



Engineering Note

Topic: NeoFox Communication Interfaces

Products Affected: NeoFox

Date Issued: 04/18/2011

Description

NeoFox is a dynamic measurement system that has been designed to work with a variety of optical probes to measure oxygen in a variety of conditions. To support the various probe technologies and the broad spectrum of applications that employ the system, the NeoFox measurement system itself has a multitude of configuration options that must be considered by developers and advanced scientists. This document will discuss the process by which NeoFox parameters are written to and read from the device. It assumes that the reader already has a basic understanding of how the NeoFox measures oxygen and a basic understanding of general programming concepts.

Topics addressed in this Engineering Note include the following:

<u>Topic</u>	<u>Page</u>
Architecture Overview	2
NeoFox Serial Interface	3
DLL Interface.....	5
NeoFox DLL Function Reference.....	6
DLL Documentation - Basic Sample C++ Program	10
DLL Documentation – NeoFoxDLL.h.....	12
Driver Interface and Hardware Interface.....	14
NeoFox Variable Reference Table.....	15
Additional Firmware Controls in FW Version 2.25.....	26

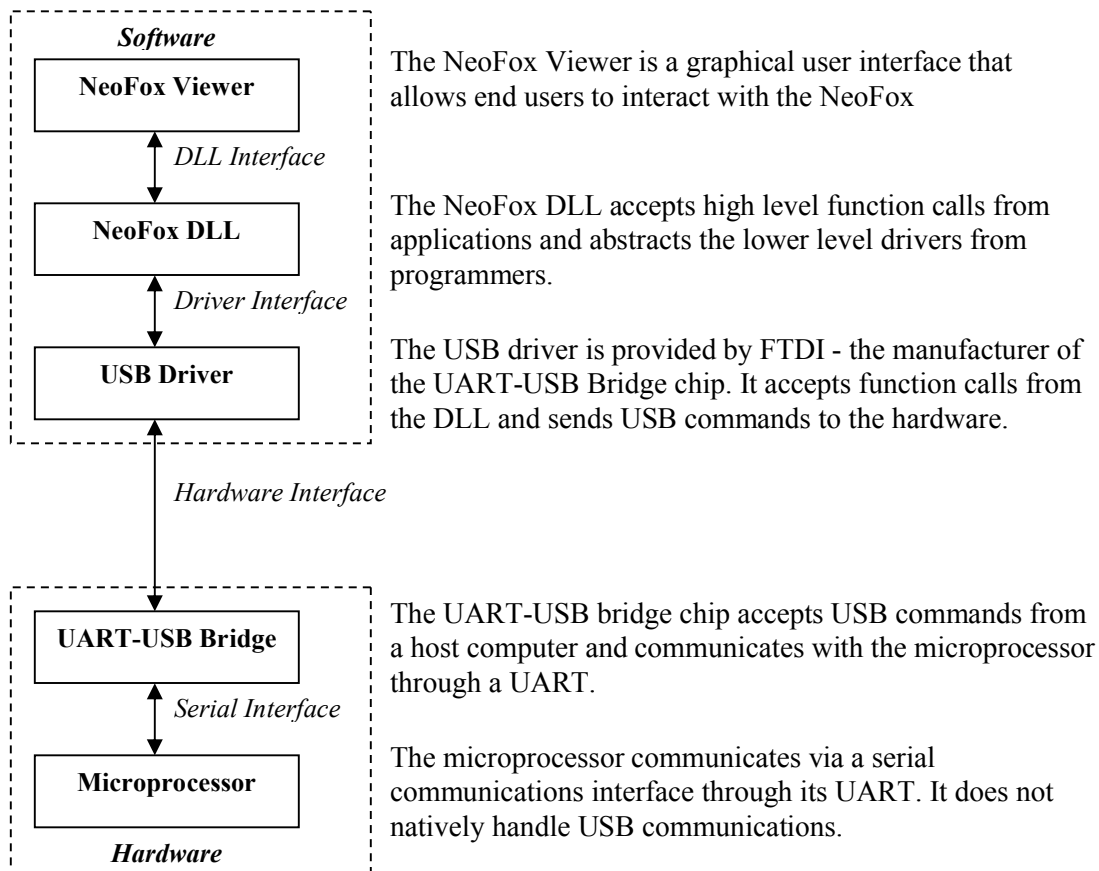
Architecture Overview

Variables and Parameters

All communication with the NeoFox centers upon either reading or setting variables on the hardware device. For instance, a developer may want to set the number of scans to average parameter or read the current sensor temperature value. In either case, there is a specific variable in the system's memory which should be set or read. This document includes reference information for all public variables and also describes the various methods with which to interact with these variables.

Communication Architecture

The diagram below illustrates the various layers involved in NeoFox's communication stack. Developers may want to interact with the device at any level. This document will provide enough information to do so by documenting the DLL interface and the serial interface. The two other interfaces – the driver interface and the hardware interface can be inferred through knowledge of the serial interface and developer information from FTDI.



NeoFox Serial Interface

Overview

The NeoFox serial protocol can be used for a number of purposes. Most significantly, future versions of the NeoFox hardware may have an RS232 serial output that will directly expose this protocol. Additionally, knowledge of the serial protocol can be used in conjunction with knowledge of the FTDI USB protocol to create custom USB drivers for the NeoFox, which are necessary for embedded host applications or non-windows based PC host applications.

From the perspective of the embedded microcontroller, the serial protocol essentially has only two functions: to read and write parameters and variables.

Settings

Setting	Value
Baud Rate:	750,000 baud (default)
Stop Bits:	1
Parity:	None
Flow Control:	None
Byte Order:	Little ending
Integer Size:	4 bytes
Float Size:	4 bytes

Writing Data to the NeoFox

All of the writable parameters on the NeoFox fall into two categories: floating point types and integer types (including char, short, etc). Regardless of type, all parameters accept the same command frame structure to set their values. This structure is documented below. There are two fields in this command that require detailed explanation.

First, bytes 8 through 11 hold the ParamType field. This field contains a code that indicates which parameter is to be set by the command. A list of parameters and codes is available at the end of this document.

The ParamValue field (bytes 12 through 15) contains the actual value that is to be set by the command. This field does not have an explicit type listed in the table below. The type is determined by the type of the parameter value (also listed in the table at the end of this document). If the parameter is an integer type (short, char, long, etc), then these four bytes are interpreted as a 4 byte signed integer (little endian) and then cast to the actual type. If the parameter is a floating point type, then the four bytes are explicitly interpreted as a little endian floating point value.

NeoFox Communication Interfaces

Addr	Type	Name	Description
0	uchar	Stx	Start of transmission character. Set to 0x03.
1	uchar	PacketType	Set to 0xC8
2	ushort	FrameSize	Set to 20 (0x14).
4	ulong	CmdNumber	Reserved. Set to 0.
8	ulong	ParamType	Type code for the parameter that is to be set.
12	n/a	ParamValue	Value for the parameter to be set. Can be a float or integer type.
16	uchar	Reserved1	Reserved. Set to 0.
17	uchar	Reserved2	Reserved. Set to 0.
18	uchar	Checksum	Set to the aggregate sum of all previous bytes modulus 256.
19	uchar	Eof	End of transmission character. Set to 0x04.

Reading Data from the NeoFox

The NeoFox streams a “data dump” of all its publicly exposed parameters after each sample. This occurs approximately once every tenth of a second. The location of each particular value within this stream is given in the table at the end of this document. The data stream is structured as follows.

Addr	Type	Name	Description
0	uchar	Stx	Start of transmission character. Set to 0x03.
1	uchar	PacketType	Set to 0xDC
2	ushort	FrameSize	Set to 5036 (0x13AC)
4	uchar	FrameCount	Number of frames uploaded since the device powered on. This rolls over after 255. It can be used to determine whether any frames have been missed.
5	uchar	ProtocolRev	Set to 0x01. There is currently only one version of the protocol. However, future versions may change and this byte will indicate that change. If this value is not 0x01, this structure is not valid.
6	uchar	Reserved1	Reserved. Do not depend on this value.
7	uchar	Reserved2	Reserved. Do not depend on this value.
8 ... 927	n/a	Data	Parameters are stored in this region of the structure. See the table at the end of this document for variable-specific documentation and addresses.
928 ... 2979	struct	SensorBlue	Blue sensor waveform raw data. This contains the data that can be seen in the “sensor waveform window” in the NeoFox Viewer.
2980 ... 5031	struct	SensorRed	Red sensor waveform raw data. This contains the data that can be seen in the “sensor waveform window” in the NeoFox Viewer.

Addr	Type	Name	Description
5032	uchar	Reserved3	Reserved. Do not depend on this value.
5033	uchar	Reserved4	Reserved. Do not depend on this value.
5034	uchar	Checksum	Set to the aggregate sum of all previous bytes modulus 256.
5035	uchar	Eof	End of transmission character. Set to 0x04

DLL Interface

The NeoFox DLL can be used by developers without the need to understand the USB or serial interface layers. The DLL itself abstracts these layers and provides an interface of high-level function calls. This DLL interface can be easily used with almost any programming language: Labview, C, C++, Visual Basic, Java, or even VBA for excel.

The DLL enables the user to perform any of the following four common actions: It allows users to open a connection to a NeoFox device, read values from the device, set parameters on the device, and close the connection to the device. The functions which enable these actions are described below.

Opening a Connection

There are two steps to creating a connection to a NeoFox. First, the user must call the DevicePerformDiscovery function, which internally creates an index list of all unopened NeoFox units that are connected to the computer. In addition to creating an indexed list, this function will also inform the user of the number of unopened NeoFox units that are connected.

