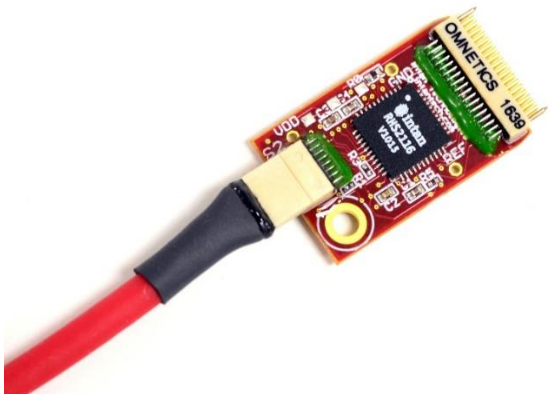
RHS

Application Note: Data File Formats

7 July 2017; updated 29 April 2022



This application note describes the file formats used by the Intan Stimulation/Recording Controller 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. The software supports 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 16-bit integer	-32,768 to 32,767
uint32	unsigned 32-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
ASCII String	ASCII unsigned 8-bit char array ending with 0	No length limit
Name Array	ASCII unsigned 8-bit char array	Exactly 5 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.

Some 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.

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 RHS Header

All Intan RHS data file formats make use of the Standard Intan RHS 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 RHS header has a filename ending with the .rhs prefix. These are binary files that begin with the following file type and version information.

DATA TYPE	NAME	DESCRIPTION
uint32	Intan RHS Header identifier	This "magic number" always has a value of 0xD69127AC to indicate a traditional Standard Intan RHS 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 lower settle bandwidth	Low cutoff frequency for amplifier settle (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 lower settle bandwidth	User-requested low cutoff frequency for amplifier settle (units: Hz).
single	Desired upper bandwidth	User-requested amplifier low-pass filter cutoff frequency (units: Hz).

The RHS2116 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_RHS2000_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.

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

Next are entries indicating the state of amp settle and charge recovery modes, which can improve stimulation recovery time.

int16	Amp settle mode	0: Switch lower bandwidth. 1: Traditional fast settle.
int16	Charge recovery mode	0: Current-limited charge recovery circuit. 1: Charge recovery switch.

Next are three floating-point numbers indicating the stimulation step size, maximum current supplied by the current-limited charge recovery circuit, and voltage to be used by the current-limited charge recovery circuits.

single	Stim step size	Step size of the 8-bit current-output DACs in each on-chip stimulator (units: A).
single	Charge recovery current limit	Maximum current supplied (per channel) by current-limited charge recovery circuit (units: A).
single	Charge recovery target voltage	Output voltage of an 8-bit DAC used to generate a voltage in the range of -1.225V to +1.215V used by current-limited charge recovery circuits (units: V).

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.

Next is a variable indicating whether or not the DC amplifier data of all enabled amplifiers are saved.

int16	DC amplifier data saved	0: DC amplifier data of all enabled channels are not saved. 1: DC amplifier data of all enabled channels are saved.

Next is the “board mode”. This integer is set by hardware, and should always be 14 for the Intan Stim/Record Controller.

int16	Board mode	Integer ranging from 0-15 indicating global properties of the hardware used to acquire the data. 14 corresponds to the Intan Stim/Record Controller.

Next is the name of the channel used for digital re-referencing. The waveform from this channel may be added to other amplifier channels to undo the effects of digital re-referencing, if desired. If hardware referencing was selected, this string is set to “n/a”.

QString	Reference channel name	Native channel name of the channel used as digital reference (e.g., “A-001” or “C-027”. If hardware referencing was selected, this string is set to “n/a”.

The next number indicates the number of "signal groups" present in the data file. This number is typically equal to eight: Port A, Port B, Port C, Port D, Board ADC Inputs, Board Digital Inputs, Board DAC Outputs, 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 board analog or digital inputs or outputs. There will typically be eight signal groups representing the eight items listed under "Ports" in the GUI.

