

XPLPro

Generated by Doxygen 1.9.6

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 XPLPro Class Reference	5
3.1.1 Detailed Description	6
3.1.2 Constructor & Destructor Documentation	6
3.1.2.1 XPLPro()	6
3.1.3 Member Function Documentation	7
3.1.3.1 begin()	7
3.1.3.2 commandEnd()	7
3.1.3.3 commandStart()	8
3.1.3.4 commandTrigger() [1/2]	8
3.1.3.5 commandTrigger() [2/2]	8
3.1.3.6 connectionStatus()	9
3.1.3.7 datarefReadElement()	9
3.1.3.8 datarefReadFloat()	9
3.1.3.9 datarefReadInt()	10
3.1.3.10 datarefWrite() [1/6]	10
3.1.3.11 datarefWrite() [2/6]	10
3.1.3.12 datarefWrite() [3/6]	11
3.1.3.13 datarefWrite() [4/6]	11
3.1.3.14 datarefWrite() [5/6]	11
3.1.3.15 datarefWrite() [6/6]	12
3.1.3.16 registerCommand()	12
3.1.3.17 registerDataRef()	12
3.1.3.18 requestUpdates() [1/2]	13
3.1.3.19 requestUpdates() [2/2]	13
3.1.3.20 sendDebugMessage()	14
3.1.3.21 sendResetRequest()	14
3.1.3.22 sendSpeakMessage()	14
3.1.3.23 setScaling()	15
3.1.3.24 xloop()	15
4 File Documentation	17
4.1 Direct inputs/main.cpp	17
4.2 MUX inputs/main.cpp	18
4.3 XPLPro.cpp	19
4.4 XPLPro.h	25
Index	29

Chapter 1

Class Index

1.1 Class List

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

XPLPro	Core class for the XPLPro Arduino library	5
------------------------	---	-------------------

Chapter 2

File Index

2.1 File List

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

Direct inputs/main.cpp	17
MUX inputs/main.cpp	18
XPLPro.cpp	??
XPLPro.h	??

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(*inboundHandler)(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

<i>device</i>	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

<i>devicename</i>	Device name
<i>initFunction</i>	Callback for DataRef and Command registration
<i>stopFunction</i>	Callback for XPlane shutdown or plane change
<i>inboundHandler</i>	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

<i>commandHandle</i>	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

<i>commandHandle</i>	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

<i>commandHandle</i>	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

<i>commandHandle</i>	Handle of the command to trigger
<i>triggerCount</i>	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

<i>handle</i>	Handle of the DataRef to write
<i>value</i>	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

<i>handle</i>	Handle of the DataRef to write
<i>value</i>	Value to write to the DataRef
<i>arrayElement</i>	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

<i>handle</i>	Handle of the DataRef to write
<i>value</i>	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

<i>handle</i>	Handle of the DataRef to write
<i>value</i>	Value to write to the DataRef
<i>arrayElement</i>	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

<i>handle</i>	Handle of the DataRef to write
<i>value</i>	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

<i>handle</i>	Handle of the DataRef to write
<i>value</i>	Value to write to the DataRef
<i>arrayElement</i>	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

<i>commandName</i>	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

<i>datarefName</i>	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

<i>handle</i>	Handle of the DataRef to subscribe to
<i>rate</i>	Maximum rate for updates to reduce traffic
<i>precision</i>	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

<i>handle</i>	Handle of the DataRef to subscribe to
<i>rate</i>	Maximum rate for updates to reduce traffic
<i>precision</i>	Floating point precision
<i>arrayElement</i>	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

<i>msg</i>	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

<i>msg</i>	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

```
00001 #include <Arduino.h>
00002 #include <XPLPro.h>
00003
00004 // The XPLDirect library is automatically installed by PlatformIO with XPLDevices
00005 // Optional defines for XPLDirect can be set in platformio.ini
00006 // This sample contains all the important defines. Modify or remove as needed
00007
00008 // A simple Pushbutton on Arduino pin 2
00009 Button btnStart(2);
00010
00011 // An Encoder with push functionality. 3&4 are the encoder pins, 5 the push pin.
00012 // configured for an Encoder with 4 counts per mechanical notch, which is the standard
00013 Encoder encHeading(3, 4, 5, enc4Pulse);
00014
00015 // A simple On/Off switch on pin 6
00016 Switch swStrobe(6);
00017
00018 // A handle for a DataRef
00019 int drefStrobe;
00020
00021 void xpInit()
00022 {
00023     // Register Command for the Button
00024     btnStart.setCommand(F("sim/starters/engage_starter_1"));
00025
00026     // Register Commands for Encoder Up/Down/Push function.
00027     encHeading.setCommand(F("sim/autopilot/heading_up"),
00028                           F("sim/autopilot/heading_down"),
00029                           F("sim/autopilot/heading_sync"));
00030
00031     // Register Commands for Switch On and Off transitions. Commands are sent when Switch is moved
00032     swStrobe.setCommand(F("sim/lights/strobe_lights_on"),
00033                        F("sim/lights/strobe_lights_off"));
00034
00035     // Register a DataRef for the strobe light. Subscribe to updates from XP, 100ms minimum Cycle time,
no divider
00036     drefStrobe = XP.registerDataRef(F("sim/cockpit/electrical/strobe_lights_on"));
00037     XP.requestUpdates(drefStrobe, 100, 0);
00038 }
00039
00040 void xpStop()
00041 {
00042     // nothing to do on unload
00043 }
00044
00045 void xpUpdate(int handle)
00046 {
00047     if (handle == drefStrobe)
00048     { // Show the status of the Strobe on the internal LED
00049         digitalWrite(LED_BUILTIN, (XP.datarefReadInt() > 0));
00050     }
00051 }
00052
00053 // Arduino setup function, called once
00054 void setup() {
00055     // setup interface
00056     Serial.begin(XPLDIRECT_BAUDRATE);
00057     XP.begin("Sample", &xpInit(), &xpStop(), &xpUpdate());
```

```

00058 }
00059
00060 // Arduino loop function, called cyclic
00061 void loop() {
00062     // Handle XPlane interface
00063     XP.xloop();
00064
00065     // handle all devices and automatically process commands
00066     btnStart.handleXP();
00067     encHeading.handleXP();
00068     swStrobe.handleXP();
00069 }

