IEEE P1159.3

Recommended Practice for a Power Quality Data Interchange Format An Extensible File Format for the Exchange of Power Quality Measurement and Simulation Data

Sponsored by
Standards Coordinating Committee 22 (Power Quality)

Draft 2
July 1999

Table of Contents

Introduction

(This introduction is not part of IEEE P1159.3 - A Recommended Practice for a Power Quality Data Interchange Format.)

The current version of this documentation and related information (header files, sample code, etc.) can be found at URL:

http://grouper.ieee.org/groups/1159/3/docs.html

For additional support, questions, or suggestions, please send an e-mail to **pqdif@electrotek.com**.

The Working Group on Developing a Power Quality Data Interchange Format, which undertook the development of this recommended practice, had the following membership:

Scott Peele, *Chair*
David Kreiss, *Secretary*
Erich W. Gunther, *Technical Editor*

Working Group Members

John Csomay
Randy Collins
Larry Conrad
Vladimir Basch
Roger Bergeron
Daniel Brooks
Arshad Mansoor
Ram Mukerji
Dan Nordel
Greg Olsen
Greg Rauch
Richard Bingham
Surya Santosa
Van Wagner
James Wikston
Charlie Williams
Cheri Warren
Rajaie Abu-Hashim
Mike Bridgewood
Richard Brown
James Crane
Fouad Dagher
Behnam Danai
Joe Enk
Stan Fabinwski
Gil Hensley
Mark Kempker
Ajay Koliwad
Jeff Lamoree
Larry Morgan
Allen Morinec
David Mueller
Tamara Ottersetter
Pragasen Pillay
Brian Prokuda
V. Rajagopalan
Larry Ray
Dan Sabin
Peter Shah
Linh Tu
David Vannoy
Marek Waclawiak
Steve Whisenant

The following individuals provided substantial contributions to the development of this recommended practice:

Core Documents
Erich W. Gunther
Robert F. Scott
Bill Dabbs

Independent Review
Dr. Baran
Daniel Brooks

Example Software
Robert F. Scott
Erich W. Gunther
Jack King

1. Introduction

1.1. Scope

Develop a recommended practice for a file format suitable for exchanging power quality related measurement and simulation data in a vendor independent manner. Appropriate definitions and event categories to be developed by other task forces under the IEEE Standards Coordinating Committee 22 (SC22) on Power Quality and the IEEE 1159 Working Group on Power Quality Monitoring.

1.2. Purpose

A variety of simulation, measurement and analysis tools for power quality engineers are now available from many vendors. Generally, the data created, measured, and analyzed by these tools are incompatible between vendors. The proposed file format will provide a common ground that all vendors could export to, import from to allow the end user maximum flexibility in choice of tool and vendor.

1.3. The PQDIF structure

There are two "layers" to the PQDIF format: The *physical* layer and the *logical* layer. The physical layer describes the physical structure of the file without regard to what will actually be stored in it. It uses tags to identify particular elements of the file. This is similar in concept to TIFF (Tagged Image File Format), used for storing images.

The logical layer uses the structure defined by the physical layer; it specifies specific tags to use when building up elements in the file.

The physical layer is based on:

- Specific "physical" data types (e.g. INT1, INT2, INT4, REAL4, REAL8, etc.) for portability; and a specific list of IDs for physical representation (e.g. ID_SERIES_PHYS_TYPE_INTEGER1, etc.)
- 4-byte alignment for efficient processing
- *Tags*—using GUIDs (Globally Unique Identifiers)—for unique identification of elements (hereafter called "tags")

The logical layer is based on:

- Specific lists of tags to identify elements of a file
- A hierarchy of tags and expected physical types
- Extensibility using user-defined tags for private data
- Extensibility of the standard format using tags defined in the future

To keep things simple, many elements in the logical layer are based on an explicit list of enumerated IDs, such as:

- Vendor list (ID_VENDOR_BMI, ID_VENDOR_BPA, etc.)
- Equipment list (ID_EQUIP_BMI_7100, ID_EQUIP_BMI_8010, etc.)
- Phase (ID_PHASE_AN, ID_PHASE_BN, etc.)

- IEEE 1159 disturbance category (ID_1159_TRANSIENT, ID_1159_SHORTDUR, etc.)
- High-level quantity type (ID_QT_WAVEFORM, ID_QT_RMS, etc.)
- Series quantity units (ID_QU_TIMESTAMP, ID_QU_VOLTS, ID_QU_AMPS, etc.)
- Series value type (ID_SERIES_VALUE_TYPE_MIN, ID_SERIES_VALUE_TYPE_MAX, etc.)

1.4. Other recommended reading

Once you understand the physical and logical formats of PQDIF, you will still need some guidelines on how to use it to represent real-world data. Please refer to section 5 - Logical Device Modeling, Annex A and Annex B for details on how to represent various types of data. Also, third parties have created PQDIF tools and development kits to facilitate getting up to speed on PQDIF.

2. Physical structure

2.1. Overall file structure

2.1.1. Standard PQDIF flat file

The file is made up of a series of records that are arranged in a linked list. These links are provided by an *absolute link* (absolute file offset or position) in the header of the records. This allows new records to be inserted, and old records to be deleted.

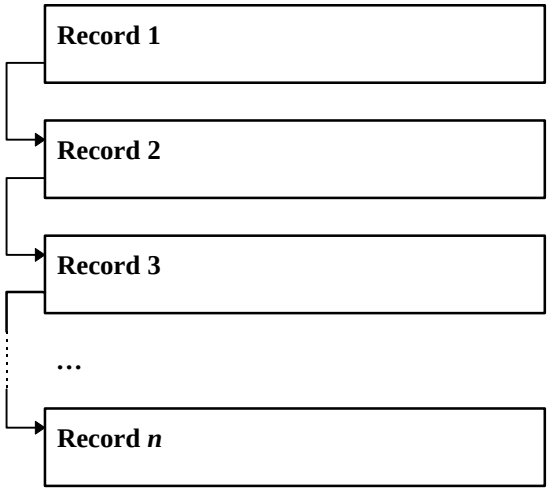


Figure 1 -- Standard record linkages

For the records shown in Figure 1, the order of the records is Record 1, Record 2, Record 3 … Record *n*.

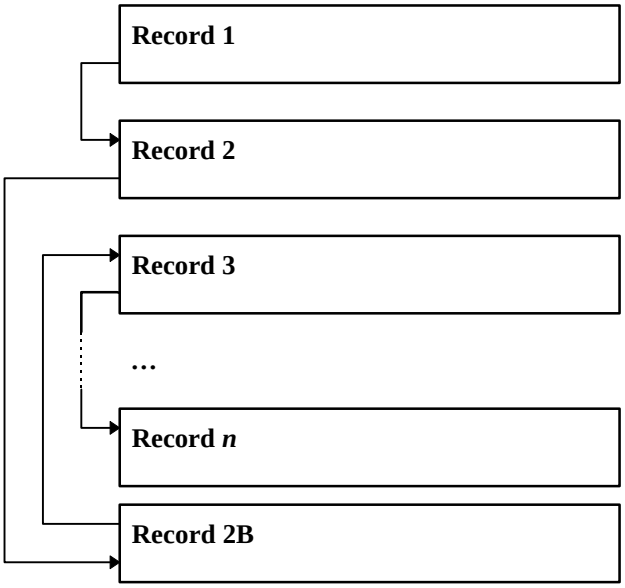


Figure 2 -- Inserting a new record

After the operation in Figure 2, the order of the records is now Record 1, Record 2, Record 2B, Record 3 … Record *n*.

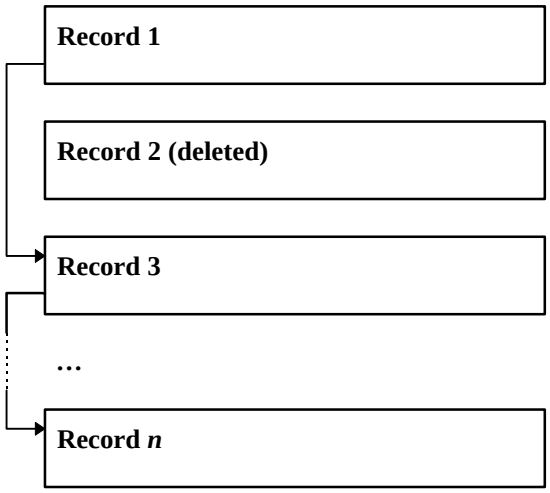


Figure 3 -- Deleting a record

After the operation in Figure 3, the order of the records is now Record 1, Record 3 … Record *n*.

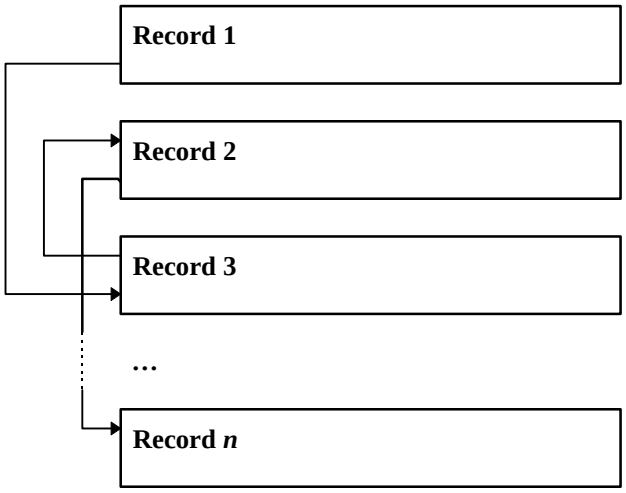


Figure 4 -- Reordering records

After the operation in Figure 4, the order of the records is now Record 1, Record 3, Record 2 … Record *n*.

2.1.2. Alternate PQDIF structures

The PQDIF format is flexible enough that other structures besides the flat file can be used. For example, it would be possible to store individual PQDIF records in a database or a structured storage medium of some sort. In this case, some of the items in the header (the *absolute link*, for example) are not necessary and can be ignored. In some cases the header can be discarded entirely. Using PQDIF in this fashion is out of the scope of this document, since the primary focus is the standard flat file that will be used for interchanging data.

2.2. Internal record structure

2.2.1. Record Header

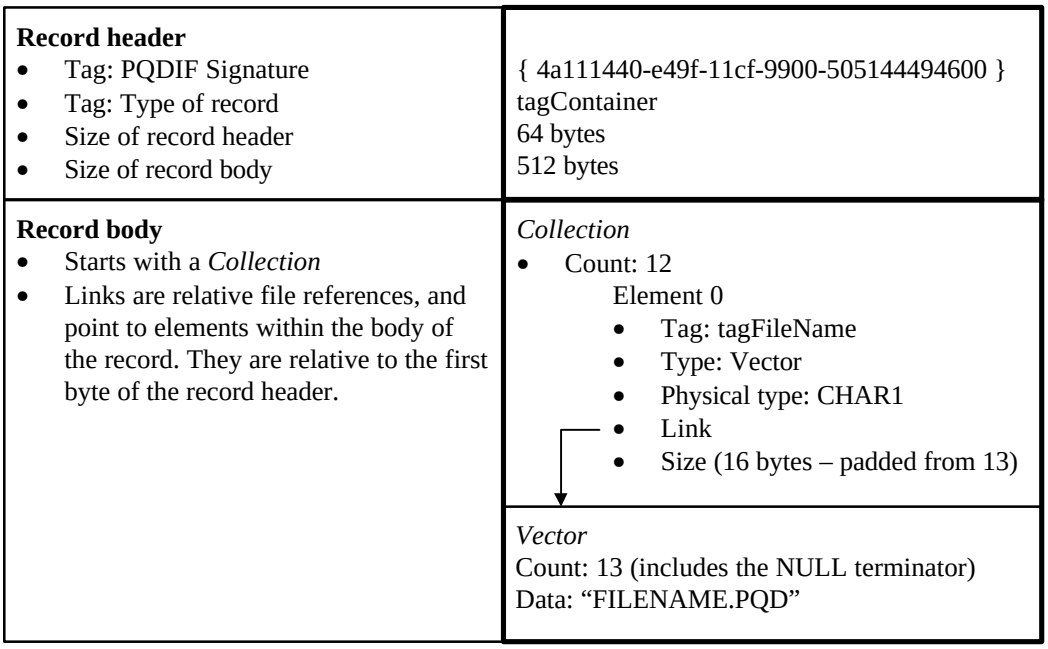
Each record has a standard header, which is marked with a unique "signature" – in this case, a GUID (shown below). The other items in the header specify the *tag* of the record (container, data source, etc.), its *size*, and the *absolute link* to the next record. There may be others, such as *status* (active/deleted).

2.2.2. Record Body

The record body is made up of a set of *elements*. There are three types of elements: *Collection*, *Scalar*, and *Vector*.

- A collection is essentially an array of *tags* and *relative links* to other elements. Because a collection can link to another collection, it is simple to create a hierarchical structure.
- A scalar is a single value of a specific physical type (INT4, REAL8, etc.).
- A vector is an arbitrarily-sized array of a specific physical type.

Each element is given a *tag* by the collection that contains it (except for the first collection). The record body always starts with a collection element. This element is tagged by its location in the record – i.e., the first element. Therefore, it is identified by the tag found in the record header.



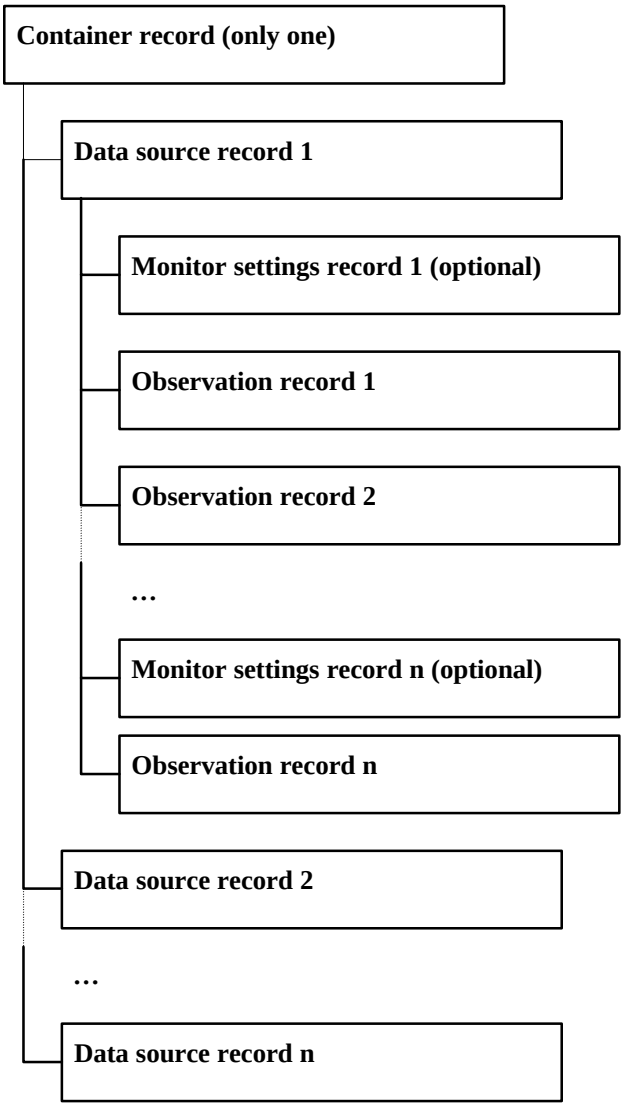
A *relative link* is a file position that is relative to the beginning of the record header.

3. Logical Structure

3.1. Logical record structure

The basic structure is made up of a logical hierarchy of “records” – consisting of a single container, following by one or more data sources, monitor settings, and observations. There can also be “extended” records by defining new tags.

The order of the records is defined by the absolute links in the header of each record. By following the list of records, a linear list of records is defined. This linear list has a logical hierarchy defined as follows:



Note that several monitor settings records can appear in the middle of the observations for a particular data source. This allows the monitor (instrument) setup to change. As a result some parameters in the observations may depend on the information in the appropriate monitor

settings record. If we did not allow multiple settings records, a totally new data source would have to be created if any settings changed.

The Date Effective time stamps of data source and monitor settings records are used to determine which of these records applies to a given observation record. The data source and monitor settings records with a Date Effective time stamp equal to or earlier the subject observation record are used.

