XPLPro

Generated by Doxygen 1.9.6

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 XPLPro Class Reference

Core class for the XPLPro Arduino library.

```
#include <XPLPro.h>
```

Public Member Functions

- XPLPro (Stream ∗device)

    Constructor.
- void begin (const char ∗devicename, void(∗initFunction)(void), void(∗stopFunction)(void), void(∗inbound↩ Handler)(int))

    Register device and set callback functions.
- int connectionStatus ()

    Return connection status.
- int commandTrigger (int commandHandle)

    Trigger a command once.
- int commandTrigger (int commandHandle, int triggerCount)

    Trigger a command multiple times.
- int commandStart (int commandHandle)

    Start a command. All commandStart must be balanced with a commandEnd.
- int commandEnd (int commandHandle)

    End a command. All commandStart must be balanced with a commandEnd.
- void datarefWrite (int handle, long value)

    Write an integer DataRef.
- void datarefWrite (int handle, int value)

    Write an integer DataRef. Maps to long DataRefs.
- void datarefWrite (int handle, long value, int arrayElement)

    Write a Integer DataRef to an array element.
- void datarefWrite (int handle, int value, int arrayElement)

    Write a Integer DataRef to an array element. Maps to long DataRefs.
- void datarefWrite (int handle, float value)

    Write a float DataRef.
- void datarefWrite (int handle, float value, int arrayElement)

Write a float DataRef to an array element.
- void requestUpdates (int handle, int rate, float precision)

  Request DataRef updates from the plugin.
- void requestUpdates (int handle, int rate, float precision, int arrayElement)

  Request DataRef updates from the plugin for an array DataRef.
- void setScaling (int handle, int inLow, int inHigh, int outLow, int outHigh)

  set scaling factor for a DataRef (offload mapping to the plugin)
- int registerDataRef (XPString_t ∗datarefName)

  Register a DataRef and obtain a handle.
- int registerCommand (XPString_t ∗commandName)

  Register a Command and obtain a handle.
- float datarefReadFloat ()

  Read the received float DataRef.
- long datarefReadInt ()

  Read the received integer DataRef.
- int datarefReadElement ()

  Read the received array element.
- int sendDebugMessage (const char ∗msg)

  Send a debug message to the plugin.
- int sendSpeakMessage (const char ∗msg)

  Send a speech message to the plugin.
- void sendResetRequest (void)

  Request a reset from the plugin.
- int xloop ()

  Cyclic loop handler, must be called in idle task.

3.1.1 Detailed Description

Core class for the XPLPro Arduino library.

Definition at line 99 of file XPLPro.h.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 XPLPro()

```
XPLPro::XPLPro (
            Stream ∗ device )
```

Constructor.

Parameters

device	Device to use (should be &Serial)

Definition at line 5 of file XPLPro.cpp.

3.1.3 Member Function Documentation

3.1.3.1 begin()

```
void XPLPro::begin (
             const char * devicename,
             void(*)(void) initFunction,
             void(*)(void) stopFunction,
             void(*)(int) inboundHandler )
```

Register device and set callback functions.

Parameters

devicename	Device name
initFunction	Callback for DataRef and Command registration
stopFunction	Callback for XPlane shutdown or plane change
inboundHandler	Callback for incoming DataRefs

Definition at line 11 of file XPLPro.cpp.

3.1.3.2 commandEnd()

```
int XPLPro::commandEnd (
             int commandHandle )
```

End a command. All commandStart must be balanced with a commandEnd.

Parameters

commandHandle	Handle of the command to start

Returns

0: OK, -1: command was not registered

Definition at line 58 of file XPLPro.cpp.

3.1.3.3 commandStart()

```
int XPLPro::commandStart (
            int commandHandle )
```

Start a command. All commandStart must be balanced with a commandEnd.

Parameters

commandHandle	Handle of the command to start

Returns

0: OK, -1: command was not registered

Definition at line 48 of file XPLPro.cpp.

3.1.3.4 commandTrigger() [1/2]

```
int XPLPro::commandTrigger (
            int commandHandle )  [inline]
```

Trigger a command once.

Parameters

commandHandle	of the command to trigger

Returns

0: OK, -1: command was not registered

Definition at line 120 of file XPLPro.h.

3.1.3.5 commandTrigger() [2/2]

```
int XPLPro::commandTrigger (
            int commandHandle,
            int triggerCount )
```

Trigger a command multiple times.

Parameters

commandHandle	Handle of the command to trigger
triggerCount	Number of times to trigger the command

Returns

0: OK, -1: command was not registered

Definition at line 37 of file XPLPro.cpp.

3.1.3.6 connectionStatus()

```
int XPLPro::connectionStatus ( )
```

Return connection status.

Returns

True if connection to XPlane established

Definition at line 68 of file XPLPro.cpp.

3.1.3.7 datarefReadElement()

```
int XPLPro::datarefReadElement ( )  [inline]
```

Read the received array element.

Returns

Received array element

Definition at line 207 of file XPLPro.h.