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

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: RHS2000 amplifier channel. 3: Analog input channel. 4: Analog output channel. 5: Digital input channel. 6: Digital output channel.
int16	Channel enabled?	0: channel disabled. 1: channel enabled.
int16	Chip channel	RHS2000 channel number (0-15).
int16	Command stream	USB board data stream (0-7); each data stream supports up to 16 channels. Each RHS2216 chip uses an entire data stream.
int16	Board stream	USB board data stream (0-7); each data stream supports up to 16 channels. Each RHS2216 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 RHS 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 (RHS2000 amplifier channels, including stimulation current if stimulation was enabled, and DC amplifier channels, if DC amplifier data were selected to be saved, analog inputs and outputs, and digital inputs and outputs) to one file, along with the Standard Intan RHS 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 RHS Header and approximately N minutes of saved data. These .rhs data files may be read into MATLAB using **read_Intan_RHS2000_file.m**, which is provided on the Intan Technologies web site.

Immediately following the Standard Intan RHS Header in each data file is the waveform data. The traditional Intan file format saves waveforms in “data blocks” corresponding to the **Rhs2000DataBlock** object in the C++ code. Each data block contains data from **128** amplifier samples.

Each data block is organized as follows:

128 × 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 max sample rate 30 kS/s, or 29.8 hours with a sample rate of 20 kS/s.

For each enabled RHS2000 amplifier channel, 128 ADC samples:

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

If ‘DC amp data’ saved from the header is 1, for each enabled RHS2000 amplifier channel, 128 ADC samples. If ‘DC amp data saved’ is 0, this data is not present.

128 × uint16	DC amplifier voltage	Units: ADC steps. To convert to DC voltage in millivolts, first subtract 512 then multiply by 19.23.

For each enabled RHS2000 amplifier channel, 128 stimulation data words:

128 × uint16	Stimulation data	Stimulation current is stored in lower 9 bits. Units: ‘Stim Step Size’, set in header file. To convert to current in amps, multiply lowest 8 bits by ‘Stim Step Size’. Then, for each 16-bit unsigned integer entry, if the 9th bit is 1 (signifying a negative current), multiply the magnitude by -1. Bit 16 (the MSB) indicates a compliance limit was reached. Bit 15 is one if charge recovery is activated. Bit 14 is one if amplifier settle is activated. Bits 10-13 are unused and always zero.

For each board ADC channel, 128 samples:

128 × uint16	Board ADC input voltage	Units: ADC steps. To convert to volts, subtract 32768 and multiply by 0.0003125.

For each board DAC channel, 128 samples:

128 × uint16	Board DAC output voltage	Units: DAC steps. To convert to volts, subtract 32768 and multiply by 0.0003125.

If **any** board digital inputs are enabled, unsigned 16-bit integers record 128 samples from **all** digital inputs 0-15. If no digital inputs are enabled, these samples are not recorded.

128 × 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 $2^0 + 2^4 + 2^5 = 1 + 16 + 32 = 49$.

For each digital output channel, 128 samples:

128 × uint16	Board digital outputs	All 16 digital outputs 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 $2^0 + 2^4 + 2^5 = 1 + 16 + 32 = 49$.

The m-file **read_Intan_RHS2000_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 RHS 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, one file for all digital outputs, etc.), the 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 RHS Header file: info.rhs

This file contains the data listed in the Intan RHS 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_RHS2000_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 RHS2000 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
```

Board ADC input data file: analogin.dat

This file contains a matrix of ADC samples from the analog inputs on the board, in **uint16** format. To convert to volts, subtract 32768 and multiply by 0.0003125.

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 - 32768) * 0.0003125; % convert to volts
```

Board DAC output data file: analogout.dat

This file contains a matrix of DAC samples from the analog outputs on the board, in **uint16** format. To convert to volts, subtract 32768 and multiply by 0.0003125

If no board DAC output 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_dac_channels); % DAC output info from header file
fileinfo = dir('analogout.dat');
num_samples = fileinfo.bytes/(num_channels * 2); % uint16 = 2 bytes
fid = fopen('analogout.dat', 'r');
v = fread(fid, [num_channels, num_samples], 'uint16');
fclose(fid);
v = (v - 32768) * 0.0003125; % convert to volts
```

DC amplifier data file: dcamplifier.dat

This file contains a matrix of ADC samples from the DC amplifier inputs from all enabled RHS2000 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 DC voltage in millivolts, first subtract 512 then multiply by 19.23.

If no amplifier channels are enabled in the GUI, or if the "Save DC Amplifier Waveforms" box is not checked in the Select File Format dialog, this file will not be written.

The following MATLAB code reads a dc amplifier data file and creates a voltage matrix with units of millivolts:

```
num_channels = length(amplifier_channels); % DC channel info from header file
fileinfo = dir('dcamplifier.dat');
num_samples = fileinfo.bytes/(num_channels * 2); % uint16 = 2 bytes
fid = fopen('dcamplifier.dat', 'r');
v = fread(fid, [num_channels, num_samples], 'uint16');
fclose(fid);
v = (v - 512) * 19.23; % convert to millivolts
```

Board digital input data file: digitalin.dat

This file contains samples of digital inputs 0-15 on the 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** Stim/Record Controller 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);
```

Board digital output data file: digitalout.dat

This file contains samples of digital outputs 0-15 on the 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);
```

Stimulation output data file: stim.data

This file contains a matrix of stimulation currents applied to all enabled RHS2000 amplifier channels, in **uint16** format. For example, if four amplifier channels are enabled, data will be written in the following order:

stimdata1(t), stimdata2(t), stimdata3(t), stimdata4(t), stimdata1(t+1), stimdata2(t+1), stimdata3(t+1), stimdata4(t+1), …

Stimulation current is stored in lower 9 bits of stimdata. To convert to stimulation current in amps, multiply lowest 8 bits by 'Stim Step Size'. Then, for each 16-bit unsigned integer entry, if the 9th bit is 1 (signifying a negative current), multiply the magnitude by -1.

Bit 16 (the MSB) indicates a compliance limit was reached. Bit 15 is one if charge recovery is activated. Bit 14 is one if amplifier settle is activated. Bits 10-13 are unused and always zero.

If no amplifier channels are enabled in the GUI, this file will not be written. The following MATLAB code reads a stim data file and creates a current matrix with units of Amps:

```
num_channels = length(amplifier_channels);
fileinfo = dir('stim.dat');
num_samples = fileinfo.bytes/(num_channels * 2); % uint16 = 2 bytes
fid = fopen('stim.dat', 'r');
data = fread(fid, [num_channels, num_samples], 'uint16');
fclose(fid);
i = bitand(data, 255) * stim_parameters.stim_step_size; % current magnitude
sign = (128 - bitand(data, 256))/128; % convert sign bit to 1 or -1
i = i .* sign; % signed current in Amps
```

Spike data file: spike.dat

This file contains the timestamp and channel name of each spike that has been detected during recording. If snapshots have been saved, then each spike entry also includes the raw, unscaled data of that spike's snapshot.

The file begins with a header containing metadata about the recorded data, which is subsequently appended to spike-by-spike for each spike event that occurs.

Header

DATA TYPE	NAME	DESCRIPTION
uint32	Intan Spike File identifier	This "magic number" always has a value of 0x18F8474B to indicate an Intan Spike File (saved in One File Per Signal Type format).
uint16	Spike File version number	Which version this Spike File format follows
ASCII String	Base Filename	Base filename of recording session and time/date
ASCII String	Native channel names	Comma-separated list of all enabled amplifier native channel names
ASCII String	Custom channel names	Comma-separated list of all enabled amplifier custom channel names
single	Sample rate	Amplifier sample rate (units: Samples/s)
uint32	Pre-detect samples	Number of samples preceding a spike event to include in a snapshot
uint32	Post-detect samples	Number of samples following a spike event to include in a snapshot

Individual Spike Data (repeats until the end of the file)

DATA TYPE	NAME	DESCRIPTION
Name Array	Native channel name	Which channel this spike occurred on. Note: not null-terminated like the strings in the header, because this string is always exactly 5 characters
int32	Timestamp	Integer timestamp at which this spike occurred, with zero marking the beginning of the recording. Divide by the amplifier sampling rate (in Samples/s) to get a time with units of seconds.
uint8	Spike ID	With future releases of Intan software, could be expanded to assign a specific ID to a certain spike type. Currently, any non-zero ID indicates a spike.

If spike snapshots were saved, immediately following a spike's ID:

DATA TYPE	NAME	DESCRIPTION
N x uint16	Snapshot electrode voltage	Short snapshot of the waveform on which the spike was detected, where N is 'Pre-detect samples' + 'Post-detect samples'. Units: ADC steps. To convert to electrode voltage in microvolts, first subtract 32768 then multiply by 0.195.

The following MATLAB function reads a spike data file into a struct array containing each spike's name, timestamp, spike ID, and (if saved), snapshot:

```
function spikes = read_spike_data
total_bytes = dir('spike.dat').bytes;
fid = fopen('spike.dat', 'r');
if fread(fid, 1, 'uint32') ~= 0x18F8474B
    fprintf(1, 'Invalid magic number\n');
    return;
end
version_number = fread(fid, 1, 'uint16');
base_filename = readString(fid, 0);
native_channel_names = readString(fid, 0);
custom_channel_names = readString(fid, 0);
sample_rate = fread(fid, 1, 'single');
pre_detect_samples = fread(fid, 1, 'uint32');
post_detect_samples = fread(fid, 1, 'uint32');
N_snapshot = pre_detect_samples + post_detect_samples;

bytes_remaining = total_bytes - ftell(fid);
spike_struct = struct( ...
    'native_channel_name', {}, ...
    'timestamp', {}, ...
    'spike_id', {}, ...
    'snapshot', {} );
spikes = struct(spike_struct);
spikes_index = 1;

while bytes_remaining > 0
    spikes(spikes_index).native_channel_name = readString(fid, 5);
    spikes(spikes_index).timestamp = fread(fid, 1, 'int32');
    spikes(spikes_index).spike_id = fread(fid, 1, 'uint8');
    if N_snapshot > 0
        spikes(spikes_index).snapshot = (fread(fid, N_snapshot, 'uint16') ...
            - 32768) * 0.195;
    end
    bytes_remaining = total_bytes - ftell(fid);
    spikes_index = spikes_index + 1;
end
end

% Read a string. For null-terminated strings, num_chars should be 0. For
% known-length string, num_chars should be the length of the string.
function this_str = readString(fid, num_chars)
last_read_byte = 1;
read_bytes = [];
if num_chars == 0
    while last_read_byte ~= 0
        last_read_byte = fread(fid, 1, 'uint8');
        read_bytes = [read_bytes last_read_byte];
    end
    this_str = char(read_bytes(1:end-1));
else
    read_bytes = fread(fid, num_chars, 'uint8')';
    this_str = char(read_bytes);
end
end
```

"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 Stim/Record Controller; 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 RHS Header file: info.rhs

This file contains the data listed in the Intan RHS 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_RHS2000_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 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 RHS2000 amplifier channel in **int16** format. 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 vector with units of microvolts:

```
fileinfo = dir('amp-B-003.dat'); % amplifier channel data
num_samples = fileinfo.bytes/2; % int16 = 2 bytes
fid = fopen('amp-B-003.dat', 'r');
v = fread(fid, num_samples, 'int16');
fclose(fid);
v = v * 0.195; % convert to microvolts
```

Board ADC input data files

Each board ADC input data file has a filename that begins with **board-ANALOG-IN** followed by the channel number. For example: **board-ANALOG-IN-1.dat**, **board-ANALOG-IN-3.dat**, or **board-ANALOG-IN-6.dat**.

Each board ADC input data file contains the consecutive ADC samples from one enabled analog input on the Stim/Record Controller board, in **uint16** format. To convert to volts, subtract 32768 and multiply by 0.0003125.

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 data file and creates a waveform vector with units of volts:

```
fileinfo = dir('board-ANALOG-IN-1.dat'); % ADC input data
num_samples = fileinfo.bytes/2; % uint16 = 2 bytes
fid = fopen('board-ANALOG-IN-1.dat', 'r');
v = fread(fid, num_samples, 'uint16');
fclose(fid);
v = (v - 32768) * 0.0003125; % convert to volts
```

Board DAC output data files

Each board DAC output data file has a filename that begins with **board-ANALOG-OUT** followed by the channel number. For example: **board-ANALOG-OUT-1.dat**, **board-ANALOG-OUT-3.dat**, or **board-ANALOG-IN-6.dat**.

Each board DAC output data file contains the consecutive DAC samples from one enabled analog output on the Stim/Record Controller board, in **uint16** format. To convert to volts, subtract 32768 and multiply by 0.0003125.

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

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