3.2. Logical element structure

3.2.1. Introduction

The complete logical format is defined by a set of tags, and an expected structure associated with each tag. For example, a *tagName* element must, by definition, be a vector of CHAR1 – in other words, a string. It has no expected length (i.e, it can be as long or as short as necessary). A *tagVersionInfo* element is slightly more complex: it must be a vector of UINT4, and must have four entries: major and minor version; and major and minor version "compatible."

Even more complex is the way we define hierarchies. A *tagChannelDefns* element must be a collection, and must contain another set of collections, which are all tagged with *tagOneChannelDefn*. (The number of elements in the *tagChannelDefns* element defines how many channel definitions are available.) Inside each *tagOneChannelDefn* collection is another set of required tags, and so forth.

See Annex A and B for full documentation of all tags. See the C language header files included in these documents for the actual tag definitions and for the lists of Id's:

Header file	Contents
PQDIF_PH.H	All data structure declarations for the physical structure (Annex A).
PQDIF_LG.H	All tag declarations for the physical structure. (These tags are documented in Annex B.)
PQDIF_ID.H	An extensible list of ID declarations for things such as vendors and equipment. (These IDs are documented in Annex B.)

3.3. Logical structure of container record

The container record contains overall – or summary – information about the PQDIF file. The following text shows an ASCII dump of the container record from a typical PQDIF file produced by a simple PQDIF utility program.

```
-Collection -- tag: tagContainer (level 0)
   +-Vector -- tag: tagFileName (type: CHAR1) [ 32 ]
   |  value: 'D:\PQDIF\Native15\PQDIFr\wf.pqd'
   +-Scalar -- tag: tagCreation (type: TIMESTAMPPQDIF)
   |  value: 7/7/1999 20:40:58.000000217
   +-Vector -- tag: tagVersionInfo (type: UNS_INTEGER4) [ 4 ]
   |  values: 1, 5, 1, 5
   +-Vector -- tag: tagLanguage (type: CHAR1) [ 11 ]
   |  value: 'US English'
   +-Vector -- tag: tagTitle (type: CHAR1) [ 25 ]
   |  value: 'PASS to PQDIF Translator'
   +-Vector -- tag: tagSubject (type: CHAR1) [ 15 ]
```

```
|  value: '8010/20 Export'
+-Vector -- tag: tagAuthor (type: CHAR1) [ 17 ]
|  value: 'Erich W. Gunther'
+-Vector -- tag: tagKeywords (type: CHAR1) [ 37 ]
|  value: 'PASS 8010 8020 PQDIF Translator IEEE'
+-Vector -- tag: tagComments (type: CHAR1) [ 24 ]
|  value: 'Comments field not used'
+-Vector -- tag: tagLastSavedBy (type: CHAR1) [ 27 ]
|  value: 'lastSavedBy field not used'
+-Vector -- tag: tagApplication (type: CHAR1) [ 34 ]
|  value: 'PASS to PQDIF translator for IEEE'
+-Vector -- tag: tagSecurity (type: CHAR1) [ 12 ]
|  value: 'No security'
+-Vector -- tag: tagOwner (type: CHAR1) [ 26 ]
|  value: 'Electrotek Concepts, Inc.'
+-Vector -- tag: tagCopyright (type: CHAR1) [ 67 ]
|  value: 'Copyright (C) 1999 Electrotek Concepts, Inc.  All Rights Reserved.'
+-Vector -- tag: tagTrademarks (type: CHAR1) [ 75 ]
|  value: 'Electrotek Concepts is a registered trademark of Electrotek Concepts, Inc.'
+-Vector -- tag: tagNotes (type: CHAR1) [ 9 ]
|  value: 'No notes'
+-Scalar -- tag: tagCompressionStyleID (type: UNS_INTEGER4)
|  value: 2
+-Scalar -- tag: tagCompressionAlgorithmID (type: UNS_INTEGER4)
|  value: 1
+-(End of collection)
+---- Record information
| Size on disk: 1028
+-(End of record)
```

There are two pieces of information here which are particularly important when interpreting the contents of the file:

- Version
- Compression

3.3.1. Version

Version information is used to control and track the evolution of the format. At some point, major changes may need to be made which affect the interpretation of some of the tags. If these changes are drastic, older readers may not be able to interpret these standard tags properly. In this case, the “compatible version” (as described below) may have to be bumped up so that older readers will know not to attempt interpretation.

On the other hand, it may be possible to continue to update the standard without affecting backward compatibility. In this case, the compatible version will remain at 1.0, and all older readers will be able to interpret the file – sans any new tags, of course.

Newer readers will be able to tell what tags to expect by looking at the “writer version.” Basically, all readers should do the following:

- If it can support the writer version, read the file using this version.
- Otherwise, if it can support the compatible version, read the file using this version.
- If it cannot support either, do not attempt to read the file.

This information is contained in a vector tagged with *tagVersionInfo*. This vector has four members, denoting:

Index	Name	Meaning
0	Writer major version	The version of the logical structure being used by the writer of the file. This is for a reader to determine
1	Writer minor version	"
2	Compatible major version	The file is also compatible with this version of the logical structure. Older readers can use this to determine if they will be able to read this file.
3	Compatible minor version	"

Note: The version information is intended to apply *only* to the logical structure – the physical structure will remain standard throughout the life of the format.

The ASCII dump above indicates that the subject PQDIF file was written by a program using version 1.5 and it is compatible with version 1.5 and later reader programs.

3.3.2. Compression

The container record contains an element indicating whether or not compression is used. The ASCII dump above specifies the tagCompressionStyleID and the tagCompressionAlgorithmID. The values shown indicate that the PQDIF file uses record level compression and the ZLIB compression algorithm is used. The integers used for these types are defined in Annex B. See section 4. Compression (page 17) for more information on compression.

3.4. Logical structure of data sources and observations

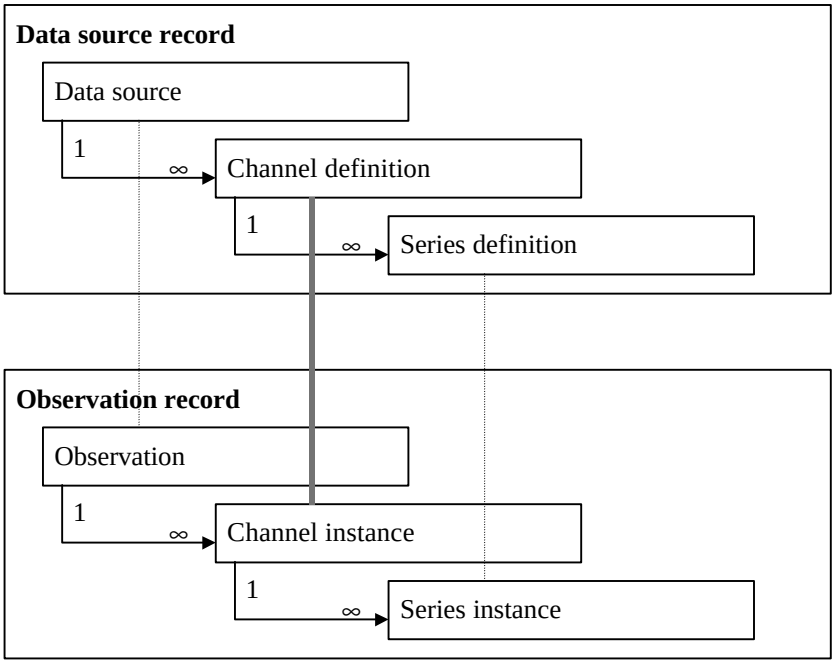
3.4.1. Structure

The internal structure of the *data source* and *observation* records parallel each other, and in so doing, are implicitly linked together. In object-oriented design language, it can be described this way (note that "many" is a technical term here; it actually means 0 or more):

A *data source* **has** many *channel definitions*; each *channel definition* **has** many *series definitions*.

An *observation* **has** many *channel instances*; each *channel instance* **has** many *series instances*.

This definition/instance structure parallels the object-oriented class/instance structure, and serves the same purpose. "Static" information common to all "instances" is kept at the definition (or class) level.



3.4.2. Definitions

The channel definition (*tagChannelDefn*) holds information about the channels that would not need to be repeated for each observation:

- Channel name
- Phase or other name
- Name if phase not applicable
- High-level quantity type (waveform, trend, etc.)
- Group name (Bus 1, Bus 2, XFMR High Side, etc.)

The series definition (*tagSeriesDefn*), likewise, holds information that is common to all to all observations:

- Quantity Measured (voltage, current, power, etc.)
- Quantity units (volts, amps, etc.)
- Quantity characteristic (instantaneous, peak, rms, etc.)
- The series value type (Min, Max, Average, etc.)
- Hints about how to display the data (expected Greek prefix, per unit vs. engineering units, etc.)

3.4.3. Instances

The specific channel instance (*tagChannelInstance)* actually holds very little except providing a link to the series instances. However, every channel definition may *or may not* be used by every single observation. Thus, there may be fewer channel instances than we have channel definitions.

The channel instance then uses a definition index (*tagChannelDefnIdx*) to indicate that it is an instance of a specific channel definition.

In other words, the channel definitions provide a “library” of possible channels to be “instantiated.” The individual channel instance can decide which definition to use. It is not necessary for each observation to use every possible channel definition; the *tagChannelDefnIdx* tag is used instead to specify which channel definitions are actually used.

Similarly, there may be more than one instance of a particular channel definition. This is most often encountered when reporting multiple attributes of a particular measured quantity characteristic. For example, if the third, fifth and seventh harmonic magnitudes of the voltage represented by a particular definition is to be recorded, three instances of this channel definition are stored in the file. Each instance contains an element that specifies for which harmonic (more generally the frequency) the instance represents. A similar technique is used to represent probability data. Multiple instances of the same channel are used to record data for a specified percentile.

The series instance (*tagSeriesInstance*) holds the array of data values which makes up its portion of the observation. The series instance is a little different from the channel instance in that each series instance must match up directly with each series definition. If there is a need for different series topologies, you will need to create additional channel definitions for each one.

3.5. How to put together a series

Proper use of the series options described in this section can result in a significant reduction in the size of a PQDIF file. Methods are provided to represent data series as scaled integers of varying size, referenced copies of another series, or series represented by a simple count and interval. Use of these options is recommended to minimize PQDIF file size.

3.5.1. Explicit and scaled series

Most series will be a straight vector of data values, typically of the types INT1, INT2, INT4, REAL4 or REAL8. The vector’s count will be the total number of data points in the series. In this case, the storage method will be ID_SERIES_METHOD_VALUES.

This can be OR’d with ID_SERIES_METHOD_SCALED to scale each value by another constant. This would typically be used when the “raw” sample (the 2-byte integer return by an A/D converter, for example) is being stored. This way, the PQDIF file is smaller, and the final scaled value (which might otherwise have to be stored a REAL4 or REAL8) can be calculated at read time.

3.5.2. Regular rate series

In some cases, you may have a series in which each number is separated by a constant value (this happens often in time series with a constant sampling rate). To represent this, you can put together an array of numbers that indicates how far apart the values are supposed to be. The storage method will be ID_SERIES_METHOD_INCREMENT. This can even have multiple sampling rates:

```
vector, count = ( N * 2 ) + 1
[ 0 ] = N (total number of rate changes)
[ 1 ] = number of points (0)
[ 2 ] = rate (0)
...
[ N * 2 - 1] = number of points (N-1)
[ N * 2    ] = rate (N-1)
```

Thus, if you had a single sampling rate (say, 0.01 seconds) for 1792 points, the series would be:

```
vector, count = 3
[0] = 1
[1] = 1792
[2] = 0.01
```

In addition, you can OR the storage method with ID_SERIES_METHOD_SCALED to represent the sample rate. In this case, the series would be:

```
vector, count = 3
[0] = 1
[1] = 1792
[2] = 1
```

and the *tagSeriesScale* element would contain 0.01.

Naturally, if you had multiple sampling rates, you would probably not use this scale factor, and would instead put the scaling factor directly into the series, as shown here. This example shows a series which still contains 1792 points, but has two different sampling rates:

```
vector, count = 5
[0] = 2
[1] = 896
[2] = 0.01
[3] = 896
[4] = 0.02
```

If your regular rate series does not start at 0, you can add a *tagSeriesOffset* element as a starting point. (Note that for time series, the first value should always be 0.)

3.5.3. Shared series

For a normal series, a *tagSeriesValues* vector is required. However, in some cases you may want to share data from another series – for space savings. In this case, the *tagSeriesValues* tag is replaced by two tags: *tagSeriesShareChannelIdx* and *tagSeriesShareSeriesIdx*. These tags indicate a “master” normal series (e.g., one that has a *tagSeriesValues* tag) which has the data we want to share.

These tags (*tagSeriesShareChannelIdx* and *tagSeriesShareSeriesIdx*) are indices into collections within the current observation.

4. Compression

4.1. Compression algorithm – ZLIB

Since PQDIF is a binary format, it is fairly compact even uncompressed. Nevertheless, compression can still help in many cases. PQDIF uses the public domain *zlib* compression library when compression is desired. This algorithm is an LZ77 variant (similar to PKZIP) and is

not covered by any patents (unlike the LZW algorithm). The *zlib* home page can be found at http://www.cdrom.com/pub/infozip/zlib/. Here is a description of the algorithm from this page:

> ***zlib*** is designed to be a free, general-purpose, legally unencumbered -- that is, not covered by any patents -- lossless data-compression library for use on virtually any computer hardware and operating system. The zlib data format is itself portable across platforms. Unlike the LZW compression method used in Unix compress(1), the compression method currently used in zlib essentially never expands the data. (LZW can double or triple the file size in extreme cases.) zlib's memory footprint is also independent of the input data and can be reduced, if necessary, at some cost in compression. A more precise, technical discussion of both points is available on another page.
>
> ***zlib*** was written by Jean-loup Gailly (compression) and Mark Adler (decompression). Jean-loup is also the primary author/maintainer of gzip(1), the author of the comp.compression FAQ list and the former maintainer of Info-ZIP's Zip; Mark is also the author of gzip's and UnZip's main decompression routines and was the original author of Zip. Not surprisingly, the compression algorithm used in zlib is essentially the same as that in gzip and Zip, namely, the `deflate' method that originated in PKWARE's PKZIP 2.x.
>
> Mark and Jean-loup can be reached by e-mail at zlib@gzip.org .
>
> Greg, Mark and/or Jean-loup will add some more stuff here when they think of something to add. For now this page is mainly a pointer to zlib itself and to the official zlib and deflate documentation. Note that the specifications both achieved official Internet RFC status in May 1996, and zlib itself was adopted by JavaSoft in version 1.1 of the Java Development Kit (JDK), both as a raw class and as a component of the JAR archive format.

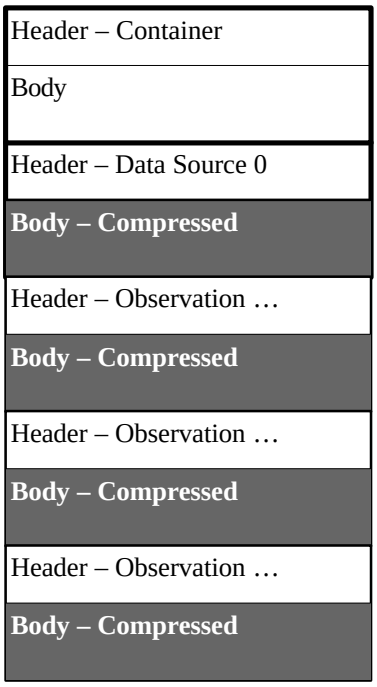
This copyright notice is included in the sample source code:

> Copyright © 1995-1998 Jean-loup Gailly and Mark Adler
>
> This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.
>
> Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:
>
> 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
> 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
> 3. This notice may not be removed or altered from any source distribution.
>
> *Jean-loup Gailly* — jloup@gzip.org
> *Mark Adler* — madler@alumni.caltech.edu

The data format used by the zlib library is described by RFCs (Request for Comments) 1950 to 1952 in the files http://www.alternic.org/rfcs/rfc1900/rfc1950.txt (zlib format), rfc1951.txt (deflate format) and rfc1952.txt (gzip format).

