KerbalSimpit Documentation

Release 1.0

KerbalSimpit

Dec 16, 2018

Contents

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 1

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.
```

(continues on next page)

(continued from previous page)

```
    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 2

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.

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>
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.

void **send**(byte *messageType*, byte *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 3

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:

SYNC_MESSAGE = 0
Sync message. Used for handshaking.

ECHO_REQ_MESSAGE = 1
Echo request. Either end can send this, and an echo response is expected.

ECHO_RESP_MESSAGE = 2
Echo response. Sent in reply to an echo request.

enum OutboundPackets
Outbound packets.

IDs for packets that go from the game to devices.

Values:

SCENE_CHANGE_MESSAGE = 3
Scene change packets are sent by the plugin when entering or leaving the flight scene.

ALTITUDE_MESSAGE = 8
Sea level and surface altitude.

Messages on this channel contain an *altitudeMessage*.

APSIDES_MESSAGE = 9
Apoapsis and periapsis.

Messages on this channel contain an *apsidesMessage*.

LF_MESSAGE = 10
Liquid fuel in the vessel.

Messages on this channel contain a *resourceMessage*.

LF_STAGE_MESSAGE = 11
Liquid fuel in the current stage.

Messages on this channel contain a *resourceMessage*.

OX_MESSAGE = 12
Oxidizer in the vessel.

Messages on this channel contain a *resourceMessage*.

OX_STAGE_MESSAGE = 13
Oxidizer in the current stage.

Messages on this channel contain a *resourceMessage*.

SF_MESSAGE = 14
Solid fuel in the vessel.

Messages on this channel contain a *resourceMessage*.

SF_STAGE_MESSAGE = 15
Solid fuel in the current stage.

Messages on this channel contain a *resourceMessage*.

MONO_MESSAGE = 16
Monoprollent in the vessel.

Messages on this channel contain a *resourceMessage*.

ELECTRIC_MESSAGE = 17
Electic Charge in the vessel.

Messages on this channel contain a *resourceMessage*.

EVA_MESSAGE = 18
EVA propellant.

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

ORE_MESSAGE = 19
Ore in the vessel.

Messages on this channel contain a *resourceMessage*.

AB_MESSAGE = 20
Ablator in the vessel.

Messages on this channel contain a *resourceMessage*.

AB_STAGE_MESSAGE = 21
Ablator in the current stage.

Messages on this channel contain a *resourceMessage*.

VELOCITY_MESSAGE = 22
Vessel velocity.

Messages on this channel contain a *velocityMessage*.

ACTIONSTATUS_MESSAGE = 23
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 & 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

APSIDESTIME_MESSAGE = 24
Time to the next apoapsis and periapsis.

Messages on this channel contain an *apsidesTimeMessage*.

TARGETINFO_MESSAGE = 25
Information about targetted object.

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

SOI_MESSAGE = 26
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.

AIRSPEED_MESSAGE = 27
Information about airspeed.

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

enum InboundPackets
Inbound packets.

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

Values:

REGISTER_MESSAGE = 8
Register to receive messages on a given channel.

DEREGISTER_MESSAGE = 9
Deregister, indicate that no further messages for the given channel should be sent.

CAGACTIVATE_MESSAGE = 10
Activate the given Custom Action Group(s).

CAGDEACTIVATE_MESSAGE = 11
Deactivate the given Custom Action Group(s).

CAGTOGGLE_MESSAGE = 12
Toggle the given Custom Action Group(s) (Active CAGs will deactivate, inactive CAGs will activate).

AGACTIVATE_MESSAGE = 13
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.

AGDEACTIVATE_MESSAGE = 14
Deactivate the given standard Action Group(s).

AGTOGGLE_MESSAGE = 15
Toggle the given standard Action Group(s).

ROTATION_MESSAGE = 16
Send vessel rotation commands.

TRANSLATION_MESSAGE = 17
Send vessel translation commands.

WHEEL_MESSAGE = 18
Send wheel steering/throttle commands.

THROTTLE_MESSAGE = 19
Send vessel throttle commands.

SAS_MODE_MESSAGE = 20
Send SAS mode commands.

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

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

Values:

STAGE_ACTION = 1
Bitmask for the Stage action group.

GEAR_ACTION = 2
Bitmask for the Gear action group.

LIGHT_ACTION = 4
Bitmask for the Light action group.

RCS_ACTION = 8
Bitmask for the RCS action group.

SAS_ACTION = 16
Bitmask for the SAS action group.

BRAKES_ACTION = 32
Bitmask for the Brakes action group.

ABORT_ACTION = 64
Bitmask for the Abort action group.

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

Values:

PITCH_ROT = 1
Bitmask for the pitch axis.

ROLL_ROT = 2
Bitmask for the roll axis.

YAW_ROT = 4
Bitmask for thw yaw axis.

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

Values:

X_TRANS = 1
Bitmask for the X axis.

Y_TRANS = 2
Bitmask for the Y axis.

Z_TRANS = 4
Bitmask for the Z axis.

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

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

Values:

AP_STABILITYASSIST = 1

AP_PROGRADE = 2

AP_RETROGRADE = 3

AP_NORMAL = 4

AP_ANTINORMAL = 5

AP_RADIALIN = 6

AP_RADIALOUT = 7

AP_TARGET = 8

AP_ANTITARGET = 9

AP_MANEUVER = 10

This documentation was build using ArduinoDocs.

CHAPTER 4

Kerbal Simpit Compound Messages

Structs for compound message types.

Functions

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

Return *altitudeMessage* A formatted *altitudeMessage* struct.

Parameters

- `msg`: The byte array of the message body.

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

Return *apsidesMessage* A formatted *apsidesMessage* struct.

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

Return *apsidesTimeMessage* A formatted *apsidesTimeMessage* struct.

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

Return *resourceMessage* A formatted *resourceMessage* struct.

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

Return *velocityMessage* A formatted *velocityMessage* struct.

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

Return *targetMessage* A formatted *targetMessage* struct.

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

Return *airspeedMessage* a formatted *airspeedMessage* struct.

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

This struct contains information about vessel rotation commands.

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
- 2: roll
- 4: yaw

struct translationMessage

#include <PayloadStructs.h> A vessel translation message.

This struct contains information about vessel translation commands.

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
- 2: Y
- 4: Z

struct wheelMessage

#include <PayloadStructs.h> A wheel control message.

This struct contains information about wheel steering and 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
- 2: throttle

This documentation was build using ArduinoDocs.

This documentation was built using ArduinoDocs.

Index

L

M

O

P

R

S

T

V

W

X

Y

Z