KerbalSimpit Documentation

Release 1.0

KerbalSimpit

Nov 06, 2021

CONTENTS

1 Quickstart Guide **3**
- 1.1 Minimal Arduino sketch . . . 3
- 1.2 Message channels . . . 4
- 1.3 Sending data . . . 4
- 1.4 Receiving data . . . 4
- 1.5 Going further . . . 5

2 KerbalSimpit Class Documentation **7**

3 Kerbal Simpit Message Types **11**

4 Kerbal Simpit Compound Messages **23**

5 Troubleshooting Guide **37**
- 5.1 First check : mod installation and configuration . . . 37
- 5.2 First example : receiving information from KSP . . . 37
- 5.3 Second example : sending information to KSP . . . 38
- 5.4 Common mistakes . . . 38

Index **41**

This library manages a serial connection to the Kerbal Simpit plugin for Kerbal Space Program. It handles low-level handshaking and packet sending and receiving, and provides data structures and methods for dealing with messages to and from the plugin.

CHAPTER

ONE

QUICKSTART GUIDE

1.1 Minimal Arduino sketch

A minimal Kerbal Simpit sketch looks like this:

```
#include <KerbalSimpit.h>

KerbalSimpit mySimpit(Serial);

void setup() {
  Serial.begin(115200);
  while (!mySimpit.init());
}

void loop() {
  mySimpit.update();
}
```

There are a few parts that are required:

1. Include the library:

   ```
   #include <KerbalSimpit.h>
   ```

2. Create a KerbalSimpit object. The constructor requires one argument, the serial device that we will use. In most cases this is "Serial":

   ```
   KerbalSimpit mySimpit(Serial);
   ```

3. Initialise the serial connection. Kerbal Simpit does not attempt to open the serial device, so the sketch should do so in its `setup()` function. The speed should match that specified by the plugin in its config file:

   ```
   Serial.begin(115200);
   ```

4. Initialise the KerbalSimpit object. The `init()` function performs a three-way handshake with the Kerbal Simpit plugin. It returns a boolean indicating handshake status, so it's easiest to just call this in a loop until a successful handshake is performed:

   ```
   while (!mySimpit.init());
   ```

5. The `update()` function is receives new data from the serial connection. It should be called regularly in the sketch `loop()`.

1.2 Message channels

Every message in to and out of the Kerbal Simpit plugin has a channel ID. Each channel is dedicated to a specific type of information, and the messages this library supports are documented in *Kerbal Simpit Message Types*.

1.3 Sending data

The low-level `send()` function can send arbitrary data to the plugin:

```
mySimpit.send(messageType, message[], messageSize);
```

- `messageType` is a byte representing the channel this message is on. The library provides constants for all supported message channels, see *Kerbal Simpit Message Types* for a full list.
- `message[]` is a byte array representing the message. The library enforces a hard limit on message size, `MAX_PAYLOAD_SIZE`, which defaults to 32 bytes.
- `messageSize` is a byte representing the size of the message.

The library provides higher-level functions encapsulating most known message channels. For example, both of these lines activate the standard Brakes action group:

```
mySimpit.send(CAGACTIVATE_MESSAGE, &BRAKES_ACTION, 1);

mySimpit.activateAction(BRAKES_ACTION);
```

Refer to *KerbalSimpit Class Documentation* for full documentation.

1.4 Receiving data

To receive data from the plugin, use the `inboundHandler()` function to register a callback handler with the library:

```
mySimpit.inboundHandler(myCallbackHandler);
```

And define the callback handler:

```
void myCallbackHandler(byte messageType, byte message[], byte messageSize) {
  switch(messageType) {
  case MESSAGE_TYPE_1:
    // Handle the first type of message.
    break;
  case MESSAGE_TYPE_2:
    // Handle the second type of message.
    break;
  }
}
```

Most messages from the plugin consist of several pieces of information. The library includes structs and helper functions to assist with working with these. For example, here's a basic callback handler for dealing with altitude information from the plugin:

```
void myCallbackHandler(byte messageType, byte mesesage[], byte messageSize) {
  switch(messageType) {
  case ALTITUDE_MESSAGE:
    if (msgSize == sizeof(altitudeMessage)) {
      altitudeMessage myAltitude;
      myAltitude = parseAltitude(msg);
      // further processing of altitude data in myAltitude here
    }
    break;
  }
}
```

For a full list of available structs and helper functions, refer to *Kerbal Simpit Compound Messages*.

1.5 Going further

The examples directory of the library contains several example sketches that demonstrate the different functionality of the library.

CHAPTER

TWO

KERBALSIMPIT CLASS DOCUMENTATION

class **KerbalSimpit**

The *KerbalSimpit* class manages a serial connection to KSP.

It automates the handshaking process, and provides utility functions to encapsulate most message types in and out of the game.

Public Functions

KerbalSimpit(Stream &serial)

Default constructor.

Parameters `serial` – The serial instance this instance will use to communicate with the plugin. Usually "Serial".

bool **init**()

Initialise the serial connection.

Performs handshaking with the plugin. Note that the KSPit library does not* call the `begin()` method on the serial object. You'll need to ensure you've run `Serial.begin(115200)` (or similar) before calling this method.

void **inboundHandler**(void (*messageHandler)(byte messageType, byte msg[], byte msgSize))

Specify a callback function to handle messages received from the plugin.

See messageHandler

Parameters `messageHandler` – The callback function.

void **registerChannel**(byte channelID)

Subscribe to a channel of messages coming from the plugin.

This function sends a channel subscription message to the plugin, indicating that this device would like to receive messages send to a given channel. This function should only be called with an ID from the OutboundPackets enum. The IDs from the InboundPackets enum are only used when sending data to KSP and should not be registered.

Parameters `channelID` – The ID of the channel to subscribe to.

void **deregisterChannel**(byte channelID)

Unsubscribe from a channel of messages coming from the plugin.

This function sends a channel subscription message to the plugin, indicating that no further messages for the given channel should be sent to this device.

Parameters **channelID** – The ID of the channel to unsubscribe from.

template<typename **T**>
inline void **send**(byte messageType, *T* &msg)
Send a formatted KSPit packet.

Sends the given message as payload of a KSPit message.

Parameters

- **messageType** – The ID of the message channel.
- **msg** – Any object to be sent. The expected object depends on the message type. No type checking is done by this library.

template<typename **T**>
inline void **send**(byte messageType, *T* &msg, byte msgSize)
Send a formatted KSPit packet.

Sends the given message as payload of a KSPit message.

Parameters

- **messageType** – The ID of the message channel.
- **msg** – A byte array representing the message contents.
- **msgSize** – The size of msg.

void **update**()
Regular library update function.

This function polls the serial device for new data, and performs other tasks that must be done regularly. The function should be called from an Arduino sketch `loop()` method.

void **activateCAG**(byte actiongroup)
Activate Custom Action Group.

Sends a message to activate the given Custom Action Group.

Parameters **actiongroup** – The ID of the Custom Action Group to activate.

void **deactivateCAG**(byte actiongroup)
Deactivate Custom Action Group.

Sends a message to deactivate the given Custom Action Group.

Parameters **actiongroup** – The ID of the Custom Action Group to deactivate.

void **toggleCAG**(byte actiongroup)
Toggle Custom Action Group.

Sends a message to toggle the state of the given Custom Action Group.

Parameters **actiongroup** – The ID of the Custom Action Group to toggle.

void **activateAction**(byte action)
Activate Action Group.

Sends a message to activate the given standard Action Group(s).

Parameters **action** – A bitfield representing one or more Action Groups to activate.

void **deactivateAction**(byte action)
Deactivate Action Group.

Sends a message to deactivate the given standard Action Group(s).

Parameters `action` – A bitfield representing one or more Action Groups to deactivate.

void **`toggleAction`**(byte action)
Toggle Action Group.

Sends a message to toggle the state of the given standard Action Group(s).