4.2. Record Level Compression

In record-level compression, all record headers are uncompressed. All record *bodies* (except the container record) are compressed as individual blocks. This compression method allows a PQDIF reader program to quickly discover the structure of the file without having to read and decode the entire contents of the file.



Since the headers are uncompressed, the absolute links work as before. The header (and the links, of course) must reflect the *compressed* size of the body, not the uncompressed size. The relative links within the body work as before – they are all set before compression occurs. In other words, create the body and then compress it. A checksum (a 32-bit CRC) for each record body must be stored in the appropriate record header.

5. Logical Device Modeling

The previous sections described the core format itself, introduced the concept of data source, monitor settings, and observation records and how to represent a basic data series. This section describes the process of modeling power quality data in more detail. PQDIF is designed to support representing simulated as well as measured power quality data. For the purposes of this section, we will focus on power quality monitoring instrument modeling.

The first thing we must realize is that PQDIF is not intended to be an optimal format for use as a native information store for an actual instrument. For some instruments it could be, but not

generally. More often, an instrument vendor will choose to store information recorded in their instrument in a manner efficient for the internal architecture of the device. A PQDIF file is only created when that information needs to be extracted in an instrument independent manner for some undefined, external use. This translation may occur in the instrument itself upon demand or in an external conversion program.

The most common types of data produced by power quality monitoring devices include:

- Voltage and current waveform captures (e.g. transients, waveform distortion)
- Voltage and current phasor captures (e.g. short and long duration RMS variations)
- Steady state value logs (e.g. true power factor at 15 minute intervals for some period of time)

PQDIF represents these data types by establishing a few basic concepts and categorization levels. The categorization levels are:

- Quantity Type
- Quantity Measured
- Quantity Characteristic
- Phase Identification

These categories uniquely identify the primary data series of a logical channel that represents a specific measured or simulated quantity. As we will show later, a logical channel can contain multiple data series that can be further qualified with attributes such as:

- Minimum value over an interval
- Maximum value over an interval
- Average value over an interval
- Instantaneous value
- Phase angle of a complex value
- Etc.

A waveform, or RMS trace, or value can be recorded as a result of a triggered event as defined by IEEE 1159, a periodic capture programmed into an instrument, a user initiated manual trigger, or some external trigger based on a contact closure or algorithm evaluation. PQDIF provides the means to identify these possible states as well as the IEEE 1159 disturbance categorization of triggered events if appropriate.

The data associated with these data types are represented in an observation record. The format of that data is defined in the most recent data source record. Further information about the state of the instrument or simulation program that created the observation record is optionally contained within the most recent monitor settings record.

For triggered events, all of the data associated with a disturbance as defined in IEEE 1159 is contained within a single observation. For example, an RMS variation is defined to begin when one or more phases of voltage goes outside of threshold and is defined to end when all phases are back within the threshold. All of the channels used to represent the RMS trace(s) and/or

waveform(s) captured during that disturbance would be stored in a single PQDIF observation record.

5.1. General Procedures

Translation of power quality data of any format requires that the following information be gleaned from the native file format and translated to PQDIF:

1. measured data type – description of what is represented by the actual quantitative measurement data. This data source information associated with a given numeric measurement data set is often referred to as the data channel definition
2. monitor information – general monitor information such as threshold settings, installation date, calibration information, etc. The extent of monitor information that needs to be translated to PQDIF is dependent on the destination of the translated PQDIF files – the final application that will use the PQDIF files as an input.
3. measured data values – actual quantitative measurement data associated with a defined data channel.

The following subsections discuss the PQDIF element tags that must be populated to relate the three pieces of information listed. The "Channel Specifications Overview" section discusses the required tags for specifying data channels (measurement data type) in the data source record. The "Monitor Settings Specifications Overview" discusses the required tags for specifying monitor information in the optional monitor settings record. Finally, the "Observation Specifications Overview" discusses the required tags for specifying measurement values recorded for a specified channel (measurement data values) in the observation record.

5.1.1. Channel Specifications Overview

Before the individual measurements can be translated to the PQDIF format, individual logical channels must be defined in the PQDIF file which relate what the measured data from the instruments physical channels represents. A logical channel in PQDIF is a unique combination of quantity type, quantity measured, quantity characteristic, and phase. The data source record is used in PQDIF to define the attributes of all of the logical channels that may be referenced by the instance data in a PQDIF file.

PQDIF requires several data fields to completely describe these elements which define each channel. Table 1 lists the required element tags that must be assigned values for each channel. Table 1 shows two series definitions. Note that a channel may consist of only one series, but most channels consist of two (e.g. a series of time values and a parallel series of magnitude values representing the instantaneous value of a waveform at each time point in the time series). Also, it should be noted that the tags listed in Table 1 are only the required tags. There are many other optional tags that may be assigned values.

Each of the required element tags listed in Table 1 have an associated set of acceptable values. The "Example Value" column of Table 1 shows the values that should be assigned to define the channel which records instantaneous voltage waveforms on phase A-neutral. See Annex B for the list of acceptable values for each of the element tags defined in Table 1.

Level	Element Tag	Example Value
0	tagRecDataSource	--NA--
1	tagNominalFrequency	60.0
1	tagChannelDfns	--NA--
2	tagOneChannelDfn	--NA--
3	tagPhaseID	*ID_Phase_AN*
3	tagQuantityTypeID	*ID_QT_Waveform*
3	tagQuantityMeasuredID	*ID_QM_Voltage*
3	tagSeriesDfns	--NA--
4	tagOneSeriesDfn	--NA--
5	tagValueTypeID	*ID_Series_Value_Type_Time*
5	tagQuantityUnitsID	*ID_QU_Seconds*
5	tagQuantityCharacteristicID	*ID_QC_None*
5	tagStorageMethodID	*ID_Series_Method_Values*
4	tagOneSeriesDfn	--NA--
5	tagValueTypeID	*ID_Series_Value_Type_Val*
5	tagQuantityUnitsID	*ID_QU_Volts*
5	tagQuantityCharacteristicID	*ID_QC_Instantaneous*
5	tagStorageMethodID	*ID_Series_Method_Values*

Table 1. Required data source record element tags

5.1.2. Monitor Settings Specifications Overview

The tags in the monitor settings record provide basic, vendor independent instrument settings information. The monitor settings record is not required. If, however, information such as the trigger values for a given channel is critical to the data being translated, it is recommended that the monitor settings record be created. Most of the defined tags are optional and are to be populated only if the data exists and would be useful to other applications.

Table 2 lists the required monitor settings tags and example values for the example channel defined in Table 1. Note that only one channel is included in Table 2, but monitor setting tag values should be specified for each channel defined. Furthermore, there are many channel level (level 3) tags that are optional if the information is available.

Level	Element Tag	Example Value
0	TagRecMonitorSettings	--NA--
1	TagEffective	Date settings were implemented.
1	TagTimeInstalled	Date monitored installed
1	TagChannelSettingsArray	--NA--
2	TagOneChannelSetting	--NA--
3	TagChannelDefnIdx	57
3	TagTriggerTypeID	--NA--

Table 2. Required monitor settings record element tags

5.1.3. Observation Specifications Overview

Assignment of the observation element tags is primarily a matter of listing the measurement values in a way that coordinates with the manner in which the associated channel has been specified. Note that a single PQDIF observation may consist of many different measurements on one or more physical channels that are associated with the same trigger. For example, a single voltage sag might trigger both waveform and rms phasor measurements. Although the data recorded due to this single sag is associated with many different channels, it is more efficient to store all of the data associated with a single trigger in a single observation. This also allows for proper association of data channels in the final application which uses the PQDIF files.

Table 3 lists the required element tags that define a given observation. As with the channel specification, there are other optional tags that can be assigned as well. Permissible values for each element tag listed in Table 3 are provided in PQDIF logical structure document. The "Example Value" column of Table 3 lists possible values for a phase A voltage waveform associated with the channel specified in Table 1.

Level	Element Tag	Example Value
0	TagRecObservation	--NA--
1	TagObservationName	*Phase A Voltage Waveform*
1	TagTimeCreate	*12/9/1998 11:42:24.453813818*
1	TagTimeStart	*2/13/1998 23:17:44.086079597*
1	TagTriggerMethodID	*ID_Trigger_Meth_Channel*
1	(tagTimeTriggered)**	*2/13/1998 23:17:44.086079597*
1	TagChannelInstances	--NA--
2	TagOneChannelInst	--NA--
3	TagChannelDefnIdx	57
3	TagSeriesInstances	--NA--
4	TagOneSeriesInstance	--NA--
5	TagSeriesValues	0,0.00013,0.00026, . . .
4	TagOneSeriesInstance	--NA--
5	TagSeriesValues	-4607.4621, -5030.9689, -5428.9193, . . .

** "tagTimeTriggered" is not a required tag but is recommended for all observations.

Table 3. Required observation record element tags

The following sections describe how to represent these common power quality data types in PQDIF by example.

5.2. Representing a Waveform Recording

A waveform recording is represented in PQDIF using channel definitions of the QT_WAVEFORM quantity type. This is the simplest of the PQDIF quantity types and one of the most commonly used types for instruments that capture waveforms of voltage and current.

The individual measurements logged by the instrument are stored in PQDIF "observation" records. If a single trigger results in saving waveforms for a number of different quantities (cross-triggered), it is useful for all of the measurements associated with the same trigger to be included in a single observation record. If an instrument can capture more than one waveform

block during a single disturbance as defined by IEEE 1159 (e.g. 5 cycles of waveform at the beginning of a sag and 5 cycles of waveform at the end of the same sag), then more than one instance of a particular channel definition may be recorded in the PQDIF observation.

The following text is an ASCII dump from a PQDIF utility program that shows the top level tags of a data source record for a real PQDIF file. These tags permit the specification of basic information about the data source.

```
-Collection -- tag: tagRecDataSource (level 0)
   | The checksum for this record is correct.
   +-Scalar -- tag: tagDataSourceTypeID (type: GUID)
   |  value: {e6b51730-f747-11cf-9d890080} - ID_DS_TYPE_MEASURE
   +-Scalar -- tag: tagVendorID (type: GUID)
   |  value: {e6b5170a-f747-11cf-9d890080} - ID_VENDOR_ELECTROTEK
   +-Scalar -- tag: tagEquipmentID (type: GUID)
   |  value: {e6b51721-f747-11cf-9d890080} - ID_EQUIP_ETK_TESTPROGRAM
   +-Vector -- tag: tagSerialNumberDS (type: CHAR1) [ 4 ]
   |  value: '1.0'
   +-Vector -- tag: tagVersionDS (type: CHAR1) [ 4 ]
   |  value: '1.0'
   +-Vector -- tag: tagNameDS (type: CHAR1) [ 14 ]
   |  value: 'PQDIF Convert'
   +-Vector -- tag: tagOwnerDS (type: CHAR1) [ 4 ]
   |  value: 'EWG'
   +-Vector -- tag: tagLocationDS (type: CHAR1) [ 1 ]
   |  value: ''
   +-Vector -- tag: tagTimeZoneDS (type: CHAR1) [ 1 ]
   |  value: ''
```

The following ASCII dump from the same PQDIF file shows the channel definition for one phase of a voltage channel waveform channel:

```
+-Collection -- tag: tagChannelDefns (level 1)
   |   +-Collection -- tag: tagOneChannelDefn (level 2)
   |   |   +-Vector -- tag: tagChannelName (type: CHAR1) [ 13 ]
   |   |   |  value: 'Waveform VAB'
   |   |   +-Scalar -- tag: tagPhaseID (type: UNS_INTEGER4)
   |   |   |  value: 1 - ID_PHASE_AB
   |   |   +-Scalar -- tag: tagQuantityMeasuredID (type: UNS_INTEGER4)
   |   |   |  value: 1 - ID_QM_VOLTAGE
   |   |   +-Scalar -- tag: tagQuantityTypeID (type: GUID)
   |   |   |  value: {67f6af80-f753-11cf-9d890080} - ID_QT_WAVEFORM
   |   |   +-Collection -- tag: tagSeriesDefns (level 3)
   |   |   |   +-Collection -- tag: tagOneSeriesDefn (level 4)
   |   |   |   |   +-Scalar -- tag: tagQuantityUnitsID (type: UNS_INTEGER4)
   |   |   |   |   |  value: 2 - ID_QU_SECONDS
   |   |   |   |   +-Scalar -- tag: tagQuantityCharacteristicID (type: GUID)
   |   |   |   |   |  value: {a6b31ae5-b451-11d1-ae170060} - ID_QC_NONE
   |   |   |   |   +-Scalar -- tag: tagValueTypeID (type: GUID)
   |   |   |   |   |  value: {c690e862-f755-11cf-9d890080} - ID_SERIES_VALUE_TYPE_TIME
   |   |   |   |   +-Scalar -- tag: tagStorageMethodID (type: UNS_INTEGER4)
   |   |   |   |   |  value: 4 - ID_SERIES_METHOD_INCREMENT
   |   |   |   |   +-(End of collection)
   |   |   |   +-Collection -- tag: tagOneSeriesDefn (level 4)
   |   |   |   |   +-Scalar -- tag: tagQuantityUnitsID (type: UNS_INTEGER4)
   |   |   |   |   |  value: 6 - ID_QU_VOLTS
   |   |   |   |   +-Scalar -- tag: tagQuantityCharacteristicID (type: GUID)
   |   |   |   |   |  value: {a6b31add-b451-11d1-ae170060} - ID_QC_INSTANTANEOUS
   |   |   |   |   +-Scalar -- tag: tagValueTypeID (type: GUID)
   |   |   |   |   |  value: {67f6af97-f753-11cf-9d890080} - ID_SERIES_VALUE_TYPE_VAL
   |   |   |   |   +-Scalar -- tag: tagStorageMethodID (type: UNS_INTEGER4)
   |   |   |   |   |  value: 3 - ID_SERIES_METHOD_VALUES | ID_SERIES_METHOD_SCALED
   |   |   |   |   +-Scalar -- tag: tagSeriesNominalQuantity (type: REAL8)
   |   |   |   |   |  value: 48083.261121
   |   |   |   |   +-(End of collection)
```

```
   |   |   |   +-(End of collection)
   |   |   +-(End of collection)
```

This channel definition specifies a logical channel of type QT_WAVEFORM and is associated with phase A to phase B voltage. By definition, a QT_WAVEFORM channel contains two series definitions - a time series and a value series. The time series is of type ID_SERIES_VALUE_TYPE_TIME using the ID_SERIES_METHOD_INCREMENT storage method. This means that the time series is stored as a relative time (relative to the time stamp of an instantiated channel in an observation) using a simple starting time, increment and count method. This is efficient for instruments that take point-on-wave samples with a fixed sampling rate.

The value series is of type ID_SERIES_VALUE_TYPE_VAL using the ID_SERIES_METHOD_SCALED | ID_SERIES_METHOD_VALUES storage method, and represents the QC_INSTANTANEOUS (point-on-wave) characteristic of the voltage. The storage method is the OR'd value of scaled and values series tags to indicate that the waveform voltage values in an instance of this channel will be represented as an array of integers to which a scale and offset are applied to obtain the original engineering units of the data.

A nominal value is set for this channel for the benefit of reader programs. This is most often used in a reader program to permit displaying the resulting waveform in percent or per-unit.

This channel definition was the fourth channel definition in the data source record. The zero based index of 3 is used in the instance data in an observation to refer to this definition.

The following text shows the ASCII dump of the tags in the monitor settings record in this same PQDIF file:

```
-Collection -- tag: tagRecMonitorSettings (level 0)
   | The checksum for this record is correct.
   +-Scalar -- tag: tagEffective (type: TIMESTAMPPQDIF)
   |  value: 7/6/1999 11:40:45.999999926
   +-Scalar -- tag: tagTimeInstalled (type: TIMESTAMPPQDIF)
   |  value: 1/1/1982 0:0:0.000000000
   +-Scalar -- tag: tagTimeRemoved (type: TIMESTAMPPQDIF)
   |  value: 1/1/2020 0:0:0.000000000
   +-Scalar -- tag: tagUseCalibration (type: BOOLEAN4)
   |  value: 'FALSE'
   +-Scalar -- tag: tagUseTransducer (type: BOOLEAN4)
   |  value: 'FALSE'
```

This monitor settings record simply specifies the time that the instrument was installed, the effective time of the monitor settings record, and whether or not any calibration or transducer settings in the monitor settings record are to be used in combination with the instance data in observation records to obtain the original measured values. Most often, transducer and calibration info is stored in the monitor settings record for reference purposes but the instance data already has these values taken into account in the data values reported there. This file therefore represents the default of FALSE for both entries and in fact, this file does not even specify the optional tags for transducer and calibration info.

