Monarch Documentation

Release v3.8.2

Project 8 Collaboration

Sep 20, 2021

Contents

Contents:

CHAPTER 1

EggStandards

1.1 Egg v1 File Standard

Usage history: 2011

Relevant software: - Katydid maintains the ability to read these files (called "2011" egg files) - Very old versions of Monarch would be able to read and write these files.

1.1.1 File Structure

1. Prelude
2. Header
3. Records

1.1.2 Prelude

This is 8-byte word describing the size of the size of the header in bytes.

1.1.3 Header

The header consists of information describing the contents of the file formatted in XML as follows:

```
<header>
  <data_format id="[Frame ID size]" ts="[timestamp size]" data="[record size]" />
  <digitizer rate="[acq'n rate]" />
  <run length="[length]" />
</header>
```

The data_format node describes the size of the various parts of the record in bytes.

The digitizer node includes the acquisition rate in MHz.

The run node includes the length of the run in ms.

1.1.4 Record

Following the header, records are written to disk sequentially.

The header for each record includes an ID number and timestamp.

The data section of the record is the array of digitized 1-byte samples.

1.2 Egg v2 File Standard

Usage history: 2012-2014

Relevant software: - Monarch v2.6.6 or newer (Monarch v3 will retain the ability to read Egg v2 files)

1.2.1 File Structure

1. Prelude
2. Header
3. Records

1.2.2 Prelude

This is 4-byte word describing the size of the size of the header in bytes.

1.2.3 Header

The header is de/serialized using Google Protocol Buffers.

The fields are specified in MonarchHeader.proto, and include the following:

- filename (string; required) – Filename
- acqRate (double; required) – Acquisition rate in MHz
- acqMode (uint32; required) – Acquisition mode describing how many channels were used:
  - 1 == one channel
  - 2 == two channels
- acqTime (uint32; required) – Run length in ms
- recSize (uint32; required) – Number of bytes in each record (sum of both channels in 2-channel mode)
- runDate (string; optional) – Timestamp string
- runInfo (string; optional) – Description of the run; in files created by Mantis this typically included the Mantis client and server configurations in JSON format.
- runSource (enum; optional) – Run source information (not typically used)

  - 0 == Mantis
  - 1 == Simulation
- runType (enum; optional) – Run content information (not typically used)
  - 0 == Background
  - 1 == Signal
  - 999 == Other
- formatMode (enum; optional) – How data from multiple channels was arranged:
  - 0 == Single
  - 1 == Multiple, separate
  - 2 == Multiple, interleaved
- dataTypeSize (uint32; optional) – Size of the integer data type used to make the data
- bitDepth (uint32; optional) – Number of bits used to digitize each sample
- voltageMin (double; optional) – Minimum voltage accepted by the digitizer
- voltageRange (double; optional) – Voltage range accepted by the digitizer (above the minimum)

1.2.4 Record

Following the header, records are written to disk sequentially.

The header for each record includes an acquisition ID, a record ID, and a timestamp. The acquisition ID is used to determine which records are part of the same acquisition (i.e. are contiguous in time). The record ID is an integer that counts up sequentially for each record. The timestamp is a integer number of nanoseconds from the start of the run.

The data section of the record is the array of digitized samples. By default, the Monarch interface presents the data array as a byte array (uint8_t), though the actual data can be larger integer types (e.g. uint16_t).

1.3 Egg v3.0.0 File Standard

Usage history:

1/2015

- First version

5/8/2015

- v3.0.0 finalized

Relevant software:

- Monarch v3

Files are written in HDF5 format. All group, dataset, and attribute names are lower case, with words separated by underscores. Numerical values in the file are stored in little-endian format.

1.3.1 File Structure

Version 3 egg files are designed to accept data from multiple sources that may or may not be taking data in a synchronized way. It is also can also record both integer and floating-point data.

Data from each source are divided into *acquisitions* and *records*. A *record* consists of a set number of data samples. An *acquisition* is the set of records that are contiguous in time.

A single source of data (e.g. a digitizer channel) is called a *channel* in the egg file. However, some devices produce data from multiple channels combined together; therefore data are actually recorded in the egg file in *streams*, where each stream contains one or more channel of data.

