OCT Engine Communications Command Specification

Axsun OCT Control Interface

1. Introduction

This document describes the interface of the Axsun OCT control. The Axsun OCT control is a windows C# control used for communication to the Axsun OCT engine. It is in the form of a single .dll called AxsunOCTControl.dll. It needs no other controls to be installed in the system besides the Microsoft .NET framework. It communicates to the OCT device over USB.

2. Communication Protocol

The Axsun OCT control has several public methods to communicate as well as events to notify the host program that the OCT USB device has either connected or disconnected.

The OCT device will only send back data in response to a command.

3. Events

The OCT control has windows events to signal when the device is connected or disconnected. These events may or may not be used to communicate with the OCT device

OCTDeviceConnectedEvent

The OCTDeviceConnectedEvent Event is called when a device is plugged into the USB or is powered on.

The event is set up to be called from a C# program like this:

```
// Create New OCT Control
AOCT = new AxsunOCTControl.AxsunOCTControl();

// Call Initialize to look for OCT USB driver and check if there is a
// device attached
AOCT.Initialize();

// Connect up the OCTDeviceConnectedEvent to a function we have locally
// called AOCT_OCTDeviceConnectedEvent
AOCT.OCTDeviceConnectedEvent += new
AxsunOCTControl.AxsunOCTControl.OCTDeviceConnectedHandler(AOCT_OCTDeviceConnectedEvent);

// Now that the event is connected, every time an OCT device is plugged
// in, the following function will be called.
// In the OCT host program, this function queries and displays the
// firmware version in the status bar

void AOCT_OCTDeviceConnectedEvent()
{
      ...
```

OCTDeviceDisconnectedEvent

The OCTDeviceDisconnectedEvent is called whenever a USB device is disconnected from the system. Usage of this event is similar to the OCTDeviceConnectedEvent above. The line to add the event handler in C# is:

```
AOCT.OCTDeviceDisconnectedEvent += new
AxsunOCTControl.AxsunOCTControl.OCTDeviceDisconnectedHandler(AOCT_OCTDeviceDisconnectedEv
ent);
```

4. Fields (Numerical Defines)

The OCT control makes public a list of numerical defines for use when programming.

USB Error Codes

The USBErrorCodes list has a list of error codes returned from the OCT control that deal with USB communication errors. There is no list of error codes for the device itself. If there is an error on the OCT device an error code will be returned that is less than 1000. The calling program can then query the OCT device for the string that defines that error code by calling GetErrorString(UInt32 ErrorCode).

For any errors that involve USB communications, however, querying the device is probably not possible and so the list of errors are provided here.

```
public const int ERROR_READING = 1000;
public const int ERROR_WRITING = 1001;
public const int NO_DEVICE_PRESENT = 1002;
public const int CHECKSUM_ERROR = 1003;
public const int TOO_MANY_PACKET_TRIES = 1004;
```

5. Fields (Numerical Defines)

The OCT control makes public a list of numerical defines for use when programming.

CommandIDs

The CommandID list has defines for all the commands used by the OCT device.

```
        public const UInt32 GET_FW_VERSION = 100;
        public const UInt32 LOAD_DAC = 101;
        public const UInt32 RETURN_DAC = 102;
        public const UInt32 START_SWEEP = 103;
        public const UInt32 END_SWEEP = 104;
        public const UInt32 LOAD_FIRMWARE = 105;
        public const UInt32 GET_BUILD_DATE = 106;
        public const UInt32 LOAD_SCAN_SETTINGS = 107;
        public const UInt32 GET_SCAN_SETTINGS = 108;
        public const UInt32 GET_ERROR_STRING = 109;
        public const UInt32 GET_LOW_SPEED_AD = 110;
        public const UInt32 WRITE_TO_DIAGNOSTIC = 111;
        public const UInt32 GET_DIAGNOSTIC = 112;
        public const UInt32 GET_USB_DATA = 113;
        public const UInt32 SET_TIMER_ENABLE = 114;
        public const UInt32 GET_TEC_STATE = 115;
        public const UInt32 RESET_TEC = 116;
        public const UInt32 SET_SERIAL_NUM = 117;
        public const UInt32 GET_SERIAL_NUM = 118;
        public const UInt32 PARK_FILTER = 119;
        public const UInt32 GET_BOOT_ERRORS = 120;
        public const UInt32 GET_SYSTEM_STATUS = 121;
```

```
public const UInt32 SW_RESET = 122;
public const UInt32 GET_SYSTEM_TYPE = 123;
public const UInt32 SET_LOW_SPEED_AD_POLLING = 124;
public const UInt32 ARE_OTHER_BOARDS_ATTACHED = 125;
public const UInt32 GET_NUMERICAL_FV_VERSION = 126;
public const UInt32 GET_DEBUG_TEXT = 127;
public const UInt32 CLEAR_DEBUG_TEXT = 128;
public const UInt32 GET_PIM_SETTINGS = 131;
public const UInt32 SET_PIM_SETTINGS = 132;


public const UInt32 LOAD_FPGA = 201;
public const UInt32 RESET_FPGA = 202;
public const UInt32 GET_FPGA_SETTINGS = 203;
public const UInt32 SET_FPGA_SETTINGS = 204;
public const UInt32 GET_FPGA_IMAGE = 205;
public const UInt32 JUST_PROGRAM_FPGA = 206;
public const UInt32 GET_FPGA_REGISTERS = 207;
public const UInt32 SET_FPGA_REGISTER = 208;
public const UInt32 FPGA_ADC_ENABLE = 209;
public const UInt32 LOAD_FPGA_MULTISECTION = 210;
public const UInt32 HEARTBEAT = 300;
public const UInt32 HOME_VDL = 301;
public const UInt32 MOVE_ABS_VDL = 302;
public const UInt32 MOVE_REL_VDL = 303;
public const UInt32 GET_VDL_STATUS = 304;
```