The following text shows the ASCII dump of the top level tags in an observation record containing a transient waveform recording in this same PQDIF file:

```
-Collection -- tag: tagRecObservation (level 0)
   | The checksum for this record is correct.
   +-Vector -- tag: tagObservationName (type: CHAR1) [ 4 ]
   |  value: 'Obs'
   +-Scalar -- tag: tagTimeCreate (type: TIMESTAMPPQDIF)
   |  value: 7/7/1999 20:40:58.000000217
   +-Scalar -- tag: tagTimeStart (type: TIMESTAMPPQDIF)
   |  value: 7/6/1999 11:40:45.999999926
   +-Scalar -- tag: tagTriggerMethodID (type: UNS_INTEGER4)
   |  value: 1 - ID_TRIGGER_METH_CHANNEL
   +-Scalar -- tag: tagTimeTriggered (type: TIMESTAMPPQDIF)
   |  value: 7/6/1999 11:40:46.030793991
   +-Vector -- tag: tagChannelTriggerIdx (type: UNS_INTEGER4) [ 1 ]
   |  values: 4
   +-Collection -- tag: tagChannelInstances (level 1)
   |   +-Collection -- tag: tagOneChannelInst (level 2)
   |   |   +-Scalar -- tag: tagChannelDefnIdx (type: UNS_INTEGER4)
   |   |   |  value: 3
   |   |   +-Collection -- tag: tagSeriesInstances (level 3)
   |   |   |   +-Collection -- tag: tagOneSeriesInstance (level 4)
   |   |   |   |   +-Vector -- tag: tagSeriesValues (type: REAL4) [ 3 ]
   |   |   |   |   +-(End of collection)
   |   |   |   +-Collection -- tag: tagOneSeriesInstance (level 4)
   |   |   |   |   +-Vector -- tag: tagSeriesValues (type: INTEGER2) [ 1536 ]
   |   |   |   |   +-Scalar -- tag: tagSeriesBaseQuantity (type: REAL8)
   |   |   |   |   |  value: 34000.000000
   |   |   |   |   +-Scalar -- tag: tagSeriesScale (type: REAL8)
   |   |   |   |   |  value: 1.568127
   |   |   |   |   +-Scalar -- tag: tagSeriesOffset (type: REAL8)
   |   |   |   |   |  value: 0.000000
   |   |   |   |   +-(End of collection)
   |   |   |   +-(End of collection)
   |   |   +-(End of collection)
```

The tags in this observation specifies an optional name "Obs", specifies the time of creation of the record, the absolute time of the first data point in the data, and the time (if applicable) of the trigger that resulted in this observation being saved. Since this is a triggered event, the trigger method tag is specified to be of type channel indicating that a specific channel based algorithm triggered the observation to be recorded. Additionally, the channel index array specifies which channel caused the trigger (channel index 4 - phase B to C voltage - in this case). The tagChannelDefnIdx indicates that this instance data is for the fourth channel definition (zero based index of 3) with in the data source record as shown earlier.

The instance data itself contains the data for both the time and the value series. The time series is first and consists of an array of 3 4 byte floating point numbers representing the number of increment/count pairs (in this case 1), count and increment of the time series. The value series consists of an array of 1,536 two byte integers. This array is scaled by the tagSeriesScale factor and then offset by the tagSeriesOffset factor to get the engineering units for the recorded waveform points.

5.3. Representing a RMS Variation Recording

The RMS trace for an RMS variation disturbance is represented in PQDIF using channel definitions of the QT_PHASOR quantity type. There is another type that will be discussed later (QT_VALUELOG) that could be used for this purpose, but the QT_PHASOR type has been

reserved for this purpose by convention. This type is capable of recording the instantaneous phasor representation of the RMS value of voltage or current.

Most instruments that gather this type of information use a one cycle window to calculate the RMS value of the signal and some are able to calculate the phase angle relative to a reference phase. These instruments can make this measurement as fast as their fundamental A/D resolution allows (e.g. every 1/256 of a cycle) or more commonly, once or twice a cycle. The QT_PHASOR type is ideally suited for representing this information.

The QT_PHASOR type and the QT_VALUELOG type to be discussed later have more options for series definitions than the QT_WAVEFORM type. QT_PHASOR uses a time series as its first series as before, but more than one value series is possible. These variants allow series to be defined to hold minimum, maximum, average, and present values over an interval for a variable. This capability is used by those instruments and simulation programs that change their recording rate for a long disturbance. These data sources typically still track the RMS value on a cycle-by-cycle of faster data rate internally and use that information to maintain max, min, and average registers for a value. Instead of saving this instantaneous data however, they save the min, max, and average values at some prescribed rate (e.g. every 6 cycles), and then reset the min, max, and average registers for the next interval.

The following text is an ASCII dump from a PQDIF utility program that shows the channel definition for one phase of a voltage channel representing recordings of type QT_PHASOR for phase A to B voltagel:

```
+-Collection -- tag: tagOneChannelDefn (level 2)
|   |   +-Vector -- tag: tagChannelName (type: CHAR1) [ 11 ]
|   |   |  value: 'Phasor VAB'
|   |   +-Scalar -- tag: tagPhaseID (type: UNS_INTEGER4)
|   |   |  value: 5 - ID_PHASE_AB
|   |   +-Scalar -- tag: tagQuantityMeasuredID (type: UNS_INTEGER4)
|   |   |  value: 1 - ID_QM_VOLTAGE
|   |   +-Scalar -- tag: tagQuantityTypeID (type: GUID)
|   |   |  value: {67f6af81-f753-11cf-9d890080} - ID_QT_PHASOR
|   |   +-Collection -- tag: tagSeriesDefns (level 3)
|   |   |   +-Collection -- tag: tagOneSeriesDefn (level 4)
|   |   |   |   +-Scalar -- tag: tagQuantityUnitsID (type: UNS_INTEGER4)
|   |   |   |   |  value: 2 - ID_QU_SECONDS
|   |   |   |   +-Scalar -- tag: tagQuantityCharacteristicID (type: GUID)
|   |   |   |   |  value: {a6b31ae5-b451-11d1-ae170060} - ID_QC_RMS
|   |   |   |   +-Scalar -- tag: tagValueTypeID (type: GUID)
|   |   |   |   |  value: {c690e862-f755-11cf-9d890080} - ID_SERIES_VALUE_TYPE_TIME
|   |   |   |   +-Scalar -- tag: tagStorageMethodID (type: UNS_INTEGER4)
|   |   |   |   |  value: 1 - ID_SERIES_METHOD_VALUES
|   |   |   |   +-(End of collection)
|   |   |   +-Collection -- tag: tagOneSeriesDefn (level 4)
|   |   |   |   +-Scalar -- tag: tagQuantityUnitsID (type: UNS_INTEGER4)
|   |   |   |   |  value: 6 - ID_QU_VOLTS
|   |   |   |   +-Scalar -- tag: tagQuantityCharacteristicID (type: GUID)
|   |   |   |   |  value: {a6b31ae5-b451-11d1-ae170060} - ID_QC_RMS
|   |   |   |   +-Scalar -- tag: tagValueTypeID (type: GUID)
|   |   |   |   |  value: {67f6af98-f753-11cf-9d890080} - ID_SERIES_VALUE_TYPE_MIN
|   |   |   |   +-Scalar -- tag: tagStorageMethodID (type: UNS_INTEGER4)
|   |   |   |   |  value: 3 - ID_SERIES_METHOD_VALUES | ID_SERIES_METHOD_SCALED
|   |   |   |   +-Scalar -- tag: tagSeriesNominalQuantity (type: REAL8)
|   |   |   |   |  value: 34000.000000
|   |   |   |   +-(End of collection)
|   |   |   +-Collection -- tag: tagOneSeriesDefn (level 4)
|   |   |   |   +-Scalar -- tag: tagQuantityUnitsID (type: UNS_INTEGER4)
```

```
|   |   |   |   |  value: 6 - ID_QU_VOLTS
|   |   |   |   +-Scalar -- tag: tagQuantityCharacteristicID (type: GUID)
|   |   |   |   |  value: {a6b31ae5-b451-11d1-ae170060} - ID_QC_RMS
|   |   |   |   +-Scalar -- tag: tagValueTypeID (type: GUID)
|   |   |   |   |  value: {67f6af99-f753-11cf-9d890080} - ID_SERIES_VALUE_TYPE_MAX
|   |   |   |   +-Scalar -- tag: tagStorageMethodID (type: UNS_INTEGER4)
|   |   |   |   |  value: 3 - ID_SERIES_METHOD_VALUES | ID_SERIES_METHOD_SCALED
|   |   |   |   +-Scalar -- tag: tagSeriesNominalQuantity (type: REAL8)
|   |   |   |   |  value: 34000.000000
|   |   |   |   +-(End of collection)
|   |   |   +-Collection -- tag: tagOneSeriesDefn (level 4)
|   |   |   |   +-Scalar -- tag: tagQuantityUnitsID (type: UNS_INTEGER4)
|   |   |   |   |  value: 6 - ID_QU_VOLTS
|   |   |   |   +-Scalar -- tag: tagQuantityCharacteristicID (type: GUID)
|   |   |   |   |  value: {a6b31ae5-b451-11d1-ae170060} - ID_QC_RMS
|   |   |   |   +-Scalar -- tag: tagValueTypeID (type: GUID)
|   |   |   |   |  value: {67f6af9a-f753-11cf-9d890080} - ID_SERIES_VALUE_TYPE_AVG
|   |   |   |   +-Scalar -- tag: tagStorageMethodID (type: UNS_INTEGER4)
|   |   |   |   |  value: 3 - ID_SERIES_METHOD_VALUES | ID_SERIES_METHOD_SCALED
|   |   |   |   +-Scalar -- tag: tagSeriesNominalQuantity (type: REAL8)
|   |   |   |   |  value: 34000.000000
|   |   |   |   +-(End of collection)
|   |   |   +-Collection -- tag: tagOneSeriesDefn (level 4)
|   |   |   |   +-Scalar -- tag: tagQuantityUnitsID (type: UNS_INTEGER4)
|   |   |   |   |  value: 17 - ID_QU_DEGREES
|   |   |   |   +-Scalar -- tag: tagQuantityCharacteristicID (type: GUID)
|   |   |   |   |  value: {a6b31add-b451-11d1-ae170060} - ID_QC_INSTANTANEOUS
|   |   |   |   +-Scalar -- tag: tagValueTypeID (type: GUID)
|   |   |   |   |  value: {3d786f9d-f76e-11cf-9d890080} - ID_SERIES_VALUE_TYPE_PHASEANGLE
|   |   |   |   +-Scalar -- tag: tagStorageMethodID (type: UNS_INTEGER4)
|   |   |   |   |  value: 3 - ID_SERIES_METHOD_VALUES | ID_SERIES_METHOD_SCALED
|   |   |   |   +-(End of collection)
|   |   |   +-(End of collection)
|   |   +-(End of collection)
```

This channel definition specifies a logical channel of type QT_PHASOR and is associated with phase A to phase B voltage. As mentioned earlier, the QT_PHASOR channel can contain multiple series definitions. In this case, series definitions are provided for minimum, maximum, average, and phase angle values from the recording. The time series is of type ID_SERIES_VALUE_TYPE_TIME using the ID_SERIES_METHOD_VALUES storage method. The implementers of this PQDIF file chose to use values instead of incremental since this instrument can report its RMS recordings with an irregular time base.

The value series are of types ID_SERIES_VALUE_TYPE_MIN, ID_SERIES_VALUE_TYPE_MAX, ID_SERIES_VALUE_TYPE_AVG, and ID_SERIES_VALUE_TYPE_PHASEANGLE. Each of these use the ID_SERIES_METHOD_SCALED | ID_SERIES_METHOD_VALUES storage method as described for the waveforms earlier. The min, max, and average series represent the RMS value (as indicated by QC_RMS) of these quantities at each time point in the parallel time series. The phase angle series contains the instantaneous value (QC_INSTANTANEOUS) of the phase angle.

A nominal value is set for this channel for the benefit of reader programs. This is most often used in a reader program to permit displaying the resulting waveform in percent or per-unit.

This channel definition was the nineteenth channel definition in the data source record. The zero based index of 18 is used in the instance data in an observation to refer to this definition.

The following text shows the ASCII dump of the tags in the monitor settings record in this same PQDIF file:

```
-Collection -- tag: tagRecMonitorSettings (level 0)
   | The checksum for this record is correct.
   +-Scalar -- tag: tagEffective (type: TIMESTAMPPQDIF)
   |  value: 6/13/1999 19:45:58.999999752
   +-Scalar -- tag: tagTimeInstalled (type: TIMESTAMPPQDIF)
   |  value: 1/1/1982 0:0:0.000000000
   +-Scalar -- tag: tagTimeRemoved (type: TIMESTAMPPQDIF)
   |  value: 1/1/2020 0:0:0.000000000
   +-Scalar -- tag: tagUseCalibration (type: BOOLEAN4)
   |  value: 'FALSE'
   +-Scalar -- tag: tagUseTransducer (type: BOOLEAN4)
   |  value: 'FALSE'
   +-Collection -- tag: tagChannelSettingsArray (level 1)
   |   +-Collection -- tag: tagOneChannelSetting (level 2)
   |   |   +-Scalar -- tag: tagChannelDefnIdx (type: UNS_INTEGER4)
   |   |   |  value: 18
   |   |   +-Scalar -- tag: tagTriggerLow (type: REAL8)
   |   |   |  value: 32300.099609
   |   |   +-Scalar -- tag: tagTriggerHigh (type: REAL8)
   |   |   |  value: 35700.000000
   |   |   +-(End of collection)
   |   +-Collection -- tag: tagOneChannelSetting (level 2)
   |   |   +-Scalar -- tag: tagChannelDefnIdx (type: UNS_INTEGER4)
   |   |   |  value: 19
   |   |   +-Scalar -- tag: tagTriggerLow (type: REAL8)
   |   |   |  value: 32300.099609
   |   |   +-Scalar -- tag: tagTriggerHigh (type: REAL8)
   |   |   |  value: 35700.000000
   |   |   +-(End of collection)
   |   +-Collection -- tag: tagOneChannelSetting (level 2)
   |   |   +-Scalar -- tag: tagChannelDefnIdx (type: UNS_INTEGER4)
   |   |   |  value: 20
   |   |   +-Scalar -- tag: tagTriggerLow (type: REAL8)
   |   |   |  value: 32300.099609
   |   |   +-Scalar -- tag: tagTriggerHigh (type: REAL8)
   |   |   |  value: 35700.000000
   |   |   +-(End of collection)
   |   +-(End of collection)
   +-(End of collection)
   +---- Record information
   | Size on disk: 431
   | The record was compressed; size after decompression: 3644
   +-(End of record)
```

This monitor settings record is similar to that for the waveform example except that this one uses the channel settings array tags to specify the trigger thresholds that were used for this instrument. This information is generally used to facilitate the display of the thresholds in a PQDIF reader and display program. They can also be used by processing algorithms that need to know the original trigger thresholds. Note that the channel settings array uses the tagChannelDefnIdx tag to reference which channel definition in the data source record the settings refer to.