Once the list has been created, the user must call the DeviceOpenChannel for each NeoFox unit for which a connection is to be established. The DeviceOpenChannel function takes an index as its parameter and it opens the particular NeoFox unit that corresponds to that index in the list that was generated by the DevicePerformDiscovery function. However, since DevicePerformDiscovery does not provide any information about the units that it discovers, the indices are essentially arbitrary and in order to select any particular NeoFox unit, the user will basically need to open all available NeoFox units and they query them each individually to determine its name. It can then close connections to the units that it does not need.

Reading Data from the NeoFox

Reading variables from the device is accomplished through either of two functions: DeviceGetParameter or DeviceGetParameterFloat. The determination of which function to use is based on the type of the variable that is to be read. Use DeviceGetParameterFloat for floating point type parameters and DeviceGetParameter for any other type of parameter.

It is important to note that the NeoFox is a discrete sampling system. New samples are only refreshed from the hardware device approximately once every tenth of a second, and operating system latency may result in even lower throughput. Therefore, developers may find it useful to poll the Millisecond Count variable to evaluate whether or not information in the DLL has been refreshed since the last time that information was read.

Writing Data to the NeoFox

Writing variables to the device is similarly accomplished through either of two functions: DeviceSetParameter or DeviceSetParameterFloat. Again, the determination of which function to use will be based on the type of the data that is to be sent.

Closing a Connection

When interaction with the device is complete, the connection should be closed by calling the DeviceClose function.

Application Maintenance

Before calling any functions from the DLL, applications should make a call to the DLL's ApplicationStartup function, which will begin a new thread in which it can run its background processes. Before exiting, applications should call ApplicationShutdown to terminate this thread.

Remarks

There is one additional issue that some early developers have faced which bears some discussion. The szText parameters, which are defined as type LPTSTR, are implemented as type char* in the DLL. There is a known issue with the Microsoft Visual C++ compiler in which sometimes the compiler has trouble casting between LPTSTR and char* types. One simple way to resolve this issue is to not include windows.h, which defines LPTSTR, and use a typedef instead (typedef char* LPTSTR). Another, more advisable, way to correct this issue is to set the project properties to use a multi-byte character set. This can be done by adjusting the "character set" project property at Properties → Configuration Properties → General.

It is also important to note that the maximum number of NeoFox units that can be supported by the DLL at one time is 8.

NeoFox DLL Function Reference

int DeviceSetParameter(int hDevice, int ParamType, int dValue)

Description

Sets an integer parameter on the NeoFox.

Parameters

int	hDevice	Handle to an open NeoFox device.
int	ParamType	Unique code for the parameter which is to be set.
int	dValue	Value to be written to the parameter.

Return Value

Number of bytes written to the hardware device in order to issue the set parameter command message. This includes the command packet overhead in addition to the ParamType code and the dValue payload. At the time of this writing, the size of this transaction is 20 bytes.

Remarks

Although the dValue parameter is a 4 byte integer, this function should also be used to set short and char type parameters as well.

int DeviceSetParameterFloat(int hDevice, int ParamType, float fValue)

Description

Sets a floating point parameter on the NeoFox.

Parameters

int	hDevice	Handle to an open NeoFox device.
int	ParamType	Unique code for the parameter which is to be set.
float	fValue	Value to be written to the parameter.

Return Value

Number of bytes written to the hardware device in order to issue the set parameter command message. This includes the command packet overhead in addition to the ParamType code and the dValue payload. At the time of this writing, the size of this transaction is 20 bytes.

int DeviceGetParameter(int hDevice, int ParamType, LPTSTR szText)

Description

Reads an integer type parameter from the NeoFox device.

Parameters

int	hDevice	Handle to an open NeoFox device
int	ParamType	Unique code for the parameter which is to be read.
LPTSTR	szText	A character buffer (char*). Be sure to pass this parameter a pointer to an initialized buffer of at least 50 bytes.

Return Value

The function returns an int that represents the value that has been queried from the device. In some cases, it will also populate the szText buffer with a textual representation of the integer value.

Remarks

Although this function returns a 4 byte integer value, it should also be used to query the values of short and char type parameters. It is also important to note that although the function will sometimes populate the szText buffer, users are advised not to use this value unless the szText parameter is explicitly discussed in a particular variable's information (from the variable table at the end of this document).

float DeviceGetParameterFloat(int hDevice, int ParamType, LPTSTR szText)

Description

Reads a floating point type parameter from the NeoFox device.

Parameters

int	hDevice	Handle to an open NeoFox device
int	ParamType	Unique code for the parameter which is to be read.
LPTSTR	szText	A character buffer (char*). Be sure to pass this parameter a pointer to an initialized buffer of at least 50 bytes.

Return Value

The function returns a float that represents the value that has been queried from the device. In some cases, it will also populate the szText buffer with a textual representation of the integer value.

Remarks

Although this function returns a 4 byte integer value, it should also be used to query the values of short and char type parameters. It is also important to note that although the function will sometimes populate the szText buffer, users are advised not to depend on this information, which is not explicitly supported.

int ApplicationStartup(int hInstance)

Description

This function starts a thread for the DLL.

Parameters

int	hInstance	Pass NULL to this parameter.
-----	-----------	------------------------------

Return Value

This function always returns a value of 1.

Remarks

This function should be called at the beginning of any user program, before any other DLL functions are called.

int DevicePerformDiscovery(int vendorID, int productID, int ShowGUI)

Description

This function instructs the DLL to create an indexed list of all NeoFox devices that are connected to the system via USB but have not been opened already.

Parameters

int	vendorID	Vendor ID for the NeoFox product. The value is 0x2457.
int	productID	Product ID for the NeoFox product. The value is 0x3000.
int	ShowGUI	This parameter is reserved. Pass it a value of 0.

Return Value

This function returns the number of NeoFox devices that have been connected to the system via USB, but have not yet been opened with the DeviceOpenChannel function.

Remarks

This function must be called before a call to DeviceOpenChannel. It creates internally an indexed list of unopened NeoFox devices. The DeviceOpenChannel function will accept one of the indices from that list as a parameter to determine which device will be opened. It is important to note that once the DeviceOpenChannel function has been called to open a connection to any of the NeoFoxes in the list, the remaining indices remain valid until the next call to DevicePerformDiscovery. When DevicePerformDiscovery is called after DeviceOpenChannel has been called, it will re-index its list without including the newly opened NeoFox. Subsequent calls to DeviceOpenChannel will need to use the new indices as appropriate.

int DeviceOpenChannel(int Select, int SubChannel, int ShowGUI)

Description

Opens a NeoFox device and returns a handle to that device.

Parameters

int	Select	Index of the unopened NeoFox which is to be opened. See remarks for more information about this parameter.
int	SubChannel	This parameter is reserved. Pass it a value of 0.
int	ShowGUI	This parameter is reserved. Pass it a value of 0.

Return Value

Returns an integer handle to the NeoFox device that has been opened by the function. A value of 0 or -1 indicates that the function has failed to open the NeoFox device properly. Handle values begin at 10000 and increment upwards as new NeoFox devices are opened.

Remarks

This function should be called immediately after the DevicePerformDiscovery function has been called. The DevicePerformDiscovery function will create an indexed list of all NeoFoxes that are connected to the computer via USB and the DevicePerformDiscovery function will then instruct it to open a connection to one of those devices by passing in a specific index from the list and receiving a handle to the connection.

One common question is how the appropriate index value for the Select parameter should be determined. For those users who can assume that only one NeoFox device will be connected to the system, the simple answer is to pass this parameter a value of 0. Some users may need to account for the scenario where multiple NeoFox units are present in the system, but only one particular unit is to be opened. In this case, the suggested approach is to open all available NeoFox units and query their names directly, then close connections to all but the target device. Note that the number of unopened devices attached to the system at any time is returned by the DevicePerformDiscovery function.

int ApplicationShutdown()

Description

Terminates the DLL's thread and performs cleanup operations.

Parameters

None

Return Value

Returns 1 if successful.

int DeviceClose(int hDevice)

Description

Terminates a connection to a particular device.

Parameters

int hDevice Handle to an open NeoFox device.

Return Value

Returns 1 if successful.

DLL Documentation - Basic Sample C++ Program

```
//This program will query the Number of averages parameter
// from the Neofox device, then increment that value by
// one and write it back.
//
//If the program is run multiple times, the output will
// increment by 1 each time that the program is run.

void BasicExample()
{
    int        hDevice;
    int        nAvg;
    char       lpchStr[50];
    char       lpchDummy[50];

    //Tell the DLL to create a background thread
    ApplicationStartup(NULL);

    //Tell the DLL to create a list of unopened Neofox units
    DevicePerformDiscovery(0x2457, 0x3000, 0);

    //Open a connection to the Neofox (index 0 in the list)
    hDevice = DeviceOpenChannel(0, 0, 0);
```

```

//Wait for 2000 ms to give the DLL time to open the device
WaitMS(2000);

//Read the value of this device's "number of averages" parameter
nAvg = DeviceGetParameter(hDevice, 129, lpchDummy);

//Increment the "number of averages" parameter and write it back
DeviceSetParameter(hDevice, 129, nAvg + 1);

//Release the device
DeviceClose(hDevice);

//Terminate the DLL background thread
ApplicationShutdown();

//Print the original number of averages
sprintf_s(lpchStr, "Number of Averages: %d\n", nAvg);
printf(lpchStr);

}