USB Error Codes

The USBErrorCodes list has a list of error codes returned from the OCT control that deal with USB communication errors. There is no list of error codes for the device itself. If there is an error on the OCT device an error code will be returned that is less than 1000. The calling program can then query the OCT device for the string that defines that error code by calling GetErrorString(UInt32 ErrorCode).

For any errors that involve USB communications, however, querying the device is probably not possible and so the list of errors are provided here.

```
public const int ERROR_READING = 1000;
public const int ERROR_WRITING = 1001;
public const int NO_DEVICE_PRESENT = 1002;
public const int CHECKSUM_ERROR = 1003;
public const int TOO_MANY_PACKET_TRIES = 1004;
```

6. Generic Methods

These are commands than are available for use on all OCT devices.

bool CheckIfOCTDevicePresent()

This function is obsolete. It is included for backward compatibility. Use IsOCTDeviceConnected() instead.

string GetControlBuildDate()

Returns a string with the build date of the OCT control. This function does not communicate with the OCT device and the OCT device does not need to be connected for this function to return its value.

string GetControlName()

Returns the product name of the control.

string GetControlVersion()

Returns the version of just the OCT control, not the firmware version.

string GetErrorString()

Returns a string to describe the error matching the ErrorCode value. Error numbers and description strings are only stored onboard the OCT device. This was designed so that changes can be made to just the OCT device firmware including adding error codes without needing to change the host program or OCT control program. \

Parameters:

UInt32	Error Code	

void Initialize()

Obsolete. This command has no functionality. Included for backward compatibility.

void GetLastCommandInfo()

Returns diagnostic information about the last command sent to the host

Parameters:

ref int	BytesSent	
ref int	PacketsSent	
ref int	PacketsRetried	
ref int	Timeouts	
ref int	ChecksumErrors	
ref int	TimeTaken	
ref int	DataBytes	

UInt32 RebootSystem()

Reboots the OCT. Useful after a firmware download. No Parameters are passed.

Returns an error code or 0 if there was no error.

UInt32 GetFirmwareVersion()

Returns a string containing the firmware version and the CPU version.

Parameters:

Ref String	Firmware Version String	

Returns an error code or 0 if there was no error.

UInt32 GetFirmwareBuildDate()

Returns the build date of the firmware.

Parameters:

Ref String	Firmware Build Date	

Returns an error code or 0 if there was no error.

UInt32 GetSystemType()

Gets the type of the currently connected board.

Type definitions are exposed in the enum OCT_SYSTEM_TYPES:

```
public const UInt32 SYSTEM_TYPE_NOT_CONNECTED = 0;
public const UInt32 SYSTEM_TYPE_OCT_LASER = 40;
public const UInt32 SYSTEM_TYPE_OCT_CAMERALINK = 41;
public const UInt32 SYSTEM_TYPE_OCT_PIM = 42;
```

Parameters:

Ref UInt32	SystemType	Number indicating system type
Ref string	SystemTypeString	String containing the system type

Returns an error code or 0 if there was no error

UInt32 GetSerialNum()

Gets the current serial number.

Parameters:

Ref String	Serial Number	Serial Number of up to 40 characters

Returns an error code or 0 if there was no error.

Bool IsOCTDeviceConnected ()

Checks if a device is connected.

Returns true if connected, false if not.

Int GetNumberOfOCTDevicesPresent ()

Returns the number of OCT devices that are connected to the system. (Plugged in and powered on)

Bool ConnectToOCTDevice ()

Connects to a specific OCT device. Number must be between 0 and the number returned from GetNumberOfOCTDevicesPresent() above.