The following text shows the ASCII dump of the top level tags in an observation record containing an RMS variation recording in this same PQDIF file:

```
-Collection -- tag: tagRecObservation (level 0)
   | The checksum for this record is correct.
   +-Vector -- tag: tagObservationName (type: CHAR1) [ 14 ]
   |  value: 'RMS Variation'
   +-Scalar -- tag: tagTimeCreate (type: TIMESTAMPPQDIF)
   |  value: 7/8/1999 8:52:53.000000138
   +-Scalar -- tag: tagTimeStart (type: TIMESTAMPPQDIF)
```

```
|  value: 6/13/1999 19:45:58.999999752
+-Scalar -- tag: tagTriggerMethodID (type: UNS_INTEGER4)
|  value: 1 - ID_TRIGGER_METH_CHANNEL
+-Scalar -- tag: tagTimeTriggered (type: TIMESTAMPPQDIF)
|  value: 6/13/1999 19:45:59.166666144
+-Vector -- tag: tagChannelTriggerIdx (type: UNS_INTEGER4) [ 1 ]
|  values: 19
+-Collection -- tag: tagChannelInstances (level 1)
|   +-Collection -- tag: tagOneChannelInst (level 2)
|   |   +-Scalar -- tag: tagChannelDefnIdx (type: UNS_INTEGER4)
|   |   |  value: 18
|   |   +-Collection -- tag: tagSeriesInstances (level 3)
|   |   |   +-Collection -- tag: tagOneSeriesInstance (level 4)
|   |   |   |   +-Vector -- tag: tagSeriesValues (type: REAL4) [ 59 ]
|   |   |   |   +-(End of collection)
|   |   |   +-Collection -- tag: tagOneSeriesInstance (level 4)
|   |   |   |   +-Vector -- tag: tagSeriesValues (type: INTEGER2) [ 59 ]
|   |   |   |   +-Scalar -- tag: tagSeriesBaseQuantity (type: REAL8)
|   |   |   |   |  value: 34000.000000
|   |   |   |   +-Scalar -- tag: tagSeriesScale (type: REAL8)
|   |   |   |   |  value: 1.077215
|   |   |   |   +-Scalar -- tag: tagSeriesOffset (type: REAL8)
|   |   |   |   |  value: 0.000000
|   |   |   |   +-(End of collection)
|   |   |   +-Collection -- tag: tagOneSeriesInstance (level 4)
|   |   |   |   +-Vector -- tag: tagSeriesValues (type: INTEGER2) [ 59 ]
|   |   |   |   +-Scalar -- tag: tagSeriesBaseQuantity (type: REAL8)
|   |   |   |   |  value: 34000.000000
|   |   |   |   +-Scalar -- tag: tagSeriesScale (type: REAL8)
|   |   |   |   |  value: 1.077215
|   |   |   |   +-Scalar -- tag: tagSeriesOffset (type: REAL8)
|   |   |   |   |  value: 0.000000
|   |   |   |   +-(End of collection)
|   |   |   +-Collection -- tag: tagOneSeriesInstance (level 4)
|   |   |   |   +-Vector -- tag: tagSeriesValues (type: INTEGER2) [ 59 ]
|   |   |   |   +-Scalar -- tag: tagSeriesBaseQuantity (type: REAL8)
|   |   |   |   |  value: 34000.000000
|   |   |   |   +-Scalar -- tag: tagSeriesScale (type: REAL8)
|   |   |   |   |  value: 1.077215
|   |   |   |   +-Scalar -- tag: tagSeriesOffset (type: REAL8)
|   |   |   |   |  value: 0.000000
|   |   |   |   +-(End of collection)
|   |   |   +-(End of collection)
|   |   +-(End of collection)
```

The tags in this observation specifies an optional name "RMS Variation", specifies the time of creation of the record, the absolute time of the first data point in the data, and the time (if applicable) of the trigger that resulted in this observation being saved. Since this is a triggered event, the trigger method tag is specified to be of type channel indicating that a specific channel based algorithm triggered the observation to be recorded. Additionally, the channel index array specifies which channel caused the trigger (channel index 19 - phase B to C voltage - in this case). The tagChannelDefnIdx indicates that this instance data is for the 19 channel definition (zero based index of 18) in the data source record as shown earlier.

The instance data itself contains the data for the time and the value series. Each of these series is of the same length (59 entries) since the time series in this case specified its values explicitly instead of using an increment. The value series use 2 byte integers to represent their values which are then scaled by the tagSeriesScale factor and then offset by the tagSeriesOffset factor to get the engineering units for the recorded RMS trace points. Note that this instance did not specify the phase series. It is permissible to omit series instances even though they are defined for the channel. For the current version of PQDIF, there is no series index specified in the series

instance so the series instances are position sensitive relative to their definitions. This means that to omit the max series, a tagOneSeriesInstance tag must still be used as a place holder. If omitting series at the end of the collection of series instances, such place holders are not required.

Some instruments capture one or more waveforms during an RMS variation in addition to the RMS trace. In this case, the data source record would have channel definitions for QT_WAVEFORM type channels as well as for the QT_PHASOR channels. The observation would contain instances of the waveform and phasor channels as needed. A PQDIF reader program would generally display such an observation in a multi-pane plot - the waveform plot in one pane, the RMS trace plot in another, and the phase angle trace in yet another pane if available.

5.4. Representing Periodically Recorded Steady State Values

Even though PQDIF is designed to represent power quality data, it must also be capable of representing steady state recordings of common electric power characteristics. This data is analogous to recording values that might be gathered by a SCADA system that periodically reports these values or reports them when they change by some amount of exceed some threshold in the steady state. This data is generally gathered over a long period of time. The QT_VALUELOG type was designed to represent this type of data. From a purist point of view, the QT_VALUELOG type could be used to represent the RMS variation data discussed previously. We have chosen however, to separate the longer time frame steady state log type data from short duration variation data by convention.

The QT_VALUELOG channel type is used to create a channelized representation of a steady state quantity log or a log of summary characteristics for disturbances. Such logs are common in a wide variety of instruments. These instruments periodically record the present value of a parameter in a simple file or buffer for later retrieval. These values are characteristics of the measured quantities or characteristics of values calculated from the measured quantities. We use the terms measured quantity and characteristic for a reason. The measured quantity may be voltage, but the recorded characteristic may be its RMS value, peak value, THD, third harmonic component, etc. Additionally, the minimum, maximum and average values recorded for a characteristic during the reporting interval may also be recorded.

Steady state value logs are often recorded in an instrument in a single file or buffer in a time ordered manner. The file or buffer can usually be set up in a fill-and-stop or overwrite mode since we have a finite amount of storage for such logs. A typical native instrument log might look like the following:

```
1998/01/01 00:00:01 UTC PhaseA Present Voltage 117.0
1998/01/01 00:00:01 UTC PhaseB Present Current 23.1
1998/01/01 00:01:00 UTC 3Phase Maximum Power 1234.0
1998/01/01 00:01:00 UTC 3Phase Minimum Power 123.0
1998/01/01 00:01:00 UTC 3Phase Average Power 423.0
1998/01/01 00:01:00 UTC 3Phase Present Power 234.0
1998/01/02 00:02:00 UTC PhaseA Present HarmonicVoltage 10.5 3.0
1998/01/02 00:02:00 UTC PhaseA Present HarmonicVoltage 10.5 5.0
1998/01/02 00:02:00 UTC PhaseA Present HarmonicVoltage 10.5 7.0
```

Note that in such a log, entries of different types with different number of parameters are mixed in the same log. Also note, that a single log will contain all of the entries the instrument has recorded up to the limits of the log file/buffer size. This is not an efficient storage method for an interchange format since it requires the user to parse variable length, non-homogeneous records of unknown time range. PQDIF channelizes, and temporalizes this type of data in a consistent manner.

With this in mind, we are now ready to come up with channel definitions for logged, steady state data. We need to add a channel for every steady state characteristic we wish to record in a PQDIF file. If we set up the instrument to record the rms value, peak value, thd, and individual harmonic magnitudes for three phase voltages, we end up with 12 channels.

Once you have generated the channel definitions to use, we need to decide on how to temporalize the data. To temporalize the data means to break up the log in nice, consistent, manageable chunks based on time stamp. The most common way of doing this is to create separate observation records in PQDIF on daily or weekly boundaries. Once we have done this, we have succeeded in taking a single, non-homogeneous, variable length record log file, and created multiple observations of finite time duration containing multiple channels of data, each of which contains a homogeneous set of characteristics. If we did not break the data up into chunks like this, we would end up with a single, large observation for the data which may not be manageable depending on the time frame. Conversely, we could put single values in a bunch of observations but the overhead would be too high.

You can see the logic of using this time grouping approach if you consider what a typical power system engineer might want to do with a steady state log. Generally, the engineer will want to get the values in a spreadsheet or data analysis program and plot a trend of a particular parameter versus time. If the engineer is given the single event log, he/she would have to manually separate out the specific value quantity, type and phase of interest - a tedious process. With PQDIF, the engineer could use a simple PQDIF to ASCII extraction program (provided with the PQDIF toolkit) to quickly extract a specific quantity, characteristic and phase of interest for one or more observation periods and plot it. A developer writing a visualization or analysis program for PQDIF files would likewise have a much simpler task of extracting the already sorted data (demonstrated in the source code provided for the PQDIF viewer included with the PQDIF toolkit).

This method can result in a large number of logical channels being generated. Are all of these channels necessary? Well, there are many ways to store the contents of our hypothetical steady state log. We could store the data in the native format of the instrument on one extreme or we can store it in a fully decomposed, structured fashion. PQDIF chooses to do the latter to ensure that any kind of data can be readily represented in the format and the result can be easily decoded for analysis and visualization. We point out that PQDIF is an interchange format that is used to extract some subset of data from a source for some period of time and make it available to a third party in a standard way. Even though we could dream up hundreds of possible logical PQDIF channels of information, we need only create channel definitions and channel instances in our PQDIF file for those items that are actually available for the requested time range.

The following ASCII dump of a PQDIF file shows the channel definition for a steady state voltage THD recording:

```
|   +-Collection -- tag: tagOneChannelDefn (level 2)
|   |   +-Vector -- tag: tagChannelName (type: CHAR1) [ 11 ]
|   |   |  value: 'SS RMS VAB'
|   |   +-Scalar -- tag: tagPhaseID (type: UNS_INTEGER4)
|   |   |  value: 5 - ID_PHASE_AB
|   |   +-Scalar -- tag: tagQuantityMeasuredID (type: UNS_INTEGER4)
|   |   |  value: 1 - ID_QM_VOLTAGE
|   |   +-Scalar -- tag: tagQuantityTypeID (type: GUID)
|   |   |  value: {67f6af82-f753-11cf-9d890080} - ID_QT_VALUELOG
|   |   +-Collection -- tag: tagSeriesDefns (level 3)
|   |   |   +-Collection -- tag: tagOneSeriesDefn (level 4)
|   |   |   |   +-Scalar -- tag: tagQuantityUnitsID (type: UNS_INTEGER4)
|   |   |   |   |  value: 2 - ID_QU_SECONDS
|   |   |   |   +-Scalar -- tag: tagQuantityCharacteristicID (type: GUID)
|   |   |   |   |  value: {a6b31ae5-b451-11d1-ae170060} - ID_QC_THD
|   |   |   |   +-Scalar -- tag: tagValueTypeID (type: GUID)
|   |   |   |   |  value: {c690e862-f755-11cf-9d890080} - ID_SERIES_VALUE_TYPE_TIME
|   |   |   |   +-Scalar -- tag: tagStorageMethodID (type: UNS_INTEGER4)
|   |   |   |   |  value: 1 - ID_SERIES_METHOD_VALUES
|   |   |   |   +-(End of collection)
|   |   |   +-Collection -- tag: tagOneSeriesDefn (level 4)
|   |   |   |   +-Scalar -- tag: tagQuantityUnitsID (type: UNS_INTEGER4)
|   |   |   |   |  value: 6 - ID_QU_VOLTS
|   |   |   |   +-Scalar -- tag: tagQuantityCharacteristicID (type: GUID)
|   |   |   |   |  value: {a6b31ae5-b451-11d1-ae170060} - ID_QC_THD
|   |   |   |   +-Scalar -- tag: tagValueTypeID (type: GUID)
|   |   |   |   |  value: {67f6af98-f753-11cf-9d890080} - ID_SERIES_VALUE_TYPE_MIN
|   |   |   |   +-Scalar -- tag: tagStorageMethodID (type: UNS_INTEGER4)
|   |   |   |   |  value: 3 - ID_SERIES_METHOD_VALUES | ID_SERIES_METHOD_SCALED
|   |   |   |   +-Scalar -- tag: tagSeriesNominalQuantity (type: REAL8)
|   |   |   |   |  value: 34000.000000
|   |   |   |   +-(End of collection)
|   |   |   +-Collection -- tag: tagOneSeriesDefn (level 4)
|   |   |   |   +-Scalar -- tag: tagQuantityUnitsID (type: UNS_INTEGER4)
|   |   |   |   |  value: 6 - ID_QU_VOLTS
|   |   |   |   +-Scalar -- tag: tagQuantityCharacteristicID (type: GUID)
|   |   |   |   |  value: {a6b31ae5-b451-11d1-ae170060} - ID_QC_THD
|   |   |   |   +-Scalar -- tag: tagValueTypeID (type: GUID)
|   |   |   |   |  value: {67f6af99-f753-11cf-9d890080} - ID_SERIES_VALUE_TYPE_MAX
|   |   |   |   +-Scalar -- tag: tagStorageMethodID (type: UNS_INTEGER4)
|   |   |   |   |  value: 3 - ID_SERIES_METHOD_VALUES | ID_SERIES_METHOD_SCALED
|   |   |   |   +-Scalar -- tag: tagSeriesNominalQuantity (type: REAL8)
|   |   |   |   |  value: 34000.000000
|   |   |   |   +-(End of collection)
|   |   |   +-Collection -- tag: tagOneSeriesDefn (level 4)
|   |   |   |   +-Scalar -- tag: tagQuantityUnitsID (type: UNS_INTEGER4)
|   |   |   |   |  value: 6 - ID_QU_VOLTS
|   |   |   |   +-Scalar -- tag: tagQuantityCharacteristicID (type: GUID)
|   |   |   |   |  value: {a6b31ae5-b451-11d1-ae170060} - ID_QC_THD
|   |   |   |   +-Scalar -- tag: tagValueTypeID (type: GUID)
|   |   |   |   |  value: {67f6af9a-f753-11cf-9d890080} - ID_SERIES_VALUE_TYPE_AVG
|   |   |   |   +-Scalar -- tag: tagStorageMethodID (type: UNS_INTEGER4)
|   |   |   |   |  value: 3 - ID_SERIES_METHOD_VALUES | ID_SERIES_METHOD_SCALED
|   |   |   |   +-Scalar -- tag: tagSeriesNominalQuantity (type: REAL8)
|   |   |   |   |  value: 34000.000000
|   |   |   |   +-(End of collection)
|   |   |   +-(End of collection)
|   |   +-(End of collection)
```

Note that this value log series definition is very similar to that for an RMS variation. In this case the quantity characteristic is specified as ID_QC_THD to indicate which characteristic of the voltage we are recording. We could have another channel definition for phase A to B voltage with a quantity characteristic of ID_QC_TIF if we wanted to store a recording of the voltage

telephone influence factor. The value of a specific harmonic or arbitrary frequency can be trended by specifying a quantity characteristic of ID_QC_SPECTRA and then using a tag in the instance data to specify for which frequency that instance is for. The following ASCII dump of a PQDIF file shows how this is done:

```
+-Collection -- tag: tagChannelInstances (level 1)
|   +-Collection -- tag: tagOneChannelInst (level 2)
|   |   +-Scalar -- tag: tagChannelDefnIdx (type: UNS_INTEGER4)
|   |   |  value: 29
|   |   +-Scalar -- tag: tagChannelFrequency (type: REAL8)
|   |   |  value: 180.0
|   |   +-Collection -- tag: tagSeriesInstances (level 3)
|   |   |   +-Collection -- tag: tagOneSeriesInstance (level 4)
|   |   |   |   +-Vector -- tag: tagSeriesValues (type: REAL4) [ 96 ]
|   |   |   |   +-(End of collection)
|   |   |   +-Collection -- tag: tagOneSeriesInstance (level 4)
|   |   |   |   +-Vector -- tag: tagSeriesValues (type: INTEGER2) [ 96 ]
|   |   |   |   +-Scalar -- tag: tagSeriesBaseQuantity (type: REAL8)
|   |   |   |   |  value: 34000.000000
|   |   |   |   +-Scalar -- tag: tagSeriesScale (type: REAL8)
|   |   |   |   |  value: 1.093739
|   |   |   |   +-Scalar -- tag: tagSeriesOffset (type: REAL8)
|   |   |   |   |  value: 0.000000
|   |   |   |   +-(End of collection)
|   |   |   +-Collection -- tag: tagOneSeriesInstance (level 4)
|   |   |   |   +-Vector -- tag: tagSeriesValues (type: INTEGER2) [ 96 ]
|   |   |   |   +-Scalar -- tag: tagSeriesBaseQuantity (type: REAL8)
|   |   |   |   |  value: 34000.000000
|   |   |   |   +-Scalar -- tag: tagSeriesScale (type: REAL8)
|   |   |   |   |  value: 1.093739
|   |   |   |   +-Scalar -- tag: tagSeriesOffset (type: REAL8)
|   |   |   |   |  value: 0.000000
|   |   |   |   +-(End of collection)
|   |   |   +-Collection -- tag: tagOneSeriesInstance (level 4)
|   |   |   |   +-Vector -- tag: tagSeriesValues (type: INTEGER2) [ 96 ]
|   |   |   |   +-Scalar -- tag: tagSeriesBaseQuantity (type: REAL8)
|   |   |   |   |  value: 34000.000000
|   |   |   |   +-Scalar -- tag: tagSeriesScale (type: REAL8)
|   |   |   |   |  value: 1.093739
|   |   |   |   +-Scalar -- tag: tagSeriesOffset (type: REAL8)
|   |   |   |   |  value: 0.000000
|   |   |   |   +-(End of collection)
|   |   |   +-(End of collection)
|   |   +-(End of collection)
```

In this observation, the minimum, maximum, and average values over a 24 hour period at 15 minute intervals for the third harmonic (180 Hz) is recorded. If additional frequencies are to be recorded for the same voltage, additional channel instances are instantiated referencing the same channel definition but with different tagChannelFrequency values.

