pit in terms of lap
distance

  float mBestLapSector1;         // sector 1 time from best lap (not
necessarily the best sector 1 time)
  float mBestLapSector2;         // sector 2 time from best lap (not
necessarily the best sector 2 time)

  // Future use
  // tag.2012.04.06 - SEE ABOVE!
  unsigned char mExpansion[48];  // for future use
};


struct ScoringInfoV01
{
  char mTrackName[64];           // current track name
  long mSession;                 // current session (0=testday 1-
4=practice 5-8=qual 9=warmup 10-13=race)
  double mCurrentET;             // current time
  double mEndET;                 // ending time
  long  mMaxLaps;                // maximum laps
  double mLapDist;               // distance around track
  char *mResultsStream;          // results stream additions since last
update (newline-delimited and NULL-terminated)

  long mNumVehicles;             // current number of vehicles

  // Game phases:
  // 0 Before session has begun
```

Page | lv

```
  // 1 Reconnaissance laps (race only)
  // 2 Grid walk-through (race only)
  // 3 Formation lap (race only)
  // 4 Starting-light countdown has begun (race only)
  // 5 Green flag
  // 6 Full course yellow / safety car
  // 7 Session stopped
  // 8 Session over
  // 9 Paused (tag.2015.09.14 - this is new, and indicates that this is
a heartbeat call to the plugin)
  unsigned char mGamePhase;

  // Yellow flag states (applies to full-course only)
  // -1 Invalid
  //  0 None
  //  1 Pending
  //  2 Pits closed
  //  3 Pit lead lap
  //  4 Pits open
  //  5 Last lap
  //  6 Resume
  //  7 Race halt (not currently used)
  signed char mYellowFlagState;

  signed char mSectorFlag[3];      // whether there are any local
yellows at the moment in each sector (not sure if sector 0 is first or
last, so test)
  unsigned char mStartLight;       // start light frame (number depends
on track)
  unsigned char mNumRedLights;     // number of red lights in start
sequence
  bool mInRealtime;                // in realtime as opposed to at the
monitor
  char mPlayerName[32];            // player name (including possible
multiplayer override)
  char mPlrFileName[64];           // may be encoded to be a legal
filename

  // weather
  double mDarkCloud;               // cloud darkness? 0.0-1.0
  double mRaining;                 // raining severity 0.0-1.0
  double mAmbientTemp;             // temperature (Celsius)
  double mTrackTemp;               // temperature (Celsius)
  TelemVect3 mWind;                // wind speed
```

Page | lvi

```
  double mMinPathWetness;          // minimum wetness on main path 0.0-
1.0
  double mMaxPathWetness;          // maximum wetness on main path 0.0-
1.0

  // multiplayer
  unsigned char mGameMode; // 1 = server, 2 = client, 3 = server and
client
  bool mIsPasswordProtected; // is the server password protected
  unsigned short mServerPort; // the port of the server (if on a
server)
  unsigned long mServerPublicIP; // the public IP address of the server
(if on a server)
  long mMaxPlayers; // maximum number of vehicles that can be in the
session
  char mServerName[32]; // name of the server
  float mStartET; // start time (seconds since midnight) of the event

  //
  double mAvgPathWetness;          // average wetness on main path 0.0-
1.0

  // Future use
  unsigned char mExpansion[200];

  // keeping this at the end of the structure to make it easier to
replace in future versions
  VehicleScoringInfoV01 *mVehicle; // array of vehicle scoring info's
};


struct CommentaryRequestInfoV01
{
  char mName[32];                  // one of the event names in the
commentary INI file
  double mInput1;                  // first value to pass in (if any)
  double mInput2;                  // first value to pass in (if any)
  double mInput3;                  // first value to pass in (if any)
  bool mSkipChecks;                // ignores commentary detail and
random probability of event

  // constructor (for noobs, this just helps make sure everything is
initialized to something reasonable)
  CommentaryRequestInfoV01()       { mName[0] = 0; mInput1 = 0.0;
mInput2 = 0.0; mInput3 = 0.0; mSkipChecks = false; }
};
```

Page | lvii