```

4.2 MUX inputs/main.cpp

```

00001 #include <Arduino.h>
00002 #include <XPLPro.h>
00003
00004 // The XPLDirect library is automatically installed by PlatformIO with XPLDevices
00005 // Optional defines for XPLDirect can be set in platformio.ini
00006 // This sample contains all the important defines. Modify or remove as needed
00007
00008 // This sample shows how to use 74HC4067 Multiplexers for the inputs as commonly used by SimVim
00009
00010 // A simple Pushbutton on MUX0 pin 0
00011 Button btnStart(0, 0);
00012
00013 // An Encoder with push functionality. MUX1 pin 8&9 are the encoder pins, 10 the push pin.
00014 // configured for an Encoder with 4 counts per mechanical notch, which is the standard
00015 Encoder encHeading(1, 8, 9, 10, enc4Pulse);
00016
00017 // A simple On/Off switch on MUX0, pin 15
00018 Switch swStrobe(0, 15);
00019
00020 // A handle for a DataRef
00021 int drefStrobe;
00022
00023 void xpInit()
00024 {
00025     // Register Command for the Button
00026     btnStart.setCommand(F("sim/starters/engage_starter_1"));
00027
00028     // Register Commands for Encoder Up/Down/Push function.
00029     encHeading.setCommand(F("sim/autopilot/heading_up"),
00030                           F("sim/autopilot/heading_down"),
00031                           F("sim/autopilot/heading_sync"));
00032
00033     // Register a DataRef for the strobe light. Subscribe to updates from XP, 100ms minimum Cycle time,
00034     // no divider
00035     drefStrobe = XP.registerDataRef(F("sim/cockpit/electrical/strobe_lights_on"));
00036     XP.requestUpdates(drefStrobe, 100, 0);
00037 }
00038 void xpStop()
00039 {
00040     // nothing to do on unload
00041 }
00042
00043 void xpUpdate(int handle)
00044 {
00045     if (handle == drefStrobe)
00046     { // Show the status of the Strobe on the internal LED
00047         digitalWrite(LED_BUILTIN, (XP.datarefReadInt() > 0));
00048     }
00049 }
00050
00051 // Arduino setup function, called once
00052 void setup() {
00053     // setup interface
00054     Serial.begin(XPLDIRECT_BAUDRATE);
00055     XP.begin("Sample", &xpInit(), &xpStop(), &xpUpdate());
00056
00057     // Connect MUX address pins to Pin 22-25 (SimVim Pins)
00058     DigitalIn.setMux(22, 23, 24, 25);
00059     // Logical MUX0 on Pin 38
00060     DigitalIn.addMux(38);
00061     // Logical MUX1 on Pin 39
00062     DigitalIn.addMux(39);
00063 }
00064
00065 // Arduino loop function, called cyclic
00066 void loop() {
00067     // Handle XPlane interface

```

```

00068   XP.xloop();
00069
00070   // handle all devices and automatically process commands in background
00071   btnStart.handleXP();
00072   encHeading.handleXP();
00073   swStrobe.handleXP();
00074 }