```

DLL Documentation - Advanced Sample C++ Program

```

//This function continuously polls all attached Neofox devices for oxygen and temperature
// readings, and prints the values to the console.
void AdvancedExample()
{
    int     hDevice[MAX_NUM_NEOFOXES]; //array of handles to Neofox units
    char    lpchName[50];              //stores the name of the neofox device
    char    lpchDummy[50];             //dummy variable for szText parameter
    int     nIndex;                    //iterating index
    int     nTemperature;              //fixed point readback value of temp
    float   fTemperature;              //floating point readback value of temp
    float   fOxygen;                  //floating point readback value of oxygen
    unsigned int uiMsCount;            //millisecond count from most recent sample
    unsigned int uiLastMsCount[MAX_NUM_NEOFOXES]; //millisecond count from previous samples

    //Initialize the array of sample times
    for (nIndex = 0; nIndex < MAX_NUM_NEOFOXES; nIndex++)
        uiLastMsCount[nIndex] = 0;

    //Intialization
    ApplicationStartup(NULL);
    DevicePerformDiscovery(0x2457, 0x3000, 0);

    //Try to open 8 neofoxes. If there are less than 8 present,
    // the returned handle for unoccupied indices will be 0.
    for (nIndex = 0; nIndex < MAX_NUM_NEOFOXES; nIndex++)
        hDevice[nIndex] = DeviceOpenChannel(nIndex, 0, 0);

    while (!_kbhit())
        for (nIndex = 0; nIndex < MAX_NUM_NEOFOXES; nIndex++)
            if (hDevice[nIndex])
            {
                //Check to see if a fresh sample has been loaded and skip if not
            }

```

NeoFox Communication Interfaces

```

    uiMsCount = DeviceGetParameter(hDevice[nIndex], NEOFOX_UP_TIME, lpchDummy);
    if (uiMsCount != uiLastMsCount[nIndex])
    {
uiLastMsCount[nIndex] = uiMsCount;

        //Get device name
        DeviceGetParameter(hDevice[nIndex], NEOFOX_NAME, lpchName);

        //Get device temperature. Note the division by 65536 to convert from fixed point
        nTemperature = DeviceGetParameter(hDevice[nIndex], NEOFOX_TEMPERATURE, lpchDummy);
        fTemperature = (float) nTemperature / 65536.0f;

        //Get oxygen
        fOxygen = DeviceGetParameterFloat(hDevice[nIndex], NEOFOX_OXYGEN_CONVERTED, lpchDummy);

        //Print the output
        printf(lpchName);
        printf(": %d, %f, %f \n", uiMsCount, fTemperature, fOxygen);
    }
}

//Close the devices
for (nIndex = 0; nIndex < MAX_NUM_NEOFOXES; nIndex++)
    if (hDevice[nIndex])
        DeviceClose(hDevice[nIndex]);

ApplicationShutdown();
}

```

DLL Documentation – NeoFoxDLL.h

```

#ifndef _NEOFOXDLL_H_
#define _NEOFOXDLL_H_

//*****
//*****
//ParamType Codes -->
//*****
//*****

#define NEOFOX_OXYGEN 20 // Calculated O2 level
#define NEOFOX_OXYGEN_CONVERTED 23 // Calculated oxygen in alternate units
#define NEOFOX_OXYGEN_UNITS 152 // Units of measurement for oxygen
#define NEOFOX_TAU 19 // Calculated TAU
#define NEOFOX_CAL_TEMP_SOURCE 165 // Temperature source to use when computing O2
#define NEOFOX_CAL_DEFAULT_TEMP 164 // Default temperature to use if no sensor
#define NEOFOX_TEMPERATURE 10 // Temperature sensor reading
#define NEOFOX_PHASE_AVG_CNT_BLUE 129 // # phase results to average for blue LED
#define NEOFOX_LED_CONTROL 121 // Misc. LED control
#define NEOFOX_NAME 1 // Name/serial number (read only)
#define NEOFOX_AOUT_VOLTAGE_SOURCE 212 // Selects the data source for the analog voltage output
#define NEOFOX_AOUT_CURRENT_SOURCE 213 // Selects the data source for the analog 4-20 mA output
#define NEOFOX_AOUT_VOLTAGE_LBOUND 214 // Lower bound for the analog voltage output
#define NEOFOX_AOUT_VOLTAGE_UBOUND 215 // Upper bound for the analog boltage output
#define NEOFOX_AOUT_CURRENT_LBOUND 216 // Lower bound for the analog 4-20 mA output
#define NEOFOX_AOUT_CURRENT_UBOUND 217 // Upper bound for the analog 4-20 mA output
#define NEOFOX_SALINITY_CORRECTION 218 // Salinity correction factor for computing dissolved o2.
#define NEOFOX_AUTOGAIN_ENABLE 101 // Enable autogain sequence
#define NEOFOX_RED_LED_PGA_GAIN 105 // Programmable gain amplifier - Red LED

```

```

#define NEOFOX_BLUE_LED_PGA_GAIN 102 // Programmable gain amplifier - Blue LED
#define NEOFOX_DAC_LED_CURRENT_RED 140 // Red LED current level
#define NEOFOX_DAC_LED_CURRENT_BLUE 143 // Blue LED current level
#define NEOFOX_DAC_APD_GAIN 141 // Avalanche photodiode gain (voltage)
#define NEOFOX_ADC_APD_VOLTAGE 17 // Measured power supply level of APD supply
#define NEOFOX_ADC_PRESSURE 15 // Air pressure reading
#define NEOFOX_FLASH_WRITE 93 // Write all non-volatile parameters to flash
#define NEOFOX_FPGA_STATUS 18 // Status register of FPGA
#define NEOFOX_FIRMWARE_VER 2 // Target ARM Main Processor Revision
#define NEOFOX_UP_TIME 74 // Number of milliseconds Neofox has been on (read only)
#define NEOFOX_CAL_METHOD 163 // Type of calibration to use
#define NEOFOX_CAL_2PT_SLOPE 174 // Slope - Two point calibration equation
#define NEOFOX_CAL_2PT_OFFSET 175 // Offset - Two point calibration equation
#define NEOFOX_CAL_2PT_TAU_0 170 // Two point calibration data point
#define NEOFOX_CAL_MULTI_A0 200 // Calibration constant
#define NEOFOX_CAL_MULTI_A1 201 // Calibration constant
#define NEOFOX_CAL_MULTI_A2 202 // Calibration constant
#define NEOFOX_CAL_MULTI_B0 203 // Calibration constant
#define NEOFOX_CAL_MULTI_B1 204 // Calibration constant
#define NEOFOX_CAL_MULTI_B2 205 // Calibration constant
#define NEOFOX_CAL_MULTI_C0 206 // Calibration constant
#define NEOFOX_CAL_MULTI_C1 207 // Calibration constant
#define NEOFOX_CAL_MULTI_C2 208 // Calibration constant
#define NEOFOX_CAL_MULTI_T0 209 // Calibration constant
#define NEOFOX_CAL_MULTI_T1 210 // Calibration constant
#define NEOFOX_CAL_MULTI_T2 211 // Calibration constant

//*****
//Enum Values -->
//*****

#define NEOFOX_O2_UNITS_PERCENT 0 // Percent - partial pressure
#define NEOFOX_O2_UNITS_DO_PPM 1 // ppm - concentration
#define NEOFOX_O2_UNITS_TORR 4 // Torr - Partial Pressure
#define NEOFOX_O2_UNITS_UMOL_L 7 // umol/L - Concentration

#define NEOFOX_TEMP_SOURCE_NONE 0
#define NEOFOX_TEMP_SOURCE_SENSOR 1
#define NEOFOX_TEMP_SOURCE_FIXED 2

#define NEOFOX_FLASHING_ON 3
#define NEOFOX_FLASHING_OFF 0

#define NEOFOX_AUTOGAIN_ENABLED 1
#define NEOFOX_AUTOGAIN_DISABLED 0

#define NEOFOX_GAIN_1X 0
#define NEOFOX_GAIN_2X 1
#define NEOFOX_GAIN_4X 2
#define NEOFOX_GAIN_8X 3
#define NEOFOX_GAIN_20X 4
#define NEOFOX_GAIN_40X 5
#define NEOFOX_GAIN_80X 6
#define NEOFOX_GAIN_160X 7

#define NEOFOX_CALMETHOD_NONE 0
#define NEOFOX_CALMETHOD_TWOPT 1
#define NEOFOX_CALMETHOD_MULTIPT 2
#define NEOFOX_CALMETHOD_SINGLEPT 3

#define MAX_NUM_NEOFOXES 8

//*****
//Funtion Prototypes -->
//*****

#define NEOFOX_API __declspec(dllimport) int _stdcall
#define NEOFOX_API_FLOAT __declspec(dllimport) float _stdcall

//If you do not include windows.h, which defines LPTSTR,

```

NeoFox Communication Interfaces

```
// then uncomment the following line or replace the LPTSTR in the function prototypes with char*
// #typedef char* LPTSTR

// Standard Routines
NEOFOX_API ApplicationStartup(int hInstance);
NEOFOX_API DeviceGetParameter(int hDevice, int ParamType, LPTSTR szText);
NEOFOX_API FLOAT DeviceGetParameterFloat(int hDevice, int ParamType, LPTSTR szText);
NEOFOX_API DeviceSetParameter(int hDevice, int ParamType, int dValue);
NEOFOX_API DeviceSetParameterFloat(int hDevice, int ParamType, float dValue);
NEOFOX_API DeviceClose(int hDevice);
NEOFOX_API ApplicationShutdown();
NEOFOX_API DeviceOpenChannel(int Select, int SubChannel, int ShowGUI);
NEOFOX_API DevicePerformDiscovery(int vendorID, int productID, int ShowGUI);
NEOFOX_API DeviceUSBInitialize(struct TDeviceInfo *DeviceInfo, int SubChannel, int ShowGUI);
NEOFOX_API DeviceGetStatus(int hDeviceHandle);
NEOFOX_API DeviceCheckForNewData(int hDeviceHandle);
NEOFOX_API DeviceGetSensorData(int hDevice, unsigned int *arrayPtr, int NumWaveforms, int NumSamples);

#endif
```

Driver Interface and Hardware Interface

Some developers may wish to access the NeoFox device at the USB driver level or the hardware level. To do this, developers will need to combine the information from the serial interface description in this document with information from the USB-UART bridge chip manufacturer, FTDI. The chip that is used on the NeoFox is the FT232RQ.