```
//##############################################################################
####
//# Version02
Structures                                                    #
//##############################################################################
#####

struct PhysicsOptionsV01
{
  unsigned char mTractionControl;  // 0 (off) - 3 (high)
  unsigned char mAntiLockBrakes;   // 0 (off) - 2 (high)
  unsigned char mStabilityControl; // 0 (off) - 2 (high)
  unsigned char mAutoShift;        // 0 (off), 1 (upshifts), 2
(downshifts), 3 (all)
  unsigned char mAutoClutch;       // 0 (off), 1 (on)
  unsigned char mInvulnerable;     // 0 (off), 1 (on)
  unsigned char mOppositeLock;     // 0 (off), 1 (on)
  unsigned char mSteeringHelp;     // 0 (off) - 3 (high)
  unsigned char mBrakingHelp;      // 0 (off) - 2 (high)
  unsigned char mSpinRecovery;     // 0 (off), 1 (on)
  unsigned char mAutoPit;          // 0 (off), 1 (on)
  unsigned char mAutoLift;         // 0 (off), 1 (on)
  unsigned char mAutoBlip;         // 0 (off), 1 (on)

  unsigned char mFuelMult;         // fuel multiplier (0x-7x)
  unsigned char mTireMult;         // tire wear multiplier (0x-7x)
  unsigned char mMechFail;         // mechanical failure setting; 0
(off), 1 (normal), 2 (timescaled)
  unsigned char mAllowPitcrewPush; // 0 (off), 1 (on)
  unsigned char mRepeatShifts;     // accidental repeat shift
prevention (0-5; see PLR file)
  unsigned char mHoldClutch;       // for auto-shifters at start of
race: 0 (off), 1 (on)
  unsigned char mAutoReverse;      // 0 (off), 1 (on)
  unsigned char mAlternateNeutral; // Whether shifting up and down
simultaneously equals neutral

  // tag.2014.06.09 - yes these are new, but no they don't change the
size of the structure nor the address of the other variables in it
(because we're just using the existing padding)
  unsigned char mAIControl;        // Whether player vehicle is
currently under AI control
  unsigned char mUnused1;          //
  unsigned char mUnused2;          //
```

Page | lviii

```
  float mManualShiftOverrideTime;  // time before auto-shifting can
resume after recent manual shift
  float mAutoShiftOverrideTime;    // time before manual shifting can
resume after recent auto shift
  float mSpeedSensitiveSteering;   // 0.0 (off) - 1.0
  float mSteerRatioSpeed;          // speed (m/s) under which lock gets
expanded to full
};


struct EnvironmentInfoV01
{
  // TEMPORARY buffers (you should copy them if needed for later use)
containing various paths that may be needed.  Each of these
  // could be relative ("UserData\") or full
("C:\BlahBlah\rFactorProduct\UserData\").
  // mPath[ 0 ] points to the UserData directory.
  // mPath[ 1 ] points to the CustomPluginOptions.JSON filename.
  // mPath[ 2 ] points to the latest results file
  // (in the future, we may add paths for the current garage setup,
fully upgraded physics files, etc., any other requests?)
  const char *mPath[ 16 ];
  unsigned char mExpansion[256];   // future use
};


struct ScreenInfoV01
{
  HWND mAppWindow;                      // Application window handle
  void *mDevice;                        // Cast type to
LPDIRECT3DDEVICE9
  void *mRenderTarget;                  // Cast type to
LPDIRECT3DTEXTURE9
  long mDriver;                         // Current video driver index

  long mWidth;                          // Screen width
  long mHeight;                         // Screen height
  long mPixelFormat;                    // Pixel format
  long mRefreshRate;                    // Refresh rate
  long mWindowed;                       // Really just a boolean
whether we are in windowed mode

  long mOptionsWidth;                   // Width dimension of screen
portion used by UI
```

Page | lix

```
  long mOptionsHeight;                  // Height dimension of screen
portion used by UI
  long mOptionsLeft;                    // Horizontal starting
coordinate of screen portion used by UI
  long mOptionsUpper;                   // Vertical starting coordinate
of screen portion used by UI

  unsigned char mOptionsLocation;       // 0=main UI, 1=track loading,
2=monitor, 3=on track
  char mOptionsPage[ 31 ];              // the name of the options page

  unsigned char mExpansion[ 224 ];      // future use
};


struct CustomControlInfoV01
{
  // The name passed through CheckHWControl() will be the
mUntranslatedName prepended with an underscore (e.g. "Track Map Toggle"
-> "_Track Map Toggle")
  char mUntranslatedName[ 64 ];         // name of the control that
will show up in UI (but translated if available)
  long mRepeat;                         // 0=registers once per hit,
1=registers once, waits briefly, then starts repeating quickly,
2=registers as long as key is down
  unsigned char mExpansion[ 64 ];       // future use
};


struct WeatherControlInfoV01
{
  // The current conditions are passed in with the API call. The
following ET (Elapsed Time) value should typically be far
  // enough in the future that it can be interpolated smoothly, and
allow clouds time to roll in before rain starts. In
  // other words you probably shouldn't have mCloudiness and mRaining
suddenly change from 0.0 to 1.0 and expect that
  // to happen in a few seconds without looking crazy.
  double mET;                           // when you want this weather
to take effect

