InSim

From LFS Manual
Jump to navigationJump to search

InSim is a protocol which allows an external program to communicate with Live for Speed. It allows you to create a socket connection with the game and to send and receive packets of data. The InSim protocol describes how each of these packets is formatted, and any programming language which can create a network connection and send and receive strings of binary data can interface with it.

The official documentation is included in the file InSim.txt, found in the games `docs` folder. It consists of a C++ header file that contains the definition for each packet, as well as comments from Scawen as to how each should be used. The documentation here is intended as an ancillary to this file.

Contents

UDP vs TCP

InSim supports both UDP and TCP connections. In UDP mode only a single connection can be made, however up to eight connections can be made to the game in TCP. Whether connected in TCP or UDP, it's possible to specify a separate UDP socket for receiving car position updates, such as IS_MCI and IS_NLP.

InSim examples

How you go about creating an InSim connection is of course dependent on which programming language you are using, but here we make an attempt to document the process with some examples from the popular Python programming language (https://www.python.org/). As mentioned above any language capable of making a socket connection can be used to interface with LFS, however the principle remain the same regardless.

Creating a connection

First of all we must establish a socket connection with the game, in this case in TCP.

```
# Import Python's socket module.
import socket
```

```
# Initialise the socket in TCP mode.
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Connect to LFS.
sock.connect(('localhost', 29999))
```

Initialising InSim

After establishing the connection we must initialise the InSim system by sending the IS_ISI packet. Before we can do this however we must first intitailse InSim within LFS itself. To do this start the game and enter the chat command `/insim 29999`. The port number used can be any valid port, but 29999 generally tends to be the accepted default.

Here is the definition for the IS_ISI packet from InSim.txt.

```
struct IS_ISI // InSim Init - packet to initialise the InSim system
{
        byte    Size;           // 44
        byte    Type;           // ISP_ISI
        byte    ReqI;           // If non-zero LFS will send an IS_VER packet
        byte    Zero;           // 0

        word    UDPPort;        // Port for UDP replies from LFS (0 to 65535)
        word    Flags;          // Bit flags for options (see below)

        byte    Sp0;            // 0
        byte    Prefix;         // Special host message prefix character
        word    Interval;       // Time in ms between NLP or MCI (0 = none)

        char    Admin[16];      // Admin password (if set in LFS)
        char    IName[16];      // A short name for your program
};
```

Each InSim packet begins with a header, consisting of 4 bytes. The first byte is the size of the packet, followed by the packet type from the ISP_ enumeration, and then the `ReqI` (standing for Request Id). Whenever a request is made to LFS the value of the `ReqI` must be set to non-zero, whereby LFS will reply with the same value set in the `ReqI` of the requested packet. Finally the fourth byte varies depending on the type of packet in question, which in this case is blank.

As you can see the IS_ISI packet contains various options and flags that are used when initialising the InSim system. We must pack this data into a binary formatted string to send it to LFS.

```
# Import Python's struct module, which allows us to pack and unpack strings.
import struct

# Pack the IS_ISI data into a string.
isi = struct.pack('BBBBHHBcH16s16s',
                  44,           # Size
                  1,            # Type
                  1,            # ReqI
                  0,            # Zero
                  0,            # UDPPort
                  0,            # Flags
                  0,            # Sp0
                  ' ',          # Prefix
                  0,            # Interval
                  'password',   # Admin
                  'MyProgram',) # IName

# Send the string to InSim
sock.send(isi)
```

Receiving Data

After creating the connection and initialising InSim we must then setup the packet receive loop. As data in TCP mode is sent as a constant stream of data, multiple packets may arrive in a single receive call and some packets may arrive incomplete. This means we must store all incoming data in a buffer and then read each packet out when we are sure it is complete.

```
# We use a string as the buffer.
buffer = ''

while True:
    # Receive up to 1024 bytes of data.
    data = sock.recv(1024)

    # If no data is received the connection has closed.
    if data:
        # Append received data onto the buffer.
        buffer += data

        # Loop through each completed packet in the buffer. The first byte of
        # each packet is the packet size, so check that the length of the
        # buffer is at least the size of the first packet.
        while len(buffer) > 0 and len(buffer) > ord(buffer[0]):
            # Copy the packet from the buffer.
            packet = buffer[:ord(buffer[0])]

            # Remove the packet from the buffer.
            buffer = buffer[ord(buffer[0]):]

            # The packet is now complete! :)
            # doSomethingWithPacket(packet)
    else:
        break

# Release the socket.
sock.close()
```