Parameters:

int	DeviceNum	

Returns an error code or 0 if there was no error.

UInt32 GetBootErrors()

Gets a list of any errors that occurred during system startup

Parameters:

Ref UInt32[]	Boot Errors	Array of errors. Array size must be at least 50.
Ref Uint32	Num Boot Errors	Number of boot errors that occurred

Returns an error code or 0 if there was no error.

UInt32 GetUSBInfo()

Gets an array containg USB Information.

Parameters:

Ref UInt32[]	Status Array of size 5	PacketsRead - number of packets since power on ChecksumErrors - number of checksum errors since power on PacketErrors - number of packet errors since power on TimeoutErrors- number of timeout errors since power on CommandsReceived - number of commands recieved since power on ConnectCount - number of connections made since power on
Ref Uint32	Words Returned	Size of USB status array

Returns an error code or 0 if there was no error.

UInt32 GetPartNum()

Gets the current part number.

Parameters:

Ref String	Serial Number	Serial Number of up to 40 characters

Returns an error code or 0 if there was no error.

UInt32 GetManufacturer()

Gets the manufacturer. Always returns “Axsun”.

Parameters:

Ref String	Serial Number	String containing “Axsun”

Returns an error code or 0 if there was no error.

UInt32 GetNumericalFirmwareVersion()

Gets the firmware version as a set of integers. If the version is “V 1.17.15” then V1 = 1, V2 = 17 and V3 = 15.

Parameters:

Ref UInt32	V1	First version digit
Ref UInt32	V2	Second version digit
Ref UInt32	V3	Third version digit

Returns an error code or 0 if there was no error.

7. Engine Specific Methods

These are commands that are available only on OCT laser engines.

UInt32 StartScan()

Start the OCT Engine Scanning. No Parameters are passed.

Returns an error code or 0 if there was no error.

UInt32 StopScan()

Stops the OCT Engine Scanning. No Parameters are passed.

Returns an error code or 0 if there was no error.

UInt32 GetSystemStatus()

Gets an array containing system Status.

Parameters:

Ref UInt32[]	Status Array of size 20	Uptime - ms system has been powered on Sweeping - 1 = sweeping, 0 = not sweeping DMAError - a DMA error has occurred PPIError - a PPI error has occurred TECError - a TEC error has occurred PPIErrors - number of PPI errors since power on Boot Reg 1 – Reason for last Reboot Boot Reg 2– Reason for last Reboot
Ref Uint32	Status Words	Size of status array

Returns an error code or 0 if there was no error.

UInt32 GetTECStatus()

Gets debug information about the TEC. The most current information is returned in the data fields. History data for the last 50 seconds is also returned in the data field.

If a TEC error occurs the TEC is turned off and history data stops collecting.

Parameters:

Ref Uint32	TEC State	Current TEC State 0 TEC_UNINT 1 TEC_WARMING_UP 2 TEC_WAITING_IN_RANGE 3 TEC_READY 4 TEC_OFF_ERROR
Ref Uint32	Counts in range	Number of seconds that the TEC has been in range
Ref Uint32	Counts Warming up	Number of seconds the TEC has been warming up for.
Ref Uint32	Counts out of range	Number of seconds the TEC has been out of range after coming in range.
Ref Uint32	TEC Error	If the TEC was shut down, the reason it was shut down. 16 TEC_NEVER_GOT_TO_READY 17 TEC_WENT_OUT_OF_RANGE The string for this error code can be returned by using the GetErrorString command (109)
Ref int	TEC Temp	Current temperature. This temperature is the float temperature multiplied by 100 and returned as an integer. So, a temperature of 24.174 will be returned as an integer 2417.
Ref Uint32	Points Returned	The number of data points returned in the following 3 arrays. Right now the system

		returns 50.
Ref int[]	TEC Temps	Temperature as an integer of the temperature multiplied by 100.
Ref Uint32[]	TEC Times	Time in milliseconds as a 32 bit integer
Ref Uint32[]	TEC States	TEC state as a 32 bit integer.

Returns an error code or 0 if there was no error.

UInt32 ResetTEC()

Resets the TEC. Turns the TEC off, sets the temperature with a D-A write, turns it on, and resets any error code and all the TEC counters. No Parameters are passed.

Returns an error code or 0 if there was no error.

UInt32 GetLowSpeedADData()

Returns the 16 channels of low speed A→D data. The data is returned as 3 arrays of data.

Parameters:

Ref UInt32[]	RawVals	Raw integer data in 16 bit integers
Ref Single[]	ScaledVals	Converted values in 32 bit floats
Ref String[]	FieldNames	Strings to label each channel
Ref UInt32	ChannelsReturned	Number of channels of data returned

Using this technique, the firmware can add or delete A→D channels without needing new host software.

