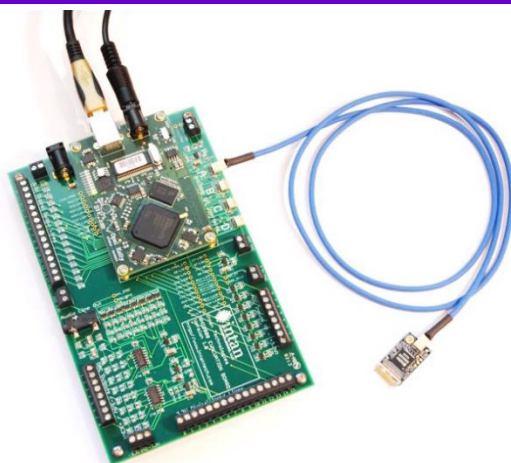


RHD2000-EVAL

Application Note: Data File Formats

19 September 2013; updated 27 February 2015



This application note describes the file formats used by the RHD2000 Evaluation System GUI software to save acquired waveforms to disk. While Intan provides some m-files for reading saved data files into MATLAB, some users may wish to write their own software to access this information. This document provides the necessary information for parsing these files.

Version 1.2 of the GUI interface software introduces multiple file format options (selected by clicking the “Select File Format” button), and each of these formats will be described in a following section.

Data Types

Most of the data types described in this document will be familiar to those with rudimentary programming experience. The following table summarizes the data types referenced in this document:

NAME	DESCRIPTION	RANGE
uint16	unsigned 16-bit integer	0 to 65,535
int16	signed 32-bit integer	-32,768 to 32,767
uint32	unsigned 16-bit integer	0 to 4,294,967,295
int32	signed 32-bit integer	-2,147,483,648 to 2,147,483,647
single	32-bit single-precision floating point number	$\pm 3.4 \times 10^{\pm 38}$ with 7-digit accuracy
double	64-bit double-precision floating point number	$\pm 1.7 \times 10^{\pm 308}$ with 15-digit accuracy
QString	length-prefixed Unicode string (see below)	0 to 2,147,483,647 Unicode characters

All numbers are saved to disk with “little endian” byte order. That is, the least-significant byte is written first and the most-significant byte is written last. MATLAB reads data in little endian format by default.

Text fields are not stored as null-terminated strings as is common in the C family of languages. Rather, they are stored as length-prefixed strings using the **QString** style from the open-source Qt framework for C++. In the **QString** format, each string begins with a 32-bit unsigned number (**uint32**) that indicates the length of the string, in bytes. If this number equals 0xFFFFFFFF, the string is null. A series of 16-bit (2-byte) Unicode characters follows, and there is no special character to indicate the end of the string.

RHD2000-EVAL Application Note

The following MATLAB function reads a **QString** from a file identifier `fid` and translates it into a MATLAB-format string `a`:

```
function a = fread_QString(fid)

a = '';
length = fread(fid, 1, 'uint32');
if length == hex2num('ffffffff')
    return;
end
length = length / 2; % convert length from bytes to 16-bit Unicode words
for i = 1:length
    a(i) = fread(fid, 1, 'uint16');
end

return
```

Standard Intan RHD2000 Header

All Intan RHD2000 data file formats make use of the Standard Intan RHD2000 Header which is described in this section. This header contains records of sampling rate, amplifier bandwidth, channel names, and other useful information.

Each file containing a Standard Intan RHD2000 header has a filename ending with the .rhd prefix. These are a binary files that begin with the following file type and version information.

DATA TYPE	NAME	DESCRIPTION
uint32	Intan RHD2000 Header identifier	This “magic number” always has a value of 0xC6912702 to indicate a traditional Standard Intan RHD2000 Header.
int16	Data file main version number	These two integers indicate the version of the data file (e.g., v1.2 would be encoded by a 1 followed by a 2).
int16	Data file secondary version number	

Next is a block of data containing global sampling rate and amplifier frequency parameters.