Driver Interface

An example of an application that would use the driver interface level would be an application that needs to run on Linux or Macintosh computers – systems for which a USB driver is already available from FTDI but support for the NeoFox DLL is not.

Driver information is available from FTDI at the link below. This documentation will explain how to use function calls to the USB driver to exercise the USB-UART bridge’s UART. In other words, it will explain how to write bytes to the UART and read bytes from the UART. In this way, the developer can use the serial interface information from this document to interact with the microcontroller.

The driver documentation is located here: <http://www.ftdichip.com/Drivers/D2XX.htm>.

Hardware Interface

An example of an application that would use the hardware interface level would be a system that uses an embedded microcontroller’s USB “On the Go” host feature to host the NeoFox. In this case, the developer would need to write custom USB drivers for his device. To do this, he would need to understand the way that the NeoFox hardware communicates over USB – the hardware interface.

Hardware interface documentation is also available from FTDI. This documentation explains how USB commands can be sent to the device to exercise the UART. Again, this information can be combined with knowledge of the serial interface to produce a complete interface description of the device.

Unfortunately, the hardware interface information from FTDI is not publicly available. However, it can be obtained from FTDI by entering into an NDA with them. To obtain the USB communication documentation, send an email to support1@ftdichip.com.

NeoFox Variable Reference Table

The following table lists all publicly exposed parameters and variables that are available to the developer. Because the DLL interface and serial interface both operate on the same variable space, the information contains information that is relevant to both.

The table information is organized as follows:

Parameter Name	Type	Range	Param Type	Address V1
DLL ParamType Constant	Set Function		Get Function	

- **Parameter Name** – The name of the parameter, as you will find it throughout this document and others that describe the NeoFox.
- **Type** – The data type of the parameter.
- **Range** – The valid range of values for the parameter or value. For writable parameters, developers should ensure that the values written are within the acceptable range. For readable parameters, the developer should consider values that are outside of the given range invalid.
- **Param Type** – The parameter code, which is used in the both the DLL interface and the serial interface. In the DLL, this value is passed to DeviceGetParameter as the ParamType argument in order to read a value and it is passed to DeviceSetParameter as the ParamType argument in order to set a value. In the serial protocol, it serves as the ParamType code within a command message to set a parameter.
- **Address V1** – The address for the parameter within the serial interface’s readback “data dump”. This address is only valid for protocol version 1 (ie: the ProtocolRev field = 1). Future versions of the protocol may have parameters located at different addresses.
- **DLL ParamType Constant** – The constant which has been defined in the NeoFoxDLL.h file to represent the ParamType code for this parameter.
- **Set Function** – The DLL function to call in order to set the parameter.
- **Get Function** – The DLL function to call in order to read the parameter.

Percent Oxygen	Float	$0 \leq X$	20	740
NEOFOX_OXYGEN	Read-Only		DeviceGetParameterFloat	
Percent Oxygen is the partial pressure of oxygen, expressed as percent of 1 ATM. Note that it does not actually represent the concentration of oxygen relative to other gasses in the measurement sample.				
Converted Oxygen	Float	$0 \leq X$	23	864
NEOFOX_OXYGEN_CONVERTED	Read-Only		DeviceGetParameterFloat	
Converted oxygen is the current oxygen reading, converted from percent of 1ATM to an alternate unit. The Oxygen Units parameter documentation has more information about which units are available.				

Oxygen Units	Uint	Enum	152	488
NEOFOX_OXYGEN_UNITS	DeviceSetParameter		DeviceGetParameter	
Sets the unit type for the Converted Oxygen parameter.				
Value	Units	Description	DLL Constant	
0	Percent (of 1 ATM)	Gaseous – Partial Pressure	NEOFOX_O2_UNITS_PERCENT	
1	Parts per Million	Dissolved – Concentration	NEOFOX_O2_UNITS_DO_PPM	
4	Torr	Gaseous – Partial Pressure	NEOFOX_O2_UNITS_TORR	
7	Micro Mole per Liter	Dissolved - Concentration	NEOFOX_O2_UNITS_UMOL_L	
Tau	Float	-1 < X	19	736
NEOFOX_TAU	Read-Only		DeviceGetParameterFloat	
Tau is an intermediate value, expressed in microseconds, that is calculated in the conversion from phase to oxygen. It should be used for diagnostic purposes only.				
Temperature Source	Uint	Enum	165	316
NEOFOX_CAL_TEMP_SOURCE	DeviceSetParameter		DeviceGetParameter	
Temperature Source determines which source should be used for temperature measurement during calibration. When the value is 0 (no temperature source), measurements which are dependant upon temperature such as dissolved O2 readings and multipoint calibrations should return -1.				
Value	Description		DLL Constant	
0	No temperature source is used.		NEOFOX_TEMP_SOURCE_NONE	
1	The external temperature sensor is used.		NEOFOX_TEMP_SOURCE_SENSOR	
2	The “Fixed Temperature” is used.		NEOFOX_TEMP_SOURCE_FIXED	
Fixed Temperature	Float	X < 200	164	304
NEOFOX_CAL_DEFAULT_TEMP	DeviceSetParameterFloat		DeviceGetParameterFloat	
Fixed Temperature is a user defined fixed temperature that can be used instead of the external temperature sensor in the conversion process.				
Sensor Temperature	Int	X < 200*2 ¹⁶	10	796
NEOFOX_TEMPERATURE	Read-Only		DeviceGetParameter	
Sensor Temperature is the temperature value of the sensor in degrees Celsius.				
It is important to note that this parameter is stored in a fixed point representation in the NeoFox’s variable space. Therefore, in order to use this value as a floating point number, the raw value must be converted from fixed point (integer type) to floating point with the formula below.				
$\text{Floating Point Temperature Value (in degrees Celsius)} = \text{Raw Integer Temperature Value} / 2^{16}$				
Number of Averages	Uint	1 ≤ X ≤ 300	129	88
NEOFOX_PHASE_AVG_CNT_BLUE	DeviceSetParameter		DeviceGetParameter	
Number of Averages is the number of oxygen samples that are averaged together on the NeoFox to produce the current value of the Percent Oxygen and Converted Oxygen values. This is a running average.				

Flashing On/Off	Uint	Enum	121	528									
NEOFOX_LED_CONTROL	DeviceSetParameter		DeviceGetParameter										
<p>The Flashing On/Off parameter enables and disables sampling. When this parameter is set to off, the LEDs will not flash. Tau and oxygen measurements will be 0 or -1 and should be considered invalid.</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Description</th> <th>DLL Constant</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Flashing Off</td> <td>NEOFOX_FLASHING_OFF</td> </tr> <tr> <td>3</td> <td>Flashing On</td> <td>NEOFOX_FLASHING_ON</td> </tr> </tbody> </table>					Value	Description	DLL Constant	0	Flashing Off	NEOFOX_FLASHING_OFF	3	Flashing On	NEOFOX_FLASHING_ON
Value	Description	DLL Constant											
0	Flashing Off	NEOFOX_FLASHING_OFF											
3	Flashing On	NEOFOX_FLASHING_ON											
Name	char[]	n/a	1	n/a									
NEOFOX_NAME	Read-Only		DeviceGetParameter										
<p>This is the unique serial number that is programmed in to every NeoFox unit. Unfortunately, it cannot be read through the serial interface. To read this parameter from the DLL, call DeviceGetParameter and the serial number will be populated into the szText buffer. The serial number cannot be set through the DLL.</p>													
0-5V Lower Bound	Float	n/a	214	472									
NEOFOX_AOUT_VOLTAGE_LBOUND	DeviceSetParameterFloat		DeviceGetParameterFloat										
<p>The 0-5V Lower Bound parameter is one of the 0-5V analog output settings. The value of this particular parameter sets the lower bound value for the following formula, which determines the 0-5 Volt analog output voltage. More information about this parameter and how to set up the analog outputs is available in the user manual.</p> $\text{Output (Volts)} = (\text{Source Value} - \text{Lower Bound Value}) / (\text{Upper Bound Value} - \text{Lower Bound Value}) * 5 \text{ Volts}$													
0-5V Upper Bound	Float	n/a	215	476									
NEOFOX_AOUT_VOLTAGE_UBOUND	DeviceSetParameterFloat		DeviceGetParameterFloat										
<p>The 0-5V Upper Bound parameter is one of the 0-5V analog output settings. The value of this particular parameter sets the upper bound value for the following formula, which determines the 0-5 Volt analog output voltage. More information about this parameter and how to set up the analog outputs is available in the user manual.</p> $\text{Output (Volts)} = (\text{Source Value} - \text{Lower Bound Value}) / (\text{Upper Bound Value} - \text{Lower Bound Value}) * 5 \text{ Volts}$													
4-20mA Lower Bound	Float	n/a	216	480									
NEOFOX_AOUT_CURRENT_LBOUND	DeviceSetParameterFloat		DeviceGetParameterFloat										
<p>The 4-20mA Lower Bound parameter is one of the 4-20mA analog output settings. The value of this particular parameter sets the lower bound value for the following formula, which determines the 4-20mA analog output current. More information about this parameter and how to set up the analog outputs is available in the user manual.</p> $\text{Output (mA)} = [16\text{mA} * (\text{Source Value} - \text{Lower Bound Value}) / (\text{Upper Bound Value} - \text{Lower Bound Value})] + 4\text{mA}$													
4-20mA Upper Bound	Float	n/a	217	484									
NEOFOX_AOUT_CURRENT_UBOUND	DeviceSetParameterFloat		DeviceGetParameterFloat										
<p>The 4-20mA Upper Bound parameter is one of the 4-20mA analog output settings. The value of this particular parameter sets the upper bound value for the following formula, which determines the 4-20mA analog output current. More information about this parameter and how to set up the analog outputs is available in the user manual.</p> $\text{Output (mA)} = [16\text{mA} * (\text{Source Value} - \text{Lower Bound Value}) / (\text{Upper Bound Value} - \text{Lower Bound Value})] + 4\text{mA}$													