Currently, the channels returned are:

0	Reference Monitor
1	Signal Monitor
2	Pointer laser current
3	Filter Voltage
4	Laser Current
5	Laser TEC thermistor
6	Laser TEC current
7	Laser TEC voltage
8	Board temperature
9	Ground
10	Spare
11	3.3V reference
12	-5V
13	+5V
14	15V
15	150V

Returns an error code or 0 if there was no error.

UInt32 SwitchConfiguration()

Switches the DAC table to another configuration. Configuration numbers are numbered from 0 to the number of loaded configurations - 1. If 3 configurations are loaded, valid values are from 0 to 2. If a configuration is selected that is not yet set up, the error CONFIGURATION_NOT_SETUP (140) will be returned.

Parameters:

int	ConfigNumber	Different configuration to switch to.

Returns an error code or 0 if there was no error.

UInt32 GetConfiguration ()

Returns the number of the current DAC table configuration. Configuration numbers are numbered from 0 to the number of loaded configurations - 1.

Parameters:

Ref int	ConfigNumber	Which configuration is currently active.

Returns an error code or 0 if there was no error.

UInt32 GetLoadedConfiguration()

Returns the number of DAC table configurations that have been loaded into this device. This number can range from 1 to the number of loaded configurations.

Parameters:

Ref int	LoadedConfigurations	The number of configurations that have been loaded to this device.

Returns an error code or 0 if there was no error.

UInt32 GetSystemErrorStates()

Returns a class containing error states for individual voltages and TECs as well as an overall "System is in Error" state that is active whenever there is at least 1 error. Individual error states are booleans (True = error on that channel, false = no error on that channel)

Parameters:

Ref SystemErrorStateType	SystemErrorStates	Class containing Booleans for each individual system error state as defined below.

```
class SystemErrorStateType
{
    bool SystemIsInError;           Overall Error State
```

```
        bool BPowerMonitorInError;        Power Monitor Channel
        bool B33RefInError;               3.3V Reference Channel
        bool BMinus5VInError;             -5V Channel
        bool B5VInError;                  5V Channel
        bool B15VInError;                 15V Channel
        bool B180VInError;                180V Channel
        bool TEC0Error;                   TEC 1 Channel
        bool TEC1Error;                   TEC 2 Channel
        bool BoardTempError;              Board Temperature
    }
```

Returns an error code or 0 if there was no error.

UInt32 ResetSystemErrorState()

This will reset any error state including a TEC error. If a TEC is in an error state it will be re-started as if it was powered on.

Parameters:
None

Returns an error code or 0 if there was no error.

UInt32 SourceCurrentOn()

Turns on the light source current.

Parameters:
None
Returns an error code or 0 if there was no error.

UInt32 SourceCurrentOff()

Turns on the light source current.

Parameters:
None
Returns an error code or 0 if there was no error.

UInt32 GetLaserOnTime()

Returns the number of seconds that the laser has been in use for the lifetime of the laser on this system.

Parameters:

Ref UInt32	LaserOnTimeSeconds	Number of seconds that the laser has been on.

Returns an error code or 0 if there was no error.

UInt32 GetSystemOnTime()

Returns the number of seconds that the system has been powered on for the lifetime of this device.

Parameters:

Ref UInt32	SystemOnTimeSeconds	Number of seconds that the device has been powered on.

Returns an error code or 0 if there was no error.

UInt32 GetWavelengthAtSweepTrigger()

Returns the wavelength at the sweep trigger for this system in nm. If multiple DAC tables are loaded, this will return the wavelength at the sweep trigger for the currently selected DAC table.

Parameters:

Ref double	Wavelength	Wavelength at sweep trigger in nm for this system

Returns an error code or 0 if there was no error.

UInt32 GetWavelengthAtEndTrigger()

Returns the wavelength at the end trigger for this system in nm.

Parameters:

Ref double	Wavelength	Wavelength at end trigger in nm for this system

Returns an error code or 0 if there was no error.

UInt32 GetKClockFSR ()

Returns the KClock Free Spectral Range for this system in GHz.

Parameters:

Ref double	Free Spectral Range	KClock FSR for this system in GHz

Returns an error code or 0 if there was no error.

UInt32 HomeVDL ()

Sends the VDL motor to its home position

Parameters:
None

Returns an error code or 0 if there was no error.

UInt32 MoveRelVDL ()

Moves the VDL to a relative position at a specific speed. Positive values will move the VDL forward away from home negative values will move the VDL backward.

Parameters:

Single	Position	Relative amount to move in mm
Single	Speed	Speed to move at in mm/sec

Returns an error code or 0 if there was no error.

UInt32 MoveAbsVDL ()

Moves the VDL to an absolute position at a specific speed. Position needs to be a positive number of millimeters away from home.

