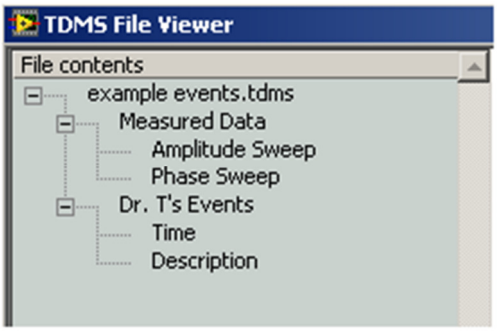
TDMS File Format Internal Structure

Overview

This article provides a detailed description of the internal structure of the TDM Streaming (TDMS) file format.

Logical Structure

TDMS files organize data in a three-level hierarchy of objects. The top level is comprised of a single object that holds file-specific information like author or title. Each file can contain an unlimited number of groups, and each group can contain an unlimited number of channels. In the following illustration, the file `example events.tdms` contains two groups, each of which contains two channels.



Every TDMS object is uniquely identified by a path. Each path is a string including the name of the object and the name of its owner in the TDMS hierarchy, separated by `/`. Each name is enclosed by the `' '` symbols. Any`'` symbol within an object name is replaced with two `'` symbols. The following table illustrates path formatting examples for each type of TDMS object:

Object Name	Object	Path
--	File	`/`
`Measured Data`	Group	`/'Measured Data'`
`Amplitude Sweep`	Channel	`/'Measured Data'/'Amplitude Sweep'`
`Dr. T's Events`	Group	`/'Dr. T''s Events'`
`Time`	Channel	`/'Dr. T''s Events'/'Time'`

In order for all TDMS client applications to work properly, every TDMS file must contain a `file`object. It must also contain a `group` object for each group name used in a channel path. In addition, it can contain an arbitrary number of group objects with no channels.

Every TDMS object can have an unlimited number of properties. Each TDMS property consists of a combination of a name (always a string), a type identifier, and a value. Typical data types for properties include numeric types such as integers or floating-point numbers, time stamps or strings. TDMS properties do not support arrays or complex data types. If a TDMS file is located within a search area of the National Instruments DataFinder, all properties automatically are available for searching.

Only channel objects in TDMS files can contain raw data arrays. In current TDMS versions, only one-dimensional arrays are supported.

Binary Layout

Every TDMS file contains two types of data: meta data and raw data. Meta data is descriptive data stored in objects or properties. Data arrays attached to channel objects are referred to as raw data. TDMS files contain raw data for multiple channels in one contiguous block. In order to be able to extract raw data from that block, TDMS files use a raw data index, which includes information about the data block composition, including the channel that corresponds to the data, the amount of values the block contains for that channel and the order in which the data was stored.

TDMS Segment Layout

Data is written to TDMS files in segments. Every time data is appended to a TDMS file, a new segment is created. Refer to the Raw Data section of this article for exceptions to this rule. A segment consists of the following three parts:

- **Lead In**—This part contains basic information, e.g. a tag that identifies files as TDMS, a version number and the overall length of the segment.
- **Meta Data**—This part contains names and properties of all objects in this segment. For objects that include raw data (channels), this part also contains index information that is used to locate the raw data for this object in the segment.
- **Raw Data**—This part is a contiguous block of all raw data associated with any of the objects included in the segment. The raw data part can contain interleaved data values or a series of contiguous data chunks. It can furthermore contain raw data from DAQmx.

This article does not describe how to decode DAQmx data. If you need to read a TDMS file with software that implements native support for TDMS (without using any components provided by National Instruments), you will *not* be able to interpret this data.

All strings in TDMS files (e.g. object paths, property names, property values, raw data values) are encoded in UTF-8 Unicode. All of them, except for raw data values, are preceded by a 32-bit unsigned integer that contains the length of the string in bytes, not including the length value itself. Strings in TDMS can be null-terminated, but since the length information is stored, the null terminator will be ignored when you read from the file.