NeoFox Communication Interfaces

0-5V Data Source	Char	Enum	212	468									
NEOFOX_AOUT_VOLTAGE_SOURCE	DeviceSetParameter		DeviceGetParameter										
<p>The 0-5V Data Source parameter is one of the 0-5V analog output settings. The value of this particular parameter sets the source value for the following formula, which determines the 0-5V analog output voltage. In the formula, the source value and bound parameters are considered dimensionless scalars – that is, only the actual values matter. The units are irrelevant. More information about this parameter and how to set up the analog outputs is available in the user manual.</p> $\text{Output (Volts)} = (\text{Source Value} - \text{Lower Bound Value}) / (\text{Upper Bound Value} - \text{Lower Bound Value}) * 5 \text{ Volts}$													
4-20mA Data Source	Char	Enum	213	469									
NEOFOX_AOUT_CURRENT_SOURCE	DeviceSetParameter		DeviceGetParameter										
<p>The 4-20mA Data Source parameter is one of the 4-20mA analog output settings. The value of this particular parameter sets the source value for the following formula, which determines the 4-20mA analog output current. In the formula, the source value and bound parameters are considered dimensionless scalars – that is, only the actual values matter. The units are irrelevant. More information about this parameter and how to set up the analog outputs is available in the user manual.</p> $\text{Output (mA)} = [16\text{mA} * (\text{Source Value} - \text{Lower Bound Value}) / (\text{Upper Bound Value} - \text{Lower Bound Value})] + 4\text{mA}$ <p>The 4-20mA Data Source parameter is one of the 4-20mA analog output settings. The value of this particular parameter sets the source value for the following formula, which determines the 4-20mA analog output current. In the formula, the source value and bound parameters are considered dimensionless scalars – that is, only the actual values matter. The units are irrelevant. More information about this parameter and how to set up the analog outputs is available in the user manual.</p> $\text{Output (mA)} = [16\text{mA} * (\text{Source Value} - \text{Lower Bound Value}) / (\text{Upper Bound Value} - \text{Lower Bound Value})] + 4\text{mA}$													
Salinity Correction Factor	Float	$0 \leq X$	218	492									
NEOFOX_SALINITY_CORRECTION	DeviceSetParameterFloat		DeviceGetParameterFloat										
<p>The Salinity Correction Factor is used in the conversion partial pressure to dissolved oxygen. More information about how this parameter is used is available in the user manual.</p>													
Autogain Enable	Uint	Enum	101	600									
NEOFOX_AUTOGAIN_ENABLE	DeviceSetParameter		DeviceGetParameter										
<p>The Autogain Enable parameter controls whether the NeoFox device will set its gain settings automatically. When autogain is enabled, the APD voltage, LED drive levels, and gain levels will be optimized automatically. However, it is recommended that the autogain should be run at the beginning of experiments to set up the device properly and then disabled during the course of the experiment itself.</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Description</th> <th>DLL Constant</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Autogain disabled</td> <td>NEOFOX_AUTOGAIN_DISABLED</td> </tr> <tr> <td>1</td> <td>Autogain enabled</td> <td>NEOFOX_AUTOGAIN_ENABLED</td> </tr> </tbody> </table>					Value	Description	DLL Constant	0	Autogain disabled	NEOFOX_AUTOGAIN_DISABLED	1	Autogain enabled	NEOFOX_AUTOGAIN_ENABLED
Value	Description	DLL Constant											
0	Autogain disabled	NEOFOX_AUTOGAIN_DISABLED											
1	Autogain enabled	NEOFOX_AUTOGAIN_ENABLED											

Reference PGA Gain	Uint	Enum	105	500																											
NEOFOX_RED_LED_PGA_GAIN	DeviceSetParameter		DeviceGetParameter																												
<p>The Reference PGA Gain parameter controls the amount of electronic gain that is applied to the input signal before analog to digital conversion. This parameter applies only at times when the reference LED is being sampled and not when the actual response of the chemistry is being sampled.</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Description</th> <th>DLL Constant</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>1X</td> <td>NEOFOX_GAIN_1X</td> </tr> <tr> <td>1</td> <td>2X</td> <td>NEOFOX_GAIN_2X</td> </tr> <tr> <td>2</td> <td>4X</td> <td>NEOFOX_GAIN_4X</td> </tr> <tr> <td>3</td> <td>8X</td> <td>NEOFOX_GAIN_8X</td> </tr> <tr> <td>4</td> <td>20X</td> <td>NEOFOX_GAIN_20X</td> </tr> <tr> <td>5</td> <td>40X</td> <td>NEOFOX_GAIN_40X</td> </tr> <tr> <td>6</td> <td>80X</td> <td>NEOFOX_GAIN_80X</td> </tr> <tr> <td>7</td> <td>160X</td> <td>NEOFOX_GAIN_160X</td> </tr> </tbody> </table>					Value	Description	DLL Constant	0	1X	NEOFOX_GAIN_1X	1	2X	NEOFOX_GAIN_2X	2	4X	NEOFOX_GAIN_4X	3	8X	NEOFOX_GAIN_8X	4	20X	NEOFOX_GAIN_20X	5	40X	NEOFOX_GAIN_40X	6	80X	NEOFOX_GAIN_80X	7	160X	NEOFOX_GAIN_160X
Value	Description	DLL Constant																													
0	1X	NEOFOX_GAIN_1X																													
1	2X	NEOFOX_GAIN_2X																													
2	4X	NEOFOX_GAIN_4X																													
3	8X	NEOFOX_GAIN_8X																													
4	20X	NEOFOX_GAIN_20X																													
5	40X	NEOFOX_GAIN_40X																													
6	80X	NEOFOX_GAIN_80X																													
7	160X	NEOFOX_GAIN_160X																													
Stimulus LED Current	Uint	0<X<25000	143	516																											
NEOFOX_DAC_LED_CURRENT_BLUE	DeviceSetParameter		DeviceGetParameter																												
<p>The Reference LED Current parameter determines the brightness of the reference LED. The value 0 in this parameter corresponds to the lowest possible intensity and the value 25000 corresponds to the highest possible intensity.</p>																															
APD Gain	Uint	3500<X<9251	141	572																											
NEOFOX_DAC_APD_GAIN	DeviceSetParameter		DeviceGetParameter																												
<p>This parameter sets the open loop APD bias voltage gain, with 3500 representing the maximum possible gain and 9251 representing the minimum possible gain. Note that there is an inverse relationship between this parameter and the actual gain. To effectively use this parameter, the APD Voltage value should be monitored and the open loop gain should be adjusted as appropriate to achieve the target APD voltage. Users should be careful to constrain values to the appropriate range (never lower than 3500), as excessive gain levels may result in damage to the APD.</p>																															
APD Voltage	Uint	n/a	17	768																											
NEOFOX_ADC_APD_VOLTAGE	Read-Only		DeviceGetParameter																												
<p>This parameter allows the user to monitor the APD bias voltage. Higher reverse bias voltages on the APD correspond to higher “optical gain”, or conversion efficiency, from the detector. The tradeoff of higher APD bias voltage is an increased risk of detector saturation and higher levels of noise. This value is stored in the NeoFox with a fixed point representation and hence it must be scaled with the following formula in order to correctly deduce the actual APD voltage.</p> $\text{Floating Point APD Voltage Value (in Volts)} = \text{Raw Integer APD Value} / 2^{16}$																															
Ambient Pressure	Uint	n/a	15	780																											
NEOFOX_ADC_PRESSURE	Read-Only		DeviceGetParameter																												
<p>This parameter represents the ambient pressure sensor that is embedded in the NeoFox electronics. Because the NeoFox is only sensitive to the partial pressure of oxygen, this value is not used in the calculation of oxygen readings. It is also important to note that because the ambient pressure sensor is embedded in the electronics and not the probe itself, this sensor will not be sensitive to the pressure in the measurement environment if that environment is distinct from the environment of the electronics unit. Because this value is stored as a fixed point value, the following formula must be scaled with the following formula in order to deduce the floating point value of the ambient pressure.</p> $\text{Floating Point Ambient Pressure Value (in kPa)} = \text{Raw Integer Ambient Pressure Value} / 2^{16}$																															