In a stream with multiple channels, the individual samples from each channel can either be *separate* or *interleaved*. For example, one record of data from a device with two channels, A and B, recorded separately in a single stream would look like this in the stream:

```
AAAAABBBBB
```

One record of data from a device with two channels, C and D, recorded interleaved in a single stream would look like this:

```
CDCDCDCDCD
```

In these cases, the record length for each channel is 5, and the record length for each stream is 10.

Header information describing the run is stored in the file attributes. Channel and stream header information is stored in the attributes for each channel and stream group.

```
File (w/ attr)
|
+-- channels (group)
|   |
|   +-- channel0 (group w/ attr)
|   |
|   +-- channel1 (group w/ attr)
|   |
|   +-- . . .
|
+-- streams (group)
    |
    +-- stream0 (group w/ attr)
    |   |
    |   +-- acquisitions (group)
    |       |
    |       +-- 0 (dataset w/ attr)
    |       |
    |       +-- 1 (dataset w/ attr)
    |       |
    |       +-- . . .
    |
    +-- stream1 (group w/ attr)
        |
        +-- acquisitions (group)
            |
            +-- 0 (dataset w/ attr)
            |
            +-- 1 (dataset w/ attr)
            |
            +-- . . .
```

1.3.2 File Attributes

These attributes constitute the header for the file, and describe its contents. They consist of information that apply to the all streams and channels.

String attributes have a maximum length of 65536 characters (64 kB).

- egg_version (string) – Egg file version
- filename (string) – Filename
- run_duration (uint32) – Length of the run in ms
- timestamp (string) – Timestamp string
- description (string) – Description of the run
- n_channels (uint32) – Number of channels present
- n_streams (uint32) – Number of streams used
- channel_streams (vector< uint32 >) – Map of channel number to stream number
- channel_coherence (vector< vector< bool > >) – Matrix of channel coherence (stored as uint8_t)

1.3.3 Stream Attributes

Each stream group has a set of attributes needed to fully describe the stream. Information there is common to all of the channels included in that stream.

String attributes have a maximum length of 65536 characters (64 kB).

- number (uint32) – Sequential integer used to uniquely identify each stream in the file; the stream's group will be named "stream[Number]".
- source (string) – Device used to produce the stream
- n_channels (uint32) – Number of channels in the stream
- channels (vector< uint32 >) – Global channel numbers for the channels in this stream
- channel_format (uint32) – Whether the samples from multiple channels are interleaved or separate (either is valid for single-channel streams)
  - 0 == interleaved
  - 1 == separate
- acquisition_rate (uint32) – Acquisition rate in MHz
- record_size (uint32) – Number of samples in each channel record (stream record size = # channels * record size)
- data_type_size (uint32) – Number of bytes used to represent each sample (whether integer or floating point)
- data_format_type (uint32) – Whether the data is analog or digitized
  - 0 == digitized
  - 1 == analog
- bit_depth (uint32) – Number of bits with which the data was digitized
- n_acquisitions (uint32) – Number of acquisitions recorded
- n_records (uint32) – Number of records recorded

1.3.4 Channel Attributes

While much of the information regarding multiple channels in a single stream will be the same for all of those channels, some can certainly be different. Therefore each channel is given its own header information.

Some information is duplicated between channel and stream headers. This is for convenience, since when writing files the user cares about how the data is divided into streams, and when reading files most users will care about how the data is divided into channels.

String attributes have a maximum length of 65536 characters (64 kB).

- number (uint32) – Sequential integer used to uniquely identify each channel in the file; the channel's group will be named "channel[Number]".
- source (string) – Device used to produce the channel
- acquisition_rate (uint32) – Acquisition rate in MHz
- record_size (uint32) – Number of samples in each channel record (stream record size = # channels * record size)
- data_type_size (uint32) – Number of bytes used to represent each sample (whether integer or floating point)
- data_format_type (uint32) – Whether the data is analog or digitized
  - 0 == digitized
  - 1 == analog
- bit_depth (uint32) – Number of bits with which the data was digitized
- voltage_offset (double) – Voltage value represented by an ADC value of 0 (the minimum voltage for unsigned digital data, and the center voltage for signed digital data)
- voltage_range (double) – Voltage range accepted above the minimum
- dac_gain (double) – Gain value needed to convert digital values to analog (analog = digital * gain + voltage_min)
- frequency_min (double) – For band-pass data, minimum frequency recorded
- frequency_range (double) – For band-pass data, range of frequencies recorded above the minimum

1.3.5 Acquisitions

Each stream contains an "acquisitions" group that holds the acquisition datasets. Each acquisition dataset is named simply with the number of the acquisition, starting from 0 and incrementing for each new acquisition.