Unpacking Packets

Once we have received the packet data as a binary formatted string, we then have to unpack this data into a format which is useful to us. In our previous example when we sent the IS_ISI initailisation packet, we set the ReqI to non-zero, meaning that LFS responded with an IS_VER version packet, however we didn't do anything with it. Firstly lets look at the definition for the IS_VER packet from InSim.txt.

```
struct IS_VER // VERsion
{
        byte    Size;                   // 20
        byte    Type;                   // ISP_VERSION
        byte    ReqI;                   // ReqI as received in the request packet
        byte    Zero;

        char    Version[8];             // LFS version, e.g. 0.3G
        char    Product[6];             // Product : DEMO or S1
        word    InSimVer;               // InSim Version : increased when InSim packets change
};
```

Now lets look at how we would unpack that data in Python.

```
# Import Python's struct module.
import struct

# Unpack the binary formatted packet data into the values we need.
size, type, reqi, zero, version, product, insimver = struct.unpack('BBBB8s6sH', packet)

# Check the InSim version.
if insimver != 4:
    print 'Invalid InSim version!'
    sock.close()
```

Keep Alive

In order to keep the connection open LFS will send a "keep alive" packet every 30 or so seconds. This packet is an IS_TINY with a SubT (sub-type) of TINY_NONE. We must respond to this packet every time it is received in order to prevent the connection with InSim from timing-out.

```
# Some constants.
ISP_TINY = 3
TINY_NONE = 0

# Check the packet type.
if ord(packet[1]) == ISP_TINY:
    # Unpack the packet data.
    tiny = struct.unpack('BBBB', packet)
    # Check the SubT.
    if tiny[3] == TINY_NONE:
       # Send the keep alive packet back to LFS.
       sock.send(packet)
```

Further examples

You can see the full example of this code as well as others on the InSim examples page.

InSim libraries

Of course as the old adage goes you shouldn't try to reinvent the wheel (unless you're trying to learn more about wheels) and there are several mature InSim libraries available for use in your own code.

InSim Libraries			
Library	**Platform**	**License**	**Webpage / repo / download**
LFSLib	.NET Framework	GPL	Project page (http://sourceforge.net/projects/lfslibnet/)
LFS_External	.NET Framework	Freeware	LFS Forum (http://www.lfsforum.net/showthread.php?t=30012)
JInSim	Java	Mozilla	LFS Forum (http://www.lfsforum.net/showthread.php?t=11568)
pyinsim	Python	LGPL	LFS Forum (http://www.lfsforum.net/showthread.php?t=70545)
CInSim	C/C++	Freeware	LFS Forum (http://www.lfsforum.net/showthread.php?t=47717)
phplfs	PHP5	Apache License V2.0	Project page (http://sourceforge.net/projects/phplfs/)
PRISM	PHP7	MIT	LFS Forum (https://www.lfs.net/forum/312-PHPInSimMod---PRISM)
InSim.NET	.NET Framework	LGPL	LFS Forum (http://www.lfsforum.net/showthread.php?t=68564)
insim.rs	Rust	MIT	GitHub (https://github.com/theangryangel/insim.rs), Documentation (https://docs.rs/insim/), Crates.io (https://crates.io/crates/insim), LFS Forum (https://www.lfs.net/forum/thread/107171-Rust---insim-rs---a-suite-of-crates-to-help-you-work-with-LFS)
PIE	PHP7	Freeware	LFS Forum (https://www.lfs.net/forum/528-PIE-%28PHP%3B-Insim%3B-Easy%29)

Node InSim	Node.js	MIT	LFS Forum (https://www.lfs.net/forum/thread/103431), GitHub (https://github.com/simbroadcasts/node-insim), NPM (https://npmjs.org/node-insim)
ktinsim	Kotlin	MIT	LFS Forum (https://www.lfs.net/forum/thread/102184-Kotlin-InSim-library---ktinsim), GitHub (https://github.com/verde-lfs/ktinsim)
GodotInSim	Godot	MIT	LFS Forum (https://www.lfs.net/forum/thread/106812-Godot-InSim), GitHub (https://github.com/Cykyrios/GodotInSim)

InSim reference

Here is an attempt to reference the complete InSim protocol.