single	Sample rate	Amplifier sample rate (units: Samples/s)
int16	DSP enabled?	0: DSP offset removal high-pass filter was disabled 1: DSP offset removal high-pass filter was enabled
single	Actual DSP cutoff frequency	DSP offset removal high-pass filter cutoff frequency (units: Hz)
single	Actual lower bandwidth	Amplifier analog high-pass filter cutoff frequency (units: Hz)
single	Actual upper bandwidth	Amplifier analog low-pass filter cutoff frequency (units: Hz)
single	Desired DSP cutoff frequency	User-requested DSP offset removal filter cutoff frequency (units: Hz)
single	Desired lower bandwidth	User-requested amplifier high-pass filter cutoff frequency (units: Hz)
single	Desired upper bandwidth	User-requested amplifier low-pass filter cutoff frequency (units: Hz)

The RHD2000 chips are not always capable of achieving the precise cutoff frequencies specified by the user, so both the values requested in the GUI and the actual values realized on the chip are saved.

The next parameter records the state of the software-implemented 50/60 Hz notch filter in the GUI during recording. This notch filter is never applied to saved data, but this information may be used to re-apply the notch filter to recorded data, if desired. (The m-file `read_Intan_RHD2000_file.m` re-implements this notch filter on amplifier data that was saved with the filter enabled.)

int16	Notch filter mode	0: Software notch filter was disabled 1: Software notch filter was enabled and set to 50 Hz 2: Software notch filter was enabled and set to 60 Hz
-------	-------------------	---

RHD2000-EVAL Application Note

Next are two floating-point numbers indicating the latest user-requested electrode impedance test frequency and the impedance test frequency actually realized on the RHD2000 chip.

single	Desired impedance test frequency	Electrode impedance test frequency last requested by user (units: Hz)
single	Actual impedance test frequency	Closest realizable electrode impedance test frequency (units: Hz)

In the “Configure” tab of the Intan GUI, there are three general-purpose text fields that may be used to enter notes on particular recording sessions. The contents of these text fields are saved here.

QString	Note 1	User text from the “Note 1” field in the “Configure” tab in the GUI.
QString	Note 2	User text from the “Note 2” field in the “Configure” tab in the GUI.
QString	Note 3	User text from the “Note 3” field in the “Configure” tab in the GUI.

Versions 1.1 and later support saving on-chip temperature sensor readings. The following number is saved in these versions, indicating the number of temperature sensors recorded. This number is typically equal to the number of RHD2000 chips plugged into the SPI ports, or zero if the temperature recording option is disabled. Note that some file formats do not support saving temperature sensor data; in those formats, this number will always be zero.

int16	Number of temperature sensors	Number of temperature sensor channels saved in file. This number is set to zero if the option for saving temperature data is not selected.
--------------	-------------------------------	--

Version 1.3 and later saves the “board mode”. This integer is set by FPGA digital input pins on the RHD2000 USB interface board. Currently, a board mode of zero indicates that the on-board ADCs operate over a range of 0-3.3V; a board mode of one indicates that the on-board ADCs operate over a range of $\pm 5.0V$.

int16	Board mode	Integer ranging from 0-15 indicating global properties of the evaluation board.
--------------	------------	---

The next number indicates the number of “signal groups” present in the data file. This number is typically equal to seven: Port A, Port B, Port C, Port D, Board ADC Inputs, Board Digital Inputs, and Board Digital Outputs.

int16	Number of signal groups in data file	Each “signal group” includes all signals from a particular SPI port or the USB board ADC inputs or digital inputs. There will typically be seven signal groups representing the six items listed under “Ports” in the GUI, plus Board Digital Outputs.
--------------	--------------------------------------	--

For each signal group, the following “signal group header” is saved, along with a description of each channel in the signal group.

QString	Signal group name	e.g., “Port B” or “Board Digital Inputs”
QString	Signal group prefix	e.g., “B” or “DIN”
int16	Signal group enabled?	0: disabled 1: enabled
int16	Number of channels in signal group	Total number of channels in signal group
int16	Number of amplifier channels in signal group	Of the total number of channels in the signal group, the number that are amplifier channels
	List of channels	See below

RHD2000-EVAL Application Note

Immediately following a signal group (before the remaining signal group headers) is a list of channel descriptions. If a signal group is enabled **and** has more than zero channels, then for each channel the following information is saved.