Each acquisition has a single attribute:

- n_records (uint32) – Number of records in that acquisition

Here are some examples how data might be arranged in an acquisition dataset:

- Single channel; channel record size == 10; 3 records:

```
AAAAAAAAAA  <-- record 0
AAAAAAAAAA  <-- record 1
AAAAAAAAAA  <-- record 2
```

- Two channels; channel record size == 5; separate samples; 2 records:

```
AAAAABBBBB
AAAAABBBBB
```

- Three channels; channel record size == 5; interleaved samples; 2 records:

```
ABCABCABCABCABC
ABCABCABCABCABC
```

1.3.6 Known Issues

- Multi-channel, multi-sample, floating-point data doesn't save properly. A multi-channel RSA might use this format, but we do not currently have such a device.

1.4 Egg v3.1.0 File Standard

Usage history:

1/2015

- First version

5/8/2015

- v3.0.0 finalized

11/19/2015

- v3.1.0 finalized – Added Bit Alignment

Relevant software:

- Monarch v3

Files are written in HDF5 format. All group, dataset, and attribute names are lower case, with words separated by underscores. Numerical values in the file are stored in little-endian format.

1.4.1 File Structure

Version 3 egg files are designed to accept data from multiple sources that may or may not be taking data in a synchronized way. It is also can also record both integer and floating-point data.

Data from each source are divided into *acquisitions* and *records*. A *record* consists of a set number of data samples. An *acquisition* is the set of records that are contiguous in time.

A single source of data (e.g. a digitizer channel) is called a *channel* in the egg file. However, some devices produce data from multiple channels combined together; therefore data are actually recorded in the egg file in *streams*, where each stream contains one or more channel of data.

In a stream with multiple channels, the individual samples from each channel can either be *separate* or *interleaved*. For example, one record of data from a device with two channels, A and B, recorded separately in a single stream would look like this in the stream:

```
AAAAABBBBB
```

One record of data from a device with two channels, C and D, recorded interleaved in a single stream would look like this:

```
CDCDCDCDCD
```

In these cases, the record length for each channel is 5, and the record length for each stream is 10.

Header information describing the run is stored in the file attributes. Channel and stream header information is stored in the attributes for each channel and stream group.

```
File (w/ attr)
|
+-- channels (group)
|   |
|   +-- channel0 (group w/ attr)
|   |
|   +-- channel1 (group w/ attr)
|   |
|   +-- . . .
|
+-- streams (group)
    |
    +-- stream0 (group w/ attr)
    |   |
    |   +-- acquisitions (group)
    |       |
    |       +-- 0 (dataset w/ attr)
    |       |
    |       +-- 1 (dataset w/ attr)
    |       |
    |       +-- . . .
    |
    +-- stream1 (group w/ attr)
        |
        +-- acquisitions (group)
            |
            +-- 0 (dataset w/ attr)
            |
            +-- 1 (dataset w/ attr)
            |
            +-- . . .
```

1.4.2 File Attributes

These attributes constitute the header for the file, and describe its contents. They consist of information that apply to the all streams and channels.

String attributes have a maximum length of 65536 characters (64 kB).

- egg_version (string) – Egg file version
- filename (string) – Filename
- run_duration (uint32) – Length of the run in ms
- timestamp (string) – Timestamp string
- description (string) – Description of the run
- n_channels (uint32) – Number of channels present
- n_streams (uint32) – Number of streams used
- channel_streams (vector< uint32 >) – Map of channel number to stream number
- channel_coherence (vector< vector< bool > >) – Matrix of channel coherence (stored as uint8_t)

1.4.3 Stream Attributes

Each stream group has a set of attributes needed to fully describe the stream. Information there is common to all of the channels included in that stream.