3.1.3.8 datarefReadFloat()

```
float XPLPro::datarefReadFloat ( )  [inline]
```

Read the received float DataRef.

Returns

Received value

Definition at line 199 of file XPLPro.h.

3.1.3.9 datarefReadInt()

```
long XPLPro::datarefReadInt ( )  [inline]
```

Read the received integer DataRef.

Returns

Received value

Definition at line 203 of file XPLPro.h.

3.1.3.10 datarefWrite() [1/6]

```
void XPLPro::datarefWrite (
             int handle,
             float value )
```

Write a float DataRef.

Parameters

handle	Handle of the DataRef to write
value	Value to write to the DataRef

Definition at line 127 of file XPLPro.cpp.

3.1.3.11 datarefWrite() [2/6]

```
void XPLPro::datarefWrite (
             int handle,
             float value,
             int arrayElement )
```

Write a float DataRef to an array element.

Parameters

handle	Handle of the DataRef to write
value	Value to write to the DataRef
arrayElement	Array element to write to

Definition at line 144 of file XPLPro.cpp.

3.1.3.12 datarefWrite() [3/6]

```
void XPLPro::datarefWrite (
             int handle,
             int value )
```

Write an integer DataRef. Maps to long DataRefs.

Parameters

handle	Handle of the DataRef to write
value	Value to write to the DataRef

Definition at line 87 of file XPLPro.cpp.

3.1.3.13 datarefWrite() [4/6]

```
void XPLPro::datarefWrite (
             int handle,
             int value,
             int arrayElement )
```

Write a Integer DataRef to an array element. Maps to long DataRefs.

Parameters

handle	Handle of the DataRef to write
value	Value to write to the DataRef
arrayElement	Array element to write to

Definition at line 97 of file XPLPro.cpp.

3.1.3.14 datarefWrite() [5/6]

```
void XPLPro::datarefWrite (
             int handle,
             long value )
```

Write an integer DataRef.

Parameters

handle	Handle of the DataRef to write
value	Value to write to the DataRef

Definition at line 107 of file XPLPro.cpp.

3.1.3.15 datarefWrite() [6/6]

```
void XPLPro::datarefWrite (
             int handle,
             long value,
             int arrayElement )
```

Write a Integer DataRef to an array element.

Parameters

handle	Handle of the DataRef to write
value	Value to write to the DataRef
arrayElement	Array element to write to

Definition at line 117 of file XPLPro.cpp.

3.1.3.16 registerCommand()

```
int XPLPro::registerCommand (
             XPString_t * commandName )
```

Register a Command and obtain a handle.

Parameters

commandName	Name of the Command (or abbreviation)

Returns

Assigned handle for the Command, -1 if Command was not found

Definition at line 459 of file XPLPro.cpp.

3.1.3.17 registerDataRef()

```
int XPLPro::registerDataRef (
             XPString_t * datarefName )
```

Register a DataRef and obtain a handle.

Parameters

datarefName	Name of the DataRef (or abbreviation)

Returns

Assigned handle for the DataRef, -1 if DataRef was not found

Definition at line 434 of file XPLPro.cpp.

3.1.3.18 requestUpdates() `[1/2]`

```
void XPLPro::requestUpdates (
             int handle,
             int rate,
             float precision )
```

Request DataRef updates from the plugin.

Parameters

handle	Handle of the DataRef to subscribe to
rate	Maximum rate for updates to reduce traffic
precision	Floating point precision

Definition at line 476 of file XPLPro.cpp.

3.1.3.19 requestUpdates() `[2/2]`

```
void XPLPro::requestUpdates (
             int handle,
             int rate,
             float precision,
             int arrayElement )
```

Request DataRef updates from the plugin for an array DataRef.

Parameters

handle	Handle of the DataRef to subscribe to
rate	Maximum rate for updates to reduce traffic
precision	Floating point precision
arrayElement	Array element to subscribe to

Definition at line 490 of file XPLPro.cpp.

3.1.3.20 sendDebugMessage()

```
int XPLPro::sendDebugMessage (
             const char * msg )
```

Send a debug message to the plugin.

Parameters

msg	Message to show as debug string

Returns

Definition at line 73 of file XPLPro.cpp.

3.1.3.21 sendResetRequest()

```
void XPLPro::sendResetRequest (
             void  )
```

Request a reset from the plugin.

Definition at line 171 of file XPLPro.cpp.

3.1.3.22 sendSpeakMessage()

```
int XPLPro::sendSpeakMessage (
             const char * msg )
```

Send a speech message to the plugin.

Parameters

msg	Message to speak

Returns

Definition at line 79 of file XPLPro.cpp.

3.1.3.23 setScaling()

```
void XPLPro::setScaling (
             int handle,
             int inLow,
             int inHigh,
             int outLow,
             int outHigh )
```

set scaling factor for a DataRef (offload mapping to the plugin)

Definition at line 505 of file XPLPro.cpp.

3.1.3.24 xloop()

```
int XPLPro::xloop (
             void  )
```

Cyclic loop handler, must be called in idle task.