Parameters:

Single	Position	Absolute amount to move in mm
Single	Speed	Speed to move at in mm/sec

Returns an error code or 0 if there was no error.

8. DAQ Specific Methods

UInt32 SetFPGARegister ()

Sets one FPGA register

Parameters:

UInt32	RegNum	Register number
UInt32	RegVal	Value to set register to

Returns an error code or 0 if there was no error.

UInt32 GetFPGARegister()

Gets a value for a single FPGA Register

Parameters:

UInt32[]	Regnum	Register number
Ref UInt16	Regval	Returned register value

UInt32 SetFPGASettings()

Sends FPGA settings to device. These settings are an array of integers and are stored in flash. They are written to the FPGA after FPGA configuration.

Parameters:

UInt32	NumIntScanSettings	Number of scan settings to write
Int[]	IntScanSettings	Array of integers

Returns an error code or 0 if there was no error.

UInt32 GetFPGASettings()

Returns FPGA settings from a device.

Parameters:

Ref UInt32	NumIntScanSettings	Number of scan settings returned
Ref Int[]	IntScanSettings	Array of integers
Ref string[]	IntScanSettingNames	Strings containing descriptions of each scan setting

UInt32 SendFPGAData()

Send a block of data to the DAQ FPGA.

Parameters:

UInt16	FPGAData	Data array to send
Int	nWords	Number of integers in data array
UInt32	Location	FPGA register to load the data into
UInt32	StartAddr	Starting FPGA address to write to

Returns an error code or 0 if there was no error.

UInt32 WriteDAQRegisterBit ()

Changes 1 bit in an FPGA register

Parameters:

UInt32	RegisterNumber	Register Number
UInt32	Bit	Which bit to change (0-16)
UInt32	Value	Bit value (0 or 1)

Returns an error code or 0 if there was no error.

9. Cameralink ADC Specific Methods

UInt32 SetFPGASettings()

Sends FPGA settings to the Cameralink ADC device and writes them to flash

Parameters:

UInt32	NumIntScanSettings	Number of FPGA settings to write
Int[]	IntScanSettings	Array of integers to be stored as FPGA settings

Returns an error code or 0 if there was no error.

UInt32 GetFPGASettings()

Gets FPGA settings from the Cameralink ADC device

Parameters:

Ref UInt32	NumIntScanSettings	Number of FPGA settings to write
Ref Int[]	IntScanSettings	Array of integers to be stored as FPGA settings
Ref string[]	IntScanSettingNames	Array of strings containing the names of the scan settings

Returns an error code or 0 if there was no error.

UInt32 FPGAADCEnable()

Turns on or off the A→D converter on Cameralink ADC

Parameters:

Bool	Enable	Boolean to turn ADC on or off

Returns an error code or 0 if there was no error.

10. Example C# Code

The following code demonstrates how to use the OCT Control to connect up the **device connected** event, connect to a device, request the firmware version and get a description of any error that is returned.

```
public class Form1 : Form
{
    public AxsunOCTControl.AxsunOCTControl AOCT;
    bool OCTConnected = false;

    public Form1()
    {
        InitializeComponent();

        AOCT = new AxsunOCTControl.AxsunOCTControl();

        // The next line wires up the device connected event.
        AOCT.OCTDeviceConnectOrDisconnectEvent += new
AxsunOCTControl.AxsunOCTControl.OCTDeviceConnectOrDisconnectHandler(AOCT_OCTDeviceConnect
OrDisconnectEvent);

        this.Controls.Add(AOCT);
        ConnectToFirstDevice();

        if (OCTConnected)
        {
            Label1.Text = "OCT Connected";
            label2.Text = GetFirmwareVersion(false);
        }
    }

    // Connects to the first enumerated OCT device.
    // This may be modified to connect to another device
    // by changing the value in ConnectToOCTDevice().
    void ConnectToFirstDevice()
    {
        bool result;

        if (AOCT.GetNumberOfOCTDevicesPresent() > 0)
        {
            result = AOCT.ConnectToOCTDevice(0);
            if (result)
            {
                OCTConnected = true;
            }
            else
            {
                OCTConnected = false;
            }
        }
        else
        {
            OCTConnected = false;
        }
    }

    // Gets the firmware version from the OCT device
    public string GetFirmwareVersion()
```

```
    {
        UInt32 result;
        string FirmwareVersion = "";

        result = AOCT.GetFirmwareVersion(ref FirmwareVersion);
        ShowOCTError(result);
        return FirmwareVersion;
    }

    // Displays a message box with the string to match the
    // numerc error code
    private void ShowOCTError(UInt32 ErrorVal)
    {
        string ErrorString;

        if (ErrorVal != 0)
        {
            ErrorString = AOCT.GetErrorString(ErrorVal);
            MessageBox.Show(ErrorString + " (" + ErrorVal.ToString() + ")",
Application.ProductName);
        }
    }

    // This event is called whenever a device is either
    // plugged in or unplugged.
    void AOCT_OCTDeviceConnectOrDisconnectEvent()
    {
        ConnectToFirstDevice();
    }
}
```

11. Error Codes

Below is a reference list of all error codes. Alternatively, GetErrorString() can be called to return the string for the error.