Parameters `action` – A bitfield representing one or more Action Groups to toggle.

void **`setSASMode`**(byte mode)
Set SAS mode Send a message to set the desired Autopilot (SAS) mode.

Parameters `mode` – The mode to set. Possible modes are listed in the AutopilotMode enum.

This documentation was built using ArduinoDocs.

CHAPTER

THREE

KERBAL SIMPIT MESSAGE TYPES

Constants for inbound and outbound message IDs.

Enums

enum **CommonPackets**
Common packets.

These packet types are used for both inbound and outbound messages.

Values:

enumerator **SYNC_MESSAGE**
Sync message. Used for handshaking.

enumerator **ECHO_REQ_MESSAGE**
Echo request. Either end can send this, and an echo response is expected.

enumerator **ECHO_RESP_MESSAGE**
Echo response. Sent in reply to an echo request.

enum **OutboundPackets**
Outbound packets.

IDs for packets that go from the game to devices.

Values:

enumerator **LF_MESSAGE**
Liquid fuel in the vessel.

Messages on this channel contain a *resourceMessage*. Need ARP to work.

enumerator **LF_STAGE_MESSAGE**
Liquid fuel in the current stage.

Messages on this channel contain a *resourceMessage*. Need ARP to work.

enumerator **OX_MESSAGE**
Oxidizer in the vessel.

Messages on this channel contain a *resourceMessage*. Need ARP to work.

enumerator **OX_STAGE_MESSAGE**
Oxidizer in the current stage.

Messages on this channel contain a *resourceMessage*. Need ARP to work.

enumerator **SF_MESSAGE**
Solid fuel in the vessel.

Messages on this channel contain a *resourceMessage*. Need ARP to work.

enumerator **SF_STAGE_MESSAGE**
Solid fuel in the current stage.

Messages on this channel contain a *resourceMessage*. Need ARP to work.

enumerator **XENON_GAS_MESSAGE**
Xenon gas in the vessel.

Messages on this channel contain a *resourceMessage*. Need ARP to work.

enumerator **XENON_GAS_STAGE_MESSAGE**
Xenon Gas in the current stage.

Messages on this channel contain a *resourceMessage*. Need ARP to work.

enumerator **MONO_MESSAGE**
Monoprollent in the vessel.

Messages on this channel contain a *resourceMessage*. Need ARP to work.

enumerator **EVA_MESSAGE**
EVA propellant.

Only available for Kerbals on EVA. Messages on this channel contain a *resourceMessage*.

enumerator **ELECTRIC_MESSAGE**
Electic Charge in the vessel.

Messages on this channel contain a *resourceMessage*. Need ARP to work.

enumerator **ORE_MESSAGE**
Ore in the vessel.

Messages on this channel contain a *resourceMessage*. Need ARP to work.

enumerator **AB_MESSAGE**
Ablator in the vessel.

Messages on this channel contain a *resourceMessage*. Need ARP to work.

enumerator **AB_STAGE_MESSAGE**
Ablator in the current stage.

Messages on this channel contain a *resourceMessage*. Need ARP to work.

enumerator **TACLS_RESOURCE_MESSAGE**
TAC Life Support ressources.

Messages on this channel contain a TACLSRessourceMessage. Need ARP and TACLS to work.

enumerator **TACLS_WASTE_MESSAGE**
TAC Life Support waste ressources.

Messages on this channel contain a *TACLSWasteMessage*. Need ARP and TACLS to work.

enumerator **CUSTOM_RESOURCE_1_MESSAGE**
Custom ressources.

Specific ressource to use to be set in the configuration file. Messages on this channel contain a *CustomResourceMessage*. Need ARP and Community Resource Pack to work.

enumerator **CUSTOM_RESOURCE_2_MESSAGE**
Custom ressources.

Specific ressource to use to be set in the configuration file. Messages on this channel contain a *CustomResourceMessage*. Need ARP and Community Resource Pack to work.

enumerator **ALTITUDE_MESSAGE**
Sea level and surface altitude.

Messages on this channel contain an *altitudeMessage*.

enumerator **VELOCITY_MESSAGE**
Vessel velocity.

Messages on this channel contain a *velocityMessage*.

enumerator **AIRSPEED_MESSAGE**
Information about airspeed.

This channel delivers messages containing indicated airspeed and mach number for the active vessel.

enumerator **APSIDES_MESSAGE**
Apoapsis and periapsis.

Messages on this channel contain an *apsidesMessage*.

enumerator **APSIDESTIME_MESSAGE**
Time to the next apoapsis and periapsis.

Messages on this channel contain an *apsidesTimeMessage*.

enumerator **MANEUVER_MESSAGE**
Data about the planned maneuvers.

Messages on this channel contain an *maneuverMessage*.

enumerator **SAS_MODE_INFO_MESSAGE**
Data about the current SAS mode.

Messages on this channel contain a *SASInfoMessage*.

enumerator **ORBIT_INFO**
Data about the current orbit.

Messages on this channel contain an *orbitInfoMessage*.

enumerator **ACTIONSTATUS_MESSAGE**
Action groups.

Messages on this channel contain a single byte representing the currently active action groups. A given action group can be checked by performing a bitwise AND with the message. For example:

```
if (msg[0] & SAS_ACTION) {
    // code to execute if SAS is active
}
```

Possible action groups are:

- STAGE_ACTION
- GEAR_ACTION
- LIGHT_ACTION
- RCS_ACTION
- SAS_ACTION
- BRAKES_ACTION
- ABORT_ACTION

enumerator **DELTAV_MESSAGE**
Amount of deltaV of the current vessel in the current situation.

Messages on this channel contain an *deltaVMessage*.

enumerator **DELTAVENV_MESSAGE**
Amount of deltaV of the current vessel in different situations (Atmospheric sea level and in vacuum).

Messages on this channel contain an *deltaVEnvMessage*.

enumerator **BURNTIME_MESSAGE**
Amount of burn time of the current vessel.

Messages on this channel contain an *burnTimeMessage*.

enumerator **CAGSTATUS_MESSAGE**
Current status of all the custom action groups.

Messages on this channel contains a *cagStatusMessage*. This *cagStatusMessage* has a is_action_activated method taking the action ground number as an argumetnt.

enumerator **TEMP_LIMIT_MESSAGE**
Current maximum temperature of the any vessel part.

This message contains the maximum percentage value of the temperature of any part of the vessel. It contains a percentage for both core and skin temperature. The two maximum percentage can come from different parts. Messages on this channel contain an *tempLimitMessage*.

enumerator **TARGETINFO_MESSAGE**
Information about targetted object.

This channel delivers messages about the object targetted by the active vessel. Messages on this channel contain a targetInfoMessage.

enumerator **SOI_MESSAGE**
Name of current Sphere of Influence.

This channel delivers an ASCII string containing the name of the body the active vessel is currently orbiting. Note that this is always the English name, regardless of the language the game is currently set to.

enumerator **SCENE_CHANGE_MESSAGE**
Scene change packets are sent by the plugin when entering or leaving the flight scene.

enumerator **FLIGHT_STATUS_MESSAGE**
Information about the current fligth (warp speed, status, crew, com).

Messages on this channel contain a *flightStatusMessage*.

enum **InboundPackets**
Inbound packets.

These packet types are used for packets going from devices to the game.

Values:

enumerator **REGISTER_MESSAGE**
Register to receive messages on a given channel.

enumerator **DEREGISTER_MESSAGE**
Deregister, indicate that no further messages for the given channel should be sent.

enumerator **CAGACTIVATE_MESSAGE**
Activate the given Custom Action Group(s).

enumerator **CAGDEACTIVATE_MESSAGE**
Deactivate the given Custom Action Group(s).

enumerator **CAGTOGGLE_MESSAGE**
Toggle the given Custom Action Group(s) (Active CAGs will deactivate, inactive CAGs will activate).

enumerator **AGACTIVATE_MESSAGE**
Activate the given standard Action Group(s).