Returns

Connection status

Definition at line 22 of file XPLPro.cpp.

The documentation for this class was generated from the following files:

- XPLPro.h
- XPLPro.cpp

Chapter 4

File Documentation

4.1 Direct inputs/main.cpp

```
#include <Arduino.h>
#include <XPLPro.h>

// The XPLDirect library is automatically installed by PlatformIO with XPLDevices
// Optional defines for XPLDirect can be set in platformio.ini
// This sample contains all the important defines. Modify or remove as needed

// A simple Pushbutton on Arduino pin 2
Button btnStart(2);

// An Encoder with push functionality. 3&4 are the encoder pins, 5 the push pin.
// configured for an Encoder with 4 counts per mechanical notch, which is the standard
Encoder encHeading(3, 4, 5, enc4Pulse);

// A simple On/Off switch on pin 6
Switch swStrobe(6);

// A handle for a DataRef
int drefStrobe;

void xpInit()
{
  // Register Command for the Button
  btnStart.setCommand(F("sim/starters/engage_starter_1"));

  // Register Commands for Encoder Up/Down/Push function.
  encHeading.setCommand(F("sim/autopilot/heading_up"),
                        F("sim/autopilot/heading_down"),
                        F("sim/autopilot/heading_sync"));

  // Register Commands for Switch On and Off transitions. Commands are sent when Switch is moved
  swStrobe.setCommand(F("sim/lights/strobe_lights_on"),
                      F("sim/lights/strobe_lights_off"));

  // Register a DataRef for the strobe light. Subscribe to updates from XP, 100ms minimum Cycle time,
no divider
  drefStrobe = XP.registerDataRef(F("sim/cockpit/electrical/strobe_lights_on"));
  XP.requestUpdates(drefStrobe, 100, 0);
}

void xpStop()
{
  // nothing to do on unload
}

void xpUpdate(int handle)
{
  if (handle == drefStrobe)
  { // Show the status of the Strobe on the internal LED
    digitalWrite(LED_BUILTIN, (XP.datarefReadInt() > 0));
  }
}

// Arduino setup function, called once
void setup() {
  // setup interface
  Serial.begin(XPLDIRECT_BAUDRATE);
  XP.begin("Sample", &xpInit(), &xpStop(), &xpUpdate());
}

// Arduino loop function, called cyclic
void loop() {
  // Handle XPlane interface
  XP.xloop();

  // handle all devices and automatically process commands
  btnStart.handleXP();
  encHeading.handleXP();
  swStrobe.handleXP();
}
```

4.2 MUX inputs/main.cpp

```
#include <Arduino.h>
#include <XPLPro.h>

// The XPLDirect library is automatically installed by PlatformIO with XPLDevices
// Optional defines for XPLDirect can be set in platformio.ini
// This sample contains all the important defines. Modify or remove as needed

// This sample shows how to use 74HC4067 Multiplexers for the inputs as commonly used by SimVim

// A simple Pushbutton on MUX0 pin 0
Button btnStart(0, 0);

// An Encoder with push functionality. MUX1 pin 8&9 are the encoder pins, 10 the push pin.
// configured for an Encoder with 4 counts per mechanical notch, which is the standard
Encoder encHeading(1, 8, 9, 10, enc4Pulse);

// A simple On/Off switch on MUX0, pin 15
Switch swStrobe(0, 15);

// A handle for a DataRef
int drefStrobe;

void xpInit()
{
  // Register Command for the Button
  btnStart.setCommand(F("sim/starters/engage_starter_1"));

  // Register Commands for Encoder Up/Down/Push function.
  encHeading.setCommand(F("sim/autopilot/heading_up"),
                        F("sim/autopilot/heading_down"),
                        F("sim/autopilot/heading_sync"));

  // Register a DataRef for the strobe light. Subscribe to updates from XP, 100ms minimum Cycle time,
no divider
  drefStrobe = XP.registerDataRef(F("sim/cockpit/electrical/strobe_lights_on"));
  XP.requestUpdates(drefStrobe, 100, 0);
}

void xpStop()
{
  // nothing to do on unload
}

void xpUpdate(int handle)
{
  if (handle == drefStrobe)
  { // Show the status of the Strobe on the internal LED
    digitalWrite(LED_BUILTIN, (XP.datarefReadInt() > 0));
  }
}

// Arduino setup function, called once
void setup() {
  // setup interface
  Serial.begin(XPLDIRECT_BAUDRATE);
  XP.begin("Sample", &xpInit(), &xpStop(), &xpUpdate());

  // Connect MUX adress pins to Pin 22-25 (SimVim Pins)
  DigitalIn.setMux(22, 23, 24, 25);
  // Logical MUX0 on Pin 38
  DigitalIn.addMux(38);
  // Logical MUX1 on Pin 39
  DigitalIn.addMux(39);
}

// Arduino loop function, called cyclic
void loop() {
  // Handle XPlane interface
  XP.xloop();

  // handle all devices and automatically process commandsin background
  btnStart.handleXP();
  encHeading.handleXP();
  swStrobe.handleXP();
}
```