String attributes have a maximum length of 65536 characters (64 kB).

- number (uint32) – Sequential integer used to uniquely identify each stream in the file; the stream's group will be named "stream[Number]".
- source (string) – Device used to produce the stream
- n_channels (uint32) – Number of channels in the stream
- channels (vector< uint32 >) – Global channel numbers for the channels in this stream
- channel_format (uint32) – Whether the samples from multiple channels are interleaved or separate (either is valid for single-channel streams)
  - 0 == interleaved
  - 1 == separate
- acquisition_rate (uint32) – Acquisition rate in MHz
- record_size (uint32) – Number of samples in each channel record (stream record size = # channels * record size)
- data_type_size (uint32) – Number of bytes used to represent each sample (whether integer or floating point)
- data_format_type (uint32) – Whether the data is analog or digitized
  - 0 == digitized
  - 1 == analog
- bit_depth (uint32) – Number of bits with which the data was digitized
- bit_alignment (uint32) – Whether the bits within each sample are left-aligned or right-aligned within the sample data word (if the bit depth is less that the data type size)
  - 0 == left-aligned
  - 1 == right-aligned
- n_acquisitions (uint32) – Number of acquisitions recorded
- n_records (uint32) – Number of records recorded

1.4.4 Channel Attributes

While much of the information regarding multiple channels in a single stream will be the same for all of those channels, some can certainly be different. Therefore each channel is given its own header information.

Some information is duplicated between channel and stream headers. This is for convenience, since when writing files the user cares about how the data is divided into streams, and when reading files most users will care about how the data is divided into channels.

String attributes have a maximum length of 65536 characters (64 kB).

- number (uint32) – Sequential integer used to uniquely identify each channel in the file; the channel's group will be named "channel[Number]".
- source (string) – Device used to produce the channel
- acquisition_rate (uint32) – Acquisition rate in MHz

- record_size (uint32) – Number of samples in each channel record (stream record size = # channels * record size)
- data_type_size (uint32) – Number of bytes used to represent each sample (whether integer or floating point)
- data_format_type (uint32) – Whether the data is analog or digitized
  - 0 == digitized
  - 1 == analog
- bit_depth (uint32) – Number of bits with which the data was digitized
- bit_alignment (uint32) – Whether the bits within each sample are left-aligned or right-aligned within the sample data word (if the bit depth is less that the data type size)
  - 0 == left-aligned
  - 1 == right-aligned
- voltage_offset (double) – Voltage value represented by an ADC value of 0 (the minimum voltage for unsigned digital data, and the center voltage for signed digital data)
- voltage_range (double) – Voltage range accepted above the minimum
- dac_gain (double) – Gain value needed to convert digital values to analog (analog = digital * gain + voltage_min)
- frequency_min (double) – For band-pass data, minimum frequency recorded
- frequency_range (double) – For band-pass data, range of frequencies recorded above the minimum

1.4.5 Acquisitions

Each stream contains an "acquisitions" group that holds the acquisition datasets. Each acquisition dataset is named simply with the number of the acquisition, starting from 0 and incrementing for each new acquisition.

Each acquisition has a single attribute:

- n_records (uint32) – Number of records in that acquisition

Here are some examples how data might be arranged in an acquisition dataset:

- Single channel; channel record size == 10; 3 records:

```
AAAAAAAAAA  <-- record 0
AAAAAAAAAA  <-- record 1
AAAAAAAAAA  <-- record 2
```

- Two channels; channel record size == 5; separate samples; 2 records:

```
AAAAABBBBB
AAAAABBBBB
```

- Three channels; channel record size == 5; interleaved samples; 2 records:

```
ABCABCABCABCABC
ABCABCABCABCABC
```

1.4.6 Known Issues

- Multi-channel, multi-sample, floating-point data doesn't save properly. A multi-channel RSA might use this format, but we do not currently have such a device.

1.5 Egg v3.2.0 File Standard

Usage history:

1/2015

- First version

5/8/2015

- v3.0.0 finalized

11/19/2015

- v3.1.0 finalized – Added Bit Alignment

3/3/2016

- v3.2.0 finalized – Added tracking of record time and ID

Relevant software:

- Monarch v3

Files are written in HDF5 format. All group, dataset, and attribute names are lower case, with words separated by underscores. Numerical values in the file are stored in little-endian format.