This example had minimum, maximum, and average series for this interval data. There is often the need to have a series that represents some other statistical calculation. The most common example of this is for recording flicker or harmonic values measured using the IEC methodology. The IEC methodology results in measurements of flicker or a harmonic characteristic every 10 minutes. Each 10 minute value is based on a statistical calculation of a lot of individual measurements taken during the 10 minute interval using a method prescribed in the relevant IEC standards. Most often, the 95 percentile value is reported but other percentiles may be calculated and retained as well (e.g. 99%, 5%, 1%, etc.). When series of this type are required, the ID_SERIES_VALUE_TYPE_VAL is used for the series definitions in the channel definition. This is then qualified by using the tagProbPercentile tag to specify the percentile for this

statistically calculated value. Optionally, the tagProbInterval can be used to indicate the time interval over which the calculation was done - in this case it would be reported as 600 seconds. Individual series definitions are then generated for each desired probability percentile/interval combination. In the case of IEC harmonic measurements, this technique is combined with the multiple series instance technique described previously to instantiate the same channel definition multiple times - one for each frequency component.

In addition to the IEC 10 minute intervals, IEC flicker and harmonic measurements have additional time frames that are often reported. For example, it is common to save a 95 percentile value for flicker and harmonic quantities at three hour, daily, and weekly intervals. Separate channel definitions are created for these alternate time intervals since they can not share the same common time series as the 10 minute interval data series.

The next type of value log we have is for value transitions. The examples given so far for value logs have been for periodically recorded data. Periodic data is denoted in the instance data by setting the tagTriggerMethodID tag to ID_TRIGGER_METH_PERIODIC. Many instruments record the value of a "steady state" variable when a transition of some sort occurs. This may be a simple threshold transition, or a deadband, or some external algorithm. When this kind of data is available, there are two methods provided in PQDIF to record it. It is differentiated from the periodic data by setting the tagTriggerMethodID tag to ID_TRIGGER_METH_CHANNEL, ID_TRIGGER_METH_EXTERNAL, or ID_TRIGGER_METH_OTHER. As we will describe in the following paragraphs, more information about the trigger type is stored in another tag or series.

The first method uses the same basic approach for the series definition as used in the periodic value log examples shown previously. The difference is that an additional series definition is used that contains a code for the transition type for each recorded value in the value series. For example, the transition code ID_CTT_NORMAL_TO_LO indicates that the value was recorded because the LO threshold was crossed and its previous known state was within the normal range. The actual value of the thresholds used by the instrument to make these transition decisions are optionally provided in a monitor settings record. Using a third series for the transition codes allows a single observation to contain many transition events and is more efficient from a storage point of view.

The second method is to store only one transition event in a single observation. This is more in line with the philosophy of representing events in PQDIF. In general we represent steady state data in arbitrarily grouped chunks and we represent event type data in distinct observation records. To do this for transition events, we use the same channel definition as before for value log items with a time series and a value series. In this case however, the instance data series are of length 1 and the transition code is placed in the channel instance data (tagChanTriggerTypeID). This is a lot of overhead just to store a single value, time stamp, and transition code but it is the most general and in keeping with the design philosophy. For this version of the PQDIF standard both methods described here should be supported by PQDIF reader programs.

In addition to the basic transition codes, there are codes to indicate that the value was saved due to a manual, periodic, internal cross trigger, external cross trigger, or algorithmic defined trigger mechanism. In the case of a value recorded due to some special algorithm, the ID_CTT_MODULE transition code can be used. Tags also exist (tagChanTriggerModuleName and tagChanTriggerModuleInfo) to specify the name of the module or algorithm that caused the trigger as well as an associated 32 bit integer that is meaningful in this context. ID's and tags are also provided to qualify the source of external and internal cross triggers. See the descriptions of these tags in Annex B.

The final type of log that can be represented using QT_VALUELOG is a power quality event summary log. Such a log may be generated from an instrument that simply records the fact that a particular type of IEEE 1159 defined event has occurred and may optionally report the basic characteristics of magnitude, duration and if appropriate, frequency. Tags are provided for the instance data to specify the IEEE 1159 disturbance type and the magnitude, duration, and frequency characteristics. In the most basic case, there is no need for any series definitions or instance data - just the IEEE 1159 type and characteristic tags are used in the observation data.

Generally, a single set of channel definitions can be used for all usage's of the QT_VALUELOG type. Proper decoding of the trigger type and method tags, transition series, and other special tags described in the paragraphs above allow a PQDIF reader to separate out triggered versus non-triggered values and steady state versus power quality event transitions for display purposes.

6. Annex A - Physical format definitions

```
/*

**  PQDIF - Power Quality Data Interchange Format

**  Version 1.5

**

**  File name:          $Workfile: pqdif_ph.h $
**  Last modified:      $Modtime: 3/05/98 2:48p $
**  Last modified by:   $Author: Rob $
**
**  VCS archive path:   $Archive: /PQDIF/Document/Version15/pqdif_ph.h $
**  VCS revision:       $Revision: 16 $
**
**  PHYSICAL FORMAT DEFINITIONS
**  ===========================================================================
**  This file contains the complete definitions for the physical
**  format of a PQDIF file. It contains no information about the
**  _logical_ structure, which is defined in PQDIF_LG.H
**
**  ===========================================================================
**  The current version of this file and related information
**  can be found at URL:
**
**  http://www.pqnet.electrotek.com/pqnet/main/measure/pqdif/pqdif.htm
**
**  This site can be freely accessed after registration at root URL
**  (http://www.pqnet.electrotek.com)
**  ===========================================================================
*/
#ifndef PQDIF_PH_H
#define PQDIF_PH_H


//  The structures must be 1-byte packed.
#ifdef __BORLANDC__
    #pragma pack( 1 )
#elif _MSC_VER
    #pragma pack( push, 1 )
#else
    #pragma pack( 1 )
#endif


/*
**  PHYSICAL TYPE IDs AND DEFINITIONS
**  ===========================================================================
**  The physical types are intended to be fully portable.
**  ===========================================================================
*/

/*
**  Physical representation IDs
**
**  NOTE:   Larger objects such as strings and BLOBs (Binary Large OBject)
**          are represented as vectors of other primitive types.
**
**  Examples
**  --------
**      ASCII string    Vector of ID_PHYS_TYPE_CHAR1, NULL-terminated
**                      (i.e., the last character must be a NULL, to
**                      correspond to a C-style string).
**
**      Unicode string  Vector of ID_PHYS_TYPE_CHAR2.
**
**      BLOB            vector of ID_PHYS_TYPE_INTEGER1 or
**                      ID_PHYS_TYPE_UNS_INTEGER1.
*/
```

```
#define ID_PHYS_TYPE_BOOLEAN1          1
#define ID_PHYS_TYPE_BOOLEAN2          2
#define ID_PHYS_TYPE_BOOLEAN4          3

#define ID_PHYS_TYPE_CHAR1             10  //  ASCII
#define ID_PHYS_TYPE_CHAR2             11  //  Unicode

//  Signed integers
#define ID_PHYS_TYPE_INTEGER1          20
#define ID_PHYS_TYPE_INTEGER2          21
#define ID_PHYS_TYPE_INTEGER4          22

//  Unsigned integers
#define ID_PHYS_TYPE_UNS_INTEGER1      30
#define ID_PHYS_TYPE_UNS_INTEGER2      31
#define ID_PHYS_TYPE_UNS_INTEGER4      32

//  Real/complex
#define ID_PHYS_TYPE_REAL4             40
#define ID_PHYS_TYPE_REAL8             41
#define ID_PHYS_TYPE_COMPLEX8          42  //  Two REAL4s: real, imag
#define ID_PHYS_TYPE_COMPLEX16         43  //  Two REAL8s: real, imag

//  Date/time variations
#define ID_PHYS_TYPE_TIMESTAMPPQDIF    50  //  Physical: TIMESTAMPPQDIF (total 12 bytes)

//  GUID
#define ID_PHYS_TYPE_GUID              60  //  Physical: GUID (total 16 bytes)


/*
**  Portable primitive type definitions
*/
typedef char            BOOL1;
typedef short           BOOL2;
typedef long            BOOL4;

typedef char            CHAR1;  //  ASCII string character
typedef short           CHAR2;  //  Unicode string character

typedef char            INT1;
typedef short           INT2;
typedef long            INT4;

typedef unsigned char   UINT1;
typedef unsigned short  UINT2;
typedef unsigned long   UINT4;

typedef float           REAL4;
typedef double          REAL8;

typedef struct _complex8
    {
    REAL4   real;
    REAL4   imag;
    } COMPLEX8;

typedef struct _complex16
    {
    REAL8   real;
    REAL8   imag;
    } COMPLEX16;

/*
**  Types used in structures (but not used as primitives for elements)
*/
typedef long            LINKABS4;
typedef long            LINKREL4;
typedef long            SIZE4;


/*
**  Portable physical time structure definition
**
**  (Modified version of time tracking used by Excel
```

```
**  Excel counts days since 1900, converts to a double
**  and adds to this a number less than 1.0 that
**  is the fractional day.  This structure enhances
**  accuracy of this method by separating out the
**  days since 1900 and seconds since midnight.)
*/
typedef struct ts
    {
    UINT4   day;        //  days since January 1, 1900 UCT
                        //  0xFFFFFFFF -> 4294967295 days -> a long time
    REAL8   sec;        //  fractional seconds since midnight of day
    } TIMESTAMPPQDIF;


/*
**  GUID (Globally Unique IDentifier) definition
**  (Used for tagging sections to identify them logically)
**
**  The Unique Universal Identifier (UUID) is also known
**  as a Globally Unique Identifier (GUID).  It is
**  a randomly generated number that due to its size,
**  is guaranteed to be unique in the universe.  This
**  guarantees that the tags will be unique and anyone
**  can generate "private" tags which will never conflict
**  with the standard tags.
**
**  GUID tags are used to mark the header of each record,
**  as well as identify the elements in the internal structures.
*/
#ifndef GUID_DEFINED
#define GUID_DEFINED
typedef struct _GUID
    {
    unsigned long  Data1;
    unsigned short Data2;
    unsigned short Data3;
    unsigned char  Data4[8];
    } GUID;
#endif /* GUID_DEFINED */


/*
**  PHYSICAL TYPE HELPER INFORMATION
**  ================================================================
**  Time structure helpers
**
**  The following constant is the number of days between
**  1/1/1900 and 1/1/1970.  This is used to convert between
**  "C Time" and Excel style day counts
**  "C Time" is a 4 byte integer representing the number of
**  seconds elapsed since  January 1, 1970.
**  Excel time is an 8 byte real number that represents
**  the number of days elapsed since 1/1/1900.  The fractional
**  part is therefore convertable to seconds since midnight.
*/
#define EXCEL_DAYCOUNT_ADJUST   25569L
#define SECONDS_PER_DAY         86400L


/*
**  Define GUID helpers
*/
#define PQDIF_IsEqualGUID(rguid1, rguid2) (!memcmp(&rguid1, &rguid2, sizeof(GUID)))
#define PQDIF_DEFINE_GUID(name, l, w1, w2, b1, b2, b3, b4, b5, b6, b7, b8) \
        const GUID name = { l, w1, w2, { b1, b2,  b3,  b4,  b5,  b6,  b7,  b8 } }



/*
**  HIGH-LEVEL FILE STRUCTURE
**  ================================================================
**  The top-level structure is a series of independent records with
**  header and body sections.
**
**  ================================================================
```

```
**  Fundamental premise for file format writing: File must be capable of
**  being written to incrementally. For this reason, the basic structure
**  is a series of independent records. Additional records can be appended
**  to the file at any time.
**
**  ========================================================================
**  The top-level structure is a series of independent records with
**  header and body sections:
**
**      +--------------------------------+
**      | Record 0 Header                |
**  /---+--------------------------------+
**  |   | Record 0 Body                  |
**  |   +--------------------------------+
**  \-->+--------------------------------+
**      | Record 1 Header                |
**  /---+--------------------------------+
**  |   | Record 1 Body                  |
**  |   +--------------------------------+
** ... ...
**  \-->+--------------------------------+
**      | Record n Header                |
**      +--------------------------------+
**      | Record n Body                  |
**      +--------------------------------+
**
**  ========================================================================
**  Note that each record header has a LINKABS4 -- an absolute offset in
**  the file -- to the next record. This allows new records to be inserted
**  in the middle of the file, and obsolete records to be deleted.
**
**  ========================================================================
**  All references which are defined by the type LINKREL4 are offsets
**  within the record body itself -- they are not absolute offsets to the
**  entire file. Thus, each record body is independent of the others. Only
**  the record header need be modified if a record is moved, deleted, or
**  inserted.
**
**  ========================================================================
*/


/*
**  Record header
**
**  The header contains a GUID signature (for testing corruptness),
**  header size, the size of the record that follows, record tag, and
**  fill bytes to make the header 64 bytes in length
*/
struct c_record_mainheader
    {
    GUID        guidRecordSignature;    //  Signature to verify a valid header

    GUID        tagRecordType;          //  Tag to identify the record type
                                        //  (This also identifies the first
                                        //  collection in the record.)

    SIZE4       sizeHeader;             //  Size of this header in bytes

    SIZE4       sizeData;               //  Size of the body in bytes (record data
                                        //  that follows header)

    LINKABS4    linkNextRecord;         //  Offset to the next record -- absolute
                                        //  reference within the file. If 0, this
                                        //  is the last record in the file.

    UINT4       checksum;               //  Optional checksum (such as a 32-bit CRC)
                                        //  of the record body to verify decompression.

    UINT4       auiReserved[ 4 ];       //  Reserved to fill structure to 64 bytes
                                        //  -- should be filled with 0
    };
```

```
/*
**  RECORD BODY STRUCTURE
**  ====================================================================
**  The record body always begins with a Collection element. This is
**  defined below in the record structure.
**
**  ====================================================================
**  The fundamental premise for the standard structures is to
**  assure 4 byte alignment. All of the structures conform to this
**  premise. In addition, the data values associated with scalars and
**  vectors must be padded out to 4-byte multiples. This _total_ size
**  will be specified in the c_collection_element structure (sizeElement).
**
**  ====================================================================
**  The three main elements are:
**
**      1. Collection   Holds an array of pointers to other elements
**                      (it could contain another collection)
**      2. Scalar       Holds a single data value (of a physical type)
**      3. Vector       Holds an array of data values (of a physical type)
**  ====================================================================
*/

#define ID_ELEMENT_TYPE_COLLECTION  1
#define ID_ELEMENT_TYPE_SCALAR      2
#define ID_ELEMENT_TYPE_VECTOR      3


/*
**  The collection element
**
**  ... is made by combining a c_collection with an array of
**  c_collection_element, which provide pointers to the
**  other elements in the collection.
*/
struct c_collection
    {
    SIZE4                   count;          //  The number of elements in this
                                            //  collection.

//  c_collection_element    ceElements[];   //  Array [count] of the elements
                                            //  in the collection.
    };

struct c_collection_element
    {
    GUID        tagElement;     //  Identifier for this element in the collection.

    //  (4) 1-byte members keep the structure 4-byte aligned.
    INT1        typeElement;    //  Type of element (ID_ELEMENT_TYPE_COLLECTION,
                                //  _SCALAR or _VECTOR).

    INT1        typePhysical;   //  Physical type of the value which follows
                                //  (ID_PHYS_TYPE_INTEGER1, etc.). This is unused
                                //  if the element is a collection and should be
                                //  set to 0.

    BOOL1       isEmbedded;     //  FALSE (0) - use the link to find the next element
                                //  TRUE  (1) - the scalar data value is embedded
                                //              (may not be used with vectors)

    INT1        reserved;       //  Fill with 0

    //  The following 8 bytes can point to another location in the record
    //  or it can hold the actual data value (if it fits in 8 bytes).
    //  This allows small scalars to be stored with much less space overhead.
    union
        {
        //  isEmbedded  What to use
        //  ----------  -----------------------------------------------
        //  FALSE       Use link to find the next element (and its size).
        //  TRUE        Use valueEmbedded to find the data directly.
        struct
            {
            LINKREL4    linkElement;    //  Offset to the element -- this offset
```

```
                                    //  is relative within the record body.

            SIZE4       sizeElement;    //  Specifies actual size of the element
                                        //  -- should be padded to even
                                        //  multiples of 4 bytes
            } link;

        UINT1       valueEmbedded[ 8 ]; //  The scalar data value
                                        //  (less than or equal to 8 bytes)
        };
    };


/*
**  The scalar element
**
**  This structure has no members, but is included
**  for completeness.
**
**      struct c_scalar
**          {
**      //  (type)      value;          //  A single value follows of variable length,
**                                      //  depending on the physical type.
**          };
*/

/*
**  The vector element
*/
struct c_vector
    {
    SIZE4       count;
//  (type)      values[];       //  An array of values [count] follows of
                                //  variable length, depending on the
                                //  physical type.
    };


/*
**  NOTE
**  ========================================================================
**  For more detailed information about how to use these physical
**  structures to create a PQDIF file, see the logical structure header
**  file -- PQDIF_LG.H
*/

//  Return to previous packing value
#ifdef __BORLANDC__
  #include <poppack.h>
#elif _MSC_VER
    #pragma pack( pop, 1 )
#else
    #pragma pack()
#endif

#endif  //  PQDIF_PH_H
```