```

4.3 XPLPro.cpp

```

00001 // XPLPro.cpp
00002 // Created by Curiosity Workshop, Michael Gerlicher, 2023.
00003 #include "XPLPro.h"
00004
00005 XPLPro::XPLPro(Stream *device)
00006 {
00007     _streamPtr = device;
00008     _streamPtr->setTimeout(XPL_RX_TIMEOUT);
00009 }
00010
00011 void XPLPro::begin(const char *devicename, void (*initFunction)(void), void (*stopFunction)(void),
00012                   void (*inboundHandler)(int))
00013 {
00014     _deviceName = (char *)devicename;
00015     _connectionStatus = 0;
00016     _receiveBuffer[0] = 0;
00017     _registerFlag = 0;
00018     _xplInitFunction = initFunction;
00019     _xplStopFunction = stopFunction;
00020     _xplInboundHandler = inboundHandler;
00021 }
00022 int XPLPro::xloop(void)
00023 {
00024     // handle incoming serial data
00025     _processSerial();
00026     // when device is registered, perform handle registrations
00027     if (_registerFlag)
00028     {
00029         _xplInitFunction();
00030         _registerFlag = 0;
00031     }
00032     // return status of connection
00033     return _connectionStatus;
00034 }
00035
00036 // TODO: is a return value necessary? These could also be void like for the data refs
00037 int XPLPro::commandTrigger(int commandHandle, int triggerCount)
00038 {
00039     if (commandHandle < 0)
00040     {
00041         return XPL_HANDLE_INVALID;
00042     }
00043     sprintf(_sendBuffer, "%c%c,%i,%i%c", XPL_PACKETHEADER, XPLCMD_COMMANDTRIGGER, commandHandle,
00044           triggerCount, XPL_PACKETTRAILER);
00045     _transmitPacket();
00046     return 0;
00047 }
00048 int XPLPro::commandStart(int commandHandle)
00049 {
00050     if (commandHandle < 0)
00051     {
00052         return XPL_HANDLE_INVALID;
00053     }
00054     _sendPacketVoid(XPLCMD_COMMANDSTART, commandHandle);
00055     return 0;
00056 }
00057
00058 int XPLPro::commandEnd(int commandHandle)
00059 {
00060     if (commandHandle < 0)
00061     {
00062         return XPL_HANDLE_INVALID;
00063     }
00064     _sendPacketVoid(XPLCMD_COMMANDEND, commandHandle);
00065     return 0;
00066 }
00067
00068 int XPLPro::connectionStatus()
00069 {
00070     return _connectionStatus;
00071 }

```

```

00072
00073 int XPLPro::sendDebugMessage(const char *msg)
00074 {
00075     _sendPacketString(XPLCMD_PRINTDEBUG, msg);
00076     return 1;
00077 }
00078
00079 int XPLPro::sendSpeakMessage(const char *msg)
00080 {
00081     _sendPacketString(XPLCMD_SPEAK, msg);
00082     return 1;
00083 }
00084
00085 // these could be done better:
00086
00087 void XPLPro::datarefWrite(int handle, int value)
00088 {
00089     if (handle < 0)
00090     {
00091         return;
00092     }
00093     sprintf(_sendBuffer, "%c%c,%i,%i%c", XPL_PACKETHEADER, XPLCMD_DATAREFUPDATEINT, handle, value,
XPL_PACKETTRAILER);
00094     _transmitPacket();
00095 }
00096
00097 void XPLPro::datarefWrite(int handle, int value, int arrayElement)
00098 {
00099     if (handle < 0)
00100     {
00101         return;
00102     }
00103     sprintf(_sendBuffer, "%c%c,%i,%i,%i%c", XPL_PACKETHEADER, XPLCMD_DATAREFUPDATEINTARRAY, handle,
value, arrayElement, XPL_PACKETTRAILER);
00104     _transmitPacket();
00105 }
00106
00107 void XPLPro::datarefWrite(int handle, long value)
00108 {
00109     if (handle < 0)
00110     {
00111         return;
00112     }
00113     sprintf(_sendBuffer, "%c%c,%i,%ld%c", XPL_PACKETHEADER, XPLCMD_DATAREFUPDATEINT, handle, value,
XPL_PACKETTRAILER);
00114     _transmitPacket();
00115 }
00116
00117 void XPLPro::datarefWrite(int handle, long value, int arrayElement)
00118 {
00119     if (handle < 0)
00120     {
00121         return;
00122     }
00123     sprintf(_sendBuffer, "%c%c,%i,%ld,%i%c", XPL_PACKETHEADER, XPLCMD_DATAREFUPDATEINTARRAY, handle,
value, arrayElement, XPL_PACKETTRAILER);
00124     _transmitPacket();
00125 }
00126
00127 void XPLPro::datarefWrite(int handle, float value)
00128 {
00129     if (handle < 0)
00130     {
00131         return;
00132     }
00133     char tBuf[20]; // todo: rewrite to eliminate this buffer. Write directly to _sendBuffer
00134     dtostrf(value, 0, XPL_FLOATPRECISION, tBuf);
00135     sprintf(_sendBuffer, "%c%c,%i,%s%c",
XPL_PACKETHEADER,
XPLCMD_DATAREFUPDATEFLOAT,
handle,
tBuf,
XPL_PACKETTRAILER);
00140     _transmitPacket();
00141 }
00142
00143
00144 void XPLPro::datarefWrite(int handle, float value, int arrayElement)
00145 {
00146     if (handle < 0)
00147     {
00148         return;
00149     }
00150     char tBuf[20]; // todo: rewrite to eliminate this buffer. Write directly to _sendBuffer
00151     dtostrf(value, 0, XPL_FLOATPRECISION, tBuf);
00152     sprintf(_sendBuffer, "%c%c,%i,%s,%i%c",
XPL_PACKETHEADER,
XPLCMD_DATAREFUPDATEFLOATARRAY,