Firmware Error Codes:

```
OK                                  = 0,
USB_CHECKSUM_ERROR                  = 1,
USB_TIMEOUT_ERROR                   = 2,
USB_READ_ERROR                      = 3,
USB_WRITE_ERROR                     = 4,
HIGHEST_USB_ERROR                   = 5,
DAC_TABLE_TOO_LARGE                 = 6,
SCAN_SETTING_COUNT_MISMATCH          = 7,
INVALID_SCAN_SETTINGS               = 8,
INVALID_ERROR_CODE                  = 9,
INVALID_DA_CHANNEL                  = 10,
INVALID_DIAGNOSTIC_TYPE             = 11,
INVALID_DIAGNOSTIC_IO               = 12,
INVALID_CLOCK_DELAY                 = 13,
INVALID_COMMAND                     = 14,
INVALID_CHANNEL                     = 15,
TEC_NEVER_GOT_TO_READY              = 16,
TEC_WENT_OUT_OF_RANGE               = 17,
FLASH_ERR_POLL_TIMEOUT              = 18, // Polling toggle bit failed
FLASH_ERR_VERIFY_WRITE              = 19, // Verifying write to flash failed
```

```
	FLASH_ERR_INVALID_SECTOR		= 20, // Invalid Sector
	FLASH_ERR_INVALID_BLOCK		= 21, // Invalid Block
	FLASH_UNKNOWN_COMMAND		= 22, // Unknown Command
	FLASH_ERR_PROCESS_COMMAND		= 23, // Processing command
	FLASH_NOT_READ_ERROR		= 24, // Could not read memory from target
	FLASH_DRV_NOTAT_BREAK		= 25, // The drive was not at AFP_BreakReady
	FLASH_BUFFER_IS_NULL		= 26, // Could not allocate storage for the buffer
	FLASH_NO_ACCESS_SECTOR		= 27, // Cannot access the sector( could be locked or
something is stored there that should not be touched )
	FLASH_NUM_ERROR_CODES		= 28,
	FLASH_ERR_FLASH_WRITE		= 29,
	FLASH_ERR_FLASH_VERIFY		= 30,
	FLASH_ERR_FILE_OVERSIZE		= 31,
	FLASH_ERR_FILE_EMPTY		= 32,
	FLASH_UNKNOWN_FILE_TYPE		= 33,
	FLASH_ERR_INVALID_DAC		= 34,
	FLASH_ERR_FILE_CKSUM		= 35,
	FLASH_FILE_SIZE_INCORRECT		= 36,
	SWEEP_SPEED_TOO_FAST		= 37,
	TRIGGER_TIMER_TOO_LARGE		= 38,
	SWEEP_ALREADY_RUNNING		= 39,
	SWEEP_ALREADY_STOPPED		= 40,
	SERIAL_NUMBER_TOO_LARGE		= 41,
	INVALID_PARK_COMMAND_SETTINGS	= 42,
	INVALID_FLASH_SCAN_SETTINGS		= 43,
	INVALID_FLASH_DAC_TABLE		= 44,
	INVALID_FLASH_SERIAL_NUM		= 45,
	SWEEP_UNDERRUN			= 46,
	SWEEP_UNDERFLOW			= 47,
	SWEEP_DMA_ERROR			= 48,
	SWEEP_ON_OFF_TOO_CLOSE		= 49,
	CAMERALINK_FPGA_PARAM_COUNT_MISMATCH	= 50,
	INVALID_FLASH_FPGA_IMAGE		= 51,
	INVALID_FPGA_SETTINGS		= 52,
	FPGA_IMAGE_WRONG_SIZE		= 53,
	FPGA_PROGRAM_TIMEOUT		= 54,
	INVALID_COMMAND_FOR_THIS_BOARD_TYPE	= 55,
	COMM_TIMEOUT			= 56,
	RS232_BUFFER_ERROR			= 57,
	RS232_EXIT_FROM_WAIT_FOR_COMMAND	= 58,
	RS232_DATA_COUNT_MISMATCH_ERROR	= 59,
	RS232_DEVICE_NOT_ATTACHED		= 60,
	USB_BUFFER_SIZE_ERROR		= 61,
	ETHERNET_INIT_ERROR			= 62,
	ETHERNET_READ_ERROR			= 63,
	ETHERNET_WRITE_ERROR		= 64,
	ETHERNET_DISCONNECT			= 65,
	COMMAND_ID_INVALID			= 66,
	HANDPIECE_PARAM_COUNT_MISMATCH	= 67,
	DAQ_PARAM_COUNT_MISMATCH		= 68,
	INVALID_HANDPIECE_SETTINGS		= 69,
	INVALID_DAQ_SETTINGS		= 70,
	STEPPER_LIMIT_SWITCH_ACTIVATED	= 71,
	STEPPER_HOME_SWITCH_ACTIVATED	= 72,
	DAQ_FPGA_PROGRAM_TIMEOUT		= 73,
	BAD_FPGA_PROGRAM_IN_FLASH		= 74,
	NO_RESPONSE_FROM_FPGA_REG_READ	= 75,
	SYNC_PACKET_ERROR_FROM_HOST		= 76,
	DAQ_IMAGE_TOO_LARGE		= 77,
	STEPPER_TABLE_TOO_LARGE		= 78,
	INVALID_SPIN_SPEED			= 79,
```

```
COMM_PORT_NOT_SUPPORTED                   = 80,
INVALID_PULLBACK_PARAMETERS               = 81,
TOO_MUCH_FPGA_DATA                        = 82,
STEPPER_NEVER_REACHED_HOME                = 83,
NVRAM_CRC_ERROR                           = 84,
NO_DATA_IN_NVRAM                          = 85,
SPIN_MOTOR_NOT_READY                      = 86,
