Ocean Optics SeaBreeze

1 Introduction

SeaBreeze is a simplified device driver intended for use with Ocean Optics spectrometers and other devices. This document describes SeaBreeze and compares it to other Ocean Optics device drivers. Note that SeaBreeze is not a spectral math library and provides only limited spectral correction or manipulation. SeaBreeze is intended for use by experienced C/C++ programmers and may not be suitable for all audiences. It is provided as a starting point for developers that would otherwise have to create an equivalent device interface themselves because OmniDriver is too large or complex for their application. SeaBreeze does not support every Ocean Optics device, feature, or target platform at this time, but it is extensible so other developers can add required enhancements themselves.

This archive is provided as-is, without any warranty, express or implied. Use the enclosed files at your own risk. Ocean Optics is not obliged to provide any support for these files or any notification of defects. These files are not licensed for further redistribution, but resulting or derived compiled products or libraries may be freely distributed. Some included files (in the `windows-support` directory) are copyrighted by and licensed from Microsoft (as part of redistribution of the DDK) and can be redistributed according to Microsoft's license. Any use of the files in this archive implies understanding of and agreement with these terms.

The remainder of this document is organized as follows. Section 2 presents the design goals for SeaBreeze and an overview of the software architecture that was created to meet these requirements. Section 3 provides a comparison of SeaBreeze to OmniDriver. Section 4 describes how to use the SeaBreeze distribution. Section 5 discusses known limitations.

2 Design Goals and Architecture

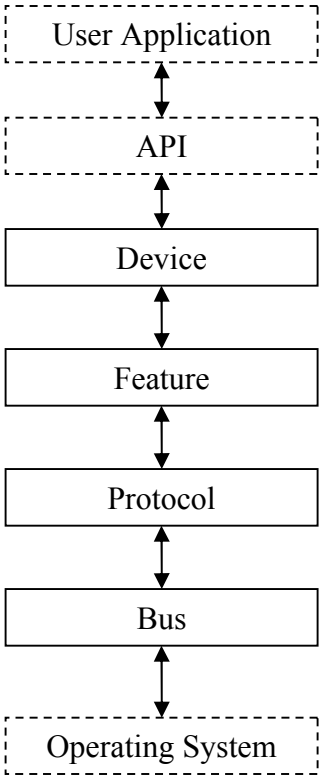
SeaBreeze was designed to address specific limitations of existing driver products. The general principles that guided the design to avoid these limitations are provided below.

- *SeaBreeze should be small*. Prior Ocean Optics drivers required a Java Virtual Machine (JVM) to operate, which when installed would require 40-70MiB of disk space and significant memory when executed. SeaBreeze is implemented in C++ which does not require a special runtime on most platforms, and can be built into a library that is smaller than 700KiB. As a result, SeaBreeze is suitable for embedded platforms. Where source code is available, developers can also remove functionality they do not require to further reduce the code size.
- *SeaBreeze should be fast*. OmniDriver is complex, and on some computers, does not easily reach the fastest cycle time of some Ocean Optics spectrometers. SeaBreeze is intended to have minimal overhead between requesting and reading out data.

- *SeaBreeze should be flexible*. Ocean Optics devices communicate via USB, Ethernet, RS232, and other buses. SeaBreeze was designed to communicate with devices using different protocols and buses. This is accomplished by allowing any number of protocols and buses to be defined for a particular device, and letting the combination of these to be determined at run-time.
- *SeaBreeze should be extensible*. SeaBreeze is designed and organized such that support for new features within existing devices can be added easily, and new devices that do not resemble anything already supported can also be added without difficulty or conflict.
- *SeaBreeze should be portable*. Most of the code in SeaBreeze is generic C++, and where it interfaces to a specific computer bus, an abstract bus interface is used. SeaBreeze can be ported to new platforms simply by providing a native implementation of the bus interfaces for the target.
- *SeaBreeze should provide uniform support for all aspects of a device*. In prior device drivers, the device was assumed to be a spectrometer with some other related functionality (e.g. thermo-electric cooler, GPIO). In SeaBreeze, devices are treated simply as a collection of features, which may or may not include a spectrometer. This provides a more uniform interface to these features.
- *SeaBreeze should be designed for and used by experienced C/C++ developers only*. OmniDriver and OOIWinIP provide numerous language interfaces (e.g. LabVIEW, Visual Basic) that require very little programming experience to use. No such language interfaces are planned for SeaBreeze, though calling functions in a SeaBreeze library (e.g. DLL) may be possible from other languages. SeaBreeze is intended for use by experienced programmers that are proficient with object oriented programming, exception handling, threading, and other modern programming language features.