NeoFox Communication Interfaces

Flash Write	n/a	n/a	93	n/a																																																			
NEOFOX_FLASH_WRITE	DeviceSetParameter		Write-Only																																																				
<p>This command will cause the variables in the Flash data structure to be written to flash memory. Those variables will then retain their state when the device has been turned back on. To invoke this command, write to the device as for any other parameter, using the NEOFOX_FLASH_WRITE code (93). The data value to be written is irrelevant – the action will happen when the code (93) is received.</p>																																																							
Flash Write	n/a	n/a	93	n/a																																																			
NEOFOX_FLASH_WRITE	DeviceSetParameter		Write-Only																																																				
<p>This command will cause the variables in the Flash data structure to be written to flash memory. Those variables will then retain their state when the device has been turned back on. To invoke this command, write to the device as for any other parameter, using the NEOFOX_FLASH_WRITE code (93). The data value to be written is irrelevant – the action will happen when the code (93) is received.</p>																																																							
FPGA Status	Uint	n/a	18	804																																																			
NEOFOX_FPGA_STATUS	Read-Only		DeviceGetParameter																																																				
<p>The FPGA status register contains various status indicators from the onboard FPGA. However, the only information that will be relevant to developers is the FPGA version code, which is contained in bits 4 through 7. In the NeoFox Viewer, the FPGA firmware version is displayed as the last two digits of the “firmware” display on the “status” panel.</p> <table border="1"> <thead> <tr> <th>Bit Number</th> <th>Name</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>15</td> <td>Fifo Sensor Full</td> <td>Reserved – do not alter or depend on this value</td> </tr> <tr> <td>14</td> <td>Fifo Sensor Empty</td> <td>Reserved – do not alter or depend on this value</td> </tr> <tr> <td>13</td> <td>Fifo Sensor Full</td> <td>Reserved – do not alter or depend on this value</td> </tr> <tr> <td>12</td> <td>Fifo CPU Empty</td> <td>Reserved – do not alter or depend on this value</td> </tr> <tr> <td>11</td> <td>Software Reset</td> <td>Reserved – do not alter or depend on this value</td> </tr> <tr> <td>10</td> <td>0</td> <td>Reserved – do not alter or depend on this value</td> </tr> <tr> <td>9</td> <td>Clip Hi</td> <td>Reserved – do not alter or depend on this value</td> </tr> <tr> <td>8</td> <td>Clip Lo</td> <td>Reserved – do not alter or depend on this value</td> </tr> <tr> <td>7</td> <td>FPGA Version Number (3)</td> <td>FPGA version code bit 3</td> </tr> <tr> <td>6</td> <td>FPGA Version Number (2)</td> <td>FPGA version code bit 2</td> </tr> <tr> <td>5</td> <td>FPGA Version Number (1)</td> <td>FPGA version code bit 1</td> </tr> <tr> <td>4</td> <td>FPGA Version Number (0)</td> <td>FPGA version code bit 0</td> </tr> <tr> <td>3</td> <td>0</td> <td>Reserved – do not alter or depend on this value</td> </tr> <tr> <td>2</td> <td>FifoCPU Available</td> <td>Reserved – do not alter or depend on this value</td> </tr> <tr> <td>1</td> <td>FifoCPU Prog Empty</td> <td>Reserved – do not alter or depend on this value</td> </tr> <tr> <td>0</td> <td>FifoCPU Prog Full</td> <td>Reserved – do not alter or depend on this value</td> </tr> </tbody> </table>					Bit Number	Name	Description	15	Fifo Sensor Full	Reserved – do not alter or depend on this value	14	Fifo Sensor Empty	Reserved – do not alter or depend on this value	13	Fifo Sensor Full	Reserved – do not alter or depend on this value	12	Fifo CPU Empty	Reserved – do not alter or depend on this value	11	Software Reset	Reserved – do not alter or depend on this value	10	0	Reserved – do not alter or depend on this value	9	Clip Hi	Reserved – do not alter or depend on this value	8	Clip Lo	Reserved – do not alter or depend on this value	7	FPGA Version Number (3)	FPGA version code bit 3	6	FPGA Version Number (2)	FPGA version code bit 2	5	FPGA Version Number (1)	FPGA version code bit 1	4	FPGA Version Number (0)	FPGA version code bit 0	3	0	Reserved – do not alter or depend on this value	2	FifoCPU Available	Reserved – do not alter or depend on this value	1	FifoCPU Prog Empty	Reserved – do not alter or depend on this value	0	FifoCPU Prog Full	Reserved – do not alter or depend on this value
Bit Number	Name	Description																																																					
15	Fifo Sensor Full	Reserved – do not alter or depend on this value																																																					
14	Fifo Sensor Empty	Reserved – do not alter or depend on this value																																																					
13	Fifo Sensor Full	Reserved – do not alter or depend on this value																																																					
12	Fifo CPU Empty	Reserved – do not alter or depend on this value																																																					
11	Software Reset	Reserved – do not alter or depend on this value																																																					
10	0	Reserved – do not alter or depend on this value																																																					
9	Clip Hi	Reserved – do not alter or depend on this value																																																					
8	Clip Lo	Reserved – do not alter or depend on this value																																																					
7	FPGA Version Number (3)	FPGA version code bit 3																																																					
6	FPGA Version Number (2)	FPGA version code bit 2																																																					
5	FPGA Version Number (1)	FPGA version code bit 1																																																					
4	FPGA Version Number (0)	FPGA version code bit 0																																																					
3	0	Reserved – do not alter or depend on this value																																																					
2	FifoCPU Available	Reserved – do not alter or depend on this value																																																					
1	FifoCPU Prog Empty	Reserved – do not alter or depend on this value																																																					
0	FifoCPU Prog Full	Reserved – do not alter or depend on this value																																																					
Firmware Version Hi	Char	n/a	2	12																																																			
NEOFOX_FIRMWARE_VER	Read-Only		DeviceGetParameter																																																				
<p>The microcontroller firmware version is stored in two distinct bytes – the Firmware Version Hi value and the Firmware Version Lo value. Concatenated together, these produce the firmware version. Usually, the firmware version is expressed as a hex short, such as 0x0208. This is how the firmware version is displayed in the NeoFox Viewer.</p> <p>It is important to note the following. A call to the DeviceGetParameter function with code 2 (NEOFOX_FIRMWARE_VER) will return an integer whose 2 lowest order bytes represent the Firmware Version Hi and Firmware Version Lo values. There is no way to query these two bytes independently through the DLL.</p>																																																							

Firmware Version Lo	Char	n/a	2	13															
NEOFOX_FIRMWARE_VER	Read-Only		DeviceGetParameter																
<p>The microcontroller firmware version is stored in two distinct bytes – the Firmware Version Hi value and the Firmware Version Lo value. Concatenated together, these produce the firmware version. Usually, the firmware version is expressed as a hex short, such as 0x0208. This is how the firmware version is displayed in the NeoFox Viewer.</p> <p>It is important to note the following. A call to the DeviceGetParameter function with code 2 (NEOFOX_FIRMWARE_VER) will return an integer whose 2 lowest order bytes represent the Firmware Version Hi and Firmware Version Lo values. There is no way to query these two bytes independently through the DLL.</p>																			
Millisecond Count	Uint	Enum	163	171															
NEOFOX_UP_TIME	Read-Only		DeviceGetParameter																
<p>This value represents the number of milliseconds that have elapsed since the device was powered on for a given sample. It can be used to verify that data has been refreshed or stored to provide a “time stamp” for each sample. It will “roll over” after approximately 50 days of continuous use.</p>																			
Calibration Method	Uint	Enum	163	171															
NEOFOX_CAL_METHOD	DeviceSetParameter		DeviceGetParameter																
<p>This parameter selects the calibration method that will be used by the NeoFox in order to calculate oxygen from its phase measurement. The two point method uses the Two Point Slope, Two Point Offset, and Two Point Tau0 parameters to calculate oxygen. The Multi Point method uses 12 coefficients (A0...T2) to calculate oxygen. The Single Point option is essentially the same as the Multi Point method, except that it uses a copy of the Multi Point coefficients that have been modified through a single point reset. More information about these calibration methods is available in the user guide and the calibration methods app note. It is strongly recommended that users use the Single Point option instead of the Multi Point option.</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Description</th> <th>DLL Constant</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>No calibration is used. Oxygen values are invalid.</td> <td>NEOFOX_CALMETHOD_NONE</td> </tr> <tr> <td>1</td> <td>Uses the two point calibration method</td> <td>NEOFOX_CALMETHOD_TWOPT</td> </tr> <tr> <td>2</td> <td>Uses the multipoint calibration method, without single point reset updates to the calibration coefficients</td> <td>NEOFOX_CALMETHOD_MULTIPT</td> </tr> <tr> <td>3</td> <td>Uses the multipoint calibration method, with single point reset updates to the calibration coefficients.</td> <td>NEOFOX_CALMETHOD_SINGLEPT</td> </tr> </tbody> </table>					Value	Description	DLL Constant	0	No calibration is used. Oxygen values are invalid.	NEOFOX_CALMETHOD_NONE	1	Uses the two point calibration method	NEOFOX_CALMETHOD_TWOPT	2	Uses the multipoint calibration method, without single point reset updates to the calibration coefficients	NEOFOX_CALMETHOD_MULTIPT	3	Uses the multipoint calibration method, with single point reset updates to the calibration coefficients.	NEOFOX_CALMETHOD_SINGLEPT
Value	Description	DLL Constant																	
0	No calibration is used. Oxygen values are invalid.	NEOFOX_CALMETHOD_NONE																	
1	Uses the two point calibration method	NEOFOX_CALMETHOD_TWOPT																	
2	Uses the multipoint calibration method, without single point reset updates to the calibration coefficients	NEOFOX_CALMETHOD_MULTIPT																	
3	Uses the multipoint calibration method, with single point reset updates to the calibration coefficients.	NEOFOX_CALMETHOD_SINGLEPT																	
Two Point Slope	Float	n/a	174	196															
NEOFOX_CAL_2PT_SLOPE	DeviceSetParameterFloat		DeviceGetParameterFloat																
<p>This is a two point calibration parameter. See the calibration methods app note and the user guide for more information about this parameter.</p>																			
Two Point Offset	Float	n/a	175	200															
NEOFOX_CAL_2PT_OFFSET	DeviceSetParameterFloat		DeviceGetParameterFloat																
<p>This is a two point calibration parameter. See the calibration methods app note and the user guide for more information about this parameter.</p>																			