Time stamps in TDMS are stored as a structure of two components:

- (i64) seconds: since the epoch 01/01/1904 00:00:00.00 UTC (using the Gregorian calendar and ignoring leap seconds)
- (u64) positive fractions: (2^-64) of a second

Boolean values are stored as 1 byte each, where 1 represents TRUE and 0 represents FALSE.

Lead In

The lead in contains information used to validate a segment. It also contains information used for random access to TDMS files. The following example shows the binary footprint of the lead in part of a TDMS file:

Binary layout (hexadecimal)	Description
54 44 53 6D	"TDSm" tag
0E 00 00 00	ToC mask 0x1110 (segment contains object list, meta data, raw data)
69 12 00 00	Version number (4713)
E6 00 00 00 00 00 00 00	Next segment offset (value: 230)
DE 00 00 00 00 00 00 00	Raw data offset (value: 222)

The lead in part in the previous table contains the following information:

- The lead in starts with a 4-byte tag that identifies a TDMS segment ("TDSm").
- The next four bytes are used as a bit mask in order to indicate what kind of data the segment contains. This bit mask is referred to as ToC (Table of Contents). Any combination of the following flags can be encoded in the ToC:

Flag		Description
#define kTocMetaData	(1L<<1)	Segment contains meta data
#define kTocRawData	(1L<<3)	Segment contains raw data
#define kTocDAQmxRawData	(1L<<7)	Segment contains DAQmx raw data
#define kTocInterleavedData	(1L<<5)	Raw data in the segment is interleaved (if flag is not set, data is contiguous)
#define kTocBigEndian	(1L<<6)	All numeric values (properties, raw data…) in the segment are big-endian formatted (if flag is not set, data is little-endian)
#define kTocNewObjList	(1L<<2)	Segment contains new object list (e.g. channels in this segment are not the same channels the previous segment contains)

- The next four bytes contain a version number (32-bit unsigned integer), which specifies the oldest TDMS revision a segment complies with. At the time of this writing, the version number is 4713. The only previous version of TDMS has number 4712.
- The next eight bytes (64-bit unsigned integer) describe the length of the remaining segment (overall length of the segment minus length of the lead in). If further segments are appended to the file, this number can be used to locate the starting point of the following segment. If an application encountered a severe problem while writing to a TDMS file (crash, power outage), all bytes of this integer can be 0xFF. This can only happen to the last segment in a file.
- The last eight bytes (64-bit unsigned integer) describe the overall length of the meta information in the segment. This information is used for random access to the raw data. If the segment contains no meta data at all (properties, index information, object list), this value will be 0.

Meta Data

TDMS meta data consists of a three-level hierarchy of data objects including a file, groups, and channels. Each of these object types can include any number of properties. The meta data section has the following binary layout on disk:

- Number of new objects in this segment (unsigned 32-bit integer).
- Binary representation of each of these objects.

The binary layout of a single TDMS object on disk consists of components in the following order. Depending on the information stored in a particular segment, the object might contain only a subset of these components.

- Object path (string)
- Raw data index
  - If this object does not have any raw data assigned to it in this segment, an unsigned 32-bit integer (`0xFFFFFFFF`) will be stored instead of the index information.
  - If the raw data index of this object in this segment exactly matches the index the same object had in the previous segment, an unsigned 32-bit integer (`0x0000000`) will be stored instead of the index information.
  - If the object contains raw data that doesn't match the index information assigned to this object in the previous segment, a new index for that raw data will be stored:
    - Length of index information (unsigned 32-bit integer)
    - Data type (`tdsDataType` enum, stored as 32-bit integer)
    - Array dimension (unsigned 32-bit integer) (right now 1 is the only valid value)
    - Number of values (unsigned 64-bit integer)
    - Total size in bytes (unsigned 64-bit integer) (only stored for variable length data types, e.g. strings)
- Number of properties (unsigned 32-bit integer)
- Properties. For each property, the following information is stored:
  - Name (string)
  - Data type (`tdsDataType`)
  - Value (numerics stored binary, strings stored as explained above).