Note that *every request* to activate the Stage action group will result in the next stage being activated. For all other action groups, multiple activate requests will have no effect.

enumerator **AGDEACTIVATE_MESSAGE**
Deactivate the given standard Action Group(s).

enumerator **AGTOGGLE_MESSAGE**
Toggle the given standard Action Group(s).

enumerator **ROTATION_MESSAGE**
Send vessel rotation commands.

enumerator **TRANSLATION_MESSAGE**
Send vessel translation commands.

enumerator **WHEEL_MESSAGE**
Send wheel steering/throttle commands.

enumerator **THROTTLE_MESSAGE**
Send vessel throttle commands.

enumerator **SAS_MODE_MESSAGE**
Send SAS mode commands.

The payload should be a single byte, possible SAS modes are listed in the AutopilotMode enum.

enumerator **CAMERA_CONTROL_MODE**

enumerator **CAMERA_ROTATION_MESSAGE**

enumerator **TIMEWARP_MESSAGE**
Send a time warp commands.

The payload should be a single byte, possible commands are listed in the Timewarp enum.

enumerator **CUSTOM_LOG**
Send a custom log message.

The message will be printed in the KSP log. The options are defined in the CustomLogStatus enum. The message should not be more than 31 char.

enumerator **KEYBOARD_EMULATOR**
Send a message to emulate a key press.

The message contains a key modifyer and a key code. The key code indicate which key to press. It is taken from the file named VirtualKeyCode in the folder *KerbalSimpit*. For instance :

- 0x4D for 'm' to open the map
- 0x74 for 'F5' to quicksave The modifier is used to emulate key press while holding keys such as CTRL, SHIFT, etc This only works on Windows, not on iOS or Linux.

enum **ActionGroupIndexes**
Action Group Indexes These are used to mask out elements of an ACTIONSTATUS_MESSAGE.

Values:

enumerator **STAGE_ACTION**
Bitmask for the Stage action group.

enumerator **GEAR_ACTION**
Bitmask for the Gear action group.

enumerator **LIGHT_ACTION**
Bitmask for the Light action group.

enumerator **RCS_ACTION**
Bitmask for the RCS action group.

enumerator **SAS_ACTION**
Bitmask for the SAS action group.

enumerator **BRAKES_ACTION**
Bitmask for the Brakes action group.

enumerator **ABORT_ACTION**
Bitmask for the Abort action group.

enum **Timewarp**
Timewarp command These are used for a TIMEWARP_MESSAGE.

Values:

enumerator **TIMEWARP_X1**
Set Timewarp to x1.

enumerator **TIMEWARP_X5**
Set Timewarp to x5 (no effect in atmosphere).

enumerator **TIMEWARP_X10**
Set Timewarp to x10 (no effect in atmosphere).

enumerator **TIMEWARP_X50**
Set Timewarp to x50 (no effect in atmosphere).

enumerator **TIMEWARP_X100**
Set Timewarp to x100 (no effect in atmosphere).

enumerator **TIMEWARP_X1000**
Set Timewarp to x1000 (no effect in atmosphere).

enumerator **TIMEWARP_X10000**
Set Timewarp to x10000 (no effect in atmosphere).

enumerator **TIMEWARP_X100000**
Set Timewarp to x100000 (no effect in atmosphere).

enumerator **TIMEWARP_X1_PHYSICAL**
Set Timewarp to x1 in atmosphere (no effect out of atmosphere).

enumerator **TIMEWARP_X2_PHYSICAL**
Set Timewarp to x2 in atmosphere (no effect out of atmosphere).

enumerator **TIMEWARP_X3_PHYSICAL**
Set Timewarp to x3 in atmosphere (no effect out of atmosphere).

enumerator **TIMEWARP_X4_PHYSICAL**
Set Timewarp to x4 in atmosphere (no effect out of atmosphere).

enumerator **TIMEWARP_UP**
Set Timewarp the next rate available.

enumerator **TIMEWARP_DOWN**
Set Timewarp the previous rate available.

enumerator **TIMEWARP_NEXT_MANEUVER**
Warp to the next maneuver.

enumerator **TIMEWARP_NEXT_SOI**
Warp to the next SOI change.

enumerator **TIMEWARP_APOAPSIS**
Warp to the apoapsis.

enumerator **TIMEWARP_PERIAPSIS**
Warp to the periapsis.

enumerator **TIMEWARP_NEXT_MORNING**
Warp to the next morning.

enumerator **TIMEWARP_CANCEL_AUTOWARP**
Cancel the current auto-timewarp and reset it to x1.

enum **CustomLogStatus**
Values:

enumerator **VERBOSE_ONLY**
If set, the message will only be put in the KSP log is the Simpit mod is in verbose mode.

enumerator **PRINT_TO_SCREEN**
If set, the message will also be displayed to the user (and not only in the log)

enumerator **NO_HEADER**
If set, the message will *not* be prefixed with 'Simpit :'.

enum **KeyboardEmulatorModifier**
Values:

enumerator **SHIFT_MOD**
If set, emulate the use of Shift.

enumerator **CTRL_MOD**
If set, emulate the use of CTRL.

enumerator **ALT_MOD**
If set, emulate the use of ALT.

enumerator **KEY_DOWN_MOD**
If set, emulate only the key down.

enumerator **KEY_UP_MOD**
If set, emulate only the key up.

enum **RotationAxes**
Rotation Axes These are used to indicate which axes in a ROTATION_MESSAGE or CAMERA_ROTATION_MESSAGE are active.

Values:

enumerator **PITCH_ROT**
Bitmask for the pitch axis.

enumerator **ROLL_ROT**
Bitmask for the roll axis.

enumerator **YAW_ROT**
Bitmask for the yaw axis.

enumerator **ZOOM_ROT**
Bitmask for the zoom axis.

Added here purely for camera control.

enum **TranslationAxes**
Translation Axes These are used to indicate which axes in a TRANSLATION_MESSAGE are active.

Values:

enumerator **X_TRANS**
Bitmask for the X axis.

enumerator **Y_TRANS**
Bitmask for the Y axis.

enumerator **Z_TRANS**
Bitmask for the Z axis.

enum **WheelAxes**
Translation Axes These are used to indicate which axes in a WHEEL_MESSAGE are active.

Values:

enumerator **STEER_WHEEL**
Bitmask for the steer.

enumerator **THROTTLE_WHEEL**
Bitmask for the throttle.

enum **AutopilotMode**
Autopilot Mode The possible Autopilot (SAS) modes.

This enum corresponds with VesselPilot.AutopilotMode in the KSP API.

Values:

enumerator **AP_STABILITYASSIST**

enumerator **AP_PROGRADE**

enumerator **AP_RETROGRADE**

enumerator **AP_NORMAL**

enumerator **AP_ANTINORMAL**

enumerator **AP_RADIALIN**

enumerator **AP_RADIALOUT**

enumerator **AP_TARGET**

enumerator **AP_ANTITARGET**

enumerator **AP_MANEUVER**

enum **CameraControlMode**

Camera Mode The possible camera mode control options.

Values:

enumerator **CAMERA_MODE_FLIGHT**

enumerator **FLIGHT_CAMERA_AUTO**

enumerator **FLIGHT_CAMERA_FREE**

enumerator **FLIGHT_CAMERA_ORBITAL**

enumerator **FLIGHT_CAMERA_CHASE**

enumerator **FLIGHT_CAMERA_LOCKED**

enumerator **CAMERA_NEXT**

enumerator **CAMERA_PREVIOUS**

enumerator **CAMERA_NEXT_MODE**

enumerator **CAMERA_PREVIOUS_MODE**

enum **FligthStatusFlags**

Flags used for the FlightStatus message.

Values:

enumerator **FLIGHT_IN_FLIGHT**

True if the game is currently in the flight screen.

enumerator **FLIGHT_IS_EVA**

True if the current flight is an EVA.

enumerator **FLIGHT_IS_RECOVERABLE**

True if the current vesset is recoverable.

enumerator **FLIGHT_IS_ATMO_TW**

True if the current Timewarp mode is for atmosphere (i.e.

physical timewarp).