  // mRaining[1][1] is at the origin (2013.12.19 - and currently the
only implemented node), while the others
  // are spaced at <trackNodeSize> meters where <trackNodeSize> is the
maximum absolute value of a track vertex
  // coordinate (and is passed into the API call).
```

Page | lx

```
  double mRaining[ 3 ][ 3 ];            // rain (0.0-1.0) at different
nodes

  double mCloudiness;                   // general cloudiness
(0.0=clear to 1.0=dark), will be automatically overridden to help
ensure clouds exist over rainy areas
  double mAmbientTempK;                 // ambient temperature (Kelvin)
  double mWindMaxSpeed;                 // maximum speed of wind
(ground speed, but it affects how fast the clouds move, too)

  bool mApplyCloudinessInstantly;       // preferably we roll the new
clouds in, but you can instantly change them now
  bool mUnused1;                        //
  bool mUnused2;                        //
  bool mUnused3;                        //

  unsigned char mExpansion[ 512 ];      // future use (humidity,
pressure, air density, etc.)
};


//##########################################################################
####
//# Version07
Structures                                                    #
//##########################################################################
#####

struct CustomVariableV01
{
  char mCaption[ 128 ];                 // Name of variable. This will
be used for storage. In the future, this may also be used in the UI
(after attempting to translate).
  long mNumSettings;                    // Number of available
settings. The special value 0 should be used for types that have
limitless possibilities, which will be treated as a string type.
  long mCurrentSetting;                 // Current setting (also the
default setting when returned in GetCustomVariable()). This is zero-
based, so: ( 0 <= mCurrentSetting < mNumSettings )

  // future expansion
  unsigned char mExpansion[ 256 ];
};

struct CustomSettingV01
{
```

Page | lxi

```
  char mName[ 128 ];                    // Enumerated name of setting
(only used if CustomVariableV01::mNumSettings > 0). This will be stored
in the JSON file for informational purposes only. It may also possibly
be used in the UI in the future.
};

struct MultiSessionParticipantV01
{
  // input only
  long mID;                             // slot ID (if loaded) or -1
(if currently disconnected)
  char mDriverName[ 32 ];               // driver name
  char mVehicleName[ 64 ];              // vehicle name
  unsigned char mUpgradePack[ 16 ];     // coded upgrades

  float mBestPracticeTime;              // best practice time
  long mQualParticipantIndex;           // once qualifying begins, this
becomes valid and ranks participants according to practice time if
possible
  float mQualificationTime[ 4 ];        // best qualification time in
up to 4 qual sessions
  float mFinalRacePlace[ 4 ];           // final race place in up to 4
race sessions
  float mFinalRaceTime[ 4 ];            // final race time in up to 4
race sessions

  // input/output
  bool mServerScored;                   // whether vehicle is allowed
to participate in current session
  long mGridPosition;                   // 1-based grid position for
current race session (or upcoming race session if it is currently
warmup), or -1 if currently disconnected
// long mPitIndex;
// long mGarageIndex;

  // future expansion
  unsigned char mExpansion[ 128 ];
};

struct MultiSessionRulesV01
{
  // input only
  long mSession;                        // current session (0=testday
1-4=practice 5-8=qual 9=warmup 10-13=race)
```

Page | lxii

```
  long mSpecialSlotID;                  // slot ID of someone who just
joined, or -2 requesting to update qual order, or -1 (default/general)
  char mTrackType[ 32 ];                // track type from GDB
  long mNumParticipants;                // number of participants
(vehicles)

  // input/output
  MultiSessionParticipantV01 *mParticipant;       // array of
partipants (vehicles)
  long mNumQualSessions;                // number of qualifying
sessions configured
  long mNumRaceSessions;                // number of race sessions
configured
  long mMaxLaps;                        // maximum laps allowed in
current session (LONG_MAX = unlimited) (note: cannot currently edit in
*race* sessions)
  long mMaxSeconds;                     // maximum time allowed in
current session (LONG_MAX = unlimited) (note: cannot currently edit in
*race* sessions)
  char mName[ 32 ];                     // untranslated name override
for session (please use mixed case here, it should get uppercased if
necessary)

  // future expansion
  unsigned char mExpansion[ 256 ];
};


enum TrackRulesCommandV01               //
{
  TRCMD_ADD_FROM_TRACK = 0,             // crossed s/f line for first
time after full-course yellow was called
  TRCMD_ADD_FROM_PIT,                   // exited pit during full-
course yellow
  TRCMD_ADD_FROM_UNDQ,                  // during a full-course yellow,
the admin reversed a disqualification
  TRCMD_REMOVE_TO_PIT,                  // entered pit during full-
course yellow
  TRCMD_REMOVE_TO_DNF,                  // vehicle DNF'd during full-
course yellow
  TRCMD_REMOVE_TO_DQ,                   // vehicle DQ'd during full-
course yellow
  TRCMD_REMOVE_TO_UNLOADED,             // vehicle unloaded (possibly
kicked out or banned) during full-course yellow
```

Page | lxiii