1.5.1 File Structure

Version 3 egg files are designed to accept data from multiple sources that may or may not be taking data in a synchronized way. It is also can also record both integer and floating-point data.

Data from each source are divided into *acquisitions* and *records*. A *record* consists of a set number of data samples. An *acquisition* is the set of records that are contiguous in time.

A single source of data (e.g. a digitizer channel) is called a *channel* in the egg file. However, some devices produce data from multiple channels combined together; therefore data are actually recorded in the egg file in *streams*, where each stream contains one or more channel of data.

In a stream with multiple channels, the individual samples from each channel can either be *separate* or *interleaved*. For example, one record of data from a device with two channels, A and B, recorded separately in a single stream would look like this in the stream:

```
AAAAABBBBB
```

One record of data from a device with two channels, C and D, recorded interleaved in a single stream would look like this:

```
CDCDCDCDCD
```

In these cases, the record length for each channel is 5, and the record length for each stream is 10.

Header information describing the run is stored in the file attributes. Channel and stream header information is stored in the attributes for each channel and stream group.

```
File (w/ attr)
|
+-- channels (group)
|   |
|   +-- channel0 (group w/ attr)
```

(continues on next page)

(continued from previous page)

```
|   |
|   +-- channel1 (group w/ attr)
|   |
|   +-- . . .
|
+-- streams (group)
    |
    +-- stream0 (group w/ attr)
    |   |
    |   +-- acquisitions (group)
    |       |
    |       +-- 0 (dataset w/ attr)
    |       |
    |       +-- 1 (dataset w/ attr)
    |       |
    |       +-- . . .
    |
    +-- stream1 (group w/ attr)
        |
        +-- acquisitions (group)
            |
            +-- 0 (dataset w/ attr)
            |
            +-- 1 (dataset w/ attr)
            |
            +-- . . .
```

1.5.2 File Attributes

These attributes constitute the header for the file, and describe its contents. They consist of information that apply to the all streams and channels.

String attributes have a maximum length of 65536 characters (64 kB).

- egg_version (string) – Egg file version
- filename (string) – Filename
- run_duration (uint32) – Length of the run in ms
- timestamp (string) – Timestamp string
- description (string) – Description of the run
- n_channels (uint32) – Number of channels present
- n_streams (uint32) – Number of streams used
- channel_streams (vector< uint32 >) – Map of channel number to stream number
- channel_coherence (vector< vector< bool > >) – Matrix of channel coherence (stored as uint8_t)

1.5.3 Stream Attributes

Each stream group has a set of attributes needed to fully describe the stream. Information there is common to all of the channels included in that stream.

String attributes have a maximum length of 65536 characters (64 kB).

- number (uint32) – Sequential integer used to uniquely identify each stream in the file; the stream's group will be named "stream[Number]".
- source (string) – Device used to produce the stream
- n_channels (uint32) – Number of channels in the stream
- channels (vector< uint32 >) – Global channel numbers for the channels in this stream
- channel_format (uint32) – Whether the samples from multiple channels are interleaved or separate (either is valid for single-channel streams)
  - 0 == interleaved
  - 1 == separate
- acquisition_rate (uint32) – Acquisition rate in MHz
- record_size (uint32) – Number of samples in each channel record (stream record size = # channels * record size)
- data_type_size (uint32) – Number of bytes used to represent each sample (whether integer or floating point)
- data_format_type (uint32) – Whether the data is analog or digitized
  - 0 == digitized
  - 1 == analog
- bit_depth (uint32) – Number of bits with which the data was digitized
- bit_alignment (uint32) – Whether the bits within each sample are left-aligned or right-aligned within the sample data word (if the bit depth is less that the data type size)
  - 0 == left-aligned
  - 1 == right-aligned
- n_acquisitions (uint32) – Number of acquisitions recorded
- n_records (uint32) – Number of records recorded

1.5.4 Channel Attributes

While much of the information regarding multiple channels in a single stream will be the same for all of those channels, some can certainly be different. Therefore each channel is given its own header information.

Some information is duplicated between channel and stream headers. This is for convenience, since when writing files the user cares about how the data is divided into streams, and when reading files most users will care about how the data is divided into channels.