This documentation was build using ArduinoDocs.

CHAPTER

FOUR

KERBAL SIMPIT COMPOUND MESSAGES

Structs for compound message types.

Functions

cagStatusMessage **parseCAGStatusMessage**(byte msg[])
Parse a message containing status of all the CAG.

Parameters **msg** – The byte array of the message body.

Returns *cagStatusMessage* A formatted *cagStatusMessage* struct.

SASInfoMessage **parseSASInfoMessage**(byte msg[])
Parse a message containing status of the SAS.

Parameters **msg** – The byte array of the message body.

Returns *SASInfoMessage* A formatted *SASInfoMessage* struct.

altitudeMessage **parseAltitude**(byte msg[])
Parse a message containing Altitude data.

Parameters **msg** – The byte array of the message body.

Returns *altitudeMessage* A formatted *altitudeMessage* struct.

apsidesMessage **parseApsides**(byte msg[])
Parse a message containing Apsides data.

Returns *apsidesMessage* A formatted *apsidesMessage* struct.

orbitInfoMessage **parseOrbitInfo**(byte msg[])
Parse a message containing orbital information.

Returns *orbitInfoMessage* A formatted *orbitInfoMessage* struct.

apsidesTimeMessage **parseApsidesTime**(byte msg[])
Parse a message containing Apsides Time data.

Returns *apsidesTimeMessage* A formatted *apsidesTimeMessage* struct.

resourceMessage **parseResource**(byte msg[])
Parse a message countaining Resource data.

Returns *resourceMessage* A formatted *resourceMessage* struct.

TACLSResourceMessage **parseTACLSResource**(byte msg[])
Parse a message countaining *TACLSResourceMessage* data.

Returns *TACLSResourceMessage* A formatted *TACLSResourceMessage* struct.

TACLSWasteMessage **parseTACLSWaste**(byte msg[])
Parse a message countaining *TACLSWasteMessage* data.

Returns *TACLSWasteMessage* A formatted *TACLSWasteMessage* struct.

CustomResourceMessage **parseCustomResource**(byte msg[])
Parse a message countaining *CustomResourceMessage* data.

Returns *CustomResourceMessage* A formatted *CustomResourceMessage* struct.

velocityMessage **parseVelocity**(byte msg[])
Parse a message containing Velocity data.

Returns *velocityMessage* A formatted *velocityMessage* struct.

targetMessage **parseTarget**(byte msg[])
Parse a message containing Target data.

Returns *targetMessage* A formatted *targetMessage* struct.

airspeedMessage **parseAirspeed**(byte msg[])
Parse a message containing Airspeed data.

Returns *airspeedMessage* a formatted *airspeedMessage* struct.

maneuverMessage **parseManeuver**(byte msg[])
Parse a message containing Maneuver data.

Returns *maneuverMessage* a formatted *maneuverMessage* struct.

deltaVMessage **parseDeltaV**(byte msg[])
Parse a message containing DeltaV data.

Returns *deltaVMessage* a formatted *deltaVMessage* struct.

deltaVEnvMessage **parseDeltaVEnv**(byte msg[])
Parse a message containing DeltaVEnv data.

Returns *deltaVEnvMessage* a formatted *deltaVEnvMessage* struct.

burnTimeMessage **parseBurnTime**(byte msg[])
Parse a message containing BurnTime data.

Returns *burnTimeMessage* a formatted *burnTimeMessage* struct.

tempLimitMessage **parseTempLimitMessage**(byte msg[])
Parse a message containing *tempLimitMessage* data.

Returns *tempLimitMessage* a formatted *tempLimitMessage* struct.

flightStatusMessage **parseFlightStatusMessage**(byte msg[])
Parse a message containing *flightStatusMessage* data.

Returns *flightStatusMessage* a formatted *flightStatusMessage* struct.

struct **cagStatusMessage**
#include <PayloadStructs.h> An Altitude message.

Public Functions

inline bool **is_action_activated**(byte i)

Public Members

byte **status**[32]
List of all the action status organised by bytes.

Read them with the is_action_activated method.

struct **SASInfoMessage**
#include <PayloadStructs.h> An SAS info message to reprsent the current SAS state.

Public Members

byte **currentSASMode**
Current SAS mode.

SAS modes are listed in the AutopilotMode enum. 255 is used to indicate a disabled SAS.

int16_t **SASModeAvailability**
bitmask for the availability of each SAS mode.

If a mode is not available, a SAS_MODE_MESSAGE setting this mode will be ignored.

struct **altitudeMessage**
#include <PayloadStructs.h> An Altitude message.

Public Members

float **sealevel**
Altitude above sea level.

float **surface**
Surface altitude at current position.

struct **apsidesMessage**
#include <PayloadStructs.h> An Apsides message.

Public Members

float **periapsis**
Current vessel's orbital periapsis.

float **apoapsis**
Current vessel's orbital apoapsis.

struct **apsidesTimeMessage**
#include <PayloadStructs.h> An Apsides Time message.

Public Members

int32_t **periapsis**

int32_t **apoapsis**
Time until the current vessel's orbital periapsis, in seconds.

struct **orbitInfoMessage**
#include <PayloadStructs.h> An message containing orbital information.

Public Members

float **eccentricity**
Current vessel's orbital eccentricity.

float **semiMajorAxis**
Current vessel's orbital semi major axis.

float **inclination**
Current vessel's orbital inclination.

float **longAscendingNode**
Current vessel's orbital longitude of ascending node.

float **argPeriapsis**
Current vessel's orbital argument of periapsis.

float **trueAnomaly**
Current vessel's orbital true anomaly.

float **meanAnomaly**
Current vessel's orbital mean anomaly.

float **period**
Current vessel's orbital period.

struct **flightStatusMessage**
#include <PayloadStructs.h> An message containing information about the current flight.

Public Functions

inline bool **isInFligth**()

inline bool **isInEVA**()

inline bool **isRecoverable**()

inline bool **isInAtmoTW**()

Public Members

byte **flightStatusFlags**
Different booleans as defined by FligthStatusFlags.

You can access them with the helper funtions.

byte **vesselSituation**
Current situation of the vessel, as defined by the Vessel.Situations enum in the KSP API (1 for Landed, 8 for flying, etc.).

byte **currentTWIndex**
Current TW index.

byte **crewCapacity**
Current vessel crew total capacity.

byte **crewCount**
Current vessel crew count.

byte **commNetSignalStrenghPercentage**
Current vessel commNet signal strengh (in percentage).

0 when CommNet is not used

struct **resourceMessage**
#include <PayloadStructs.h> A Resource message.

All resource messages use this struct for sending data.

Public Members

float **total**
Maximum capacity of the resource.

float **available**
Current resource level.

struct **TACLSResourceMessage**
#include <PayloadStructs.h> A Resource message for TACLS ressources.

Public Members

float **currentFood**
Current resource level for food.

float **maxFood**
Maximum capacity of food.

float **currentWater**
Current resource level for water.

float **maxWater**
Maximum capacity of water.

float **currentOxygen**
Current resource level for oxygen.

float **maxOxygen**
Maximum capacity of oxygen.

struct **TACLSWasteMessage**
#include <PayloadStructs.h> A Resource message for TACLS ressources.

Public Members

float **currentWaste**
Current resource level for waste.

float **maxWaste**
Maximum capacity of waste.

float **currentLiquidWaste**
Current resource level for liquid waste.

float **maxLiquidWaste**
Maximum capacity of liquid waste.

float **currentCO2**
Current resource level for CO2.

float **maxCO2**
Maximum capacity of CO2.

struct **CustomResourceMessage**
#include <PayloadStructs.h> A Resource message for custom ressources.

The resources must be set in the configuration file.

Public Members

float **currentResource1**
Current resource level resource 1.

float **maxResource1**
Maximum capacity of resource 1.

float **currentResource2**
Current resource level resource 2.

float **maxResource2**
Maximum capacity of resource 2.

float **currentResource3**
Current resource level resource 3.

float **maxResource3**
Maximum capacity of resource 3.

float **currentResource4**
Current resource level resource 4.

float **maxResource4**
Maximum capacity of resource 4.

struct **velocityMessage**
#include <PayloadStructs.h> A Velocity message.