```
  TRCMD_MOVE_TO_BACK,                    // misbehavior during full-
course yellow, resulting in the penalty of being moved to the back of
their current line
  TRCMD_LONGEST_LINE,                    // misbehavior during full-
course yellow, resulting in the penalty of being moved to the back of
the longest line
  //------------------
  TRCMD_MAXIMUM                          // should be last
};

struct TrackRulesActionV01
{
  // input only
  TrackRulesCommandV01 mCommand;         // recommended action
  long mID;                              // slot ID if applicable
  double mET;                            // elapsed time that event
occurred, if applicable
};

enum TrackRulesColumnV01
{
  TRCOL_LEFT_LANE = 0,                   // left (inside)
  TRCOL_MIDLEFT_LANE,                    // mid-left
  TRCOL_MIDDLE_LANE,                     // middle
  TRCOL_MIDRIGHT_LANE,                   // mid-right
  TRCOL_RIGHT_LANE,                      // right (outside)
  //------------------
  TRCOL_MAX_LANES,                       // should be after the valid
static lane choices
  //------------------
  TRCOL_INVALID = TRCOL_MAX_LANES,       // currently invalid (hasn't
crossed line or in pits/garage)
  TRCOL_FREECHOICE,                      // free choice (dynamically
chosen by driver)
  TRCOL_PENDING,                         // depends on another
participant's free choice (dynamically set after another driver
chooses)
  //------------------
  TRCOL_MAXIMUM                          // should be last
};

struct TrackRulesParticipantV01
{
  // input only
  long mID;                              // slot ID
```

Page | lxiv

```
  short mFrozenOrder;                    // 0-based place when caution
came out (not valid for formation laps)
  short mPlace;                          // 1-based place (typically
used for the initialization of the formation lap track order)
  float mYellowSeverity;                 // a rating of how much this
vehicle is contributing to a yellow flag (the sum of all vehicles is
compared to TrackRulesV01::mSafetyCarThreshold)
  double mCurrentRelativeDistance;       // equal to ( (
ScoringInfoV01::mLapDist * this->mRelativeLaps ) +
VehicleScoringInfoV01::mLapDist )

  // input/output
  long mRelativeLaps;                    // current formation/caution
laps relative to safety car (should generally be zero except when
safety car crosses s/f line); this can be decremented to implement
'wave around' or 'beneficiary rule' (a.k.a. 'lucky dog' or 'free pass')
  TrackRulesColumnV01 mColumnAssignment;// which column (line/lane)
that participant is supposed to be in
  long mPositionAssignment;              // 0-based position within
column (line/lane) that participant is supposed to be located at (-1 is
invalid)
  unsigned char mPitsOpen;               // whether the rules allow this
particular vehicle to enter pits right now (input is 2=false or 3=true;
if you want to edit it, set to 0=false or 1=true)
  bool mUpToSpeed;                       // while in the frozen order,
this flag indicates whether the vehicle can be followed (this should be
false for somebody who has temporarily spun and hasn't gotten back up
to speed yet)
  bool mUnused[ 2 ];                     //
  double mGoalRelativeDistance;          // calculated based on where
the leader is, and adjusted by the desired column spacing and the
column/position assignments
  char mMessage[ 96 ];                   // a message for this
participant to explain what is going on (untranslated; it will get run
through translator on client machines)

  // future expansion
  unsigned char mExpansion[ 192 ];
};

enum TrackRulesStageV01                  //
{
  TRSTAGE_FORMATION_INIT = 0,            // initialization of the
formation lap
  TRSTAGE_FORMATION_UPDATE,              // update of the formation lap
```

Page | lxv

```
  TRSTAGE_NORMAL,                         // normal (non-yellow) update
  TRSTAGE_CAUTION_INIT,                   // initialization of a full-
course yellow
  TRSTAGE_CAUTION_UPDATE,                 // update of a full-course
yellow
  //-----------------
  TRSTAGE_MAXIMUM                         // should be last
};

struct TrackRulesV01
{
  // input only
  double mCurrentET;                      // current time
  TrackRulesStageV01 mStage;              // current stage
  TrackRulesColumnV01 mPoleColumn;        // column assignment where pole
position seems to be located
  long mNumActions;                       // number of recent actions
  TrackRulesActionV01 *mAction;           // array of recent actions
  long mNumParticipants;                  // number of participants
(vehicles)

  bool mYellowFlagDetected;               // whether yellow flag was
requested or sum of participant mYellowSeverity's exceeds
mSafetyCarThreshold
  unsigned char mYellowFlagLapsWasOverridden;     // whether
mYellowFlagLaps (below) is an admin request (0=no 1=yes 2=clear yellow)