4.3 XPLPro.cpp

```
// XPLPro.cpp
// Created by Curiosity Workshop, Michael Gerlicher, 2023.
#include "XPLPro.h"

XPLPro::XPLPro(Stream *device)
{
  _streamPtr = device;
  _streamPtr->setTimeout(XPL_RX_TIMEOUT);
}

void XPLPro::begin(const char *devicename, void (*initFunction)(void), void (*stopFunction)(void),
void (*inboundHandler)(int))
{
  _deviceName = (char *)devicename;
  _connectionStatus = 0;
  _receiveBuffer[0] = 0;
  _registerFlag = 0;
  _xplInitFunction = initFunction;
  _xplStopFunction = stopFunction;
  _xplInboundHandler = inboundHandler;
}

int XPLPro::xloop(void)
{
  // handle incoming serial data
  _processSerial();
  // when device is registered, perform handle registrations
  if (_registerFlag)
  {
    _xplInitFunction();
    _registerFlag = 0;
  }
  // return status of connection
  return _connectionStatus;
}

// TODO: is a return value necessary? These could also be void like for the datarefs
int XPLPro::commandTrigger(int commandHandle, int triggerCount)
{
  if (commandHandle < 0)
  {
    return XPL_HANDLE_INVALID;
  }
  sprintf(_sendBuffer, "%c%c,%i,%i%c", XPL_PACKETHEADER, XPLCMD_COMMANDTRIGGER, commandHandle,
triggerCount, XPL_PACKETTRAILER);
  _transmitPacket();
  return 0;
}

int XPLPro::commandStart(int commandHandle)
{
  if (commandHandle < 0)
  {
    return XPL_HANDLE_INVALID;
  }
  _sendPacketVoid(XPLCMD_COMMANDSTART, commandHandle);
  return 0;
}

int XPLPro::commandEnd(int commandHandle)
{
  if (commandHandle < 0)
  {
    return XPL_HANDLE_INVALID;
  }
  _sendPacketVoid(XPLCMD_COMMANDEND, commandHandle);
  return 0;
}

int XPLPro::connectionStatus()
{
  return _connectionStatus;
}
```

```

int XPLPro::sendDebugMessage(const char *msg)
{
  _sendPacketString(XPLCMD_PRINTDEBUG, msg);
  return 1;
}

int XPLPro::sendSpeakMessage(const char *msg)
{
  _sendPacketString(XPLCMD_SPEAK, msg);
  return 1;
}

// these could be done better:

void XPLPro::datarefWrite(int handle, int value)
{
  if (handle < 0)
  {
    return;
  }
  sprintf(_sendBuffer, "%c%c,%i,%i%c", XPL_PACKETHEADER, XPLCMD_DATAREFUPDATEINT, handle, value, XPL_PACKETTRAILER);
  _transmitPacket();
}

void XPLPro::datarefWrite(int handle, int value, int arrayElement)
{
  if (handle < 0)
  {
    return;
  }
  sprintf(_sendBuffer, "%c%c,%i,%i,%i%c", XPL_PACKETHEADER, XPLCMD_DATAREFUPDATEINTARRAY, handle, value, arrayElement, XPL_PACKETTRAILER);
  _transmitPacket();
}

void XPLPro::datarefWrite(int handle, long value)
{
  if (handle < 0)
  {
    return;
  }
  sprintf(_sendBuffer, "%c%c,%i,%ld%c", XPL_PACKETHEADER, XPLCMD_DATAREFUPDATEINT, handle, value, XPL_PACKETTRAILER);
  _transmitPacket();
}

void XPLPro::datarefWrite(int handle, long value, int arrayElement)
{
  if (handle < 0)
  {
    return;
  }
  sprintf(_sendBuffer, "%c%c,%i,%ld,%i%c", XPL_PACKETHEADER, XPLCMD_DATAREFUPDATEINTARRAY, handle, value, arrayElement, XPL_PACKETTRAILER);
  _transmitPacket();
}

void XPLPro::datarefWrite(int handle, float value)
{
  if (handle < 0)
  {
    return;
  }
  char tBuf[20]; // todo:  rewrite to eliminate this buffer.  Write directly to _sendBuffer
  dtostrf(value, 0, XPL_FLOATPRECISION, tBuf);
  sprintf(_sendBuffer, "%c%c,%i,%s%c",
          XPL_PACKETHEADER,
          XPLCMD_DATAREFUPDATEFLOAT,
          handle,
          tBuf,
          XPL_PACKETTRAILER);
  _transmitPacket();
}

void XPLPro::datarefWrite(int handle, float value, int arrayElement)
{
  if (handle < 0)
  {
    return;
  }
  char tBuf[20]; // todo:  rewrite to eliminate this buffer.  Write directly to _sendBuffer
  dtostrf(value, 0, XPL_FLOATPRECISION, tBuf);
  sprintf(_sendBuffer, "%c%c,%i,%s,%i%c",
          XPL_PACKETHEADER,
          XPLCMD_DATAREFUPDATEFLOATARRAY,
          handle,
          tBuf,
          arrayElement,
          XPL_PACKETTRAILER);
  _transmitPacket();
}

void XPLPro::_sendname()
{
  // register device on request only when we have a valid name
  if (_deviceName != NULL)
  {
    _sendPacketString(XPLRESPONSE_NAME, _deviceName);
  }
}

void XPLPro::sendResetRequest()
{
  // request a reset only when we have a valid name
  if (_deviceName != NULL)
  {
    _sendPacketVoid(XPLCMD_RESET, 0);
  }
}

void XPLPro::_processSerial()
{
  // read until package header found or buffer empty
  while (_streamPtr->available() && _receiveBuffer[0] != XPL_PACKETHEADER)
  {