Two Point Tau0	Float	n/a	170	180
NEOFOX_CAL_2PT_TAU_0	DeviceSetParameterFloat		DeviceGetParameterFloat	
This is a two point calibration parameter. See the calibration methods app note and the user guide for more information about this parameter.				
Multi Point Orig A0	Float	n/a	200	208
NEOFOX_CAL_MULTI_A0	DeviceSetParameterFloat		DeviceGetParameterFloat	
This is a Multi Point calibration parameter. See the calibration methods app note and the user guide for more information about this parameter.				
Multi Point Orig A1	Float	n/a	201	212
NEOFOX_CAL_MULTI_A1	DeviceSetParameterFloat		DeviceGetParameterFloat	
This is a Multi Point calibration parameter. See the calibration methods app note and the user guide for more information about this parameter.				
Multi Point Orig A2	Float	n/a	202	216
NEOFOX_CAL_MULTI_A2	DeviceSetParameterFloat		DeviceGetParameterFloat	
This is a Multi Point calibration parameter. See the calibration methods app note and the user guide for more information about this parameter.				
Multi Point Orig B0	Float	n/a	203	220
NEOFOX_CAL_MULTI_B0	DeviceSetParameterFloat		DeviceGetParameterFloat	
This is a Multi Point calibration parameter. See the calibration methods app note and the user guide for more information about this parameter.				
Multi Point Orig B1	Float	n/a	204	224
NEOFOX_CAL_MULTI_B1	DeviceSetParameterFloat		DeviceGetParameterFloat	
This is a Multi Point calibration parameter. See the calibration methods app note and the user guide for more information about this parameter.				
Multi Point Orig B2	Float	n/a	205	228
NEOFOX_CAL_MULTI_B2	DeviceSetParameterFloat		DeviceGetParameterFloat	
This is a Multi Point calibration parameter. See the calibration methods app note and the user guide for more information about this parameter.				
Multi Point Orig C0	Float	n/a	206	232
NEOFOX_CAL_MULTI_C0	DeviceSetParameterFloat		DeviceGetParameterFloat	
This is a Multi Point calibration parameter. See the calibration methods app note and the user guide for more information about this parameter.				
Multi Point Orig C1	Float	n/a	207	236
NEOFOX_CAL_MULTI_C1	DeviceSetParameterFloat		DeviceGetParameterFloat	
This is a Multi Point calibration parameter. See the calibration methods app note and the user guide for more information about this parameter.				

Multi Point Orig C2	Float	n/a	208	240
NEOFOX_CAL_MULTI_C2	DeviceSetParameterFloat		DeviceGetParameterFloat	
This is a Multi Point calibration parameter. See the calibration methods app note and the user guide for more information about this parameter.				
Multi Point Orig T0	Float	n/a	209	244
NEOFOX_CAL_MULTI_T0	DeviceSetParameterFloat		DeviceGetParameterFloat	
This is a Multi Point calibration parameter. See the calibration methods app note and the user guide for more information about this parameter.				
Multi Point Orig T1	Float	n/a	210	248
NEOFOX_CAL_MULTI_T1	DeviceSetParameterFloat		DeviceGetParameterFloat	
This is a Multi Point calibration parameter. See the calibration methods app note and the user guide for more information about this parameter.				
Multi Point Orig T2	Float	n/a	211	252
NEOFOX_CAL_MULTI_T2	DeviceSetParameterFloat		DeviceGetParameterFloat	
This is a Multi Point calibration parameter. See the calibration methods app note and the user guide for more information about this parameter.				
Multi Point SP A0	Float	n/a		256
n/a	n/a		n/a	
This is a Single Point calibration parameter. It is a copy of its counterpart from the multi-point calibration parameter that is copied during a single point reset. It cannot be read or set through DLL function calls. It also cannot be set through the serial protocol, but it can be read. See the calibration methods app note and the user guide for more information about this parameter.				
Multi Point SP A1	Float	n/a		260
n/a	n/a		n/a	
This is a Single Point calibration parameter. It is a copy of its counterpart from the multi-point calibration parameter that is copied during a single point reset. It cannot be read or set through DLL function calls. It also cannot be set through the serial protocol, but it can be read. See the calibration methods app note and the user guide for more information about this parameter.				
Multi Point Orig C1	Float	n/a	207	236
NEOFOX_CAL_MULTI_C1	DeviceSetParameterFloat		DeviceGetParameterFloat	
This is a Multi Point calibration parameter. See the calibration methods app note and the user guide for more information about this parameter.				
Multi Point Orig C2	Float	n/a	208	240
NEOFOX_CAL_MULTI_C2	DeviceSetParameterFloat		DeviceGetParameterFloat	
This is a Multi Point calibration parameter. See the calibration methods app note and the user guide for more information about this parameter.				

Multi Point Orig T0	Float	n/a	209	244
NEOFOX_CAL_MULTI_T0	DeviceSetParameterFloat		DeviceGetParameterFloat	
<p>This is a Multi Point calibration parameter. See the calibration methods app note and the user guide for more information about this parameter.</p>				
Multi Point OrigT1	Float	n/a	210	248
NEOFOX_CAL_MULTI_T1	DeviceSetParameterFloat		DeviceGetParameterFloat	
<p>This is a Multi Point calibration parameter. See the calibration methods app note and the user guide for more information about this parameter.</p>				
Multi Point Orig T2	Float	n/a	211	252
NEOFOX_CAL_MULTI_T2	DeviceSetParameterFloat		DeviceGetParameterFloat	
<p>This is a Multi Point calibration parameter. See the calibration methods app note and the user guide for more information about this parameter.</p>				
Multi Point SP A0	Float	n/a		256
n/a	n/a		n/a	
<p>This is a Single Point calibration parameter. It is a copy of its counterpart from the multi-point calibration parameter that is copied during a single point reset. It cannot be read or set through DLL function calls. It also cannot be set through the serial protocol, but it can be read. See the calibration methods app note and the user guide for more information about this parameter.</p>				
Multi Point SP A1	Float	n/a		260
n/a	n/a		n/a	
<p>This is a Single Point calibration parameter. It is a copy of its counterpart from the multi-point calibration parameter that is copied during a single point reset. It cannot be read or set through DLL function calls. It also cannot be set through the serial protocol, but it can be read. See the calibration methods app note and the user guide for more information about this parameter.</p>				
Multi Point SP A2	Float	n/a		264
n/a	n/a		n/a	
<p>This is a Single Point calibration parameter. It is a copy of its counterpart from the multi-point calibration parameter that is copied during a single point reset. It cannot be read or set through DLL function calls. It also cannot be set through the serial protocol, but it can be read. See the calibration methods app note and the user guide for more information about this parameter.</p>				
Multi Point SP B0	Float	n/a		268
n/a	n/a		n/a	
<p>This is a Single Point calibration parameter. It is a copy of its counterpart from the multi-point calibration parameter that is copied during a single point reset. It cannot be read or set through DLL function calls. It also cannot be set through the serial protocol, but it can be read. See the calibration methods app note and the user guide for more information about this parameter.</p>				

Multi Point SP B1	Float	n/a		272
n/a	n/a		n/a	
<p>This is a Single Point calibration parameter. It is a copy of its counterpart from the multi-point calibration parameter that is copied during a single point reset. It cannot be read or set through DLL function calls. It also cannot be set through the serial protocol, but it can be read. See the calibration methods app note and the user guide for more information about this parameter.</p>				
Multi Point SP B2	Float	n/a		276
n/a	n/a		n/a	
<p>This is a Single Point calibration parameter. It is a copy of its counterpart from the multi-point calibration parameter that is copied during a single point reset. It cannot be read or set through DLL function calls. It also cannot be set through the serial protocol, but it can be read. See the calibration methods app note and the user guide for more information about this parameter.</p>				
Multi Point SP C0	Float	n/a		280
n/a	n/a		n/a	
<p>This is a Single Point calibration parameter. It is a copy of its counterpart from the multi-point calibration parameter that is copied during a single point reset. It cannot be read or set through DLL function calls. It also cannot be set through the serial protocol, but it can be read. See the calibration methods app note and the user guide for more information about this parameter.</p>				
Multi Point SP C1	Float	n/a		284
n/a	n/a		n/a	
<p>This is a Single Point calibration parameter. It is a copy of its counterpart from the multi-point calibration parameter that is copied during a single point reset. It cannot be read or set through DLL function calls. It also cannot be set through the serial protocol, but it can be read. See the calibration methods app note and the user guide for more information about this parameter.</p>				
Multi Point SP C2	Float	n/a		288
n/a	n/a		n/a	
<p>This is a Single Point calibration parameter. It is a copy of its counterpart from the multi-point calibration parameter that is copied during a single point reset. It cannot be read or set through DLL function calls. It also cannot be set through the serial protocol, but it can be read. See the calibration methods app note and the user guide for more information about this parameter.</p>				
Multi Point SP T0	Float	n/a		292
n/a	n/a		n/a	
<p>This is a Single Point calibration parameter. It is a copy of its counterpart from the multi-point calibration parameter that is copied during a single point reset. It cannot be read or set through DLL function calls. It also cannot be set through the serial protocol, but it can be read. See the calibration methods app note and the user guide for more information about this parameter.</p>				

Multi Point SP T1	Float	n/a		296
n/a	n/a		n/a	
<p>This is a Single Point calibration parameter. It is a copy of its counterpart from the multi-point calibration parameter that is copied during a single point reset. It cannot be read or set through DLL function calls. It also cannot be set through the serial protocol, but it can be read. See the calibration methods app note and the user guide for more information about this parameter.</p>				
Multi Point SP T2	Float	n/a		300
n/a	n/a		n/a	
<p>This is a Single Point calibration parameter. Unlike the other Single Point parameters, the value of this parameter is different than the value of its Multi Point counterpart. This value (the T2 parameter) is updated during a single point reset, and this is the only difference between the Multi Point calibration and the Single Point calibration parameters. Unfortunately, the value of this parameter cannot be read back through the DLL at this time. However, it can be read back through the NeoFox Viewer, in the calibration notes field. It can also be read back, but not set through the serial protocol.</p>				

Additional Firmware Controls in FW Version 2.25

As of firmware version 2.25, additional firmware control features are available for developers who wish to control the communications characteristics of the device. Those users who do not wish to invoke these features can simply use the default settings of the device as described above and the device will be completely backwards compatible.