  bool mSafetyCarExists;                  // whether safety car even
exists
  bool mSafetyCarActive;                  // whether safety car is active
  long mSafetyCarLaps;                    // number of laps
  float mSafetyCarThreshold;              // the threshold at which a
safety car is called out (compared to the sum of
TrackRulesParticipantV01::mYellowSeverity for each vehicle)
  double mSafetyCarLapDist;               // safety car lap distance
  float mSafetyCarLapDistAtStart;         // where the safety car starts
from

  float mPitLaneStartDist;                // where the waypoint branch to
the pits breaks off (this may not be perfectly accurate)
  float mTeleportLapDist;                 // the front of the teleport
locations (a useful first guess as to where to throw the green flag)

  // future input expansion
  unsigned char mInputExpansion[ 256 ];
```

Page | lxvi

```
  // input/output
  signed char mYellowFlagState;         // see ScoringInfoV01 for
values
  short mYellowFlagLaps;                // suggested number of laps to
run under yellow (may be passed in with admin command)

  long mSafetyCarInstruction;           // 0=no change, 1=go active,
2=head for pits
  float mSafetyCarSpeed;                // maximum speed at which to
drive
  float mSafetyCarMinimumSpacing;       // minimum spacing behind
safety car (-1 to indicate no limit)
  float mSafetyCarMaximumSpacing;       // maximum spacing behind
safety car (-1 to indicate no limit)

  float mMinimumColumnSpacing;          // minimum desired spacing
between vehicles in a column (-1 to indicate indeterminate/unenforced)
  float mMaximumColumnSpacing;          // maximum desired spacing
between vehicles in a column (-1 to indicate indeterminate/unenforced)

  float mMinimumSpeed;                  // minimum speed that anybody
should be driving (-1 to indicate no limit)
  float mMaximumSpeed;                  // maximum speed that anybody
should be driving (-1 to indicate no limit)

  char mMessage[ 96 ];                  // a message for everybody to
explain what is going on (which will get run through translator on
client machines)
  TrackRulesParticipantV01 *mParticipant;         // array of
partipants (vehicles)

  // future input/output expansion
  unsigned char mInputOutputExpansion[ 256 ];
};


struct PitMenuV01
{
  long mCategoryIndex;                  // index of the current
category
  char mCategoryName[ 32 ];             // name of the current category
(untranslated)

  long mChoiceIndex;                    // index of the current choice
(within the current category)
```

Page | lxvii

```
  char mChoiceString[ 32 ];             // name of the current choice
(may have some translated words)
  long mNumChoices;                     // total number of choices (0
<= mChoiceIndex < mNumChoices)

  unsigned char mExpansion[ 256 ];      // for future use
};


//############################################################################
####
//# Plugin classes used to access
internals                                #
//############################################################################
#####

// Note: use class InternalsPluginV01 and have exported function
GetPluginVersion() return 1, or
//       use class InternalsPluginV02 and have exported function
GetPluginVersion() return 2, etc.
class InternalsPlugin : public PluginObject
{
 public:

  // General internals methods
  InternalsPlugin() {}
  virtual ~InternalsPlugin() {}

  // GAME FLOW NOTIFICATIONS
  virtual void Startup( long version ) {}                       // sim
startup with version * 1000
  virtual void Shutdown() {}                                    // sim
shutdown

  virtual void Load() {}                                        //
scene/track load
  virtual void Unload() {}                                      //
scene/track unload

  virtual void StartSession() {}                                //
session started
  virtual void EndSession() {}                                  //
session ended

  virtual void EnterRealtime() {}                               //
entering realtime (where the vehicle can be driven)
```

Page | lxviii

```
  virtual void ExitRealtime() {}                                    //
exiting realtime

  // SCORING OUTPUT
  virtual bool WantsScoringUpdates() { return( false ); }       //
whether we want scoring updates
  virtual void UpdateScoring( const ScoringInfoV01 &info ) {}   //
update plugin with scoring info (approximately five times per second)

  // GAME OUTPUT
  virtual long WantsTelemetryUpdates() { return(1); }           //
whether we want telemetry updates (0=no 1=player-only 2=all vehicles)
  virtual void UpdateTelemetry( const TelemInfoV01 &info) {}    //
update plugin with telemetry info

  virtual bool WantsGraphicsUpdates() { return( false ); }      //
whether we want graphics updates
  virtual void UpdateGraphics( const GraphicsInfoV01 &info ) {} //
update plugin with graphics info

  // COMMENTARY INPUT
  virtual bool RequestCommentary( CommentaryRequestInfoV01 &info ) {
return( false ); } // to use our commentary event system, fill in data
and return true