QString	Native channel name	e.g., "B-013" or "DIN-15"
QString	Custom channel name	e.g., "MyTetrode3-4" or "TTLSensor" renamed by user
int16	Native order	The original numerical order of this channel in the GUI display (e.g., the native order of amplifier channel B-013 is 13).
int16	Custom order	The numerical order of this channel as it appears on the GUI, after possible reordering by the user.
int16	Signal type	0: RHD2000 amplifier channel 1: RHD2000 auxiliary input channel 2: RHD2000 supply voltage channel 3: USB board ADC input channel 4: USB board digital input channel 5: USB board digital output channel
int16	Channel enabled?	0: channel disabled 1: channel enabled
int16	Chip channel	RHD2000 channel number (0-31)
int16	Board stream	USB board data stream (0-7); each data stream supports up to 32 channels. Each RHD2164 chips use two data streams. Each RHD2216 chip uses an entire data stream.
int16	Spike Scope voltage trigger mode	0: trigger on digital input 1: trigger on voltage threshold
int16	Spike Scope voltage threshold	Spike voltage threshold (units: microvolts)
int16	Spike Scope digital trigger channel	USB board digital input channel used for spike trigger (0-15)
int16	Spike Scope digital edge polarity	0: trigger on digital falling edge 1: trigger on digital rising edge
single	Electrode impedance magnitude	Last measured impedance magnitude (units: Ohms)
single	Electrode impedance phase	Last measured impedance phase (units: degrees)

Even non-amplifier channels will contain fields for Spike Scope trigger parameters and electrode impedance data, but these fields will contain default values that may be ignored.

The Spike Scope feature in the GUI is used only to aid in viewing neural spikes; the software saves full waveforms, not just spikes. However, the user-specified thresholds set in the Spike Scope are saved for each channel, so it would be relatively easy to write a script to isolate action potentials based on these thresholds (e.g., for compressing saved data files after recording).

This concludes the Standard Intan RHD2000 Header contents. Typical headers consume very little disk space (a few KB) even for large numbers of enabled channels.

Traditional Intan File Format

This file format saves all types of waveforms (RHD2000 amplifier channels, auxiliary channels from RHD2000 chips, chip supply voltages, USB board ADC inputs, and USB board digital inputs) to one file, along with the Standard Intan RHD2000 Header described above. Only enabled channels of each type are saved. To keep individual file size reasonable, a new file is created every N minutes, where N is an integer that is specified by the user. New filenames are created by appending a date and time stamp to a base filename provided by the user. Each file contains both a Standard Intan RHD2000 Header and approximately N minutes of saved data. These .rhd data files may be read into MATLAB using `read_Intan_RHD2000_file.m`, which is provided on the Intan Technologies web site.

This file format includes an option to save the temperature sensor readings from each RHD2000 chip attached to the USB interface board.

RHD2000-EVAL Application Note

Immediately following the Standard Intan RHD2000 Header in each data file is the waveform data. The traditional Intan file format saves waveforms in “data blocks” corresponding to the **Rhd2000DataBlock** object in the C++ code. Each data block contains data from 60 amplifier samples.

Each data block is organized as follows:

60 × int32	Amplifier sample time index	Sequential integers (e.g., 0, 1, 2, 3...) with zero marking the beginning of a recording or a trigger point. Time indices can be negative to denote pre-trigger times. Divide by the amplifier sampling rate (in Samples/s) to get a time vector with units of seconds. The use of an int32 data type means that this number will not “roll over” until total recording times exceed 19.8 hours with the maximum sample rate of 30 kS/s, or 29.8 hours with a sample rate of 20 kS/s.
-------------------	-----------------------------	---

For each enabled RHD2000 amplifier channel, 60 ADC samples:

60 × uint16	Electrode voltage	Units: ADC steps. To convert to electrode voltage in microvolts, first subtract 32768 then multiply by 0.195.
--------------------	-------------------	---

For each enabled RHD2000 auxiliary input channel, 15 ADC samples. (The Intan GUI software samples the RHD2000 auxiliary inputs at one-fourth the sampling rate of the amplifiers.)

15 × uint16	Auxiliary input voltage	Units: ADC steps. To convert to volts, multiply by 0.0000374.
--------------------	-------------------------	---

For each enabled RHD2000 supply voltage channel, one ADC sample. (The Intan GUI software samples the RHD2000 supply voltage at one-sixtieth the sampling rate of the amplifiers.)