7. Annex B - Logical structure documentation

The next page begins a set of documentation about the logical structure; which tags are required or optional, what element type and physical type they should be, and where they should appear in the hierarchy.

A tag at the far left (level 0) means a record-level tag. These are the only tags that appear in the record header itself. Example:

Element tag	Element	Physical	Count	*VALID CONTENTS*	Required?	Description	Ver
tagContainer	Collection		*		Required	Record-level tag which identifies the container	

The other levels (1 through 5) denote various levels of hierarchy (using collections) within the records. Note that the full hierarchy will *not* be repeated if a section crosses over to the next page.

The Count column is used for collections and vectors. In general, no specific number of items is required; this is indicated by a * (asterisk):

1	tagFileName	Vector	CHAR1	*	Required	Original name of the file.	1.0

In some cases, a vector of a specific size is specified. The indices into the vector are usually documented as well:

1	tagVersionInfo	Vector	UINT4	4	Required	Specifies the format version for read/write	1.0
					…		
				Vector [0] - Major version		*Version of the PQDIF format used for this file.*	*1.0*
				…			

In other cases, there is no specific count, but the size of the collection or vector may be used elsewhere, or may be indexed from elsewhere:

1	tagChannelDefns	Collection	(n/a)	# defs	Required	The tagChannelDefns collection must be a	1.0

For many scalar elements (whose name ends in ID), the “Valid contents” column specifies the expected values.

1	tagCompressionStyleID	Scalar	UINT4		Optional	Specified how the compression is applied to the file.	1.0
				ID_COMP_STYLE_NONE = 0		*No compression is used.*	*1.0*
				ID_COMP_STYLE_RECORDLEVEL = 2		*The body of each record is compressed individually. The checksums will be found in the header of each record.*	*1.0*

Usually, these are specific values that are mutually exclusive. In other words, it expects a single element with a single value. There are a few cases (such as *tagStorageMethodID*) in which some of the values are masks—that is, you can OR them together. For example, you could specify:

scalar(tagStorageMethodID) = ID_SERIES_METHOD_VALUES
scalar(tagStorageMethodID) = ID_SERIES_METHOD_SCALED | ID_SERIES_METHOD_INCREMENT

However, this would be invalid:

scalar(tagStorageMethodID) = ID_SERIES_METHOD_VALUES | ID_SERIES_METHOD_SCALED | ID_SERIES_METHOD_INCREMENT

Updated 4/27/99 10:45:37 AM

Element tag	Element	Physical	Count	*VALID CONTENTS*	Required?	Description	Ver
tagContainer	Collection		*		Required	Record-level tag which identifies the container record (always the first one in the file, and there must be only one per file).	
1 tagVersionInfo	Vector	UINT4	4		Required	Specifies the format version for read/write compatibility. The four required numbers in the vector are described below.	1.0
				Vector [0] - Major version		*Version of the PQDIF format used for this file.*	*1.0*
				Vector [1] - Minor version			*1.0*
				Vector [2] - Major version compatible		*The "compatible" information is used when writing a file that conforms to a future version of the logical structure, but can still be read by an earlier program.*	*1.0*
				Vector [3] - Minor version compatible			*1.0*
1 tagFileName	Vector	CHAR1	*		Required	Original name of the file.	1.0
1 tagCreation	Scalar	TIMESTAMP			Required	Date/time when the file was created.	1.0
1 tagLastSaved	Scalar	TIMESTAMP			Optional	Date/time when the file was last saved.	1.0
1 tagTimesSaved	Scalar	UINT4			Optional	The number of times the file has been saved/modified.	1.0
1 tagLanguage	Vector	CHAR1	*		Optional	The language (English, etc.) of the file.	1.0
1 tagTitle	Vector	CHAR1	*		Optional	Arbitrary title.	1.0
						This and some of the following fields are part of the standard summary information used in OLE compound files. When a compound file is created from a PQDIF standard file, this info can be copied to the summary stream. This data is used by Microsoft operating system functions to assist in finding files in the file system.	*1.0*
1 tagSubject	Vector	CHAR1	*		Optional	Arbitrary subject string	1.0
1 tagAuthor	Vector	CHAR1	*		Optional	Individual/company who caused the file to be written	1.0
1 tagKeywords	Vector	CHAR1	*		Optional	Keywords for assisting searches	1.0
1 tagComments	Vector	CHAR1	*		Optional	Arbitrary comments	1.0
1 tagLastSavedBy	Vector	CHAR1	*		Optional	Individual/company who last wrote to file	1.0
1 tagApplication	Vector	CHAR1	*		Optional	Creating application	1.0
1 tagSecurity	Vector	CHAR1	*		Optional	Security descriptor information	1.0
1 tagOwner	Vector	CHAR1	*		Optional	Owner of file contents (This and some of the following fields are for copyright and trademark information)	1.0
1 tagCopyright	Vector	CHAR1	*		Optional	Copyright notice	1.0
1 tagTrademarks	Vector	CHAR1	*		Optional	Trademark notice	1.0
1 tagNotes	Vector	CHAR1	*		Optional	RTF formatted notes associated with this file (This corresponds to the IEEE COMTRADE .HDR file, for example).	1.0
1 tagCompressionStyleID	Scalar	UINT4			Optional	Specified how the compression is applied to the file.	1.0
				ID_COMP_STYLE_NONE = 0		*No compression is used.*	*1.0*
				ID_COMP_STYLE_RECORDLEVEL = 2		*The body of each record is compressed individually. The checksums will be found in the header of each record.*	*1.0*
				ID_COMP_STYLE_TOTALFILE = 1		*Everything after the container record is compressed as a single block. This feature has been deprecated under 1.5 and should not be used.*	*1.5 DEP*

Element tag	Element	Physical	Count	*VALID CONTENTS*	Required?	Description	Ver
1 tagCompressionAlgorithmID	Scalar	UINT4			Optional	Required if tagCompressionStyleID specifies that compression is turned on.	1.0
				ID_COMP_ALG_NONE = 0		*No compression algorithm is used.*	*1.0*
				ID_COMP_ALG_ZLIB = 1		*A standard compression algorithm -- ZLIB -- standardized by the IETF (Internet Engineering Task Force). See http://quest.jpl.nasa.gov/zlib/ for details.*	*1.0*
				ID_COMP_ALG_PKZIPCL = 64		*A commercial package, the PKZIP data compression library, was used to compress the data. This feature has been deprecated under 1.5 and should not be used.*	*1.5 DEP*
1 tagCompressionChecksum	Scalar	UINT4			Optional	If compression style is _TOTALFILE, this is the checksum for the entire file. This feature has been deprecated under 1.5.	1.5 DEP
1 tagName	Scalar	UINT4			Optional	General contact information (all optional)	1.0
1 tagAddress1	Scalar	UINT4			Optional	General contact information (all optional)	1.0
1 tagAddress2	Scalar	UINT4			Optional	General contact information (all optional)	1.0
1 tagCity	Scalar	UINT4			Optional	General contact information (all optional)	1.0
1 tagState	Scalar	UINT4			Optional	General contact information (all optional)	1.0
1 tagPostalCode	Scalar	UINT4			Optional	General contact information (all optional)	1.0
1 tagCountry	Scalar	UINT4			Optional	General contact information (all optional)	1.0
1 tagPhoneVoice	Scalar	UINT4			Optional	General contact information (all optional)	1.0
1 tagPhoneFAX	Scalar	UINT4			Optional	General contact information (all optional)	1.0
1 tagEMail	Scalar	UINT4			Optional	General contact information (all optional)	1.0

Element tag	Element	Physical	Count	*VALID CONTENTS*	Required?	Description	Ver
tagRecDataSource	Collection		*		Required	Record-level tag which identifies a data source (an instrument, etc.).	
1 tagDataSourceTypeID	Scalar	GUID			Required	Since this ID is a GUID, you can generate a custom ID if a standard ID is not defined.	1.0
				ID_DS_TYPE_MEASURE			*1.0*
				ID_DS_TYPE_MANUAL			*1.0*
				ID_DS_TYPE_SIMULATE			*1.0*
				ID_DS_TYPE_BENCHMARK			*1.0*
				ID_DS_TYPE_DEBUG			*1.0*
1 tagVendorID	Scalar	GUID			Optional	Since this ID is a GUID, you can generate a custom ID if a standard ID is not defined.	1.0
				ID_VENDOR_NONE			
				ID_VENDOR_BMI			
				ID_VENDOR_BPA			
				ID_VENDOR_CESI			
				ID_VENDOR_COOPER			
				ID_VENDOR_DCG			
				ID_VENDOR_DRANETZ			
				ID_VENDOR_EDF			
				ID_VENDOR_EPRI			
				ID_VENDOR_ELECTROTEK			
				ID_VENDOR_FLUKE			
				ID_VENDOR_HYDROQUEBEC			
				ID_VENDOR_IEEE			
				ID_VENDOR_KREISSJOHNSON			
				ID_VENDOR_METROSONIC			
				ID_VENDOR_PML			
				ID_VENDOR_PSI			
				ID_VENDOR_PTI			
				ID_VENDOR_PUBLICDOMAIN			
				ID_VENDOR_RPM			
				ID_VENDOR_SQUAREDPOWERLOGIC			
				ID_VENDOR_TELOG			
1 tagEquipmentID	Scalar	GUID			Optional	Since this ID is a GUID, you can generate a custom ID if a standard ID is not defined.	1.0
				ID_EQUIP_BMI_7100			
				ID_EQUIP_BMI_8010			
				ID_EQUIP_BMI_8020			
				ID_EQUIP_BMI_9010			
				ID_EQUIP_COOPER_VHARM			
				ID_EQUIP_COOPER_VFLICKER			
				ID_EQUIP_DCG_EMTP			
				ID_EQUIP_DRANETZ_656			
				ID_EQUIP_DRANETZ_658			
				ID_EQUIP_ETK_TESTPROGRAM			
				ID_EQUIP_DRANETZ_8000			
				ID_EQUIP_ETK_PQDIFEDITOR			
				ID_EQUIP_ETK_PASS			
				ID_EQUIP_ETK_SUPERHARM			
				ID_EQUIP_ETK_SUPERTRAN			
				ID_EQUIP_ETK_TOP			
				ID_EQUIP_ETK_PQVIEW			
				ID_EQUIP_ETK_HARMONI			

Element tag	Element	Physical	Count	*VALID CONTENTS*	Required?	Description	Ver
				ID_EQUIP_FLUKE_CUR *ID_EQUIP_IEEE_COMTRADE* *ID_EQUIP_FLUKE_F41* *ID_EQUIP_PUBLIC_ATP* *ID_EQUIP_METROSONIC_M1* *ID_EQUIP_SQD_SMS* *ID_EQUIP_TELOG_M1* *ID_EQUIP_PML_3710* *ID_EQUIP_PML_3720* *ID_EQUIP_PML_3800* *ID_EQUIP_PML_7300* *ID_EQUIP_PML_7700* *ID_EQUIP_PML_VIP* *ID_EQUIP_PML_LOGSERVER*			
1 tagCustomSourceInfo	Collection	(n/a)			Optional	This collection can include the standard name, address & telephone number tags -- these apply to the vendor -- as well as tags about the instrument itself.	1.0
2 tagInstrumentTypeID	Scalar	GUID			Optional		1.0
				ID_INSTR_TYPE_SCOPE			*1.0*
				ID_INSTR_TYPE_FR		*Frequency recorder.*	*1.0*
				ID_INSTR_TYPE_PQM		*Power quality meter.*	*1.0*
				ID_INSTR_TYPE_VR		*Voltage recorder.*	*1.0*
				ID_INSTR_TYPE_SA			*1.0*
2 tagInstrumentModelName	Vector	CHAR1	*		Optional	Arbitrary string	1.0
2 tagInstrumentModelNumber	Vector	CHAR1	*		Optional	Arbitrary string	1.0
1 tagSerialNumberDS	Vector	CHAR1	*		Optional	Arbitrary string to hold data souce (instrument) serial number	1.0
1 tagVersionDS	Vector	CHAR1	*		Optional	Arbitrary string to hold data souce (instrument) version number (if applicable)	1.0
1 tagNameDS	Vector	CHAR1	*		Required	Arbitrary string to hold the name of the data souce (instrument)	1.0
1 tagOwnerDS	Vector	CHAR1	*		Optional	Arbitrary string to hold data souce (instrument) owner name	1.0
1 tagLocationDS	Vector	CHAR1	*		Optional	Arbitrary string to hold data souce (instrument) location information	1.0
1 tagTimeZoneDS	Vector	CHAR1	*		Optional	EST, CST, etc.	1.0
1 tagCoordinatesDS	Vector	UINT4	2		Optional	Longitude/latitude	1.0
1 tagChannelDefns	Collection	(n/a)	# defs		Required	The tagChannelDefns collection must be a collection where the count = the number of channel definitions. Each entry must be another collection, each having the tagOneChannelDefn tag.	1.0
2 tagOneChannelDefn	Collection	(n/a)	*		Required	Must have one or more channel definitions	1.0
3 tagChannelName	Vector	CHAR1	*		Optional	Arbitrary string	1.0
3 tagPhaseID	Scalar	UINT4			Required	Phase identifier	1.0
				ID_PHASE_NONE = 0		*Phase is not applicable.*	*1.0*
				ID_PHASE_AN = 1		*A-to-neutral.*	*1.0*
				ID_PHASE_BN = 2		*B-to-neutral.*	*1.0*
				ID_PHASE_CN = 3		*C-to-neutral.*	*1.0*