```



```

00155         handle,
00156         tBuf,
00157         arrayElement,
00158         XPL_PACKETTRAILER);
00159     _transmitPacket();
00160 }
00161
00162 void XPLPro::_sendname()
00163 {
00164     // register device on request only when we have a valid name
00165     if (_deviceName != NULL)
00166     {
00167         _sendPacketString(XPLRESPONSE_NAME, _deviceName);
00168     }
00169 }
00170
00171 void XPLPro::_sendResetRequest()
00172 {
00173     // request a reset only when we have a valid name
00174     if (_deviceName != NULL)
00175     {
00176         _sendPacketVoid(XPLCMD_RESET, 0);
00177     }
00178 }
00179
00180 void XPLPro::_processSerial()
00181 {
00182     // read until package header found or buffer empty
00183     while (_streamPtr->available() && _receiveBuffer[0] != XPL_PACKETHEADER)
00184     {
00185         _receiveBuffer[0] = (char)_streamPtr->read();
00186     }
00187     // return when buffer empty and header not found
00188     if (_receiveBuffer[0] != XPL_PACKETHEADER)
00189     {
00190         return;
00191     }
00192     // read rest of package until trailer
00193     _receiveBufferBytesReceived = _streamPtr->readBytesUntil(XPL_PACKETTRAILER, (char
*)_receiveBuffer[1], XPLMAX_PACKETSIZE_RECEIVE - 1);
00194     // if no further chars available, delete package
00195     if (_receiveBufferBytesReceived == 0)
00196     {
00197         _receiveBuffer[0] = 0;
00198         return;
00199     }
00200     // add package trailer and zero byte to frame
00201     _receiveBuffer[++_receiveBufferBytesReceived] = XPL_PACKETTRAILER;
00202     _receiveBuffer[++_receiveBufferBytesReceived] = 0; // old habits die hard.
00203     // at this point we should have a valid frame
00204     _processPacket();
00205 }
00206
00207 void XPLPro::_processPacket()
00208 {
00209     int tHandle;
00210     // check whether we have a valid frame
00211     if (_receiveBuffer[0] != XPL_PACKETHEADER)
00212     {
00213         return;
00214     }
00215     // branch on received command
00216     switch (_receiveBuffer[1])
00217     {
00218         // plane unloaded or XP exiting
00219         case XPL_EXITING:
00220             _connectionStatus = false;
00221             _xplStopFunction();
00222             break;
00223
00224         // register device
00225         case XPLCMD_SENDNAME:
00226             _sendname();
00227             _connectionStatus = true; // not considered active till you know my name
00228             _registerFlag = 0;
00229             break;
00230
00231         // plugin is ready for registrations.
00232         case XPLCMD_SENDREQUEST:
00233             _registerFlag = 1; // use a flag to signal registration so recursion doesn't occur
00234             break;
00235
00236         // get handle from response to registered dataref
00237         case XPLRESPONSE_DATAREF:
00238             _parseInt(&_handleAssignment, _receiveBuffer, 2);
00239             break;
00240     }

```

```

00241 // get handle from response to registered command
00242 case XPLRESPONSE_COMMAND:
00243     _parseInt(&_handleAssignment, _receiveBuffer, 2);
00244     break;
00245
00246 // int dataref received
00247 case XPLCMD_DATAREFUPDATEINT:
00248     _parseInt(&tHandle, _receiveBuffer, 2);
00249     _parseInt(&_readValueLong, _receiveBuffer, 3);
00250     _readValueFloat = 0;
00251     _readValueElement = 0;
00252     _xplInboundHandler(tHandle);
00253     break;
00254
00255 // int array dataref received
00256 case XPLCMD_DATAREFUPDATEINTARRAY:
00257     _parseInt(&tHandle, _receiveBuffer, 2);
00258     _parseInt(&_readValueLong, _receiveBuffer, 3);
00259     _parseInt(&_readValueElement, _receiveBuffer, 4);
00260     _readValueFloat = 0;
00261     _xplInboundHandler(tHandle);
00262     break;
00263
00264 // float dataref received
00265 case XPLCMD_DATAREFUPDATEFLOAT:
00266     _parseInt(&tHandle, _receiveBuffer, 2);
00267     _parseFloat(&_readValueFloat, _receiveBuffer, 3);
00268     _readValueLong = 0;
00269     _readValueElement = 0;
00270     _xplInboundHandler(tHandle);
00271     break;
00272
00273 // float array dataref received
00274 case XPLCMD_DATAREFUPDATEFLOATARRAY:
00275     _parseInt(&tHandle, _receiveBuffer, 2);
00276     _parseFloat(&_readValueFloat, _receiveBuffer, 3);
00277     _parseInt(&_readValueElement, _receiveBuffer, 4);
00278     _readValueLong = 0;
00279     _xplInboundHandler(tHandle);
00280     break;
00281
00282 // obsolete?
00283 case XPLREQUEST_REFRESH:
00284     break;
00285
00286 default:
00287     break;
00288 }
00289 // empty receive buffer
00290 _receiveBuffer[0] = 0;
00291 }
00292
00293 void XPLPro::_sendPacketVoid(int command, int handle) // just a command with a handle
00294 {
00295     // check for valid handle
00296     if (handle < 0)
00297     {
00298         return;
00299     }
00300     sprintf(_sendBuffer, "%c%c,%i%c", XPL_PACKETHEADER, command, handle, XPL_PACKETTRAILER);
00301     _transmitPacket();
00302 }
00303
00304 void XPLPro::_sendPacketString(int command, const char *str) // for a string
00305 {
00306     sprintf(_sendBuffer, "%c%c,\"%s\"%c", XPL_PACKETHEADER, command, str, XPL_PACKETTRAILER);
00307     _transmitPacket();
00308 }
00309
00310 void XPLPro::_transmitPacket(void)
00311 {
00312     _streamPtr->write(_sendBuffer);
00313     if (strlen(_sendBuffer) == 64)
00314     {
00315         // apparently a bug in arduino with some boards when we transmit exactly 64 bytes. That took a
00316         while to track down...
00317         _streamPtr->print(" ");
00318     }
00319 }
00320
00321 int XPLPro::_parseString(char *outBuffer, char *inBuffer, int parameter, int maxSize)
00322 {
00323     int cBeg;
00324     int pos = 0;
00325     int len;
00326     for (int i = 1; i < parameter; i++)