The following table shows an example of meta information for a group and a channel. The group contains two properties, one string and one integer. The channel contains a raw data index and no properties.

Binary footprint (hexadecimal)	**Description**
`02 00 00 00`	Number of objects
`08 00 00 00`	Length of the first object path
`2F 27 47 72 6F 75 70 27`	Object path (`/'Group'`)
`FF FF FF FF`	Raw data index ("`FF FF FF FF`" means there is no raw data assigned to the object)
`02 00 00 00`	Number of properties for `/'Group'`
`04 00 00 00`	Length of the first property name
`70 72 6F 70`	Property name (`prop`)
`20 00 00 00`	Data type of the property value (`tdsTypeString`)
`05 00 00 00`	Length of the property value (only for strings)
`76 61 6C 75 65`	Value of the property `prop` (`value`)
`03 00 00 00`	Length of the second property name
`6E 75 6D`	Property name (`num`)
`03 00 00 00`	Data type of the property value (`tdsTypeI32`)
`0A 00 00 00`	Value of the property `num` (10)
`13 00 00 00`	Length of the second object path
`2F 27 47 72 6F 75 70 27 2F 27 43 68 61 6E 6E 65 6C 31 27`	Path of the second object (`/'Group'/'Channel1'`)
`14 00 00 00`	Length of index information
`03 00 00 00`	Data type of the raw data assigned to this object
`01 00 00 00`	Dimension of the raw data array (has to be 1)
`02 00 00 00 00 00 00 00`	Number of raw data values
`00 00 00 00`	Number of properties for `/'Group'/'Channel1'` (doesn't have properties)

Meta information that matches meta information in the previous segments can be omitted in following segments. This is optional, but omitting redundant meta information significantly speeds up reading the file. If you choose to write redundant information, you can later remove it using the TDMS Defragment function in LabVIEW, LabWindows/CVI or MeasurementStudio.

- Writing a new object to the next segment will imply that the segment contains all objects from the previous segment, plus the new objects described here. If the new segment does not contain any channel(s) from the previous segment, or if the order of channels in segment changes, the new segment needs to contain a new list of all objects. Refer to the *Optimization* section of this article for more information.
- Writing a new property to an object that already exists in the previous segment will add this property to the object.
- Writing a property that already exists on an object will overwrite the previous value of that property.

The following example shows the binary footprint for the meta data section of a segment directly following the segment described above. The only meta information written to the new segment is the new property value.

Binary layout (hexadecimal)	Description
01 00 00 00	Number of new/changed objects
08 00 00 00	Length of object path
2F 27 47 72 6F 75 70 27	Object path (`/'Group'`)
FF FF FF FF	Raw data index (no raw data assigned to the object)
01 00 00 00	Number of new/changed properties
03 00 00 00	Length of property name
6E 75 6D	Property name (`num`)
03 00 00 00	Data type of the property value (`tdsTypeI32`)
07 00 00 00	New value for property `num` (7)

Raw Data

The segment finally contains the raw data associated with each channel. The data arrays for all channels are concatenated in the exact order in which the channels appear in the meta information part of the segment. Numeric data needs to be formatted according to the little-endian/big-endian flag in the lead in. Note that channels cannot change their endian format or data type once they have been written for the first time.

String type channels are preprocessed for fast random access. All strings are concatenated to a contiguous piece of memory. The offset of the first character of each string in this contiguous piece of memory is stored to an array of unsigned 32-bit integers. This array of offset values is stored first, followed by the concatenated string values. This layout allows client applications to access any string value from anywhere in the file by repositioning the file pointer a maximum of three times and without reading any data that is not needed by the client.