```
fileinfo = dir('board-ANALOG-OUT-1.dat'); % DAC output data
num_samples = fileinfo.bytes/2; % uint16 = 2 bytes
fid = fopen('board-ANALOG-OUT-1.dat', 'r');
v = fread(fid, num_samples, 'uint16');
fclose(fid);
v = (v - 32768) * 0.0003125; % convert to volts
```

DC amplifier data file: dcamplifier.dat

Each amplifier data has a filename that begins with **dc** followed by the SPI port letter and channel number. For example: **dc-A-000.dat**, **dc-B-002.dat**, or **dc-B-027.dat**.

Each amplifier data file contains the consecutive DC samples from one enabled RHS2000 amplifier channel in **uint16** format. To convert to DC voltage in millivolts, first subtract 512 then multiply by 19.23.

If no amplifier channels are enabled in the GUI, or if the "Save DC Amplifier Waveforms" box is not checked in the Select File Format dialog, these files will not be written.

The following MATLAB code reads a dc amplifier data file and creates a voltage matrix with units of millivolts:

```
fileinfo = dir('dc-B-000.dat'); % DC channel data
num_samples = fileinfo.bytes/2; % uint16 = 2 bytes
fid = fopen('dc-B-000.dat', 'r');
v = fread(fid, num_samples, 'uint16');
fclose(fid);
v = (v - 512) * 19.23; % convert to millivolts
```

Board digital input data files

Each board digital input data file has a filename that begins with **board-DIGITAL-IN** followed by the channel number. For example: **board-DIGITAL-IN-01.dat**, **board-DIGITAL-IN-03.dat**, or **board-DIGITAL-IN-06.dat**.

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

Each board digital input data file contains the consecutive binary samples from one enabled digital input on the 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-DIGITAL-IN-01.dat'); % digital input data
num_samples = fileinfo.bytes/2; % uint16 = 2 bytes
fid = fopen('board-DIGITAL-IN-01.dat', 'r');
din01 = 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-DIGITAL-OUT** followed by the channel number. For example: **board-DIGITAL-OUT-01.dat**, **board-DIGITAL-OUT-03.dat**, or **board-DIGITAL-OUT-06.dat**.

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

Each board digital output data file contains the consecutive binary samples from one digital output on the 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-DIGITAL-OUT-01.dat'); % digital output data
num_samples = fileinfo.bytes/2; % uint16 = 2 bytes
fid = fopen('board-DIGITAL-OUT-01.dat', 'r');
dout01 = fread(fid, num_samples, 'uint16');
fclose(fid);
```

Stimulation output current files

Each stimulation output current file has a filename that begins with **stim** followed by the SPI port letter and channel number. For example: **stim-A-001.dat**, **stim-B-003.dat**, **stim-C-031.dat**.

Stimulation current is stored in lower 9 bits of each saved 16-bit word. To convert to stimulation current in amps, multiply lowest 8 bits by ‘Stim Step Size’. Then, for each 16-bit unsigned integer entry, if the 9th bit is 1 (signifying a negative current), multiply the magnitude by -1.

Bit 16 (the MSB) indicates a compliance limit was reached. Bit 15 is one if charge recovery is activated. Bit 14 is one if amplifier settle is activated. Bits 10-13 are unused and always zero.

If no amplifier channels are enabled in the GUI, this file will not be written. The following MATLAB code reads a stim data file and creates a current matrix with units of Amps:

```
fileinfo = dir('stim-B-001.dat');
num_samples = fileinfo.bytes/2; % uint16 = 2 bytes
fid = fopen('stim-B-001.dat', 'r');
data = fread(fid, num_samples, 'uint16');
fclose(fid);
i = bitand(data, 255) * stim_parameters.stim_step_size; % current magnitude
sign = (128 - bitand(data, 256))/128; % convert sign bit to 1 or -1
i = i .* sign; % signed current in Amps
```

Spike data files

Each spike data file has a filename that begins with **spike** followed by the channel number. For example: **spike-A-000.dat, spike-A-001.dat, or spike-B-030.dat**.