    _receiveBuffer[0] = (char)_streamPtr->read();
  }
  // return when buffer empty and header not found
  if (_receiveBuffer[0] != XPL_PACKETHEADER)
  {
    return;
  }
  // read rest of package until trailer
  _receiveBufferBytesReceived = _streamPtr->readBytesUntil(XPL_PACKETTRAILER, (char
*)&_receiveBuffer[1], XPLMAX_PACKETSIZE_RECEIVE - 1);
  // if no further chars available, delete package
  if (_receiveBufferBytesReceived == 0)
  {
    _receiveBuffer[0] = 0;
    return;
  }
  // add package trailer and zero byte to frame
  _receiveBuffer[++_receiveBufferBytesReceived] = XPL_PACKETTRAILER;
  _receiveBuffer[++_receiveBufferBytesReceived] = 0; // old habits die hard.
  // at this point we should have a valid frame
  _processPacket();
}

void XPLPro::_processPacket()
{
  int tHandle;
  // check whether we have a valid frame
  if (_receiveBuffer[0] != XPL_PACKETHEADER)
  {
    return;
  }
  // branch on receiverd command
  switch (_receiveBuffer[1])
  {
  // plane unloaded or XP exiting
  case XPL_EXITING:
    _connectionStatus = false;
    _xplStopFunction();
    break;

  // register device
  case XPLCMD_SENDNAME:
    _sendname();
    _connectionStatus = true; // not considered active till you know my name
    _registerFlag = 0;
    break;

  // plugin is ready for registrations.
  case XPLCMD_SENDREQUEST:
    _registerFlag = 1; // use a flag to signal registration so recursion doesn't occur
    break;

  // get handle from response to registered dataref
  case XPLRESPONSE_DATAREF:
    _parseInt(&_handleAssignment, _receiveBuffer, 2);
    break;

  // get handle from response to registered command
  case XPLRESPONSE_COMMAND:
    _parseInt(&_handleAssignment, _receiveBuffer, 2);
    break;

  // int dataref received
  case XPLCMD_DATAREFUPDATEINT:
    _parseInt(&tHandle, _receiveBuffer, 2);
    _parseInt(&_readValueLong, _receiveBuffer, 3);
    _readValueFloat = 0;
    _readValueElement = 0;
    _xplInboundHandler(tHandle);
    break;

  // int array dataref received
  case XPLCMD_DATAREFUPDATEINTARRAY:
    _parseInt(&tHandle, _receiveBuffer, 2);
    _parseInt(&_readValueLong, _receiveBuffer, 3);
    _parseInt(&_readValueElement, _receiveBuffer, 4);
    _readValueFloat = 0;
    _xplInboundHandler(tHandle);
    break;

  // float dataref received
  case XPLCMD_DATAREFUPDATEFLOAT:
    _parseInt(&tHandle, _receiveBuffer, 2);
    _parseFloat(&_readValueFloat, _receiveBuffer, 3);
    _readValueLong = 0;
    _readValueElement = 0;
    _xplInboundHandler(tHandle);
    break;

  // float array dataref received
  case XPLCMD_DATAREFUPDATEFLOATARRAY:
    _parseInt(&tHandle, _receiveBuffer, 2);
    _parseFloat(&_readValueFloat, _receiveBuffer, 3);
    _parseInt(&_readValueElement, _receiveBuffer, 4);
    _readValueLong = 0;
    _xplInboundHandler(tHandle);
    break;

  // obsolete?
  case XPLREQUEST_REFRESH:
    break;

  default:
    break;
  }
  // empty receive buffer
  _receiveBuffer[0] = 0;
}

void XPLPro::_sendPacketVoid(int command, int handle) // just a command with a handle
{
  // check for valid handle
  if (handle < 0)
  {
    return;
  }
  sprintf(_sendBuffer, "%c%c,%i%c", XPL_PACKETHEADER, command, handle, XPL_PACKETTRAILER);
  _transmitPacket();
}

void XPLPro::_sendPacketString(int command, const char *str) // for a string
{
  sprintf(_sendBuffer, "%c%c,\"%s\"%c", XPL_PACKETHEADER, command, str, XPL_PACKETTRAILER);
  _transmitPacket();
}

void XPLPro::_transmitPacket(void)
{
  _streamPtr->write(_sendBuffer);
  if (strlen(_sendBuffer) == 64)
  {
    // apparently a bug in arduino with some boards when we transmit exactly 64 bytes. That took a
while to track down...
    _streamPtr->print(" ");
  }
}

int XPLPro::_parseString(char *outBuffer, char *inBuffer, int parameter, int maxSize)
{
  int cBeg;
  int pos = 0;
  int len;

  for (int i = 1; i < parameter; i++)
  {
    while (inBuffer[pos] != ',' && inBuffer[pos] != 0)
    {
      pos++;
    }
    pos++;
  }

  while (inBuffer[pos] != '\"' && inBuffer[pos] != 0)
  {
    pos++;
  }
  cBeg = ++pos;

  while (inBuffer[pos] != '\"' && inBuffer[pos] != 0)
  {
    pos++;
  }
  len = pos - cBeg;
  if (len > maxSize)
  {
    len = maxSize;
  }
  strncpy(outBuffer, (char *)&inBuffer[cBeg], len);
  outBuffer[len] = 0;
  // fprintf(errlog, "_parseString, pos: %i, cBeg: %i, deviceName: %s\n", pos, cBeg, target);