Public Members

float **orbital**
Orbital velocity.

float **surface**
Surface velocity.

float **vertical**
Vertical velocity.

struct **targetMessage**
#include <PayloadStructs.h> A Target information message.

Public Members

float **distance**
Distance to target.

float **velocity**
Velocity relative to target.

struct **airspeedMessage**
#include <PayloadStructs.h> An Airspeed information message.

Public Members

float **IAS**
Indicated airspeed.

float **mach**
Mach number.

struct **maneuverMessage**
#include <PayloadStructs.h> A maneuver information message.

Public Members

float **timeToNextManeuver**
Time to the next planned maneuver.

float **deltaVNextManeuver**
Delta to the next planned maneuver.

float **durationNextManeuver**
Duration of the burn for the next planned maneuver.

float **deltaVTotal**
DeltaV of all the planned maneuvers.

struct **deltaVMessage**
#include <PayloadStructs.h> A deltaV information message.

Public Members

float **stageDeltaV**
DeltaV of the current stage.

float **totalDeltaV**
DeltaV of the whole vessel.

struct **deltaVEnvMessage**
#include <*PayloadStructs.h*> A deltaV information message in different environments.

Public Members

float **stageDeltaVASL**
DeltaV of the current stage at atmospheric sea level.

float **totalDeltaVASL**
DeltaV of the whole vessel at atmospheric sea level.

float **stageDeltaVVac**
DeltaV of the current stage in vacuum.

float **totalDeltaVVac**
DeltaV of the whole vessel in vacuum.

struct **burnTimeMessage**
#include <*PayloadStructs.h*> A burn time information message.

Public Members

float **stageBurnTime**
Burn time of the current stage.

float **totalBurnTime**
Burn time of the whole vessel.

struct **tempLimitMessage**
#include <*PayloadStructs.h*> A temperator limit message.

Public Members

byte **tempLimitPercentage**
Maximum temperature percentage (as current temp over max temp) of any part of the vessel.

byte **skinTempLimitPercentage**
Maximum temperature percentage (as current skin temp over max skin temp) of any part of the vessel.

struct **rotationMessage**
#include <PayloadStructs.h> A vessel rotation message.

This struct contains information about vessel rotation commands.

Public Functions

rotationMessage()

void **setPitch**(int16_t pitch)

void **setRoll**(int16_t roll)

void **setYaw**(int16_t yaw)

void **setPitchRollYaw**(int16_t pitch, int16_t roll, int16_t yaw)

Public Members

int16_t **pitch**
Vessel pitch.

int16_t **roll**
Vessel roll.

int16_t **yaw**
Vessel yaw.

byte **mask**
The mask indicates which elements are intentionally set.

Unset elements should be ignored. It should be one or more of:

- 1: pitch (PITCH_ROT)
- 2: roll (ROLL_ROT)
- 4: yaw (YAW_ROT)

struct **translationMessage**
#include <PayloadStructs.h> A vessel translation message.

This struct contains information about vessel translation commands.

Public Functions

translationMessage()

void **setX**(int16_t x)

void **setY**(int16_t y)

void **setZ**(int16_t z)

void **setXYZ**(int16_t x, int16_t y, int16_t z)

Public Members

int16_t **X**
Translation along the X axis.

int16_t **Y**
Translation along the Y axis.

int16_t **Z**
Translation along the Z axis.

byte **mask**
The mask indicates which elements are intentionally set.

Unset elements should be ignored. It should be one or more of:

- 1: X (X_TRANS)
- 2: Y (Y_TRANS)
- 4: Z (Z_TRANS)

struct **wheelMessage**
#include <PayloadStructs.h> A wheel control message.

This struct contains information about wheel steering and throttle.

Public Functions

wheelMessage()

void **setSteer**(int16_t steer)

void **setThrottle**(int16_t throttle)

void **setSteerThrottle**(int16_t steer, int16_t throttle)

Public Members

int16_t **steer**
Wheel steer.

int16_t **throttle**
Wheel throttle.

byte **mask**
The mask indicates which elements are intentionally set.

Unset elements should be ignored. It should be one or more of:

- 1: steer (STEER_WHEEL)
- 2: throttle (THROTTLE_WHEEL)

struct **throttleMessage**
#include <PayloadStructs.h> A throttle control message.

This struct contains information about throttle.

Public Members

int16_t **throttle**
Throttle.

struct **timewarpMessage**
#include <PayloadStructs.h> A timewarp message.

This struct contains a single timewarp command, defined in the Timewarp enum.

Public Members

byte **command**
Command in the Timewarp enum.

struct **cameraRotationMessage**
#include <PayloadStructs.h> A camera mode message.

This struct contains information about the camera mode.

Public Functions

cameraRotationMessage()

void **setPitch**(int16_t pitch)

void **setRoll**(int16_t roll)

void **setYaw**(int16_t yaw)

void **setZoom**(int16_t zoom)

void **setPitchRollYawZoom**(int16_t pitch, int16_t roll, int16_t yaw, int16_t zoom)

Public Members

int16_t **cameraPitch**

int16_t **cameraRoll**

int16_t **cameraYaw**

int16_t **cameraZoom**

byte **mask**
The mask indicates which elements are intentionally set.

Unset elements should be ignored. Based on the RotationAxes enum.

struct **keyboardEmulatorMessage**
#include <PayloadStructs.h> A keyboard emulator message.

This struct contains information about the keypress to emulate.

Public Functions

keyboardEmulatorMessage(int16_t keyCode)

keyboardEmulatorMessage(int16_t keyCode, byte modifier)

Public Members

byte **modifier**

int16_t **keyCode**

This documentation was build using ArduinoDocs.

CHAPTER

FIVE

TROUBLESHOOTING GUIDE

5.1 First check : mod installation and configuration

Go to your *KSP/GameData* folder. You should see a *KerbalSimpit* folder if SimPit is installed (else, install it with CKAN).

Open the configuration file *KSP/GameData/KerbalSimpit/PluginData/Settings.cfg*

The line `PortName = COM3` should be updated with the correct port number. It should be identical to the one used with the Arduino IDE on Windows. On a Linux or a Mac, the USB port ID will look like /dev/cu.usbmodem401 OR /dev/ttyUSBO.

The line `Verbose = False` should be replaced by `Verbose = True` to increase the level of detail written in the logs. It is assumed for this guide that Verbose is set to True.

Launch KSP and check in the logs that you can see the following lines (either by opening the in-game console with Alt+F12 or by

- `KerbalSimpit Has put a message into the console!`
- `KerbalSimpit: Settings loaded.`
- `KerbalSimpit: Using serial polling thread for XXX` where XXX is your port name (on Windows, it should be the same as the one set in the config file).

Make sure that you installed the SimPit Arduino library in the library folder of Arduino. You should see the *File/Examples* a category called *Kerbal Sim Pit*.

5.2 First example : receiving information from KSP

The next step is to check the connection from KSP to the controler. For this, we will use the provided example called *KerbalSimpitAltitudeTrigger*

Open the example with the Arduino IDE, compile it and upload it to the Arduino. Make sure that KSP is not launched when you upload the code. You should see the built-in led turned off.

Then launch KSP and go to the launchpad with a rocket (for instance the 2nd Training called *Basic Flight*).

Before taking off, you should see the folowing in the KSP.log file :

- The log line `KerbalSimpit: Opened COM3` meaning that the connection was started on KSP side.
- The log line `ACK received on port COM3. Handshake complete, Arduino library version '1.3.0` meaning that the connection was established between KSP and the controler.

- The log line `KerbalSimpit: Serial port 0 subscribing to channel 24` meaning that the controler requested an update of the altitude of the craft.

You should also see the built-in LED turned on when the connection is established.