Specifically, two firmware features have been added that will be of use to developers:

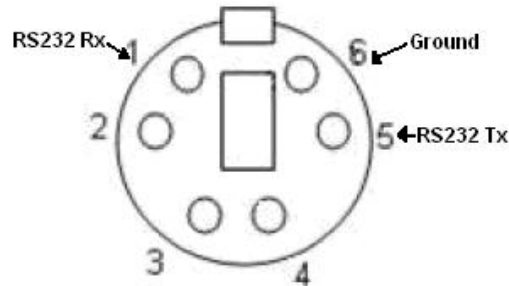
- RS232 Settings
- Data Copy Settings

RS-232 Overview

Only new hardware units (those whose serial number has an E as its third character) support RS232 communications. In these units, the RS232 uses the same serial communications protocol described earlier in this document.

To use the RS232 port on the NeoFox device, you must set the RS232 Enable parameter to '1'. This parameter is not currently supported by NeoFox Viewer software. Therefore, you must use the NeoFox Set Parameter utility program to set this parameter prior to interfacing with NeoFox's RS232 port. This program is available from the Ocean Optics [Software Downloads page](#). In this directory, there is a .bat file which has already been set up to set this parameter appropriately and save it to flash memory. Once you have run this program, RS232 communication will be enabled until you reset the flash memory.

RS-232 Pins



GPIO Connector (as viewed looking towards the back of the NeoFox unit with the graphic label down and the power jack to the left of the connector)

It is important to note that the device will disable RS232 communications in certain conditions:

- When the RS232 enable parameter is set to 0. It will also disable the RS232 port
- When the USB connector is plugged in (as detected by the presence of the 5V USB power line)

RS232 Parameters

RS232 Divisor Latch	short	$0 < X < 10000$	78	
NEOFOX_RS232_DIVISOR_LATCH	DeviceSetParameter		DeviceGetParameter	
Baud Rate = $12,000,000 / (16 * \text{Divisor Latch}) * (\text{Multiply Value} / [\text{Multiply Value} + \text{Divisor Add value}])$				
RS232 Divisor Add Value	unsigned char	$0 < X < 256$	79	
NEOFOX_RS232_DIVADDVAL	DeviceSetParameter		DeviceGetParameter	
Baud Rate = $12,000,000 / (16 * \text{Divisor Latch}) * (\text{Multiply Value} / [\text{Multiply Value} + \text{Divisor Add value}])$				
RS232 Multiply Value	unsigned char	$0 < X < 256$	80	
NEOFOX_RS232_MULVAL	DeviceSetParameter		DeviceGetParameter	
Baud Rate = $12,000,000 / (16 * \text{Divisor Latch}) * (\text{Multiply Value} / [\text{Multiply Value} + \text{Divisor Add value}])$				
RS232 Enable	unsigned char	0, 1	96	
NEOFOX_RS232_ENABLE	DeviceSetParameter		DeviceGetParameter	
This value determines whether or not RS232 communications will be enabled when the USB cable is not present. Its presence is determined by the presence of the 5V USB power line.				
	Value	Description		
	0	RS232 is disabled.		
	1	RS232 will be enabled when the USB connector is not plugged in		

Data Copy Settings

By default, the NeoFox’s “data dump” structure is big, and many host devices cannot handle a data stream of around 60,000 KBps. Therefore, settings have been developed that allow the user to reduce the size and frequency of the NeoFox’s data dump.

Specifically, there are two features which have been added:

- Data Copy Mode
- Data Copy Type

To understand what these features do, it is important to understand how the Neofox samples and transmits its data. By default, the process is as follows:

1. A sample is ready

Note: Every 100 ms, the controller calculates a new value of Tau, oxygen, and temperature. Internally, the actual sample update occurs whether or not the device is currently in the process of transmitting a previous sample.

2. Data is copied into the transmission buffer.

Note: If the device is not currently in the process of transmitting a previous sample, it copies its “data dump” of its parameters to the transmission buffer. If the device is currently transmitting, then data will not be copied until the transmission is complete. If a new sample becomes available before the current sample has begun transmitting, then the current sample is never transmitted and the new sample will take its place waiting for the end of the previous transmission.

3. Once data has been copied into the transmission buffer, it is immediately sent to the host via the UART

Note: This can be over RS232 or the USB, depending on which mode is currently selected.

Data Copy Mode

The first new feature Data Copy Mode simply allows you to determine when data should be copied into the transmission buffer. There are two options for this parameter:

- Auto – When the auto mode is enabled, the default behavior will occur, as listed above.
- Request – When the Request mode is enabled, the device will look at the current value of the Data Copy Trigger setting to determine whether or not to perform the sample copy. When the Data Copy Trigger value is 0, it will not perform this copy. When the value is 1, it will perform the copy and set the value of the Data Copy Trigger parameter to 0; it does not perform another copy until you request it to do so.

Data Copy Type

The Data Copy Type parameter can be used to reduce the size of the data dump. There are currently three data types available to users:

- Type 1: The original data structure format, as documented earlier in this document.
- Type 2: The original data structure format, as documented earlier in this document, but without the two waveform structures included at the end of the structure.
- Type 3: A measurement only structure, which includes a very small subset of the information available in type 1; specifically, it only includes measurement data.

Type 3 Structure Format

Addr	Type	Name	Description
0	uchar	Stx	Start of transmission character. Set to 0x03.
1	uchar	PacketType	Set to 0xDC
2	ushort	FrameSize	Set to 5036 (0x13AC)
4	uchar	FrameCount	Number of frames uploaded since the device powered on. This rolls over after 255. It can be used to determine whether any frames have been missed.
5	uchar	ProtocolRev	Set to 0x01. There is currently only one version of the protocol. However, future versions may change and this byte will indicate that change. If this value is not 0x01, this structure is not valid.
6	uchar	Reserved1	Reserved. Do not depend on this value.
7	uchar	Reserved2	Reserved. Do not depend on this value.
8	ulong	MillisecondCount	Millisecond Count
12	float	fOxygenConverted	Converted Oxygen
16	ulong	uiOxygenUnits	Oxygen Units
20	float	fTau	Tau
24	float	fSelectedTemperature	The sensor temperature (converted to a float, in °C or the fixed temperature depending on the value of the temperature source.
28	uchar	Reserved3	Reserved. Do not depend on this value.
29	uchar	Reserved4	Reserved. Do not depend on this value.
30	uchar	Checksum	Set to the aggregate sum of all previous bytes modulus 256.
31	uchar	Eof	End of transmission character. Set to 0x04

Data Copy Parameters

Uart Data Copy Mode	Unsigned char	0,1	88	
<p>Determines whether new samples should be copied to the Uart automatically or whether the device should wait for a request from the host before copying the next sample. When the value of this parameter is 0, the device will be in Auto mode and samples will be copied automatically. When the value of this parameter is 1, the device will be in request mode, meaning that samples will only be copied when requested by the host.</p>				
Uart Data Copy Trigger	Unsigned char	0,1	84	
<p>When the value of the Data Copy Mode parameter is 1 (Request Mode), and the value of this parameter is 1, the next sample available will be copied to the transmission buffer and sent through the Uart. This parameter will immediately be reset to 0.</p>				
Uart Data Copy Type	Unsigned char	1,2,3	87	
<p>Specifies the format of the data structure that is copied into the transmission buffer for transmission to the host through the Uart.</p>				

Single Point Reset Parameters

A single point reset works as follows:

1. The user puts the temperature sensor and oxygen sensor into an environment with a known oxygen partial pressure.
2. The user waits for the measurements to stabilize (the longer, the better).
3. The three measurement conditions are written to the device:
 - Single Point Tau – the current value of Tau that is read by the sensor in the SPR environment.
 - Single Point Temperature – the current value of temperature that is read by the sensor in the SPR environment.
 - Single Point Oxygen – The known partial pressure of oxygen (expressed as percent of 1 ATM) in the SPR environment. In ambient air, at sea level, this is 20.9%.
4. The Single Point Calculate command is issued to the device. This instructs the device to perform the single point reset, which recalculates the single point coefficients from the original multipoint coefficients and puts the device into single point calibration mode.
5. (Optional) Some users may want to issue a flash write command as well, in order to save the new SPR coefficients to memory.

Single Point Temperature	float	$X \leq 200$	188	
Temperature (in degrees C) to use for next SPR calculation. Most users will want to set this to the current value of Temperature read by the instrument while in the SPR environment.				
Single Point Tau	float	$X \leq 10.0$	186	
Tau (in microseconds) to use for next SPR calculation. Most users will want to set this to the current value of Tau read by the instrument while in the SPR environment.				
Single Point Oxygen	float	$0 \leq X$	187	
Percent Oxygen (in percent of 1 ATM) to use for next SPR calculation. Most users will want to set this to the known value of percent oxygen in the SPR environment. In ambient air, at sea level, this will be equal to 20.9.				
Single Point Calculate	N/A	N/A	189	
Writing any value to this parameter will result in a single point reset with the single point reset parameter values listed above. The new single point coefficients will be calculated and the device will be put into single point mode (as opposed to multipoint mode). The new value will not necessarily be set in flash memory until the user manually issues a flash write command.				