If meta information between segments doesn't change, the lead in and meta information parts can be completely omitted and raw data can just be appended to the end of the file. Each following raw data chunk has the same binary layout, and the number of chunks could be calculated from the lead in and meta information by the following steps:

1. Calculate the raw data size of a channel. Each channel has a **Data type**, **Array dimension**and **Number of values** in meta information. Refer to the *Meta Data* section of this article for details. Each **Data type** is associated with a type size. You can get the raw data size of the channel by: type size of **Data type** × **Array dimension** × **Number of values**. If **Total size in bytes** is valid, then the raw data size of the channel is this value.
2. Calculate the raw data size of one chunk by accumulating the raw data size of all channels.
3. Calculate the raw data size of total chunks by: **Next segment offset - Raw data offset**. If the value of **Next segment offset** is -1, use the file size as the value of **Next segment offset**.
4. Calculate the number of chunks by: **Raw data size of total chunks** ÷ **Raw data size of one chunk**.

Raw data can be organized in two types of layout: interleaved and non-interleaved. The ToC bit mask in the segment lead in declares whether or not data in the segment is interleaved. Example: Storing 32-bit integer values to channel 1 (1,2,3) and channel 2 (4,5,6) would result in the following layouts:

Data Layout	Binary Footprint (hexadecimal)
Non-interleaved	01 00 00 00 02 00 00 00 03 00 00 00 04 00 00 00 05 00 00 00 06 00 00 00
Interleaved	01 00 00 00 04 00 00 00 02 00 00 00 05 00 00 00 03 00 00 00 06 00 00 00

Data Type Values

The following enum type describes the data type of a property or channel in a TDMS file. For properties, the data type value will be stored in between the name and the binary value. For channels, the data type will be part of the raw data index.

```
typedef enum {
    tdsTypeVoid,
    tdsTypeI8,
    tdsTypeI16,
    tdsTypeI32,
    tdsTypeI64,
    tdsTypeU8,
    tdsTypeU16,
    tdsTypeU32,
    tdsTypeU64,
    tdsTypeSingleFloat,
    tdsTypeDoubleFloat,
    tdsTypeExtendedFloat,
    tdsTypeSingleFloatWithUnit=0x19,
    tdsTypeDoubleFloatWithUnit,
    tdsTypeExtendedFloatWithUnit,
    tdsTypeString=0x20,
    tdsTypeBoolean=0x21,
    tdsTypeTimeStamp=0x44,
    tdsTypeDAQmxRawData=0xFFFFFFFF
} tdsDataType;
```

Notes:

- Refer to the LabVIEW Timestamp Developer Zone article for more information about using `tdsTypeTimeStamp` in LabVIEW.
- LabVIEW floating-point types with unit translate into a floating-point channel with a property named `unit_string` that contains the unit as a string.

Use the VI-based API for writing TDMS files for further reference on how TDMS files are composed.

Predefined Properties

LabVIEW waveforms are represented in TDMS as numeric channels, where the waveform attributes are added to the channel as properties.

- **wf_start_time**–This property represents the time at which the waveform was acquired or generated. This property can be zero if the time information is relative or if the waveform is not time-domain, but e.g. frequency domain.
- **wf_start_offset**–This property is used for the LabVIEW Express Dynamic Data Type. Frequency-domain data and histogram results will use this value as the first value on the *x*-axis.
- **wf_increment**–This property represents the increment between two consecutive samples on the *x*-axis.
- **wf_samples**–This property represents the number of samples in the waveform.

Optimization

Applying the format definition as described in the previous sections creates perfectly valid TDMS files. However, TDMS allows for a variety of optimizations that are commonly used by NI software like LabVIEW, LabWindows/CVI or MeasurementStudio. Applications that are trying to read data written by NI software need to support the optimization mechanisms described in this paragraph.

Incremental Meta Information Example

Meta information such as object paths, properties, and raw indexes, is added to a segment only if it changes. Incremental meta information is best explained by the following example.