1 × uint16	Supply voltage	Units: ADC steps. To convert to volts, multiply by 0.0000748.
-------------------	----------------	---

For each enabled RHD2000 temperature sensor channel, one ADC sample. (The Intan GUI software samples the RHD2000 temperature sensor at one-sixtieth the sampling rate of the amplifiers.) The GUI calculates a running average of temperature sensor readings over a window of approximately 100 ms to improve sensor accuracy.

1 × int16	Temperature sensor reading	Units: 0.01°C per step. To convert to °C, divide by 100.
------------------	----------------------------	--

For each USB interface board ADC channel, 60 samples:

60 × uint16	Board ADC input voltage	Units: ADC steps. To convert to volts, multiply by 0.000050354. (If the board mode is not equal to zero, other scaling values may apply.)
--------------------	-------------------------	---

If **any** USB interface board digital inputs are enabled, unsigned 16-bit integers record 60 samples from **all** digital inputs 0-15. If no digital inputs are enabled, these samples are not recorded.

60 × uint16	Board digital inputs	All 16 digital inputs are encoded bit-by-bit in each 16-bit word. For example, if digital inputs 0, 4, and 5 are high and the rest low, the uint16 value for this sample time will be $2^0 + 2^4 + 2^5 = 1 + 16 + 32 = 49$.
--------------------	----------------------	---

The m-file **read_Intan_RHD2000_file.m** includes code to extract the bit-by-bit information from these 16-bit words into individual digital waveforms.

“One File Per Signal Type” Format

This file format creates a subdirectory using the base filename provided, plus a date and time stamp. Global information is saved in a Standard Intan RHD2000 Header file, and all waveform data is saved in separate raw data files. Although the raw data files are divided by signal type (i.e., one file for all amplifier channels, another file for all auxiliary input channels, etc.), the

RHD2000-EVAL Application Note

file sizes can grow large quickly: If 64 amplifier channels were recorded at 20 kS/s for one hour, the amplifier data file would be 9.2 GB in size.

The raw data files written in this format are compatible with the NeuroScope open-source software for data viewing and analysis. (See <http://neuroscope.sourceforge.net> for more information on this third-party software.)

When using this file format, the following data files are written to the subdirectory:

Standard Intan RHD2000 Header file: info.rhd

This file contains the data listed in the Intan RHD2000 Standard Header described above: sampling rate, amplifier bandwidth, channel names, and other useful information. The information in this file may be read into MATLAB data structures using `read_Intan_RHD2000_file.m`, which is provided on the Intan Technologies web site.

Timestamp data file: time.dat

This file contains `int32`-type sequential integers (e.g., 0, 1, 2, 3...) corresponding to sample times indices, with zero marking the beginning of a recording or a trigger point. Time indices can be negative to denote pre-trigger times. Divide by the amplifier sampling rate (in Samples/s) to get a time vector with units of seconds.

The following MATLAB code reads a timestamp data file and creates a time vector with units of seconds:

```
fileinfo = dir('time.dat');
num_samples = fileinfo.bytes/4; % int32 = 4 bytes
fid = fopen('time.dat', 'r');
t = fread(fid, num_samples, 'int32');
fclose(fid);
t = t / frequency_parameters.amplifier_sample_rate; % sample rate from header file
```

The use of the `int32` data type means that this number will not “roll over” until total recording times exceed 19.8 hours with the maximum sample rate of 30 kS/s, or 29.8 hours with a sample rate of 20 kS/s.

Amplifier data file: amplifier.dat

This file contains a matrix of ADC samples from all enabled RHD2000 amplifier channels in `int16` format. For example, if four amplifier channels are enabled, data will be written in the following order:

amp1(t), amp2(t), amp3(t), amp4(t), amp1(t+1), amp2(t+1), amp3(t+1), amp4(t+1), amp1(t+2), amp2(t+2), ...

To convert to electrode voltage in microvolts, multiply by 0.195.

If no amplifier channels are enabled in the GUI, this file will not be written.