This file contains the timestamp of each spike on this channel that has been detected during recording. If snapshots have been saved, then each spike entry also includes the raw, unscaled data of that spike’s snapshot.

The file begins with a header containing metadata about the recorded data, which is subsequently appended to spike-by-spike for each spike event that occurs on this channel.

Header

DATA TYPE	NAME	DESCRIPTION
uint32	Intan Spike File identifier	This “magic number” always has a value of 0x18F88C00 to indicate an Intan Spike File (saved in One File Per Channel format).
uint16	Spike File version number	Which version this Spike File format follows
ASCII String	Base Filename	Base filename of recording session and time/date

ASCII String	Native channel name	This channel's native channel name
ASCII String	Custom channel name	This channel's custom channel name
single	Sample rate	Amplifier sample rate (units: Samples/s)
uint32	Pre-detect samples	Number of samples preceding a spike event to include in a snapshot
uint32	Post-detect samples	Number of samples following a spike event to include in a snapshot

Individual Spike Data (repeats until the end of the file)

DATA TYPE	NAME	DESCRIPTION
int32	Timestamp	Integer timestamp at which this spike occurred, with zero marking the beginning of the recording. Divide by the amplifier sampling rate (in Samples/s) to get a time with units of seconds.
uint8	Spike ID	With future releases of Intan software, could be expanded to assign a specific ID to a certain spike type. Currently, any non-zero ID indicates a spike.

If spike snapshots were saved, immediately following a spike's ID:

DATA TYPE	NAME	DESCRIPTION
N x uint16	Snapshot electrode voltage	Short snapshot of the waveform on which the spike was detected, where N is 'Pre-detect samples' + 'Post-detect samples'. Units: ADC steps. To convert to electrode voltage in microvolts, first subtract 32768 then multiply by 0.195.

The following MATLAB function reads a single channel's spike data file into a struct array containing each spike's timestamp, spike ID, and (if saved), snapshot:

```
function spikes = read_spike_data
filename = 'spike-A-000.dat'; % Change this string to read different channels
total_bytes = dir(filename).bytes;
fid = fopen(filename, 'r');
if fread(fid, 1, 'uint32') ~= 0x18F88C00
    fprintf(1, 'Invalid magic number\n');
    return;
end
version_number = fread(fid, 1, 'uint16')
base_filename = readString(fid, 0)
native_name = readString(fid, 0)
custom_name = readString(fid, 0)
sample_rate = fread(fid, 1, 'single')
pre_detect_samples = fread(fid, 1, 'uint32');
post_detect_samples = fread(fid, 1, 'uint32');
N_snapshot = pre_detect_samples + post_detect_samples;

bytes_remaining = total_bytes - ftell(fid);
spike_struct = struct( ...
    'timestamp', {}, ...
    'spike_id', {}, ...
    'snapshot', {} );
spikes_index = 1;

while bytes_remaining > 0
    spikes(spikes_index).timestamp = fread(fid, 1, 'int32');
    spikes(spikes_index).spike_id = fread(fid, 1, 'uint8');
    if N_snapshot > 0
        spikes(spikes_index).snapshot = (fread(fid, N_snapshot, 'uint16') ...
            - 32768) * 0.195;
    end
    bytes_remaining = total_bytes - ftell(fid);
    spikes_index = spikes_index + 1;
end
end

% Read a string. For null-terminated strings, num_chars should be 0. For
% known-length string, num_chars should be the length of the string.
function this_str = readString(fid, num_chars)
last_read_byte = 1;
read_bytes = [];
if num_chars == 0
    while last_read_byte ~= 0
        last_read_byte = fread(fid, 1, 'uint8');
        read_bytes = [read_bytes last_read_byte];
    end
else
    read_bytes = fread(fid, num_chars, 'uint8')';
end
this_str = char(read_bytes);
end
```

Handling Large Data Files

The Intan Stimulation/Recording Controller supports up to 128 amplifier/stimulator 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.