TOO_MANY_FRAMES_TO_RECORD                 = 87,
TIMEOUT_WAITING_FOR_FPGA_RECORD_COMPLETE  = 88,
REGISTER_READ_RANGE_TOO_LARGE             = 89,
CANNOT_TURN_ON_SOURCE_CURRENT             = 90,
ERROR_SETTING_FPGA_RECORD_BIT             = 91,
FPGA_ERROR_VCCAUX_OUT_OF_RANGE            = 92,
FPGA_ERROR_VCCINT_OUT_OF_RANGE            = 93,
FPGA_ERROR_OVER_TEMP                      = 94,
FPGA_ERROR_USER_OVER_TEMP                 = 95,
FPGA_ERROR_TCD_TOO_SHORT                  = 96,
FPGA_ERROR_TDC_TOO_LONG                   = 97,
FPGA_ERROR_SWEEP_TRIGGER_OUT_OF_RANGE     = 98,
FPGA_ERROR_ADC_SAMPLE_CLOCK_NOT_DETECTED  = 99,
FPGA_ERROR_JPEG_CORE_ERROR                = 100,
FPGA_ERROR_FIFO_ERROR                     = 101,
FPGA_ERROR_A_SCAN_PADDER_DROPPED_FRAME    = 102,
FPGA_CONFIG_TABLE_TOO_LARGE               = 103,
USB_CMD_HDR_CHKSUM_ERROR                  = 104,
ETH_CMD_HDR_CHKSUM_ERROR                  = 105,
RS232_1_CMD_HDR_CHKSUM_ERROR              = 106,
RS232_2_CMD_HDR_CHKSUM_ERROR              = 107,
PACKET_CHKSUM_ERROR                       = 108,
PACKET_TOO_LARGE                          = 109,
INTER_BRD_PACKET_CHKSUM_ERROR             = 110,
INTER_BRD_PACKET_TOO_LARGE                = 111,
INTER_BRD_RS232_1_CMD_HDR_CHKSUM_ERROR    = 112,
INTER_BRD_RS232_2_CMD_HDR_CHKSUM_ERROR    = 113,
COMM_TIMEOUT_AT_PASS_THROUGH              = 114,
LAST_RESET_WAS_WD_ERROR                   = 115,
LAST_RESET_WAS_SWRST_ERROR                = 116,
LAST_RESET_WAS_DF_ERROR                   = 117,
USB_INIT_FAILURE                          = 118,
ENGINE_NOT_CONNECTED                      = 119,
INTERBOARD_COMM_ERROR                     = 120,
MOTOR_MOVE_OUT_OF_RANGE                   = 121,
SOME_SETTINGS_WERE_UPDATED                = 122,
COMMAND_TO_GO_TO_ERROR_STATE              = 123,
INVALID_BOARD_CONFIG                      = 124,
DAC_TABLE_NUM_TOO_LARGE                   = 125,
CANNOT_START_SWEEP_TEC_IS_IN_ERROR        = 126,
PARK_VALUE_FILTER_1_OUT_OF_RANGE          = 127,
PARK_VALUE_FILTER_2_OUT_OF_RANGE          = 128,
PART_NUMBER_TOO_LARGE                     = 129,
SOURCE_CURRENT_ON_OFF_NOT_ALLOWED         = 130,
INVALID_FPGA_CONFIG_TABLE                 = 131,
CANNOT_START_SWEEP_INTERLOCK_ACTIVE       = 132,
CANNOT_TURN_ON_SOURCE_CURRENT_INTERLOCK_ACTIVE = 133,
FPGA_PHY_STARTUP_TRIES                    = 134,
BIT_VALUE_TOO_LARGE                       = 135,
VALUE_MUST_BE_0_OR_1                      = 136,
UDP_PACKETS_RECEIVED                      = 137,
INVALID_CONFIGURATION_PARAMETERS          = 138,
SERIAL_NUMBER_CAN_ONLY_BE_SET_IN_CONFIGURATION_1       = 139,
CONFIGURATION_NOT_SETUP                                = 140,
```

```
MODEL_NUMBER_TOO_LARGE                                  = 141,
VOLTAGE_IS_OUT_OF_RANGE                                 = 142,

POWER_MONITOR_ERROR                                     = 143,
V33_ERROR                                               = 144,
VM5_ERROR                                               = 145,
V5_ERROR                                                = 146,
V15_ERROR                                               = 147,
V180_ERROR                                              = 148,
SYSTEM_IS_IN_ERROR                                      = 149,
ENGINE_BOARD_TEMP_OUT_OF_RANGE                          = 150,
DAQ_BOARD_TEMP_OUT_OF_RANGE                             = 151,
CANNOT_START_SWEEP_SYSTEM_IS_IN_ERROR                   = 152,
```