The following MATLAB code reads an amplifier data file and creates an electrode voltage matrix with units of microvolts:

```
num_channels = length(amplifier_channels); % amplifier channel info from header file
fileinfo = dir('amplifier.dat');
num_samples = fileinfo.bytes/(num_channels * 2); % int16 = 2 bytes
fid = fopen('amplifier.dat', 'r');
v = fread(fid, [num_channels, num_samples], 'int16');
fclose(fid);
v = v * 0.195; % convert to microvolts
```

Auxiliary input data file: auxiliary.dat

This file contains a matrix of ADC samples from all enabled RHD2000 auxiliary input channels in `uint16` format. To convert to volts, multiply by 0.0000374.

Although the RHD2000 interface software samples the RHD2000 auxiliary input channels at one-fourth the rate of the amplifiers, each auxiliary input sample is repeated four times in this file so that this data may easily be aligned with the timestamp vector in the `time.dat` file.

If no auxiliary input channels are enabled in the GUI, this file will not be written.

RHD2000-EVAL Application Note

The following MATLAB code reads an auxiliary input data file and creates a waveform matrix with units of volts:

```
num_channels = length(aux_input_channels); % aux input channel info from header file
fileinfo = dir('auxiliary.dat');
num_samples = fileinfo.bytes/(num_channels * 2); % uint16 = 2 bytes
fid = fopen('auxiliary.dat', 'r');
v = fread(fid, [num_channels, num_samples], 'uint16');
fclose(fid);
v = v * 0.0000374; % convert to volts
```

Supply voltage data file: supply.dat

This file contains a matrix of ADC samples from all enabled RHD2000 supply voltage sensor channels in **uint16** format. To convert to volts, multiply by 0.0000748.

Although the RHD2000 interface software samples the RHD2000 auxiliary input channels at one-sixtieth the rate of the amplifiers, each supply voltage sample is repeated sixty times in this file so that this data may easily be aligned with the timestamp vector in the **time.dat** file.

If no supply voltage channels are enabled in the GUI, this file will not be written.

The following MATLAB code reads a supply voltage data file and creates a waveform matrix with units of volts:

```
num_channels = length(supply_voltage_channels); % supply channel info from header file
fileinfo = dir('supply.dat');
num_samples = fileinfo.bytes/(num_channels * 2); % uint16 = 2 bytes
fid = fopen('supply.dat', 'r');
v = fread(fid, [num_channels, num_samples], 'uint16');
fclose(fid);
v = v * 0.0000748; % convert to volts
```

Board ADC input data file: analogin.dat

This file contains a matrix of ADC samples from the analog inputs on the USB interface board, in **uint16** format. To convert to volts, multiply by 0.000050354. (If the board mode is not equal to zero, other scaling values may apply.)

If no board ADC input channels are enabled in the GUI, this file will not be written.

The following MATLAB code reads a board ADC input data file and creates a waveform matrix with units of volts:

```
num_channels = length(board_adc_channels); % ADC input info from header file
fileinfo = dir('analogin.dat');
num_samples = fileinfo.bytes/(num_channels * 2); % uint16 = 2 bytes
fid = fopen('analogin.dat', 'r');
v = fread(fid, [num_channels, num_samples], 'uint16');
fclose(fid);
v = v * 0.000050354; % convert to volts
```

Board digital input data file: digitalin.dat

This file contains samples of digital inputs 0-15 on the USB interface board, in **uint16** format. All 16 digital inputs are encoded bit-by-bit in each 16-bit word. For example, if digital inputs 0, 4, and 5 are high and the rest low, the **uint16** value for this sample time will be $2^0 + 2^4 + 2^5 = 1 + 16 + 32 = 49$.

If no board digital input channels are enabled in the GUI, this file will not be written. If **any** USB interface board digital inputs are enabled, the **uint16** numbers in this file record data from **all** digital inputs 0-15.

The following MATLAB code reads a board digital input data file and creates vector of 16-bit words:

```
fileinfo = dir('digitalin.dat');
num_samples = fileinfo.bytes/2; % uint16 = 2 bytes
fid = fopen('digitalin.dat', 'r');
digital_word = fread(fid, num_samples, 'uint16');
fclose(fid);
```


RHD2000-EVAL Application Note

Board digital output data file: digitalout.dat

This file contains samples of digital outputs 0-15 on the USB interface board, in **uint16** format. All 16 digital inputs are encoded bit-by-bit in each 16-bit word. For example, if digital outputs 0, 4, and 5 are high and the rest low, the **uint16** value for this sample time will be $2^0 + 2^4 + 2^5 = 1 + 16 + 32 = 49$.