In the first segment, channel 1 and channel 2 are written. Each has three 32-bit integer values (1,2,3 and 4,5,6) and several descriptive properties. The meta information part of the first segment contains paths, properties, and raw data indexes for channel 1 and channel 2. The flags `kTocMetaData`, `kTocNewObjList`, and `kTocRawData` of the ToC bit field are set. The binary footprint of the first segment looks like:

Part	Binary Footprint (hexadecimal)
Lead In	54 44 53 6D 0E 00 00 00 68 12 00 00 8F 00 00 00 00 00 00 00 77 00 00 00 00 00 00 00
Number of objects	02 00 00 00
Meta information object 1	13 00 00 00 2F 27 67 72 6F 75 70 27 2F 27 63 68 61 6E 6E 65 6C 31 27 14 00 00 00 03 00 00 00 01 00 00 00 03 00 00 00 00 00 00 00 01 00 00 00 04 00 00 00 70 72 6F 70 20 00 00 00 05 00 00 00 76 61 6C 69 64
Meta information object 2	13 00 00 00 2F 27 67 72 6F 75 70 27 2F 27 63 68 61 6E 6E 65 6C 32 27 14 00 00 00 03 00 00 00 01 00 00 00 03 00 00 00 00 00 00 00 00 00 00 00
Raw data channel 1	01 00 00 00 02 00 00 00 03 00 00 00
Raw data channel 2	04 00 00 00 05 00 00 00 06 00 00 00

In the second segment, none of the properties have changed, channel 1 and channel 2 still have three values each, and no additional channels are written to the segment. The segment does not contain any meta data. The meta data from the previous segment is still assumed valid. Only the raw data bit is set in the ToC bit field. The binary footprint of the second segment looks like:

Part	Binary Footprint (hexadecimal)
Lead In	54 44 53 6D 08 00 00 00 68 12 00 00 18 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Raw data channel 1	01 00 00 00 02 00 00 00 03 00 00 00
Raw data channel 2	04 00 00 00 05 00 00 00 06 00 00 00

The third segment adds another three values to each channel. In channel 1, the property `status` was set to `valid` in the first segment, but now needs to be set to `error`. The meta data section of the third segment now contains the object path for channel, name, type, and value for this property. In future file reads, the `error` value will override the previously written `valid` value. However, the previous valid value remains in the file, unless it is defragmented. The binary footprint of the third segment looks like:

Part	Binary Footprint (hexadecimal)

Lead In	54 44 53 6D 0A 00 00 00 68 12 00 00 60 00 00 00 00 00 00 00 48 00 00 00 00 00 00 00
Number of objects	01 00 00 00
Meta information object 1	13 00 00 00 2F 27 67 72 6F 75 70 27 2F 27 63 68 61 6E 6E 65 6C 31 27 14 00 00 00 03 00 00 00 01 00 00 00 03 00 00 00 00 00 00 00 01 00 00 00 04 00 00 00 70 72 6F 70 20 00 00 00 05 00 00 00 65 72 72 6F 72
Raw data channel 1	01 00 00 00 02 00 00 00 03 00 00 00
Raw data channel 2	04 00 00 00 05 00 00 00 06 00 00 00

The fourth segment adds an additional channel, `voltage`, which contains five values (7,8,9,10,11). Since all other meta data from the previous segment is still valid, the meta data section of the fourth segment includes the object path, the properties, and the index information for channel voltage only. The raw data section contains three values for channel 1, three values for channel 2, and five values for channel voltage. The binary footprint of the forth segment looks like:

Part	**Binary Footprint (hexadecimal)**
Lead In	54 44 53 6D 0A 00 00 00 68 12 00 00 5E 00 00 00 00 00 00 00 32 00 00 00 00 00 00 00
Number of objects	01 00 00 00
Meta information object 3	12 00 00 00 2F 27 67 72 6F 75 70 27 2F 27 76 6F 6C 74 61 67 65 27 14 00 00 00 03 00 00 00 01 00 00 00 05 00 00 00 00 00 00 00 00 00 00 00
Raw data channel 1	01 00 00 00 02 00 00 00 03 00 00 00
Raw data channel 2	04 00 00 00 05 00 00 00 06 00 00 00
Raw data channel 3	07 00 00 00 08 00 00 00 09 00 00 00 0A 00 00 00 0B 00 00 00