String attributes have a maximum length of 65536 characters (64 kB).

- number (uint32) – Sequential integer used to uniquely identify each channel in the file; the channel's group will be named "channel[Number]".
- source (string) – Device used to produce the channel
- acquisition_rate (uint32) – Acquisition rate in MHz
- record_size (uint32) – Number of samples in each channel record (stream record size = # channels * record size)
- data_type_size (uint32) – Number of bytes used to represent each sample (whether integer or floating point)
- data_format_type (uint32) – Whether the data is analog or digitized
  - 0 == digitized

  - 1 == analog
- bit_depth (uint32) – Number of bits with which the data was digitized
- bit_alignment (uint32) – Whether the bits within each sample are left-aligned or right-aligned within the sample data word (if the bit depth is less that the data type size)
  - 0 == left-aligned
  - 1 == right-aligned
- voltage_offset (double) – Voltage value represented by an ADC value of 0 (the minimum voltage for unsigned digital data, and the center voltage for signed digital data)
- voltage_range (double) – Voltage range accepted above the minimum
- dac_gain (double) – Gain value needed to convert digital values to analog (analog = digital * gain + voltage_min)
- frequency_min (double) – For band-pass data, minimum frequency recorded
- frequency_range (double) – For band-pass data, range of frequencies recorded above the minimum

1.5.5 Acquisitions

Each stream contains an "acquisitions" group that holds the acquisition datasets. Each acquisition dataset is named simply with the number of the acquisition, starting from 0 and incrementing for each new acquisition.

Each acquisition has the following attributes:

- first_rec_time (uint64) – Time since the start of the run of the first record in the acquisition (in ns)
- first_rec_id (uint64) – Sequential integer used to identify the first record in the acquisition
- n_records (uint32) – Number of records in that acquisition

When reading a file, the time and ID for each record are calculated from those of the first record in the acquisition. When writing a file, only the time and ID of the first record in the acquisition are retained.

Backwards compatibility is maintained for files that did not retain the first-time or first-ID information. The first time and first ID in every acquisition will be 0, and they will increment by the record length and 1, respectively. However, they start over at 0 for every acquisition, and therefore should not be trusted. If the first record in an acquisition has a time of 0 (realistically this will never happen in reality), you know that you're reading a file without the time and ID information stored properly, and you should therefore ignore the record time and ID parameters reported by Monarch.

Here are some examples how data might be arranged in an acquisition dataset:

- Single channel; channel record size == 10; 3 records:

```
AAAAAAAAAA  <-- record 0
AAAAAAAAAA  <-- record 1
AAAAAAAAAA  <-- record 2
```

- Two channels; channel record size == 5; separate samples; 2 records:

```
AAAAABBBBB
AAAAABBBBB
```

- Three channels; channel record size == 5; interleaved samples; 2 records:

```
ABCABCABCABCABC
ABCABCABCABCABC
```

1.5.6 Known Issues

- Multi-channel, multi-sample, floating-point data doesn't save properly. A multi-channel RSA might use this format, but we do not currently have such a device.

CHAPTER 2

How to use Monarch3

Thread safety: Reading and writing records (via M3Stream::ReadRecord() and M3Stream::WriteRecord(), respectively) are thread-safe except that the HDF5 C library (on which the C++ library is built) is inherently non-thread-safe. Though multi-threaded writing may work even most of the time, it is inherently unstable. All other operations in Monarch (besides writing and reading records) are are explicitly not thread-safe.