If the “Save Digital Outputs” box is not checked in the Select File Format dialog, this file will not be written.

The following MATLAB code reads a board digital output data file and creates vector of 16-bit words:

```
fileinfo = dir('digitalout.dat');  
num_samples = fileinfo.bytes/2; % uint16 = 2 bytes  
fid = fopen('digitalout.dat', 'r');  
digital_word = fread(fid, num_samples, 'uint16');  
fclose(fid);
```

“One File Per Channel” Format

This file format creates a subdirectory using the base filename provided, plus a date and time stamp. The subdirectory contains separate files for each waveform recorded by the RHD2000 USB interface; if 256 amplifier channels are connected to the system, then 256 individual amplifier data files will be written. This file format has the advantage of maintaining reasonable individual file sizes even for long recordings (a one-hour recording session at 30 kS/s would generate a 216 MB file for each enabled channel) while not dividing particular waveforms between multiple files.

When using this file format, the following data files are written to the subdirectory:

Standard Intan RHD2000 Header file: info.rhd

This file contains the data listed in the Intan RHD2000 Standard Header described above: sampling rate, amplifier bandwidth, channel names, and other useful information. The information in this file may be read into MATLAB data structures using **read_Intan_RHD2000_file.m**, which is provided on the Intan Technologies web site.

Timestamp data file: time.dat

This file contains **int32**-type sequential integers (e.g., 0, 1, 2, 3...) corresponding to sample times indices, with zero marking the beginning of a recording or a trigger point. Time indices can be negative to denote pre-trigger times. Divide by the amplifier sampling rate (in Samples/s) to get a time vector with units of seconds.

The use of the **int32** data type means that this number will not “roll over” until total recording times exceed 19.8 hours with the maximum sample rate of 30 kS/s, or 29.8 hours with a sample rate of 20 kS/s.

The following MATLAB code reads a timestamp data file and creates a time vector with units of seconds:

```
fileinfo = dir('time.dat');  
num_samples = fileinfo.bytes/4; % int32 = 4 bytes  
fid = fopen('time.dat', 'r');  
t = fread(fid, num_samples, 'int32');  
fclose(fid);  
t = t / frequency_parameters.amplifier_sample_rate; % sample rate from header file
```

Amplifier data files

Each amplifier data file has a filename that begins with **amp** followed by the SPI port letter and channel number. For example: **amp-A-000.dat**, **amp-C-063.dat**, or **amp-D-027.dat**.

Each amplifier data file contains the consecutive ADC samples from one enabled RHD2000 amplifier channel in **int16** format. To convert to electrode voltage in microvolts, multiply by 0.195.

The following MATLAB code reads an amplifier data file and creates an electrode voltage vector with units of microvolts:

```
fileinfo = dir('amp-B-003.dat');
```


RHD2000-EVAL Application Note

```
num_samples = fileinfo.bytes/2; % uint16 = 2 bytes
fid = fopen('amp-B-003.dat', 'r');
v = fread(fid, num_samples, 'uint16');
fclose(fid);
v = v * 0.195; % convert to microvolts
```

Auxiliary input data files

Each auxiliary input data file has a filename that begins with **aux** followed by the port and auxiliary channel: **AUX1**, **AUX2**, or **AUX3**. (If a dual-chip amplifier board is used, **AUX4**, **AUX5**, and **AUX6** may be present.) For example: **aux-A-AUX1.dat**, **aux-B-AUX3.dat**, or **aux-C-AUX2.dat**.

Each auxiliary input data file contains the consecutive ADC samples from one enabled RHD2000 auxiliary input channel in **uint16** format. To convert to volts, multiply by 0.0000374.

Although the RHD2000 interface software samples the RHD2000 auxiliary input channels at one-fourth the rate of the amplifiers, each auxiliary input sample is repeated four times in this file so that this data may easily be aligned with the timestamp vector in the **time.dat** file.