2.1 Architectural Overview

SeaBreeze provides four layers of abstraction between application code and the device being controlled: *device*, *feature*, *protocol*, and *bus*. Each layer deals with different kinds of information and has different responsibilities. Data and control always move between these layers in a consistent manner. The provided `SeaBreezeWrapper` API provides access to the *device* layer in a simplified manner, and operating system functionality may reside below the *bus* layer. The relationship between these layers and the rest of the system is as follows:



Simplified interface between SeaBreeze and user application

Representation of a physical device; a collection of usable features

A single device capability, such as TEC, EEPROM, spectrometer, strobe/lamp enable, GPIO

A command set or encoding of parameters and data necessary to interact with a specific device feature

Representation of a hardware bus capable of sending and/or receiving arbitrary bit or byte sequences

Low-level control of hardware to accomplish data transfers

A separate document containing a detailed UML description of the architecture should accompany this introduction.

3 Comparison to Other Drivers

SeaBreeze is not intended to be a universal replacement for OmniDriver or its predecessor, OOIWinIP (OOIDrv32.DLL). SeaBreeze and OmniDriver address different needs. The following table provides a side-by-side comparison of these drivers to illustrate where each is best used.

Attribute	OmniDriver	SeaBreeze
Required resources	Requires a Java Runtime Environment (JRE) which is typically 40+ MiB on disk and has a memory overhead when executed	Links to the C++ runtime that is available on most operating systems. Can be compiled into a library that is less than 700KiB.
Language interfaces	Java, C, C++, Delphi, Visual Basic, C#, LabVIEW	C/C++
Device support	All Ocean Optics USB spectrometers	No support for discount spectrometers, e.g. RedTide or Amadeus; only most popular devices supported

Device feature support	Every major feature of Ocean Optics USB devices is supported	Only the most commonly used device features are supported to minimize total code size
Operating system support	Windows 98, ME, 2000, XP, Vista, and 7 (32- and 64-bit), Linux (32-bit and 64-bit), MacOSX 10.3-10.5 (PPC and Intel)	Windows XP, Vista, and 7 (32-bit and 64-bit), Linux, MacOSX; extensible to other platforms
Audience	Application developers or novice programmers	OEMs and system integrators
Target platforms	Workstations	Embedded systems

4 Installation and Use

SeaBreeze currently supports three development environments: Linux, Windows, and MacOSX. Each of these has specific requirements that are described below. A sample program has been provided for each operating system that illustrates the use of an API class, `SeaBreezeWrapper`. This API should be used if possible as it will greatly simplify the process of acquiring data from Ocean Optics spectrometers as compared to calling the underlying classes directly. `SeaBreezeWrapper` also provides a working example of how to use the rest of the architecture if needed.

4.1 Linux

SeaBreeze requires the `libusb` library to be installed on the development system, including the header files (e.g. `usb.h`). Most modern Linux distributions provide a suitable version of this library. A typical compiler toolchain (`g++`, `ld`, `ar`, `make`) is required. Running `make` in the top-level directory should be sufficient to build `libseabreeze.so` which can then be linked against by a user application. The top-level `Makefile` can be modified to disable debugging symbols and enable optimization (e.g. remove the `-ggdb3` flag and add `-O2` from the `CFLAGS`). This should be done prior to redistribution of the compiled library as it will improve performance and reduce its size. Note that the `Makefile` does not always detect when dependencies have changed, to it may be necessary to delete `libseabreeze.so` prior to running `make` each time to ensure a proper result.

A test program is located in `test/seabreeze_test_linux.c` that should connect to a supported device and acquire data. This will be built along with `libseabreeze.so` above. Note that if errors are displayed indicating that a device cannot be claimed, it is likely that the user does not have sufficient privileges. Running the program as `root` is one solution, or a `udev` script can be installed if the Linux system uses `udev` to detect newly connected devices. A sample `udev` script (`linux-support/10-oceanoptics.rules`) has been included and should be placed in or around `/etc/udev/rules.d` as appropriate.

4.2 Windows

SeaBreeze communicates with USB devices in Windows using the WinUSB API provided by Microsoft. This API is supported by default in Windows Vista and Windows 7, both 32-bit and 64-bit, though user approval of the driver may be required. Windows XP can also use the WinUSB API if it is installed. Note that there may be a conflict between the .INF files required to use an Ocean Optics device with WinUSB and the .INF files formerly used by Ocean Optics software to communicate via USB (the `ezusb.sys` and `ooiusb.inf` files). Only one of these may be used on a system at a time. It is expected that WinUSB will be used for future support of Windows, and `ezusb.sys` will be phased out. Support for `ezusb.sys` in SeaBreeze is possible but not currently available.

The `windows-support` directory contains .INF files for each of the supported Ocean Optics devices (and some that are not), and "CoInstaller" bundles for 32-bit and 64-bit systems. These files are required when first connecting the spectrometer to the system. The recommended way to install the driver is to get to the Windows Device Manager and to reinstall or update the driver. If the user browses to this `windows-support` directory to find the .INF file, then the rest of the files should be found automatically. Note that if existing Ocean Optics driver files (e.g. `ooiusb.inf` and `ezusb.sys`) are found on the system, there may be a conflict. Windows seems to give these older drivers priority, so it may be necessary to remove (or rename) them completely prior to using the included drivers. The enclosed drivers should appear with "(WinUSB)" in the name when presented by the device manager, which should help to identify them.

