MDS Analytical Technologies

ABF2 Programmers Guide

Author: Philip Churchward
Date: June 19, 2007

Revised October 9, 2007

Background

The ABF file format has been used for many years by Axon Instruments for the pCLAMP electrophysiology data acquisition and analysis software. In addition it has been used by third party developers.

An ABF consists of a header (containing the protocol and data file structure information), followed by numerous ‘data’ sections (the actual data, and also other information such as the length of each sweep, tags and scope information). In version 1.x (used in pCLAMP 9.2 and earlier versions), the header was a simple fixed-size header, which simply read from disk and then passed to higher level software. This had the limitation that any changes to numbers of channels or information fields had to fit within the fixed size header. . This restriction also applied to strings which were of a fixed length.

ABF2 (released with pCLAMP 10.0) sought to remove this restriction and allow for more flexible handling of extra fields, variable length strings and different numbers of channels (either input or output). However, with this extra flexibility comes an extra level of complication.

The data sections of an ABF2 file are exactly the same as the earlier ABF1.x files; the only difference is the variable-sized header at the start of the file.

The remainder of this document assumes the reader is familiar with ABF1.x files and as such it will concentrate on the differences between ABF1.x and ABF2. The File Support Pack is freely available from the Molecular Devices web site. Full source code is available for ABF1.x, but not ABF2. However, the ABF2 file header section of the source code is available (for developers on platforms other than Windows) upon request.

ABF2 File Header

The ABF2 file header is a variable length structure. Everything in ABF files is written in 512 byte blocks – the file header is typically about 10 – 15 of these blocks (i.e. 5 kB – 7.5 kB).

The C++ header file ProtocolStructs.h documents the various C style structs that are written at the start of the file. When writing new ABF2 files it is critical that none of these structs are changed, since the reading code requires the structs to be the expected size and in the expected order. The flexibility of ABF2 comes from the fact that there are a variable number of these structs depending on the protocol characteristics (e.g. the number of ADC and DAC channels). In addition, all strings are now of varible length. As a result, it is expected that this file format should remain stable as new features are added to future versions.

The order of the structs in the file is shown in ProtocolReaderABF2.cpp, in the `CABF2ProtocolReader::Read` method.

The code in ProtocolReaderABF2.cpp demonstrates how the variable-sized header information is read from disk and is used to populate the fixed-sized ABFFileHeader, which is used by the remainder of the pCLAMP applications. Third party developers do not need to read into an ABFFileHeader struct, though its header file in included in case this is desired. The ABFFileHeader is similar in structure to earlier ABF1.x versions, except that it is no longer forced to remain unchanged – this has the advantage that retired fields can be (and have been) removed.

Once the header information has been read, the remainder of the file can be read in exactly the same manner as earlier versions. The locations of the various sections is contained in the various ABF_Section structs contained in the ABF_FileInfo struct at the start of the file.

Channel Ordering
Information about the ADC and DAC channels is written in the ADCInfo and DACInfo structs. The channel ordering is indicated by the order in which these structs appears in the file. The nADCNum field in ADCInfo and nDACNum in DACInfo indicate which ADC / DAC is being referred to. Note that this order is not necessarily in numerical order, though in some situations it might be.

Distinguishing between ABF1 and ABF 2 files
The first filed in both ABF 1 and ABF 2 files is a 4 byte UNIT containing the 'File Signature', which is a specified constant indicating the type of the file. Each ProtocolReaderXXX class has a CanOpen() method that checks the file signature in order to determine whether it can open files with a matching file signature.

The next field is the file version; in ABF 1 is it a 4 byte float, whereas in ABF 2 it is a 4 byte UINT. Therefore both take up the same number of bytes and the relevant CanOpen() method tests the value as appropriate.

File Compression
ABF 2 supports a new form of data compression, which is decimation, performed independently for each epoch. There is a fixed value of `uFileCompressionRatio`, in the `ABF_ProtocolInfo` struct, which is applied to all epochs with compression enabled.

When a file is being written, one sample is written for each `uFileCompressionRatio` ‘raw’ samples. When reading, each sample is padded out `uFileCompressionRatio` times in order to return the data to the original time scale. The only benefit of file compression is to create smaller data files. Clampfit expands the compressed data to restore the original number of samples (i.e. the same number as were originally acquired and displayed in the Scope window in Clampex). There is loss of data when compression is used.