The following MATLAB code reads an auxiliary input data file and creates a waveform vector with units of volts:

```
fileinfo = dir('aux-C-AUX1.dat');
num_samples = fileinfo.bytes/2; % uint16 = 2 bytes
fid = fopen('aux-C-AUX1.dat', 'r');
v = fread(fid, num_samples, 'uint16');
fclose(fid);
v = v * 0.0000374; % convert to volts
```

Supply voltage data files

Each supply voltage data file has a filename that begins with **vdd** followed by the port and MISO channel: **VDD1** (or **VDD2** if a dual-chip amplifier board is used). For example: **vdd-A-VDD1.dat**, **vdd-B-VDD1.dat**, or **vdd-D-VDD2.dat**.

Each supply voltage data file contains the consecutive ADC samples from one enabled RHD2000 supply voltage sensor channel in **uint16** format. To convert to volts, multiply by 0.0000748.

Although the RHD2000 interface software samples the RHD2000 auxiliary input channels at one-sixtieth the rate of the amplifiers, each supply voltage sample is repeated sixty times in this file so that this data may easily be aligned with the timestamp vector in the **time.dat** file.

The following MATLAB code reads a supply voltage data file and creates a waveform vector with units of volts:

```
fileinfo = dir('vdd-D-VDD1.dat');
num_samples = fileinfo.bytes/2; % uint16 = 2 bytes
fid = fopen('vdd-D-VDD1.dat', 'r');
v = fread(fid, num_samples, 'uint16');
fclose(fid);
v = v * 0.0000748; % convert to volts
```

Board ADC input data files

Each board ADC input data file has a filename that begins with **board-ADC** followed by the channel number. For example: **board-ADC-00.dat**, **board-ADC-01.dat**, or **board-ADC-07.dat**.

Each board ADC input data file contains the consecutive ADC samples from one enabled analog input on the USB interface board, in **uint16** format. To convert to volts, multiply by 0.000050354. (If the board mode is not equal to zero, other scaling values may apply.)

The following MATLAB code reads a board ADC data file and creates a waveform vector with units of volts:

```
fileinfo = dir('board-ADC-00.dat');
num_samples = fileinfo.bytes/2; % uint16 = 2 bytes
fid = fopen('board-ADC-00.dat', 'r');
v = fread(fid, num_samples, 'uint16');
```

RHD2000-EVAL Application Note

```
fclose(fid);  
v = v * 0.000050354; % convert to volts
```

Board digital input data files

Each board digital input data file has a filename that begins with **board-DIN** followed by the channel number. For example: **board-DIN-00.dat**, **board-DIN-01.dat**, or **board-DIN-15.dat**.

Each board digital input data file contains the consecutive binary samples from one enabled digital input on the USB interface board, in uint16 format. Each **uint16** value in these files will be equal either to 0 or 1.

The following MATLAB code reads a board digital input data file and creates a waveform vector:

```
fileinfo = dir('board-DIN-07.dat');  
num_samples = fileinfo.bytes/2; % uint16 = 2 bytes  
fid = fopen('board-DIN-07.dat', 'r');  
din07 = fread(fid, num_samples, 'uint16');  
fclose(fid);
```

Board digital output data files

Each board digital output data file has a filename that begins with **board-DOUT** followed by the channel number. For example: **board-DOUT-00.dat**, **board-DOUT-01.dat**, or **board-DOUT-15.dat**.

Each board digital output data file contains the consecutive binary samples from one digital output on the USB interface board, in uint16 format. Each **uint16** value in these files will be equal either to 0 or 1.

The following MATLAB code reads a board digital output data file and creates a waveform vector:

```
fileinfo = dir('board-DOUT-00.dat');  
num_samples = fileinfo.bytes/2; % uint16 = 2 bytes  
fid = fopen('board-DOUT-00.dat', 'r');  
dout00 = fread(fid, num_samples, 'uint16');  
fclose(fid);
```

Handling Large Data Files

The RHD2000 Evaluation System supports up to 256 amplifier channels, plus several other analog and digital inputs, that may be sampled up to 30 kS/s/channel. This can quickly create enormous data files. The example MATLAB code shown above reads entire raw data files into memory, but it is also possible to read particular segments of data from a file using the MATLAB commands `fseek`, `ftell`, and `frewind` to move to specified positions in a file. Then `fread` may be used to read a subset of the data in the file beginning at that point. Similar functions are available in C++ and other programming languages.