  // GAME INPUT
  virtual bool HasHardwareInputs() { return( false ); }         //
whether plugin has hardware plugins
  virtual void UpdateHardware( const double fDT ) {}            //
update the hardware with the time between frames
  virtual void EnableHardware() {}                              //
message from game to enable hardware
  virtual void DisableHardware() {}                             //
message from game to disable hardware

  // See if the plugin wants to take over a hardware control.  If the
plugin takes over the
  // control, this method returns true and sets the value of the double
pointed to by the
  // second arg.  Otherwise, it returns false and leaves the double
unmodified.
  virtual bool CheckHWControl( const char * const controlName, double
&fRetVal ) { return false; }
```

P a g e | lxix

```
  virtual bool ForceFeedback( double &forceValue ) { return( false ); }
// alternate force feedback computation - return true if editing the
value

  // ERROR FEEDBACK
  virtual void Error( const char * const msg ) {} // Called with
explanation message if there was some sort of error in a plugin
callback
};


class InternalsPluginV01 : public InternalsPlugin  // Version 01 is the
exact same as the original
{
  // REMINDER: exported function GetPluginVersion() should return 1 if
you are deriving from this InternalsPluginV01!
};


class InternalsPluginV02 : public InternalsPluginV01  // V02 contains
everything from V01 plus the following:
{
  // REMINDER: exported function GetPluginVersion() should return 2 if
you are deriving from this InternalsPluginV02!

 public:

  // This function is called occasionally
  virtual void SetPhysicsOptions( PhysicsOptionsV01 &options ) {}
};


class InternalsPluginV03 : public InternalsPluginV02  // V03 contains
everything from V02 plus the following:
{
  // REMINDER: exported function GetPluginVersion() should return 3 if
you are deriving from this InternalsPluginV03!

 public:

  virtual unsigned char WantsToViewVehicle( CameraControlInfoV01
&camControl ) { return( 0 ); } // return values: 0=do nothing, 1=set ID
and camera type, 2=replay controls, 3=both

  // EXTENDED GAME OUTPUT
```

Page | lxx

```
  virtual void UpdateGraphics( const GraphicsInfoV02 &info
)          {} // update plugin with extended graphics info

  // MESSAGE BOX INPUT
  virtual bool WantsToDisplayMessage( MessageInfoV01 &msgInfo )       {
return( false ); } // set message and return true
};


class InternalsPluginV04 : public InternalsPluginV03  // V04 contains
everything from V03 plus the following:
{
  // REMINDER: exported function GetPluginVersion() should return 4 if
you are deriving from this InternalsPluginV04!

 public:

  // EXTENDED GAME FLOW NOTIFICATIONS
  virtual void SetEnvironment( const EnvironmentInfoV01 &info
)       {} // may be called whenever the environment changes
};


class InternalsPluginV05 : public InternalsPluginV04  // V05 contains
everything from V04 plus the following:
{
  // REMINDER: exported function GetPluginVersion() should return 5 if
you are deriving from this InternalsPluginV05!

 public:

  // SCREEN INFO NOTIFICATIONS
  virtual void InitScreen( const ScreenInfoV01 &info
)               {} // Now happens right after graphics device
initialization
  virtual void UninitScreen( const ScreenInfoV01 &info
)             {} // Now happens right before graphics device
uninitialization

  virtual void DeactivateScreen( const ScreenInfoV01 &info
)          {} // Window deactivation
  virtual void ReactivateScreen( const ScreenInfoV01 &info
)          {} // Window reactivation

  virtual void RenderScreenBeforeOverlays( const ScreenInfoV01 &info
){} // before rFactor overlays
```

Page | lxxi

```
  virtual void RenderScreenAfterOverlays( const ScreenInfoV01 &info )
{} // after rFactor overlays

  virtual void PreReset( const ScreenInfoV01 &info
)                  {} // after detecting device lost but before
resetting
  virtual void PostReset( const ScreenInfoV01 &info
)                 {} // after resetting

  // CUSTOM CONTROLS
  virtual bool InitCustomControl( CustomControlInfoV01 &info )        {
return( false ); } // called repeatedly at startup until false is
returned
};


class InternalsPluginV06 : public InternalsPluginV05  // V06 contains
everything from V05 plus the following:
{
  // REMINDER: exported function GetPluginVersion() should return 6 if
you are deriving from this InternalsPluginV06!

 public:

  // CONDITIONS CONTROL
  virtual bool WantsWeatherAccess()                                   {
return( false ); } // change to true in order to read or write weather
with AccessWeather() call:
  virtual bool AccessWeather( double trackNodeSize,
WeatherControlInfoV01 &info ) { return( false ); } // current weather
is passed in; return true if you want to change it