Packet Reference		
Packet	**Description**	**Type**
	Initialisation	
IS_ISI	InSim initialisation	Instruction
	General Purpose	
IS_TINY	General purpose 4 byte packet	Both
IS_SMALL	General purpose 8 byte packet	Both
	Version request	
IS_VER	Version information	Info
	State Reporting and Requests	
IS_STA	Sent whenever the game state changes	Info
IS_SFP	Send to set various state options	Instruction
	Screen Mode	
IS_MOD	Send to change screen mode	Instruction
	Text Messages and Key Presses	
IS_MSO	System and user messages sent from LFS	Info
IS_III	User messages to host InSim	Info
IS_MST	Send LFS a message or command (64 characters)	Instruction
IS_MSX	Extended version of IS_MST (96 characters), not for commands	Instruction
IS_MSL	Send message to local game client	Instruction
IS_MTC	Send message to specific connection or player	Instruction
IS_SCH	Send single character or key press	Instruction
	Multiplayer Notification	
IS_ISM	Sent when starting or joining a host	Info
IS_NCI	Sent when host admin password is set, contains user IP and language data	Info
	Vote Notify	
IS_VTN	Notify of player vote (restart race, qualify etc..)	Info
	Race Tracking	
IS_RST	Race starting or restarting	Info
IS_NCN	New connection joining server	Info
IS_SLC	to report changes in car state (currently start or stop)	Info

IS_CSC	reports a connection's currently selected car	Info
IS_CNL	Connection left server	Info
IS_CPR	Player changed name	Info
IS_NPL	New player joining race, or leaving pits	Info
IS_MAL	Allowed Mods - Set/Clear allowed mods (by skinID) in the server	Info
IS_PLP	Player pits (gone to garage screen)	Info
IS_PLL	Player left race (gone to spectate)	Info
IS_CRS	Car reset (pressed space bar)	Info
IS_JRR	can be used to reset or start a car at a specified location	Instruction
IS_LAP	Lap time completed	Info
IS_SPX	Split time completed	Info
IS_PIT	Pit stop started (at pit box)	Info
IS_PSF	Pit stop finished	Info
IS_PLA	Player entered pit lane (to pit or serve penalty)	Info
IS_CHH	Camera view changed (chase view, custom view etc..)	Info
IS_PEN	Penalty given or cleated	Info
IS_TOC	Player taken over another car (driver swap)	Info
IS_FLG	Player shown flag (yellow or blue)	Info
IS_PFL	Player flags changed (auto-gears, auto-clutch etc..)	Info
IS_FIN	Player finished race (crossed finish line)	Info
IS_RES	Player result awarded, confirmed finish	Info
IS_REO	Reorder starting grid	Both
	Autocross	
IS_AXI	Autocross layout loaded	Info
IS_AXO	Player hit autocross object	Info
IS_OCO	can be used to override specific or all start lights	Instruction
IS_UCO	sends info about InSim checkpoints and circles	Info
	Car Tracking	
IS_NLP	Players current node, lap and race position	Info
IS_MCI	More detailed version of IS_NLP, world-coordinates, speed, angle and heading	Info
	Camera Control	
IS_SCC	Set viewed car and select camera	Instruction
IS_CPP	Set full camera position	Instruction
	Replay Control	
IS_RIP	Load replay and move to specific destination	Instruction
	Screenshots	
IS_SSH	Take a screenshot	Instruction
	InSim Buttons	
IS_BFN	Delete a button or all buttons	Instruction
IS_BTN	Send a button to the screen	Instruction
IS_BTC	Sent when a button is clicked	Info
IS_BTT	Sent when text is entered	Info

- This page was last edited on 30 January 2025, at 14:26.