```

```

00327 {
00328     while (inBuffer[pos] != ',' && inBuffer[pos] != 0)
00329     {
00330         pos++;
00331     }
00332     pos++;
00333 }
00334
00335 while (inBuffer[pos] != '\"' && inBuffer[pos] != 0)
00336 {
00337     pos++;
00338 }
00339 cBeg = ++pos;
00340
00341 while (inBuffer[pos] != '\"' && inBuffer[pos] != 0)
00342 {
00343     pos++;
00344 }
00345 len = pos - cBeg;
00346 if (len > maxSize)
00347 {
00348     len = maxSize;
00349 }
00350 strncpy(outBuffer, (char *)&inBuffer[cBeg], len);
00351 outBuffer[len] = 0;
00352 // fprintf(errlog, "_parseString, pos: %i, cBeg: %i, deviceName: %s\n", pos, cBeg, target);
00353 return 0;
00354 }
00355
00356 int XPLPro::_parseInt(int *outTarget, char *inBuffer, int parameter)
00357 {
00358     int cBeg;
00359     int pos = 0;
00360     // search for the selected parameter
00361     for (int i = 1; i < parameter; i++)
00362     {
00363         while (inBuffer[pos] != ',' && inBuffer[pos] != 0)
00364         {
00365             pos++;
00366         }
00367         pos++;
00368     }
00369     // parameter starts here
00370     cBeg = pos;
00371     // search for end of parameter
00372     while (inBuffer[pos] != ',' && inBuffer[pos] != 0 && inBuffer[pos] != XPL_PACKETTRAILER)
00373     {
00374         pos++;
00375     }
00376     // temporarily make parameter null terminated
00377     char holdChar = inBuffer[pos];
00378     inBuffer[pos] = 0;
00379     // get integer value from string
00380     *outTarget = atoi((char *)&inBuffer[cBeg]);
00381     // restore buffer
00382     inBuffer[pos] = holdChar;
00383     return 0;
00384 }
00385
00386 int XPLPro::_parseInt(long *outTarget, char *inBuffer, int parameter)
00387 {
00388     int cBeg;
00389     int pos = 0;
00390     for (int i = 1; i < parameter; i++)
00391     {
00392         while (inBuffer[pos] != ',' && inBuffer[pos] != 0)
00393         {
00394             pos++;
00395         }
00396         pos++;
00397     }
00398     cBeg = pos;
00399     while (inBuffer[pos] != ',' && inBuffer[pos] != 0 && inBuffer[pos] != XPL_PACKETTRAILER)
00400     {
00401         pos++;
00402     }
00403     char holdChar = inBuffer[pos];
00404     inBuffer[pos] = 0;
00405     *outTarget = atoi((char *)&inBuffer[cBeg]);
00406     inBuffer[pos] = holdChar;
00407     return 0;
00408 }
00409
00410 int XPLPro::_parseFloat(float *outTarget, char *inBuffer, int parameter)
00411 {
00412     int cBeg;
00413     int pos = 0;