2.1 Reading Egg3 Files

1. Open the file: `Monarch3::OpenForReading( [filename] )`
2. Access the header information: `Monarch3::ReadHeader()`
3. Get the pointer to the header, and use as needed: `Monarch3::GetHeader()`
4. Get the pointer(s) to the stream(s) in the file: `Monarch3::GetStream( [stream number] )`
5. Setup to access the data in a stream. You can acces either the record for the entire stream with `Monarch3::GetStreamRecord()`, or for individual channels with `Monarch3::GetChannelRecord( [record number] )`. If you have only one channel in the stream, the distinction between those is irrelevant. The record objects have a function `M3Record::GetData()` to get the data array. There are two ways in which you can interact with the data array:
   - If you want to access the data as an array of bytes (e.g. because either your data is of type `uint8_t`, or you want to use `memcpy`), you can use the pointer returned by `M3Record::GetData()`;
   - If you want to access the data as an array of other integer or floating-point data types, you can pass the data pointer from `M3Record::GetData()` to an `M3DataReader` object, along with the data type size and data format flag. The type of the values that are returned is specified as a template argument for `M3DataReader`; it doesn't have to match the data type in the data array exactly, but it should have at least as many bytes as the data elements, and if the data elements are integer, it should be an integer, and if the data elements are floating-point, it should be floating-point.
   - If you want to access the data as an array of complex floating-point data types, you can pass the data pointer from `M3Record::GetData()` to an `M3ComplexDataReader` object, along with the data type size and data format flag (you can also specify the element size, but for complex data it should

be the default, 2). The type of the values that are returned is specified as a template argument for `M3ComplexDataReader`; it should either be f4_complex or f8_complex, or the equivalent.

6. When moving from record to record in the file, the memory used for the data stays the same, but it gets updated with new values. To move to a new record use the `M3Stream::ReadRecord( [offset] )` function. The offset parameter allows you to move forward and backward within the file. If the last record read was `[J]` (= `-1` for a just-opened file), ReadRecord will access the `[J+1+offset]` record. This means that the offset parameter has the following meanings:

   - if `offset == 0` (default), the next record will be accessed;
   - if `offset == -1`, the current record will be reread;
   - `offset < -1` will go backwards in the file;
   - `offset > 0` will skip forward in the file.

   The outcomes from the call are:

   - returns true if the move was successful;
   - returns false if the move was unsuccessful because it goes past the end (or beginning) of the file;
   - throws an M3Exception if there was an error.

7. When you're finished reading, use `Monarch3::FinishReading()` to close the file.

2.2 Writing Egg3 Files

1. Open the file: `Monarch3::OpenForWriting( [filename] )`
2. Get the pointer to the header, and use as needed: `Monarch3::GetHeader()`
3. Fill in the header information and setup the streams. For the latter, use the AddStream functions to add streams with one or multiple channels.
4. Write the header information: `Monarch3::WriteHeader()`
5. Get the pointer(s) to the stream(s) in the file: `Monarch3::GetStream( [stream number] )`
6. Setup to access the data in a stream. You can acces either the record for the entire stream with `Monarch3::GetStreamRecord()`, or for individual channels with `Monarch3::GetChannelRecord( [record number] )`. If you have only one channel in the stream, the distinction between those is irrelevant. The record objects have a function `M3Record::GetData()` to get the data array. There are two ways in which you can interact with the data array:

   - If you want to access the data as an array of bytes (e.g. because either your data is of type `uint8_t`, or you want to use `memcpy`), you can use the pointer returned by `M3Record::GetData()`;
   - If you want to access the data as an array of other integer or floating-point data types, you can pass the data pointer from `M3Record::GetData()` to an `M3DataWriter` object, along with the data type size and data format flag. The type of the values that are passed to the writer is specified as a template argument for `M3DataWriter`; it doesn't have to match the data type in the data array exactly, but it should be no larger than the data elements, and if the data elements are integer, it should be an integer, and if the data elements are floating-point, it should be floating-point.
   - If you want to access the data as an array of complex floating-point data types, you can pass the data pointer from `M3Record::GetData()` to an `M3ComplexDataWriter` object, along with the data type size and data format flag (you can also specify the element size, but for complex data it should be the default, 2). The type of the values that are returned is specified as a template argument for `M3ComplexDataWriter`; it should either be `f4_complex` or `f8_complex`, or the equivalent.

7. For each record, copy the data to the stream data memory using the access method you chose above, and then write to disk with `M3Stream::WriteRecord( [is new acquisition?] )`. When a record is from a different acquisition than the previous record, the flag passed to `WriteRecord` should be `true`; otherwise it should be `false`. The outcomes from the call are:
    - returns true if the write was successful;
    - throws an M3Exception if there was an error;
    - (should never return false).
8. When you're finished writing, use `Monarch3::FinishWriting()` to close the file.

Full Doxygen API Reference