Then activate the SAS and launch your craft straight up. As soon as its altitude is above 500m, you should see the built-in LED turned off.

After the fligth, exit KSP and go check in your log for any error related to KSP. If no error are found and if you saw the LED turning off when going higher than 500m, congratulation. Your controler can receive information from KSP.

5.3 Second example : sending information to KSP

For this test, we will use the provided example called *KerbalSimpitStageDemo*. The objective is to launch your rocket based on an input from the controler. You will need to use a button connected to the pin 2 of the Arduino on one side and on the ground to the other side (see this example from Arduino). You can also just plug a wire from the pin 2 to the GND pin to simulate a push on the button if you don't have one.

Start like the previous example :

- Compile and upload the code while KSP is closed
- Launch KSP and open the *Basic Flight* training
- Check for the connection in the KSP log (note : there will be no log indicating a subscription from the controler to a channel).

Then activate the button (or insert the wire). The rocket should take off.

After the fligth, exit KSP and go check in your log for any error related to KSP. If no error are found and your rocket took off, congratulation !! You are now ready to start your own controler.

5.4 Common mistakes

5.4.1 At one point, my controler stopped working

Please check for any error in the KSP log. If at any point an error occured, the SimPit mod will stop and no information can be shared with the controler, thus giving the impression that the controler is not working.

5.4.2 What does an error look like in the KSP log

Here is an example.

[EXC 20:48:03.683] IOException: A device attached to the system is not functioning.
System.IO.Ports.WinSerialStream.ReportIOError (System.String optional_arg) (at <376e8c39bbab4f1193a569c8dbe4305c>:0) System.IO.Ports.WinSerialStream.Write (System.Byte[] buffer, System.Int32 offset, System.Int32 count) (at <376e8c39bbab4f1193a569c8dbe4305c>:0) System.IO.Ports.SerialPort.Write (System.Byte[] buffer, System.Int32 offset, System.Int32 count) (at <376e8c39bbab4f1193a569c8dbe4305c>:0) (wrapper remoting-invoke-with-check) System.IO.Ports.SerialPort.Write(byte[],int,int) KerbalSimpit.Serial.KSPSerialPort.<SerialWriteQueueRunner>b__29_0 () (at <d23bc766ee8645b4813da6de1d115a35>:0) KerbalSimpit.Serial.KSPSerialPort.SerialWriteQueueRunner () (at <d23bc766ee8645b4813da6de1d115a35>:0) System.Threading.ThreadHelper.ThreadStart_Context

(System.Object state) (at <ad04dee02e7e4a85a1299c7ee81c79f6>:0) System.Threading.ExecutionContext.RunInternal (System.Threading.ExecutionContext executionContext, System.Threading.ContextCallback callback, System.Object state, System.Boolean preserveSyncCtx) (at <ad04dee02e7e4a85a1299c7ee81c79f6>:0) System.Threading.ExecutionContext.Run (System.Threading.ExecutionContext executionContext, System.Threading.ContextCallback callback, System.Object state, System.Boolean preserveSyncCtx) (at <ad04dee02e7e4a85a1299c7ee81c79f6>:0) System.Threading.ExecutionContext.Run (System.Threading.ExecutionContext executionContext, System.Threading.ContextCallback callback, System.Object state) (at <ad04dee02e7e4a85a1299c7ee81c79f6>:0) System.Threading.ThreadHelper.ThreadStart () (at <ad04dee02e7e4a85a1299c7ee81c79f6>:0) UnityEngine.UnhandledExceptionHandler:<RegisterUECatcher>m__0(Object, UnhandledExceptionEventArgs)

This documentation was built using ArduinoDocs.

INDEX

A

ActionGroupIndexes (*C++ enum*), 16
ActionGroupIndexes::ABORT_ACTION (*C++ enumerator*), 17
ActionGroupIndexes::BRAKES_ACTION (*C++ enumerator*), 17
ActionGroupIndexes::GEAR_ACTION (*C++ enumerator*), 16
ActionGroupIndexes::LIGHT_ACTION (*C++ enumerator*), 16
ActionGroupIndexes::RCS_ACTION (*C++ enumerator*), 16
ActionGroupIndexes::SAS_ACTION (*C++ enumerator*), 16
ActionGroupIndexes::STAGE_ACTION (*C++ enumerator*), 16
airspeedMessage (*C++ struct*), 30
airspeedMessage::IAS (*C++ member*), 30
airspeedMessage::mach (*C++ member*), 30
altitudeMessage (*C++ struct*), 25
altitudeMessage::sealevel (*C++ member*), 25
altitudeMessage::surface (*C++ member*), 25
apsidesMessage (*C++ struct*), 25
apsidesMessage::apoapsis (*C++ member*), 26
apsidesMessage::periapsis (*C++ member*), 26
apsidesTimeMessage (*C++ struct*), 26
apsidesTimeMessage::apoapsis (*C++ member*), 26
apsidesTimeMessage::periapsis (*C++ member*), 26
AutopilotMode (*C++ enum*), 19
AutopilotMode::AP_ANTINORMAL (*C++ enumerator*), 20
AutopilotMode::AP_ANTITARGET (*C++ enumerator*), 20
AutopilotMode::AP_MANEUVER (*C++ enumerator*), 20
AutopilotMode::AP_NORMAL (*C++ enumerator*), 20
AutopilotMode::AP_PROGRADE (*C++ enumerator*), 20
AutopilotMode::AP_RADIALIN (*C++ enumerator*), 20
AutopilotMode::AP_RADIALOUT (*C++ enumerator*), 20
AutopilotMode::AP_RETROGRADE (*C++ enumerator*), 20
AutopilotMode::AP_STABILITYASSIST (*C++ enumerator*), 19
AutopilotMode::AP_TARGET (*C++ enumerator*), 20

B

burnTimeMessage (*C++ struct*), 31
burnTimeMessage::stageBurnTime (*C++ member*), 31
burnTimeMessage::totalBurnTime (*C++ member*), 31

C

cagStatusMessage (*C++ struct*), 24
cagStatusMessage::is_action_activated (*C++ function*), 25
cagStatusMessage::status (*C++ member*), 25
CameraControlMode (*C++ enum*), 20
CameraControlMode::CAMERA_MODE_FLIGHT (*C++ enumerator*), 20
CameraControlMode::CAMERA_NEXT (*C++ enumerator*), 20
CameraControlMode::CAMERA_NEXT_MODE (*C++ enumerator*), 20
CameraControlMode::CAMERA_PREVIOUS (*C++ enumerator*), 20
CameraControlMode::CAMERA_PREVIOUS_MODE (*C++ enumerator*), 20
CameraControlMode::FLIGHT_CAMERA_AUTO (*C++ enumerator*), 20
CameraControlMode::FLIGHT_CAMERA_CHASE (*C++ enumerator*), 20
CameraControlMode::FLIGHT_CAMERA_FREE (*C++ enumerator*), 20
CameraControlMode::FLIGHT_CAMERA_LOCKED (*C++ enumerator*), 20
CameraControlMode::FLIGHT_CAMERA_ORBITAL (*C++ enumerator*), 20
cameraRotationMessage (*C++ struct*), 35
cameraRotationMessage::cameraPitch (*C++ member*), 35
cameraRotationMessage::cameraRoll (*C++ member*), 35

cameraRotationMessage::cameraRotationMessage (C++ function), 35
cameraRotationMessage::cameraYaw (C++ member), 35
cameraRotationMessage::cameraZoom (C++ member), 35
cameraRotationMessage::mask (C++ member), 35
cameraRotationMessage::setPitch (C++ function), 35
cameraRotationMessage::setPitchRollYawZoom (C++ function), 35
cameraRotationMessage::setRoll (C++ function), 35
cameraRotationMessage::setYaw (C++ function), 35
cameraRotationMessage::setZoom (C++ function), 35
CommonPackets (C++ enum), 11
CommonPackets::ECHO_REQ_MESSAGE (C++ enumerator), 11
CommonPackets::ECHO_RESP_MESSAGE (C++ enumerator), 11
CommonPackets::SYNC_MESSAGE (C++ enumerator), 11
CustomLogStatus (C++ enum), 18
CustomLogStatus::NO_HEADER (C++ enumerator), 18
CustomLogStatus::PRINT_TO_SCREEN (C++ enumerator), 18
CustomLogStatus::VERBOSE_ONLY (C++ enumerator), 18
CustomResourceMessage (C++ struct), 28
CustomResourceMessage::currentResource1 (C++ member), 29
CustomResourceMessage::currentResource2 (C++ member), 29
CustomResourceMessage::currentResource3 (C++ member), 29
CustomResourceMessage::currentResource4 (C++ member), 29
CustomResourceMessage::maxResource1 (C++ member), 29
CustomResourceMessage::maxResource2 (C++ member), 29
CustomResourceMessage::maxResource3 (C++ member), 29
CustomResourceMessage::maxResource4 (C++ member), 29