  return 0;
}

int XPLPro::_parseInt(int *outTarget, char *inBuffer, int parameter)
{
  int cBeg;
  int pos = 0;
  // search for the selected parameter
  for (int i = 1; i < parameter; i++)
  {
    while (inBuffer[pos] != ',' && inBuffer[pos] != 0)
    {
      pos++;
    }
    pos++;
  }
  // parameter starts here
  cBeg = pos;
  // search for end of parameter
  while (inBuffer[pos] != ',' && inBuffer[pos] != 0 && inBuffer[pos] != XPL_PACKETTRAILER)
  {
    pos++;
  }
  // temporarily make parameter null terminated
  char holdChar = inBuffer[pos];
  inBuffer[pos] = 0;
  // get integer value from string
  *outTarget = atoi((char *)&inBuffer[cBeg]);
  // restore buffer
  inBuffer[pos] = holdChar;
  return 0;
}

int XPLPro::_parseInt(long *outTarget, char *inBuffer, int parameter)
{
  int cBeg;
  int pos = 0;
  for (int i = 1; i < parameter; i++)
  {
    while (inBuffer[pos] != ',' && inBuffer[pos] != 0)
    {
      pos++;
    }
    pos++;
  }
  cBeg = pos;
  while (inBuffer[pos] != ',' && inBuffer[pos] != 0 && inBuffer[pos] != XPL_PACKETTRAILER)
  {
    pos++;
  }
  char holdChar = inBuffer[pos];
  inBuffer[pos] = 0;
  *outTarget = atoi((char *)&inBuffer[cBeg]);
  inBuffer[pos] = holdChar;
  return 0;
}

int XPLPro::_parseFloat(float *outTarget, char *inBuffer, int parameter)
{
  int cBeg;
  int pos = 0;
  for (int i = 1; i < parameter; i++)
  {
    while (inBuffer[pos] != ',' && inBuffer[pos] != 0)
    {
      pos++;
    }
    pos++;
  }
  cBeg = pos;
  while (inBuffer[pos] != ',' && inBuffer[pos] != 0 && inBuffer[pos] != XPL_PACKETTRAILER)
  {
    pos++;
  }
  char holdChar = inBuffer[pos];
  inBuffer[pos] = 0;
  *outTarget = atof((char *)&inBuffer[cBeg]);
  inBuffer[pos] = holdChar;
  return 0;
}

int XPLPro::registerDataRef(XPString_t *datarefName)
{
  long int startTime;

  // registration only allowed in callback (TODO: is this limitation really necessary?)
  if (!_registerFlag)
  {
    return XPL_HANDLE_INVALID;
  }
#if XPL_USE_PROGMEM
  sprintf(_sendBuffer, "%c%c,\"%S\"%c", XPL_PACKETHEADER, XPLREQUEST_REGISTERDATAREF, (wchar_t *)datarefName, XPL_PACKETTRAILER);
#else
  sprintf(_sendBuffer, "%c%c,\"%s\"%c", XPL_PACKETHEADER, XPLREQUEST_REGISTERDATAREF, (char *)datarefName, XPL_PACKETTRAILER);
#endif
  _transmitPacket();

  _handleAssignment = XPL_HANDLE_INVALID;
  startTime = millis(); // for timeout function

  while (millis() - startTime < XPL_RESPONSE_TIMEOUT && _handleAssignment < 0)
    _processSerial();

  return _handleAssignment;
}

int XPLPro::registerCommand(XPString_t *commandName)
{
  long int startTime = millis(); // for timeout function
#if XPL_USE_PROGMEM
  sprintf(_sendBuffer, "%c%c,\"%S\"%c", XPL_PACKETHEADER, XPLREQUEST_REGISTERCOMMAND, (wchar_t *)commandName, XPL_PACKETTRAILER);
#else
  sprintf(_sendBuffer, "%c%c,\"%s\"%c", XPL_PACKETHEADER, XPLREQUEST_REGISTERCOMMAND, (char *)commandName, XPL_PACKETTRAILER);
#endif
  _transmitPacket();
  _handleAssignment = XPL_HANDLE_INVALID;
  while (millis() - startTime < XPL_RESPONSE_TIMEOUT && _handleAssignment < 0)
  {
    _processSerial();
  }
  return _handleAssignment;
}

void XPLPro::requestUpdates(int handle, int rate, float precision)
{
  char tBuf[20]; // todo:  rewrite to eliminate this buffer.  Write directly to _sendBuffer?
  dtostrf(precision, 0, XPL_FLOATPRECISION, tBuf);
  sprintf(_sendBuffer, "%c%c,%i,%i,%s%c",
          XPL_PACKETHEADER,
          XPLREQUEST_UPDATES,
          handle,
          rate,
          tBuf,
          XPL_PACKETTRAILER);
  _transmitPacket();
}

void XPLPro::requestUpdates(int handle, int rate, float precision, int element)
{
  char tBuf[20]; // todo:  rewrite to eliminate this buffer.  Write directly to _sendBuffer?
  dtostrf(precision, 0, XPL_FLOATPRECISION, tBuf);
  sprintf(_sendBuffer, "%c%c,%i,%i,%s,%i%c",
          XPL_PACKETHEADER,
          XPLREQUEST_UPDATESARRAY,
          handle,
          rate,
          tBuf,
          element,
          XPL_PACKETTRAILER);
  _transmitPacket();
}

void XPLPro::setScaling(int handle, int inLow, int inHigh, int outLow, int outHigh)
{
  sprintf(_sendBuffer, "%c%c,%i,%i,%i,%i,%i%c",
          XPL_PACKETHEADER,
          XPLREQUEST_SCALING,
          handle,
          inLow,
          inHigh,
          outLow,
          outHigh,
          XPL_PACKETTRAILER);
  _transmitPacket();
}
```