DLL Error Codes:

```
  ERROR_READING                                         = 1000;
  ERROR_WRITING_1                                       = 1001;
  NO_DEVICE_PRESENT                                     = 1002;
  CHECKSUM_ERROR                                        = 1003;
  TOO_MANY_PACKET_TRIES                                 = 1004;
  INTERNAL_DATA_ERROR                                   = 1005;
  SCAN_SETTINGS_ARE_EMPTY                               = 1006;
  ERROR_READING_SCAN_SETTINGS                           = 1007;
  ERROR_WRITING_SCAN_SETTINGS                           = 1008;
  ERROR_SAVING_DAC_FILE                                 = 1009;
  ERROR_READING_DAC_FILE                                = 1010;
  ERROR_WRITING_FPGA_SETTINGS                           = 1011;
  ERROR_READING_FPGA_SETTINGS                           = 1012;
  ERROR_WRITING_DAQ_SETTINGS                            = 1013;
  ERROR_READING_DAQ_SETTINGS                            = 1014;
  ERROR_WRITING_HANDPIECE_SETTINGS                      = 1015;
  ERROR_READING_HANDPIECE_SETTINGS                      = 1016;
  FPGA_SETTINGS_ARE_EMPTY                               = 1017;
  DAQ_SETTINGS_ARE_EMPTY                                = 1018;
  HANDPIECE_SETTINGS_ARE_EMPTY                          = 1019;
  TCP_READ_ERROR                                        = 1020;
  TCP_WRITE_ERROR                                       = 1021;
  TCP_CONNECT_ERROR                                     = 1022;
  TCP_EVENT_SOCKET_ERROR                                = 1023;
  ERROR_CONNECTING_TO_DEVICE_FOR_HEARTBEAT              = 1024;
  ERROR_READING_RS232                                   = 1025;
  BOARD_DOES_NOT_SUPPORT_THIS_COMMAND                   = 1026;
  ERROR_READING_FPGA_DATA_FILE                          = 1027;
  ERROR_STRUCTURE_SIZE_DOES_NOT_MATCH                   = 1028;
  ERROR_DATA_SIZE_TOO_BIG                               = 1029;
  ERROR_FIRMWARE_IS_EMPTY                               = 1030;
  ERROR_USB_WRITE_INIT                                  = 1031;
  ERROR_USB_READ_INIT                                   = 1032;
  ERROR_USB_INIT                                        = 1033;
  USB_READ_LEVEL_1_TIMEOUT                              = 1034;
  USB_READ_LEVEL_2_TIMEOUT                              = 1035;
  USB_DEVICE_CLOSED                                     = 1036;
  USB_READ_READFILE_ERROR                               = 1037;
  USB_COMM_ERROR                                        = 1038;
  MUTEX_TIMEOUT                                         = 1039;
  DEVICE_CONNECTING                                     = 1040;
  DEVICE_REBOOTING                                      = 1041;
  HARD_ERROR                                            = 1042;
```

```
ERROR_WRITING_2                                 = 1043;
ERROR_WRITING_3                                 = 1044;
ERROR_WRITING_4                                 = 1045;
ERROR_WRITING_5                                 = 1046;
NETWORK_THREAD_ALREADY_RUNNING                  = 1047;
DAC_TABLE_NOT_LOADED                            = 1048;
```