D

deltaVEnvMessage (C++ struct), 31
deltaVEnvMessage::stageDeltaVASL (C++ member), 31
deltaVEnvMessage::stageDeltaVVac (C++ member), 31
deltaVEnvMessage::totalDeltaVASL (C++ member), 31
deltaVEnvMessage::totalDeltaVVac (C++ member), 31
deltaVMessage (C++ struct), 30
deltaVMessage::stageDeltaV (C++ member), 31
deltaVMessage::totalDeltaV (C++ member), 31

F

flightStatusMessage (C++ struct), 26
flightStatusMessage::commNetSignalStrenghPercentage (C++ member), 27
flightStatusMessage::crewCapacity (C++ member), 27
flightStatusMessage::crewCount (C++ member), 27
flightStatusMessage::currentTWIndex (C++ member), 27
flightStatusMessage::flightStatusFlags (C++ member), 27
flightStatusMessage::isInAtmoTW (C++ function), 27
flightStatusMessage::isInEVA (C++ function), 27
flightStatusMessage::isInFligth (C++ function), 27
flightStatusMessage::isRecoverable (C++ function), 27
flightStatusMessage::vesselSituation (C++ member), 27
FligthStatusFlags (C++ enum), 20
FligthStatusFlags::FLIGHT_IN_FLIGHT (C++ enumerator), 20
FligthStatusFlags::FLIGHT_IS_ATMO_TW (C++ enumerator), 20
FligthStatusFlags::FLIGHT_IS_EVA (C++ enumerator), 20
FligthStatusFlags::FLIGHT_IS_RECOVERABLE (C++ enumerator), 20

I

InboundPackets (C++ enum), 15
InboundPackets::AGACTIVATE_MESSAGE (C++ enumerator), 15
InboundPackets::AGDEACTIVATE_MESSAGE (C++ enumerator), 15
InboundPackets::AGTOGGLE_MESSAGE (C++ enumerator), 15
InboundPackets::CAGACTIVATE_MESSAGE (C++ enumerator), 15
InboundPackets::CAGDEACTIVATE_MESSAGE (C++ enumerator), 15
InboundPackets::CAGTOGGLE_MESSAGE (C++ enumerator), 15

InboundPackets::CAMERA_CONTROL_MODE (C++ enumerator), 16
InboundPackets::CAMERA_ROTATION_MESSAGE (C++ enumerator), 16
InboundPackets::CUSTOM_LOG (C++ enumerator), 16
InboundPackets::DEREGISTER_MESSAGE (C++ enumerator), 15
InboundPackets::KEYBOARD_EMULATOR (C++ enumerator), 16
InboundPackets::REGISTER_MESSAGE (C++ enumerator), 15
InboundPackets::ROTATION_MESSAGE (C++ enumerator), 15
InboundPackets::SAS_MODE_MESSAGE (C++ enumerator), 16
InboundPackets::THROTTLE_MESSAGE (C++ enumerator), 16
InboundPackets::TIMEWARP_MESSAGE (C++ enumerator), 16
InboundPackets::TRANSLATION_MESSAGE (C++ enumerator), 15
InboundPackets::WHEEL_MESSAGE (C++ enumerator), 16

K

KerbalSimpit (C++ class), 7
KerbalSimpit::activateAction (C++ function), 8
KerbalSimpit::activateCAG (C++ function), 8
KerbalSimpit::deactivateAction (C++ function), 8
KerbalSimpit::deactivateCAG (C++ function), 8
KerbalSimpit::deregisterChannel (C++ function), 7
KerbalSimpit::inboundHandler (C++ function), 7
KerbalSimpit::init (C++ function), 7
KerbalSimpit::KerbalSimpit (C++ function), 7
KerbalSimpit::registerChannel (C++ function), 7
KerbalSimpit::send (C++ function), 8
KerbalSimpit::setSASMode (C++ function), 9
KerbalSimpit::toggleAction (C++ function), 9
KerbalSimpit::toggleCAG (C++ function), 8
KerbalSimpit::update (C++ function), 8
keyboardEmulatorMessage (C++ struct), 35
keyboardEmulatorMessage::keyboardEmulatorMessage (C++ function), 36
keyboardEmulatorMessage::keyCode (C++ member), 36
keyboardEmulatorMessage::modifier (C++ member), 36
KeyboardEmulatorModifier (C++ enum), 18
KeyboardEmulatorModifier::ALT_MOD (C++ enumerator), 18
KeyboardEmulatorModifier::CTRL_MOD (C++ enumerator), 18
KeyboardEmulatorModifier::KEY_DOWN_MOD (C++ enumerator), 18
KeyboardEmulatorModifier::KEY_UP_MOD (C++ enumerator), 18
KeyboardEmulatorModifier::SHIFT_MOD (C++ enumerator), 18

M

maneuverMessage (C++ struct), 30
maneuverMessage::deltaVNextManeuver (C++ member), 30
maneuverMessage::deltaVTotal (C++ member), 30
maneuverMessage::durationNextManeuver (C++ member), 30
maneuverMessage::timeToNextManeuver (C++ member), 30

O

orbitInfoMessage (C++ struct), 26
orbitInfoMessage::argPeriapsis (C++ member), 26
orbitInfoMessage::eccentricity (C++ member), 26
orbitInfoMessage::inclination (C++ member), 26
orbitInfoMessage::longAscendingNode (C++ member), 26
orbitInfoMessage::meanAnomaly (C++ member), 26
orbitInfoMessage::period (C++ member), 26
orbitInfoMessage::semiMajorAxis (C++ member), 26
orbitInfoMessage::trueAnomaly (C++ member), 26
OutboundPackets (C++ enum), 11
OutboundPackets::AB_MESSAGE (C++ enumerator), 12
OutboundPackets::AB_STAGE_MESSAGE (C++ enumerator), 12
OutboundPackets::ACTIONSTATUS_MESSAGE (C++ enumerator), 13
OutboundPackets::AIRSPEED_MESSAGE (C++ enumerator), 13
OutboundPackets::ALTITUDE_MESSAGE (C++ enumerator), 13
OutboundPackets::APSIDES_MESSAGE (C++ enumerator), 13
OutboundPackets::APSIDESTIME_MESSAGE (C++ enumerator), 13
OutboundPackets::BURNTIME_MESSAGE (C++ enumerator), 14
OutboundPackets::CAGSTATUS_MESSAGE (C++ enumerator), 14
OutboundPackets::CUSTOM_RESOURCE_1_MESSAGE (C++ enumerator), 13
OutboundPackets::CUSTOM_RESOURCE_2_MESSAGE (C++ enumerator), 13