4.4 XPLPro.h

```
//   XPLPro.h - Library for serial interface to Xplane SDK.
//   Created by Curiosity Workshop, Michael Gerlicher,  2020-2023
//   See readme.txt file for information on updates.
//   To report problems, download updates and examples, suggest enhancements or get technical support,
please visit:
//      discord:  https://discord.gg/gzXetjEST4
//      patreon:  www.patreon.com/curiosityworkshop

#ifndef XPLPro_h
#define XPLPro_h

#include <Arduino.h>

// Parameters that can be overwritten by command line defines

// Decimals of precision for floating point datarefs. More increases dataflow (default 4)
#ifndef XPL_FLOATPRECISION
#define XPL_FLOATPRECISION 4
#endif

// Timeout after sending a registration request, how long will we wait for the response.
// This is giant because sometimes xplane says the plane is loaded then does other stuff for a while.
(default 90000 ms)
#ifndef XPL_RESPONSE_TIMEOUT
#define XPL_RESPONSE_TIMEOUT 90000
#endif

// For boards with limited memory that can use PROGMEM to store strings.
// You will need to wrap your dataref names with F() macro ie:
// Xinterface.registerDataref(F("laminar/B738/annunciator/drive2"), XPL_READ, 100, 0, &drive2);
// Disable for boards that have issues compiling: errors with strncmp_PF for instance.
#ifndef XPL_USE_PROGMEM
#ifdef __AVR_ARCH__
// flash strings are default on on AVR architecture
#define XPL_USE_PROGMEM 1
#else
// and off otherwise
#define XPL_USE_PROGMEM 0
#endif
#endif

// Package buffer size for send and receive buffer each.
// If you need a few extra bytes of RAM it could be reduced, but it needs to
// be as long as the longest dataref name + 10.  If you are using datarefs
// that transfer strings it needs to be big enough for those too.  (default 200)
#ifndef XPLMAX_PACKETSIZE_TRANSMIT
#define XPLMAX_PACKETSIZE_TRANSMIT 200
#endif

#ifndef XPLMAX_PACKETSIZE_RECEIVE
#define XPLMAX_PACKETSIZE_RECEIVE 200
#endif

// All other defines in this header must not be modified

// define whether flash strings will be used
#if XPL_USE_PROGMEM
// use Flash for strings, requires F() macro for strings in all registration calls
typedef const __FlashStringHelper XPString_t;
```

```
#else
typedef const char XPString_t;
#endif

// Parameters around the interface
#define XPL_BAUDRATE 115200   // Baudrate needed to match plugin
#define XPL_RX_TIMEOUT 500    // Timeout for reception of one frame
#define XPL_PACKETHEADER '['  // Frame start character
#define XPL_PACKETTRAILER ']' // Frame end character
#define XPL_HANDLE_INVALID -1 // invalid handle

// Items in caps generally come from XPlane. Items in lower case are generally sent from the arduino.
#define XPLCMD_SENDNAME 'N'                 // plugin request name from arduino
#define XPLRESPONSE_NAME 'n'                // Arduino responds with device name as initialized in the "begin" function
#define XPLCMD_SENDREQUEST 'Q'              // plugin sends this when it is ready to register bindings
#define XPLREQUEST_REGISTERDATAREF 'b'      // Register a dataref
#define XPLREQUEST_REGISTERCOMMAND 'm'      // Register a command
#define XPLRESPONSE_DATAREF 'D'             // Plugin responds with handle to dataref or - value if not found.  dataref handle, dataref name
#define XPLRESPONSE_COMMAND 'C'             // Plugin responds with handle to command or - value if not found.  command handle, command name
#define XPLCMD_PRINTDEBUG 'g'               // Plugin logs string sent from arduino
#define XPLCMD_SPEAK 's'                    // plugin speaks string through xplane speech
#define XPLREQUEST_REFRESH 'd'              // the plugin will call this once xplane is loaded in order to get fresh updates from arduino handles that write
#define XPLREQUEST_UPDATES 'r'              // arduino is asking the plugin to update the specified dataref with rate and divider parameters