```

```

00414     for (int i = 1; i < parameter; i++)
00415     {
00416         while (inBuffer[pos] != ',' && inBuffer[pos] != 0)
00417         {
00418             pos++;
00419         }
00420         pos++;
00421     }
00422     cBeg = pos;
00423     while (inBuffer[pos] != ',' && inBuffer[pos] != 0 && inBuffer[pos] != XPL_PACKETTRAILER)
00424     {
00425         pos++;
00426     }
00427     char holdChar = inBuffer[pos];
00428     inBuffer[pos] = 0;
00429     *outTarget = atof((char *)&inBuffer[cBeg]);
00430     inBuffer[pos] = holdChar;
00431     return 0;
00432 }
00433
00434 int XPLPro::registerDataRef(XPString_t *datarefName)
00435 {
00436     long int startTime;
00437
00438     // registration only allowed in callback (TODO: is this limitation really necessary?)
00439     if (!_registerFlag)
00440     {
00441         return XPL_HANDLE_INVALID;
00442     }
00443     #if XPL_USE_PROGMEM
00444     sprintf(_sendBuffer, "%c%c,\"%S\"%c", XPL_PACKETHEADER, XPLREQUEST_REGISTERDATAREF, (wchar_t
*)datarefName, XPL_PACKETTRAILER);
00445     #else
00446     sprintf(_sendBuffer, "%c%c,\"%s\"%c", XPL_PACKETHEADER, XPLREQUEST_REGISTERDATAREF, (char
*)datarefName, XPL_PACKETTRAILER);
00447     #endif
00448     _transmitPacket();
00449
00450     _handleAssignment = XPL_HANDLE_INVALID;
00451     startTime = millis(); // for timeout function
00452
00453     while (millis() - startTime < XPL_RESPONSE_TIMEOUT && _handleAssignment < 0)
00454         _processSerial();
00455
00456     return _handleAssignment;
00457 }
00458
00459 int XPLPro::registerCommand(XPString_t *commandName)
00460 {
00461     long int startTime = millis(); // for timeout function
00462     #if XPL_USE_PROGMEM
00463     sprintf(_sendBuffer, "%c%c,\"%S\"%c", XPL_PACKETHEADER, XPLREQUEST_REGISTERCOMMAND, (wchar_t
*)commandName, XPL_PACKETTRAILER);
00464     #else
00465     sprintf(_sendBuffer, "%c%c,\"%s\"%c", XPL_PACKETHEADER, XPLREQUEST_REGISTERCOMMAND, (char
*)commandName, XPL_PACKETTRAILER);
00466     #endif
00467     _transmitPacket();
00468     _handleAssignment = XPL_HANDLE_INVALID;
00469     while (millis() - startTime < XPL_RESPONSE_TIMEOUT && _handleAssignment < 0)
00470     {
00471         _processSerial();
00472     }
00473     return _handleAssignment;
00474 }
00475
00476 void XPLPro::requestUpdates(int handle, int rate, float precision)
00477 {
00478     char tBuf[20]; // todo: rewrite to eliminate this buffer. Write directly to _sendBuffer?
00479     dtostrf(precision, 0, XPL_FLOATPRECISION, tBuf);
00480     sprintf(_sendBuffer, "%c%c,%i,%i,%s%c",
00481             XPL_PACKETHEADER,
00482             XPLREQUEST_UPDATES,
00483             handle,
00484             rate,
00485             tBuf,
00486             XPL_PACKETTRAILER);
00487     _transmitPacket();
00488 }
00489
00490 void XPLPro::requestUpdates(int handle, int rate, float precision, int element)
00491 {
00492     char tBuf[20]; // todo: rewrite to eliminate this buffer. Write directly to _sendBuffer?
00493     dtostrf(precision, 0, XPL_FLOATPRECISION, tBuf);
00494     sprintf(_sendBuffer, "%c%c,%i,%i,%s,%i%c",
00495             XPL_PACKETHEADER,
00496             XPLREQUEST_UPDATESARRAY,

```

```

00497         handle,
00498         rate,
00499         tBuf,
00500         element,
00501         XPL_PACKETTRAILER);
00502     _transmitPacket();
00503 }
00504
00505 void XPLPro::setScaling(int handle, int inLow, int inHigh, int outLow, int outHigh)
00506 {
00507     sprintf(_sendBuffer, "%c%c,%i,%i,%i,%i,%i%c",
00508             XPL_PACKETHEADER,
00509             XPLREQUEST_SCALING,
00510             handle,
00511             inLow,
00512             inHigh,
00513             outLow,
00514             outHigh,
00515             XPL_PACKETTRAILER);
00516     _transmitPacket();
00517 }