OutboundPackets::DELTAV_MESSAGE (C++ enumerator), 14
OutboundPackets::DELTAVENV_MESSAGE (C++ enumerator), 14
OutboundPackets::ELECTRIC_MESSAGE (C++ enumerator), 12
OutboundPackets::EVA_MESSAGE (C++ enumerator), 12
OutboundPackets::FLIGHT_STATUS_MESSAGE (C++ enumerator), 15
OutboundPackets::LF_MESSAGE (C++ enumerator), 11
OutboundPackets::LF_STAGE_MESSAGE (C++ enumerator), 11
OutboundPackets::MANEUVER_MESSAGE (C++ enumerator), 13
OutboundPackets::MONO_MESSAGE (C++ enumerator), 12
OutboundPackets::ORBIT_INFO (C++ enumerator), 13
OutboundPackets::ORE_MESSAGE (C++ enumerator), 12
OutboundPackets::OX_MESSAGE (C++ enumerator), 11
OutboundPackets::OX_STAGE_MESSAGE (C++ enumerator), 11
OutboundPackets::SAS_MODE_INFO_MESSAGE (C++ enumerator), 13
OutboundPackets::SCENE_CHANGE_MESSAGE (C++ enumerator), 15
OutboundPackets::SF_MESSAGE (C++ enumerator), 12
OutboundPackets::SF_STAGE_MESSAGE (C++ enumerator), 12
OutboundPackets::SOI_MESSAGE (C++ enumerator), 15
OutboundPackets::TACLS_RESOURCE_MESSAGE (C++ enumerator), 12
OutboundPackets::TACLS_WASTE_MESSAGE (C++ enumerator), 13
OutboundPackets::TARGETINFO_MESSAGE (C++ enumerator), 14
OutboundPackets::TEMP_LIMIT_MESSAGE (C++ enumerator), 14
OutboundPackets::VELOCITY_MESSAGE (C++ enumerator), 13
OutboundPackets::XENON_GAS_MESSAGE (C++ enumerator), 12
OutboundPackets::XENON_GAS_STAGE_MESSAGE (C++ enumerator), 12

P

parseAirspeed (C++ function), 24
parseAltitude (C++ function), 23
parseApsides (C++ function), 23
parseApsidesTime (C++ function), 23
parseBurnTime (C++ function), 24
parseCAGStatusMessage (C++ function), 23
parseCustomResource (C++ function), 24
parseDeltaV (C++ function), 24
parseDeltaVEnv (C++ function), 24
parseFlightStatusMessage (C++ function), 24
parseManeuver (C++ function), 24
parseOrbitInfo (C++ function), 23
parseResource (C++ function), 23
parseSASInfoMessage (C++ function), 23
parseTACLSResource (C++ function), 23
parseTACLSWaste (C++ function), 23
parseTarget (C++ function), 24
parseTempLimitMessage (C++ function), 24
parseVelocity (C++ function), 24

R

resourceMessage (C++ struct), 27
resourceMessage::available (C++ member), 27
resourceMessage::total (C++ member), 27
RotationAxes (C++ enum), 19
RotationAxes::PITCH_ROT (C++ enumerator), 19
RotationAxes::ROLL_ROT (C++ enumerator), 19
RotationAxes::YAW_ROT (C++ enumerator), 19
RotationAxes::ZOOM_ROT (C++ enumerator), 19
rotationMessage (C++ struct), 32
rotationMessage::mask (C++ member), 32
rotationMessage::pitch (C++ member), 32
rotationMessage::roll (C++ member), 32
rotationMessage::rotationMessage (C++ function), 32
rotationMessage::setPitch (C++ function), 32
rotationMessage::setPitchRollYaw (C++ function), 32
rotationMessage::setRoll (C++ function), 32
rotationMessage::setYaw (C++ function), 32
rotationMessage::yaw (C++ member), 32

S

SASInfoMessage (C++ struct), 25
SASInfoMessage::currentSASMode (C++ member), 25
SASInfoMessage::SASModeAvailability (C++ member), 25

T

TACLSResourceMessage (C++ struct), 27
TACLSResourceMessage::currentFood (C++ member), 28
TACLSResourceMessage::currentOxygen (C++ member), 28

TACLSResourceMessage::currentWater (C++ member), 28
TACLSResourceMessage::maxFood (C++ member), 28
TACLSResourceMessage::maxOxygen (C++ member), 28
TACLSResourceMessage::maxWater (C++ member), 28
TACLSWasteMessage (C++ struct), 28
TACLSWasteMessage::currentCO2 (C++ member), 28
TACLSWasteMessage::currentLiquidWaste (C++ member), 28
TACLSWasteMessage::currentWaste (C++ member), 28
TACLSWasteMessage::maxCO2 (C++ member), 28
TACLSWasteMessage::maxLiquidWaste (C++ member), 28
TACLSWasteMessage::maxWaste (C++ member), 28
targetMessage (C++ struct), 29
targetMessage::distance (C++ member), 30
targetMessage::velocity (C++ member), 30
tempLimitMessage (C++ struct), 31
tempLimitMessage::skinTempLimitPercentage (C++ member), 32
tempLimitMessage::tempLimitPercentage (C++ member), 32
throttleMessage (C++ struct), 34
throttleMessage::throttle (C++ member), 34
Timewarp (C++ enum), 17
Timewarp::TIMEWARP_APOAPSIS (C++ enumerator), 18
Timewarp::TIMEWARP_CANCEL_AUTOWARP (C++ enumerator), 18
Timewarp::TIMEWARP_DOWN (C++ enumerator), 18
Timewarp::TIMEWARP_NEXT_MANEUVER (C++ enumerator), 18
Timewarp::TIMEWARP_NEXT_MORNING (C++ enumerator), 18
Timewarp::TIMEWARP_NEXT_SOI (C++ enumerator), 18
Timewarp::TIMEWARP_PERIAPSIS (C++ enumerator), 18
Timewarp::TIMEWARP_UP (C++ enumerator), 17
Timewarp::TIMEWARP_X1 (C++ enumerator), 17
Timewarp::TIMEWARP_X10 (C++ enumerator), 17
Timewarp::TIMEWARP_X100 (C++ enumerator), 17
Timewarp::TIMEWARP_X1000 (C++ enumerator), 17
Timewarp::TIMEWARP_X10000 (C++ enumerator), 17
Timewarp::TIMEWARP_X100000 (C++ enumerator), 17
Timewarp::TIMEWARP_X1_PHYSICAL (C++ enumerator), 17
Timewarp::TIMEWARP_X2_PHYSICAL (C++ enumerator), 17
Timewarp::TIMEWARP_X3_PHYSICAL (C++ enumerator), 17
Timewarp::TIMEWARP_X4_PHYSICAL (C++ enumerator), 17
Timewarp::TIMEWARP_X5 (C++ enumerator), 17
Timewarp::TIMEWARP_X50 (C++ enumerator), 17
timewarpMessage (C++ struct), 34
timewarpMessage::command (C++ member), 35
TranslationAxes (C++ enum), 19
TranslationAxes::X_TRANS (C++ enumerator), 19
TranslationAxes::Y_TRANS (C++ enumerator), 19
TranslationAxes::Z_TRANS (C++ enumerator), 19
translationMessage (C++ struct), 32
translationMessage::mask (C++ member), 33
translationMessage::setX (C++ function), 33
translationMessage::setXYZ (C++ function), 33
translationMessage::setY (C++ function), 33
translationMessage::setZ (C++ function), 33
translationMessage::translationMessage (C++ function), 33
translationMessage::X (C++ member), 33
translationMessage::Y (C++ member), 33
translationMessage::Z (C++ member), 33

V

velocityMessage (C++ struct), 29
velocityMessage::orbital (C++ member), 29
velocityMessage::surface (C++ member), 29
velocityMessage::vertical (C++ member), 29

W

WheelAxes (C++ enum), 19
WheelAxes::STEER_WHEEL (C++ enumerator), 19
WheelAxes::THROTTLE_WHEEL (C++ enumerator), 19
wheelMessage (C++ struct), 33
wheelMessage::mask (C++ member), 34
wheelMessage::setSteer (C++ function), 34
wheelMessage::setSteerThrottle (C++ function), 34
wheelMessage::setThrottle (C++ function), 34
wheelMessage::steer (C++ member), 34
wheelMessage::throttle (C++ member), 34
wheelMessage::wheelMessage (C++ function), 34