#define XPLREQUEST_UPDATESARRAY 't'         // arduino is asking the plugin to update the specified array dataref with rate and divider parameters
#define XPLREQUEST_SCALING 'u'              // arduino requests the plugin apply scaling to the dataref values
#define XPLCMD_RESET 'z'                    // Request a reset and reregistration from the plugin
#define XPLCMD_DATAREFUPDATEINT '1'         // Int DataRef update
#define XPLCMD_DATAREFUPDATEFLOAT '2'       // Float DataRef update
#define XPLCMD_DATAREFUPDATEINTARRAY '3'    // Int array DataRef update
#define XPLCMD_DATAREFUPDATEFLOATARRAY '4'  // Float array DataRef Update
#define XPLCMD_DATAREFUPDATESTRING '9'      // String DataRef update
#define XPLCMD_COMMANDTRIGGER 'k'           // Trigger command n times
#define XPLCMD_COMMANDSTART 'i'             // Begin command (Button pressed)
#define XPLCMD_COMMANDEND 'j'               // End command (Button released)
#define XPL_EXITING 'X'                     // XPlane sends this to the arduino device during normal shutdown of XPlane. It may not happen if xplane crashes.

class XPLPro
{
public:
    XPLPro(Stream *device);

    void begin(const char *devicename, void (*initFunction)(void), void (*stopFunction)(void), void (*inboundHandler)(int));

    int connectionStatus();

    int commandTrigger(int commandHandle) { return commandTrigger(commandHandle, 1); };

    int commandTrigger(int commandHandle, int triggerCount);

    int commandStart(int commandHandle);

    int commandEnd(int commandHandle);

    void datarefWrite(int handle, long value);

    void datarefWrite(int handle, int value);

    void datarefWrite(int handle, long value, int arrayElement);

    void datarefWrite(int handle, int value, int arrayElement);

    void datarefWrite(int handle, float value);

    void datarefWrite(int handle, float value, int arrayElement);

    void requestUpdates(int handle, int rate, float precision);

    void requestUpdates(int handle, int rate, float precision, int arrayElement);

    void setScaling(int handle, int inLow, int inHigh, int outLow, int outHigh);

    int registerDataRef(XPString_t *datarefName);

    int registerCommand(XPString_t *commandName);

    float datarefReadFloat() { return _readValueFloat; }

    long datarefReadInt() { return _readValueLong; }

    int datarefReadElement() { return _readValueElement; }

    int sendDebugMessage(const char *msg);

    int sendSpeakMessage(const char *msg);

    void sendResetRequest(void);

    int xloop();

private:
    void _processSerial();
    void _processPacket();
    void _transmitPacket();
    void _sendname();
    void _sendPacketVoid(int command, int handle);        // just a command with a handle
    void _sendPacketString(int command, const char *str); // send a string
    int _parseInt(int *outTarget, char *inBuffer, int parameter);
    int _parseInt(long *outTarget, char *inBuffer, int parameter);
    int _parseFloat(float *outTarget, char *inBuffer, int parameter);
    int _parseString(char *outBuffer, char *inBuffer, int parameter, int maxSize);

    Stream *_streamPtr;
    const char *_deviceName;
    byte _registerFlag;
    byte _connectionStatus;

    char _sendBuffer[XPLMAX_PACKETSIZE_TRANSMIT];
    char _receiveBuffer[XPLMAX_PACKETSIZE_RECEIVE];
    int _receiveBufferBytesReceived;

    void (*_xplInitFunction)(void);  // this function will be called when the plugin is ready to receive binding requests
    void (*_xplStopFunction)(void);  // this function will be called with the plugin receives message or detects xplane flight model inactive
    void (*_xplInboundHandler)(int); // this function will be called when the plugin sends dataref values

    int _handleAssignment;
    long _readValueLong;
    float _readValueFloat;
    int _readValueElement;
};
```

Index