  // ADDITIONAL GAMEFLOW NOTIFICATIONS
  virtual void ThreadStarted( long type
)                            {} // called just after a primary thread
is started (type is 0=multimedia or 1=simulation)
  virtual void ThreadStopping( long type
)                           {} // called just before a primary thread
is stopped (type is 0=multimedia or 1=simulation)
};


class InternalsPluginV07 : public InternalsPluginV06  // V07 contains
everything from V06 plus the following:
{
```

Page | lxxii

```
  // REMINDER: exported function GetPluginVersion() should return 7 if
you are deriving from this InternalsPluginV07!

 public:

  // CUSTOM PLUGIN VARIABLES
  // This relatively simple feature allows plugins to store settings in
a shared location without doing their own
  // file I/O. Direct UI support may also be added in the future so
that end users can control plugin settings within
  // rFactor. But for now, users can access the data in
UserData\Player\CustomPluginOptions.JSON.
  // Plugins should only access these variables through this interface,
though:
  virtual bool GetCustomVariable( long i, CustomVariableV01 &var )   {
return( false ); } // At startup, this will be called with increasing
index (starting at zero) until false is returned. Feel free to
add/remove/rearrange the variables when updating your plugin; the index
does not have to be consistent from run to run.
  virtual void AccessCustomVariable( CustomVariableV01 &var
)        {}                  // This will be called at startup,
shutdown, and any time that the variable is changed (within the UI).
  virtual void GetCustomVariableSetting( CustomVariableV01 &var, long
i, CustomSettingV01 &setting ) {} // This gets the name of each
possible setting for a given variable.

  // SCORING CONTROL (only available in single-player or on multiplayer
server)
  virtual bool WantsMultiSessionRulesAccess()                          {
return( false ); } // change to true in order to read or write multi-
session rules
  virtual bool AccessMultiSessionRules( MultiSessionRulesV01 &info )  {
return( false ); } // current internal rules passed in; return true if
you want to change them

  virtual bool WantsTrackRulesAccess()                                 {
return( false ); } // change to true in order to read or write track
order (during formation or caution laps)
  virtual bool AccessTrackRules( TrackRulesV01 &info )                 {
return( false ); } // current track order passed in; return true if you
want to change it (note: this will be called immediately after
UpdateScoring() when appropriate)

  // PIT MENU INFO (currently, the only way to edit the pit menu is to
use this in conjunction with CheckHWControl())
```

Page | lxxiii

```
  virtual bool WantsPitMenuAccess()                                    {
return( false ); } // change to true in order to view pit menu info
  virtual bool AccessPitMenu( PitMenuV01 &info )                       {
return( false ); } // currently, the return code should always be false
(because we may allow more direct editing in the future)
};


//##########################################################################
####
//##########################################################################
#####

// See #pragma at top of file
#pragma pack( pop )

#endif // _INTERNALS_PLUGIN_HPP_
```

PluginObjects.hpp

```
//##########################################################################
######
//#
    #
//# Module: Header file for plugin object
types                              #
//#
    #
//# Description: interface declarations for plugin
objects                  #
//#
    #
//# This source code module, and all information, data, and
algorithms      #
//# associated with it, are part of isiMotor Technology
(tm).               #
//#                 PROPRIETARY AND
CONFIDENTIAL                            #
//# Copyright (c) 2017 Studio 397 B.V.  All rights
reserved.               #
//#
    #
//# Change
history:                                                          #
```

Page | lxxiv

```
//#   tag.2008.02.15:
created                                                    #
//#
    #
//##########################################################################
######

#ifndef _PLUGIN_OBJECTS_HPP_
#define _PLUGIN_OBJECTS_HPP_


// rF currently uses 4-byte packing ... whatever the current packing is
will
// be restored at the end of this include with another #pragma.
#pragma pack( push, 4 )


//##########################################################################
####
//# types of
plugins                                                              #
//##########################################################################
#####

enum PluginObjectType
{
  PO_INVALID      = -1,
  //-------------------
  PO_GAMESTATS    =  0,
  PO_NCPLUGIN     =  1,
  PO_IVIBE        =  2,
  PO_INTERNALS    =  3,
  PO_RFONLINE     =  4,
  //-------------------
  PO_MAXIMUM
};


//##########################################################################
####
//#  PluginObject
   #
//#    - interface used by plugin
classes.                                    #
//##########################################################################
#####
```

Page | lxxv

```
class PluginObject
{
 private:

  class PluginInfo *mInfo;             // used by main executable to
obtain info about the plugin that implements this object

 public:

  void SetInfo( class PluginInfo *p )  { mInfo = p; }        // used by
main executable
  class PluginInfo *GetInfo() const    { return( mInfo ); }  // used by
main executable
  class PluginInfo *GetInfo()          { return( mInfo ); }  // used by
main executable
};