```

4.4 XPLPro.h

```

00001 // XPLPro.h - Library for serial interface to Xplane SDK.
00002 // Created by Curiosity Workshop, Michael Gerlicher, 2020-2023
00003 // See readme.txt file for information on updates.
00004 // To report problems, download updates and examples, suggest enhancements or get technical support,
00005 // please visit:
00006 // discord: https://discord.gg/gzXetjEST4
00007 // patreon: www.patreon.com/curiosityworkshop
00008
00008 #ifndef XPLPro_h
00009 #define XPLPro_h
00010
00011 #include <Arduino.h>
00012
00014 // Parameters that can be overwritten by command line defines
00016
00017 // Decimals of precision for floating point datarefs. More increases dataflow (default 4)
00018 #ifndef XPL_FLOATPRECISION
00019 #define XPL_FLOATPRECISION 4
00020 #endif
00021
00022 // Timeout after sending a registration request, how long will we wait for the response.
00023 // This is giant because sometimes xplane says the plane is loaded then does other stuff for a while.
00024 // (default 90000 ms)
00024 #ifndef XPL_RESPONSE_TIMEOUT
00025 #define XPL_RESPONSE_TIMEOUT 90000
00026 #endif
00027
00028 // For boards with limited memory that can use PROGMEM to store strings.
00029 // You will need to wrap your dataref names with F() macro ie:
00030 // Xinterface.registerDataref(F("laminar/B738/annunciator/drive2"), XPL_READ, 100, 0, &drive2);
00031 // Disable for boards that have issues compiling: errors with strncmp_PF for instance.
00032 #ifndef XPL_USE_PROGMEM
00033 #ifdef __AVR_ARCH__
00034 // flash strings are default on on AVR architecture
00035 #define XPL_USE_PROGMEM 1
00036 #else
00037 // and off otherwise
00038 #define XPL_USE_PROGMEM 0
00039 #endif
00040 #endif
00041
00042 // Package buffer size for send and receive buffer each.
00043 // If you need a few extra bytes of RAM it could be reduced, but it needs to
00044 // be as long as the longest dataref name + 10. If you are using datarefs
00045 // that transfer strings it needs to be big enough for those too. (default 200)
00046 #ifndef XPLMAX_PACKETSIZE_TRANSMIT
00047 #define XPLMAX_PACKETSIZE_TRANSMIT 200
00048 #endif
00049
00050 #ifndef XPLMAX_PACKETSIZE_RECEIVE
00051 #define XPLMAX_PACKETSIZE_RECEIVE 200
00052 #endif
00053
00055 // All other defines in this header must not be modified
00057
00058 // define whether flash strings will be used
00059 #if XPL_USE_PROGMEM
00060 // use Flash for strings, requires F() macro for strings in all registration calls
00061 typedef const __FlashStringHelper XPString_t;

```

```

00062 #else
00063 typedef const char XPString_t;
00064 #endif
00065
00066 // Parameters around the interface
00067 #define XPL_BAUDRATE 115200 // Baudrate needed to match plugin
00068 #define XPL_RX_TIMEOUT 500 // Timeout for reception of one frame
00069 #define XPL_PACKETHEADER '[' // Frame start character
00070 #define XPL_PACKETTRAILER ']' // Frame end character
00071 #define XPL_HANDLE_INVALID -1 // invalid handle
00072
00073 // Items in caps generally come from XPlane. Items in lower case are generally sent from the arduino.
00074 #define XPLCMD_SENDNAME 'N' // plugin request name from arduino
00075 #define XPLRESPONSE_NAME 'n' // Arduino responds with device name as initialized in the
    "begin" function
00076 #define XPLCMD_SENDREQUEST 'Q' // plugin sends this when it is ready to register bindings
00077 #define XPLREQUEST_REGISTERDATADEF 'b' // Register a dataref
00078 #define XPLREQUEST_REGISTERCOMMAND 'm' // Register a command
00079 #define XPLRESPONSE_DATADEF 'D' // Plugin responds with handle to dataref or - value if not
    found. dataref handle, dataref name
00080 #define XPLRESPONSE_COMMAND 'C' // Plugin responds with handle to command or - value if not
    found. command handle, command name
00081 #define XPLCMD_PRINTDEBUG 'g' // Plugin logs string sent from arduino
00082 #define XPLCMD_SPEAK 's' // plugin speaks string through xplane speech
00083 #define XPLREQUEST_REFRESH 'd' // the plugin will call this once xplane is loaded in order
    to get fresh updates from arduino handles that write
00084 #define XPLREQUEST_UPDATES 'r' // arduino is asking the plugin to update the specified
    dataref with rate and divider parameters
00085 #define XPLREQUEST_UPDATESARRAY 't' // arduino is asking the plugin to update the specified
    array dataref with rate and divider parameters
00086 #define XPLREQUEST_SCALING 'u' // arduino requests the plugin apply scaling to the dataref
    values
00087 #define XPLCMD_RESET 'z' // Request a reset and reregistration from the plugin
00088 #define XPLCMD_DATAREFUPDATEINT '1' // Int DataRef update
00089 #define XPLCMD_DATAREFUPDATEFLOAT '2' // Float DataRef update
00090 #define XPLCMD_DATAREFUPDATEINTARRAY '3' // Int array DataRef update
00091 #define XPLCMD_DATAREFUPDATEFLOATARRAY '4' // Float array DataRef Update
00092 #define XPLCMD_DATAREFUPDATESTRING '9' // String DataRef update
00093 #define XPLCMD_COMMANDTRIGGER 'k' // Trigger command n times
00094 #define XPLCMD_COMMANDSTART 'i' // Begin command (Button pressed)
00095 #define XPLCMD_COMMANDEND 'j' // End command (Button released)
00096 #define XPL_EXITING 'X' // XPlane sends this to the arduino device during normal
    shutdown of XPlane. It may not happen if xplane crashes.
00097
00099 class XPLPro
00100 {
00101 public:
00104     XPLPro(Stream *device);
00105
00111     void begin(const char *devicename, void (*initFunction)(void), void (*stopFunction)(void), void
        (*inboundHandler)(int));
00112
00115     int connectionStatus();
00116
00120     int commandTrigger(int commandHandle) { return commandTrigger(commandHandle, 1); };
00121
00126     int commandTrigger(int commandHandle, int triggerCount);
00127
00131     int commandStart(int commandHandle);
00132
00136     int commandEnd(int commandHandle);
00137
00141     void datarefWrite(int handle, long value);
00142
00146     void datarefWrite(int handle, int value);
00147
00152     void datarefWrite(int handle, long value, int arrayElement);
00153
00158     void datarefWrite(int handle, int value, int arrayElement);
00159
00163     void datarefWrite(int handle, float value);
00164
00169     void datarefWrite(int handle, float value, int arrayElement);
00170
00175     void requestUpdates(int handle, int rate, float precision);
00176
00182     void requestUpdates(int handle, int rate, float precision, int arrayElement);
00183
00185     void setScaling(int handle, int inLow, int inHigh, int outLow, int outHigh);
00186
00190     int registerDataRef(XPString_t *datarefName);
00191
00195     int registerCommand(XPString_t *commandName);
00196
00199     float datarefReadFloat() { return _readValueFloat; }
00200
00203     long datarefReadInt() { return _readValueLong; }