To develop with SeaBreeze in Windows, the latest Windows DDK is required. This can be downloaded for free from Microsoft (once an account is created) and is provided with an MSDN subscription. The DDK version used to develop Windows support for SeaBreeze was 6001.18001. To set up a Visual Studio project to use this DDK, it is necessary to specify the following include directories within the DDK installation:

- `inc\ddk`
- `inc\api`

The following external library (.lib) files must also be specified:

- `winusb.lib`
- `setupapi.lib`

The library directory must also be specified. The correct setting depends on the build target. For 32-bit targets, these are probably located in the following directory relative to the DDK:

- `lib\wxp\i386`

For 64-bit targets, these are probably located in the following directory relative to the DDK:

- `lib\wlh\amd64`

Note that some files provided with the SeaBreeze distribution are intended for Linux and will not compile with Visual Studio. In particular, the `NativeUSB.c` file in `src\native\usb\linux` should not be included in any Visual Studio project as

many of its included header files (e.g. `usb.h`) are not available in Windows. Further, the test program `test\seabreeze_test_linux.c` contains Linux-only code and will not compile under Visual Studio.

The sample program, `seabreeze_test_windows.c`, illustrates how to do simple spectrometer control and data acquisition. This has been tested using Visual Studio 2005 on a Windows Vista 32-bit operating system set up as described above. This is not intended to be a comprehensive demonstration program, but should show how to use the SeaBreezeWrapper API as a starting point.

4.3 MacOSX

SeaBreeze communicates with USB devices in OSX through the IOKit framework. As a result, no extra system files are necessary. The OSX driver has been tested using Snow Leopard but it should work for any version of OSX (Intel or PPC). No steps have been taken to make a universal binary out of the dynamic library yet, so any attempt to make a single executable that will work in both Intel and PPC, 32-bit and 64-bit may require some changes to the compiler and linker settings.

To build SeaBreeze for OSX, it is sufficient to run `make` in the top-level directory of the source tree provided that `gcc` has been installed and is in the path. It should be possible to compile this in XCode, but no project for XCode is provided. Linux and OSX both use the same Makefiles to build SeaBreeze, and `make` will determine what to do for each. As with the Linux build (see Section 4.1 above), it may be desirable to remove debugging (`-ggdb3` flag) and add optimization (`-O2` flag) to `CFLAGS` prior to a release.

When built, a program called `seabreeze_test_macosx` will be created in the `test` directory. This will attempt to open connected devices and perform some simple interactions with them.

Additional operating systems can be supported provided that native interfaces for the required communication buses are created. The existing native interfaces can be found in the `native` directories in the top-level `src` and `include` directories.

4.4 Organization

The source code for SeaBreeze is laid out as follows. There are two top-level directories, `include` and `src`, that both have the same organization (shown below). Header (`.h`) files are located in the `include` directory tree, and all other files (e.g. `.cpp`) are located in the `src` tree. Within these directories, files are organized as follows:

- `api` (simplified interface for applications to use)
- `common` (base classes, macros, and general definitions)
    - `buses` (abstract definitions of the *bus* layer)
    - `devices` (abstract definitions of the *device* layer)
    - `exceptions` (C++ exception types that may be thrown)
    - `features` (abstract definitions of the *feature* layer)

    - `protocols` (abstract definitions of the *protocol* layer)
- `native` (bus interface implementations for target platforms)
- `vendors` (device-specific definitions)
    - `OceanOptics` (definitions for Ocean Optics devices)
        - … (classes derived from those in `common`)

5 Limitations

SeaBreeze is a relatively new project and it may contain coding errors. Source code is provided so that any such errors can quickly be resolved by the developer. Device and feature support is also limited. The following Ocean Optics USB spectrometers are supported by SeaBreeze at present:

- HR2000, HR2000+, HR4000, USB2000, USB2000+, USB4000, Maya2000, MayaPro2000, QE65000, NIRQuest 256, NIRQuest 512, STS

For these devices, SeaBreeze can do the following:

- Set integration time
- Acquire and decode a spectrum
- Set strobe/lamp enable state
- Read EEPROM slots, including serial number
- Compute wavelengths for spectrometer pixels based on calibration stored in EEPROM
- Enable/disable thermoelectric cooler on equipped spectrometers, set target temperature, read current detector temperature in degrees Celsius, and read out the default settings for these parameters
- Set trigger mode
- For some spectrometers, read an irradiance calibration from the device

At present, SeaBreeze does not perform any spectral corrections, such as electric dark correction, boxcar smoothing, multi-scan averaging, linearity correction, or stray light correction. Since source code is provided, required functionality can be added as needed. Sample code that will do linearity and electric dark correction is provided in the `test` directory, but is above the level of the API.