In the fifth segment, channel 2 now has 27 values. All other channels remain unchanged. The meta data section now contains the object path for channel 2, the new raw data index for channel 2, and no properties for channel 2. The binary footprint of the fifth segment looks like:

Part	**Binary Footprint (hexadecimal)**
Lead In	54 44 53 6D 0A 00 00 00 68 12 00 00 BF 00 00 00 00 00 00 00 33 00 00 00 00 00 00 00
Number of objects	01 00 00 00
Meta information object 2	13 00 00 00 2F 27 67 72 6F 75 70 27 2F 27 63 68 61 6E 6E 65 6C 32 27 14 00 00 00 03 00 00 00 01 00 00 00 1B 00 00 00 00 00 00 00 00 00 00 00
Raw data channel 1	01 00 00 00 02 00 00 00 03 00 00 00
Raw data channel 2	01 00 00 00 02 00 00 00 03 00 00 00 04 00 00 00 05 00 00 00 06 00 00 00 07 00 00 00 08 00 00 00 09 00 00 00 0A 00 00 00 0B 00 00 00 0C 00 00 00 0D 00 00 00 0E 00 00 00 0F 00 00 00 10 00 00 00 11 00 00 00 12 00 00 00 13 00 00 00 14 00 00 00 15 00 00 00 16 00 00 00 17 00 00 00 18 00 00 00 19 00 00 00 1A 00 00 00 1B 00 00 00
Raw data channel 3	07 00 00 00 08 00 00 00 09 00 00 00 0A 00 00 00 0B 00 00 00

In the sixth segment, you stop writing to channel 2. You only continue writing to channel 1 and channel `voltage`. This constitutes a change in the channel order, which requires you to write a new list of channel paths. You must set the ToC bit `kTocNewObjList`. The meta data section of the new segment must contain a complete list of all object paths, but no properties and no raw data indexes, unless they also change. The binary footprint of the sixth segment looks like:

Part	**Binary Footprint (hexadecimal)**
Lead In	54 44 53 6D 0E 00 00 00 68 12 00 00 81 00 00 00 00 00 00 00 61 00 00 00 00 00 00 00
Number of objects	02 00 00 00
Meta information object 1	13 00 00 00 2F 27 67 72 6F 75 70 27 2F 27 63 68 61 6E 6E 65 6C 31 27 14 00 00 00 03 00 00 00 01 00 00 00 03 00 00 00 00 00 00 00 00 00 00 00
Meta information object 2	12 00 00 00 2F 27 67 72 6F 75 70 27 2F 27 76 6F 6C 74 61 67 65 27 14 00 00 00 03 00 00 00 01 00 00 00 05 00 00 00 00 00 00 00 00 00 00 00
Raw data channel 1	01 00 00 00 02 00 00 00 03 00 00 00
Raw data channel 3	07 00 00 00 08 00 00 00 09 00 00 00

	0A 00 00 00 0B 00 00 00

Index Files

All data written to a TDMS file is stored to a file with the extension `*.tdms`. TDMS files can be accompanied by a `*.tdms_index` optional index file. The index file is used to speed up reading from the `*.tdms` file. If a National Instruments application opens a TDMS file without an index file, the index file is automatically created. If a National Instruments application such as LabVIEW or LabVIEW Windows/CVI writes a TDMS file, the index file and the main file are created at the same time.

The index file is an exact copy of the `*.tdms` file, except in that it does not contain any raw data and every segment starts with a `TDSh` tag instead of a `TDSm` tag. The index file contains all information to precisely locate any value of any channel within the `*.tdms` file.