```

```

00204
00207     int datarefReadElement() { return _readValueElement; }
00208
00212     int sendDebugMessage(const char *msg);
00213
00217     int sendSpeakMessage(const char *msg);
00218
00220     void sendResetRequest(void);
00221
00224     int xloop();
00225
00226 private:
00227     void _processSerial();
00228     void _processPacket();
00229     void _transmitPacket();
00230     void _sendname();
00231     void _sendPacketVoid(int command, int handle);           // just a command with a handle
00232     void _sendPacketString(int command, const char *str); // send a string
00233     int _parseInt(int *outTarget, char *inBuffer, int parameter);
00234     int _parseInt(long *outTarget, char *inBuffer, int parameter);
00235     int _parseFloat(float *outTarget, char *inBuffer, int parameter);
00236     int _parseString(char *outBuffer, char *inBuffer, int parameter, int maxSize);
00237
00238     Stream *_streamPtr;
00239     const char *_deviceName;
00240     byte _registerFlag;
00241     byte _connectionStatus;
00242
00243     char _sendBuffer[XPLMAX_PACKETSIZE_TRANSMIT];
00244     char _receiveBuffer[XPLMAX_PACKETSIZE_RECEIVE];
00245     int _receiveBufferBytesReceived;
00246
00247     void (*_xplInitFunction)(void); // this function will be called when the plugin is ready to
receive binding requests
00248     void (*_xplStopFunction)(void); // this function will be called with the plugin receives message
or detects xplane flight model inactive
00249     void (*_xplInboundHandler)(int); // this function will be called when the plugin sends dataref
values
00250
00251     int _handleAssignment;
00252     long _readValueLong;
00253     float _readValueFloat;
00254     int _readValueElement;
00255 };

```


Index

- begin
 - [XPLPro, 7](#)
- commandEnd
 - [XPLPro, 7](#)
- commandStart
 - [XPLPro, 7](#)
- commandTrigger
 - [XPLPro, 8](#)
- connectionStatus
 - [XPLPro, 9](#)
- datarefReadElement
 - [XPLPro, 9](#)
- datarefReadFloat
 - [XPLPro, 9](#)
- datarefReadInt
 - [XPLPro, 9](#)
- datarefWrite
 - [XPLPro, 10–12](#)
- main.cpp, [17, 18](#)
- registerCommand
 - [XPLPro, 12](#)
- registerDataRef
 - [XPLPro, 12](#)
- requestUpdates
 - [XPLPro, 13](#)
- sendDebugMessage
 - [XPLPro, 14](#)
- sendResetRequest
 - [XPLPro, 14](#)
- sendSpeakMessage
 - [XPLPro, 14](#)
- setScaling
 - [XPLPro, 14](#)
- xloop
 - [XPLPro, 15](#)
- XPLPro, [5](#)
 - [begin, 7](#)
 - [commandEnd, 7](#)
 - [commandStart, 7](#)
 - [commandTrigger, 8](#)
 - [connectionStatus, 9](#)
 - [datarefReadElement, 9](#)
 - [datarefReadFloat, 9](#)
 - [datarefReadInt, 9](#)
 - [datarefWrite, 10–12](#)
 - [registerCommand, 12](#)
 - [registerDataRef, 12](#)
 - [requestUpdates, 13](#)
 - [sendDebugMessage, 14](#)
 - [sendResetRequest, 14](#)
 - [sendSpeakMessage, 14](#)
 - [setScaling, 14](#)
 - [xloop, 15](#)
 - [XPLPro, 6](#)