//#########################################################################
####
//# typedefs for dll functions - easier to use a typedef than to
type      #
//# out the crazy syntax for declaring and casting function
pointers       #
//#########################################################################
#####

typedef const char *      ( __cdecl *GETPLUGINNAME )();
typedef PluginObjectType  ( __cdecl *GETPLUGINTYPE )();
typedef int               ( __cdecl *GETPLUGINVERSION )();
typedef PluginObject *    ( __cdecl *CREATEPLUGINOBJECT )();
typedef void              ( __cdecl *DESTROYPLUGINOBJECT )(
PluginObject *obj );


//#########################################################################
####
//#########################################################################
#####

// See #pragma at top of file
#pragma pack( pop )

#endif // _PLUGIN_OBJECTS_HPP_
```

Page | lxxvi

receiverV5.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include <WinSock2.h>
#include <WS2tcpip.h>
#include <chrono>
#include <thread>
#include <ctime>            // Include for time functions
#include <sstream>          // Include for stringstream
#include <vector>           // Include for vector
#include <iomanip>          // Include for hex parsing

const char* UDPFilePath =
"C:\\Users\\Mathew\\Desktop\\rF2_data_files\\Received.csv";

// Function to handle errors
void handleError(const std::string& message) {
    std::cerr << "Error: " << message << " - " << WSAGetLastError() <<
std::endl;
    WSACleanup();
    exit(EXIT_FAILURE);
}

// Function to clear the terminal screen
void clearScreen() {
// Wait for the specified duration
    std::this_thread::sleep_for(std::chrono::milliseconds(200));
    system("cls"); // For Windows

    // For Linux/Unix, use "clear":
    // system("clear");
}

// Function to convert hexadecimal string to bytes
std::vector<unsigned char> hexStringToBytes(const std::string& hex) {
    std::vector<unsigned char> bytes;
    std::istringstream iss(hex);

    // Ensure stringstream sets hexadecimal conversion mode
    iss >> std::hex;

    unsigned int byte;
    while (iss >> byte) {
        bytes.push_back(static_cast<unsigned char>(byte));
```

Page | lxxvii

```
    }

    return bytes;
}

int main() {
    // Initialize Winsock
    WSADATA wsadata;
    if (WSAStartup(MAKEWORD(2, 2), &wsadata) != 0) {
        handleError("WSAStartup failed");
    }

    // Create a UDP socket
    SOCKET udpSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    if (udpSocket == INVALID_SOCKET) {
        handleError("Failed to create socket");
    }

    // Bind the socket to the local address and port
    sockaddr_in localAddr;
    localAddr.sin_family = AF_INET;
    localAddr.sin_port = htons(10815); // Specify your UDP port here
    localAddr.sin_addr.s_addr = INADDR_ANY;
    if (bind(udpSocket, (sockaddr*)&localAddr, sizeof(localAddr)) ==
SOCKET_ERROR) {
        handleError("Bind failed");
    }

    std::cout << "Waiting for data on UDP port 10815..." << std::endl;

    // Receive data from the UDP socket and write to CSV file
    char buffer[1024];
    sockaddr_in remoteAddr;
    int addrLen = sizeof(remoteAddr);
    int bytesReceived;

    while (true) {
        // Clear the terminal screen before each update
        clearScreen();
        bytesReceived = recvfrom(udpSocket, buffer, sizeof(buffer) - 1,
0, (sockaddr*)&remoteAddr, &addrLen);
        if (bytesReceived == SOCKET_ERROR) {
            handleError("recvfrom failed");
        }
```

Page | lxxviii

```
        buffer[bytesReceived] = '\0'; // Null-terminate the received
data

        // Process the received data
        std::string hexData(buffer);
        std::vector<unsigned char> bytes = hexStringToBytes(hexData);

        // Open or create the CSV file in append mode
        std::ofstream outputFile(UDPFilePath, std::ios::app);
        if (!outputFile.is_open()) {
            handleError("Failed to open file");
        }

        // Write received data to CSV file
        for (unsigned char byte : bytes) {
            outputFile << static_cast<int>(byte) << ",";
        }
        outputFile << std::endl;

        // Close the file
        outputFile.close();

        // Print received data to terminal
        std::cout << "Received " << bytesReceived << " bytes from " <<
inet_ntoa(remoteAddr.sin_addr) << ":" << ntohs(remoteAddr.sin_port) <<
std::endl;
        std::cout << "Type: Roll, Pitch, HeavAccel,  Yaw,  SwayAccel,
SurgeAcc" << std::endl;
        std::cout << "Data: " << hexData << std::endl;
    }

    // Close the socket and clean up Winsock
    closesocket(udpSocket);
    WSACleanup();

    return 0;
}
```