Element tag	Element	Physical	Count	*VALID CONTENTS*	Required?	Description	Ver
				ID_PHASE_NG = 4		*Neutral-to-ground.*	*1.0*
				ID_PHASE_AB = 5		*A-to-B.*	*1.0*
				ID_PHASE_BC = 6		*B-to-C.*	*1.0*
				ID_PHASE_CA = 7		*C-to-A.*	*1.0*
				ID_PHASE_RES = 8		*Residual - the vector or point-on-wave sum of Phases A, B, and C. Should be zero in a perfectly balanced system.*	*1.0*
				ID_PHASE_NET = 9		*Net - the vector or point-on-wave sum of Phases A, B, C and the Neutral phase. Should be zero in a 4 wire system with no earth return path.*	*1.0*
				ID_PHASE_PSEQ = 10		*Positive sequence. This has been deprecated under 1.5 and should not be used.*	
				ID_PHASE_NSEQ = 11		*Negative sequence. This has been deprecated under 1.5 and should not be used.*	
				ID_PHASE_ZSEQ = 12		*Zero sequence. This has been deprecated under 1.5 and should not be used.*	
				ID_PHASE_TOTAL = 13		*The value representing a total or other summarizing value in a multi-phase system.*	*1.5*
3 tagOtherChannelIdentifier	Vector	CHAR1	*		Optional	Arbitrary string	1.0
3 tagGroupName	Vector	CHAR1	*		Optional	This tag can be repeated if there are multiple groupings. The first one should be the highest-level group (example: a bus), and the next one should be a lower group (example: a	1.0
3 tagQuantityTypeID	Scalar	GUID			Required	The high-level description of the type of quantity which is being captured by this channel. In order to guarantee reader compatibility, the following series value types (in order) should be used	1.0
				ID_QT_WAVEFORM		*TIME, VAL / For point-on-wave measurements,*	*1.0*
				ID_QT_VALUELOG		*TIME, MIN, MAX, AVG, INST, VAL … / For time-based logged entries.*	*1.5*
				ID_QT_PHASOR		*TIME, MIN, MAX, AVG, INST, VAL, PHASEANGLE … / For time-domain measurements including magnitudes and (optionally) phase angle.*	*1.5*
				ID_QT_RESPONSE		*VAL (FREQUENCY), VAL, PHASEANGLE / For frequency-domain measurements including magnitude and (optionally) phase angle.*	*1.0*
				ID_QT_FLASH		*TIME, LAT, LON, VAL, POLARITY, ELLIPSE*	*1.0*
				ID_QT_HISTOGRAM		*BINLOW, BINHIGH, BINID, COUNT*	*1.0*
				ID_QT_HISTOGRAM3D		*XBINLOW, XBINHIGH, YBINLOW, YBINHIGH, BINID, COUNT*	*1.0*
				ID_QT_CPF		*PROB, VAL. (Note that the specific P1 value types have been deprecated.)*	*1.0*
				ID_QT_XY		*VAL, VAL*	*1.0*
				ID_QT_MAGDUR		*VAL, DUR*	*1.0*
				ID_QT_XYZ		*VAL, VAL, VAL*	*1.0*
				ID_QT_MAGDURTIME		*TIME, VAL, DUR*	*1.0*
				ID_QT_MAGDURCOUNT		*TIME, VAL, DUR, COUNT*	*1.0*
				ID_QT_VALUE		*TIME, VAL. This has been deprecated under 1.5 and should not be used. (It has been replaced by _PHASOR.)*	
				ID_QT_TREND		*TIME, MIN, MAX, AVG, INST, VAL. This has been deprecated under 1.5 and should not be used. (It has been replaced by _VALUELOG.)*	
				ID_QT_RMS		*TIME, MIN, MAX, AVG, INST, VAL. This has been deprecated under 1.5 and should not be used. (It has been replaced by _PHASOR.)*	
				ID_QT_SPECTRUM		*FREQ, VAL. This has been deprecated under 1.5 and should not be used. (Use _RESPONSE instead.)*	
3 tagQuantityMeasuredID	Scalar	UINT4			Required	Identifies the physical quantity under measurement -- Voltage, Current, Power, etc.	1.5

Element tag	Element	Physical	Count	*VALID CONTENTS*	Required?	Description	Ver
						In general, there is a one-to-one correspondence between this and the units of the series, but not always.	
				ID_QM_NONE = 0		*None or not applicable.*	*1.5*
				ID_QM_VOLTAGE = 1		*Voltage.*	*1.5*
				ID_QM_CURRENT = 2		*Current.*	*1.5*
				ID_QM_POWER = 3		*Power - includes all data for a quantity or characteristic derived from multiplying voltage and current components.*	*1.5*
				ID_QM_ENERGY = 4		*Energy - includes all data from an integration of a quantity or characteristic derived from multiplying voltage and current components together.*	*1.5*
				ID_QM_TEMPERATURE = 5		*Temperature.*	*1.5*
				ID_QM_PRESSURE = 6		*Pressure.*	*1.5*
				ID_QM_CHARGE = 7		*Charge.*	*1.5*
				ID_QM_EFIELD = 8		*Electrical field.*	*1.5*
				ID_QM_MFIELD = 9		*Magnetic field.*	*1.5*
3 tagPhysicalChannel	Scalar	UINT4			Optional	The instrument physical channel that this channel definition is associated with.	1.0
3 tagQuantityName	Vector	CHAR1	*		Optional	Additional quantity information.	1.0
3 tagPrimarySeriesIdx	Scalar	UINT4			Optional	Identifies the series which will be the primary. Index into the tagSeriesDefns collection.	1.0
3 tagSeriesDefns	Collection	(n/a)	# sers		Required	The tagSeriesDefns collection must be a collection where the count = the number of series definitions. Each entry must be another collection, each having the tagOneSeriesDefn	1.0
4 tagOneSeriesDefn	Collection	(n/a)	*		Required	One of these collections per series.	1.0
5 tagValueTypeID	Scalar	GUID			Required	This specifies the meaning of the series data.	1.0
				ID_SERIES_VALUE_TYPE_VAL		*This should be the default value type for a measurement -- a value.*	*1.0*
				ID_SERIES_VALUE_TYPE_TIME		*Time.*	*1.0*
				ID_SERIES_VALUE_TYPE_MIN		*Minimum.*	*1.0*
				ID_SERIES_VALUE_TYPE_MAX		*Maximum.*	*1.0*
				ID_SERIES_VALUE_TYPE_AVG		*Average.*	*1.0*
				ID_SERIES_VALUE_TYPE_INST		*Instantaneous. (depricated - use VAL instead)*	*1.5 DEP*
				ID_SERIES_VALUE_TYPE_PHASEANGLE		*Phase angle (used for a _VAL series or when it applies to all).*	*1.0*
				ID_SERIES_VALUE_TYPE_PHASEANGLE_MIN		*Phase angle which corresponds to a _MIN series (completing a complex pair).*	*1.5*
				ID_SERIES_VALUE_TYPE_PHASEANGLE_MAX		*Phase angle which corresponds to a _MAX series.*	*1.5*
				ID_SERIES_VALUE_TYPE_PHASEANGLE_AVG		*Phase angle which corresponds to an _AVG series.*	*1.5*
				ID_SERIES_VALUE_TYPE_LATITUDE		*Latitude.*	*1.0*
				ID_SERIES_VALUE_TYPE_DURATION		*Duration.*	*1.0*
				ID_SERIES_VALUE_TYPE_LONGITUDE		*Longitude.*	*1.0*
				ID_SERIES_VALUE_TYPE_POLARITY		*Polarity.*	*1.0*
				ID_SERIES_VALUE_TYPE_ELLIPSE		*Ellipse (for lightning flash density).*	*1.0*
				ID_SERIES_VALUE_TYPE_BINID			*1.0*
				ID_SERIES_VALUE_TYPE_BINHIGH			*1.0*
				ID_SERIES_VALUE_TYPE_BINLOW			*1.0*
				ID_SERIES_VALUE_TYPE_XBINHIGH			*1.0*
				ID_SERIES_VALUE_TYPE_XBINLOW			*1.0*
				ID_SERIES_VALUE_TYPE_YBINHIGH			*1.0*
				ID_SERIES_VALUE_TYPE_YBINLOW			*1.0*
				ID_SERIES_VALUE_TYPE_COUNT			*1.0*

Element tag	Element	Physical	Count	*VALID CONTENTS*	Required?	Description	Ver
				ID_SERIES_VALUE_TYPE_TRANSITION		*Transition event code series. This series contains codes cooresponding to values in a value series that indicates what kind of transition caused the event to be recorded. Used only with VALUELOG data.*	*1.5*
				ID_SERIES_VALUE_TYPE_P1		*Probability: 1%. This has been deprecated under 1.5 and should not be used.*	*1.5 DEP*
				ID_SERIES_VALUE_TYPE_P5		*Probability: 5%. This has been deprecated under 1.5 and should not be used.*	*1.5 DEP*
				ID_SERIES_VALUE_TYPE_P10		*Probability: 10%. This has been deprecated under 1.5 and should not be used.*	*1.5 DEP*
				ID_SERIES_VALUE_TYPE_P90		*Probability: 90%. This has been deprecated under 1.5 and should not be used.*	*1.5 DEP*
				ID_SERIES_VALUE_TYPE_P95		*Probability: 95%. This has been deprecated under 1.5 and should not be used.*	*1.5 DEP*
				ID_SERIES_VALUE_TYPE_P99		*Probability: 99%. This has been deprecated under 1.5 and should not be used.*	*1.5 DEP*
				ID_SERIES_VALUE_TYPE_FREQUENCY 1.5 DEP		*Frequency. This has been deprecated under 1.5 and should not be used. (It is now a characteristic instead of a value type.)*	
5 tagQuantityUnitsID	Scalar	UINT4			Required	This specifies the units of the data in this series. The expected physical type for the tagSeriesValues vector is REAL4 or REAL8 (except as noted).	1.0
				ID_QU_NONE = 0		*Unitless.*	*1.0*
				ID_QU_SECONDS = 2		*Seconds -- relative from the beginning time of the observation (using tagTimeStart as the beginning time).*	*1.0*
				ID_QU_TIMESTAMP = 1		*Absolute time. Each timestamp in the series must be in absolute time using the TIMESTAMPPQDIF physical type. This is generally *not* recommended, but is acceptible when _VALUELOG is used.*	*1.0*
				ID_QU_CYCLES = 3		*The timestamps are in cycles, relative to tagTimeStart.*	*1.0*
				ID_QU_VOLTS = 6		*Volts.*	*1.0*
				ID_QU_AMPS = 7		*Amperes.*	*1.0*
				ID_QU_VA = 8		*Volt-amperes.*	*1.0*
				ID_QU_WATTS = 9		*Watts.*	*1.0*
				ID_QU_VARS = 10		*Volt-amperes reactive.*	*1.0*
				ID_QU_OHMS = 11		*Ohms.*	*1.0*
				ID_QU_SIEMENS = 12		*Siemens.*	*1.0*
				ID_QU_VOLTSPERAMP = 13		*Volts per amp.*	*1.0*
				ID_QU_JOULES = 14		*Joules.*	*1.0*
				ID_QU_HERTZ = 15		*Hertz.*	*1.0*
				ID_QU_CELCIUS = 16		*Celcius.*	*1.0*
				ID_QU_DEGREES = 17		*Degrees of arc.*	*1.0*
				ID_QU_DB = 18		*Decibels.*	*1.0*
				ID_QU_PERCENT = 19		*Percent.*	*1.0*
				ID_QU_PERUNIT = 20		*Per-unit.*	*1.0*
				ID_QU_SAMPLES = 21		*Number of counts or samples*	*1.0*
				ID_QU_VARHOURS = 22		*Energy - var-hours*	*1.5*
				ID_QU_WATTHOURS = 23		*Energy - Watt-hours*	*1.5*
				ID_QU_VAHOURS = 24		*Energy - VA-hours*	*1.5*
5 tagQuantityCharacteristicID	Scalar	GUID			Required	This specifies additional detail about the meaning of the series data.	1.5
				General characteristics:			*1.5*
				ID_QC_NONE			*1.5*
				ID_QC_INSTANTANEOUS		*Instantaneous f(t)*	*1.5*
				ID_QC_SPECTRA		*Spectra F(F)*	*1.5*
				ID_QC_PEAK		*Peak value*	*1.5*
				ID_QC_RMS		*RMS value*	*1.5*

Element tag	Element	Physical	Count	*VALID CONTENTS*	Required?	Description	Ver
				ID_QC_HRMS		*Harmonic RMS*	*1.5*
				ID_QC_FREQUENCY		*Frequency*	*1.5*
				ID_QC_TOTAL_THD		*Total harmonic distortion (%)*	*1.5*
				ID_QC_EVEN_THD		*Even harmonic distortion (%)*	*1.5*
				ID_QC_ODD_THD		*Odd harmonic distortion (%)*	*1.5*
				ID_QC_CREST_FACTOR		*Crest factor*	*1.5*
				ID_QC_FORM_FACTOR		*Form factor*	*1.5*
				ID_QC_ARITH_SUM		*Arithmetic sum*	*1.5*
				ID_QC_S0S1		*Zero sequence component unbalance (%)*	*1.5*
				ID_QC_S2S1		*Negative sequence component unbalance (%)*	*1.5*
				ID_QC_SPOS		*Positive sequence component*	*1.5*
				ID_QC_AVG_IMBAL		*Imbalance by max deviation from average*	*1.5*
				Voltage characteristics:			*1.5*
				ID_QC_TIF		*TIF*	*1.5*
				ID_QC_FLKR_MAG_AVG		*Flicker average RMS value*	*1.5*
				ID_QC_FLKR_MAX_CVV		*dV/V base*	*1.5*
				ID_QC_FLKR_FREQ_MAX		*Frequence of maximum flicker harmonic*	*1.5*
				ID_QC_FLKR_MAG_MAX		*Magnitude of maximum flicker harmonic*	*1.5*
				ID_QC_FLKR_WGT_AVG		*Spectrum weighted average*	*1.5*
				ID_QC_FLKR_SPECTRUM		*Flicker spectrum VRMS(F)*	*1.5*
				Current characteristics:			*1.5*
				ID_QC_IT		*IT*	*1.5*
				ID_QC_RMS_DEMAND		*RMS value of current for a demand interval*	*1.5*
				Power characteristics:			*1.5*
				ID_QC_P		*Real power (watts)*	*1.5*
				ID_QC_Q		*Reactive power (VAR)*	*1.5*
				ID_QC_S		*Apparent power (VA)*	*1.5*
				ID_QC_PF		*True Power Factor - (Vrms * Irms) / P.*	*1.5*
				ID_QC_DF		*Displacement Factor - Cosine of the phase angle between fundamental frequency voltage and current phasors.*	*1.5*
				ID_QC_P_DEMAND		*Value of active power for a demand interval*	*1.5*
				ID_QC_Q_DEMAND		*Value of reactive power for a demand interval*	*1.5*
				ID_QC_S_DEMAND		*Value of apparent power for a demand interval*	*1.5*
				ID_QC_DF_DEMAND		*Value of displacement power factor for a demand interval*	*1.5*
				ID_QC_PF_DEMAND		*Value of true power factor for a demand interval*	*1.5*
				ID_QC_P_INTG		*Value of active power integrated over time (Energy - watt-hours)*	*1.5*
				ID_QC_P_INTG_POS		*Value of active power integrated over time (Energy - watt-hours) in the positive direction (toward load).*	*1.5*
				ID_QC_P_INTG_NEG		*Value of active power integrated over time (Energy - watt-hours) in the negative direction (away from load).*	*1.5*
				ID_QC_Q_INTG		*Value of reactive power integrated over time (var-hours)*	*1.5*
				ID_QC_Q_INTG_POS		*Value of reactive power integrated over time (Energy - watt-hours) in the positive direction (toward load).*	*1.5*
				ID_QC_Q_INTG_NEG		*Value of reactive power integrated over time (Energy - watt-hours) in the negative direction (away from load).*	*1.5*
				ID_QC_S_INTG		*Value of apparent power integrated over time (VA-hours)*	*1.5*
				ID_QC_DF_CO_S_DEMAND		*Value of displacement power factor coincident with apparent power demand*	*1.5*
				ID_QC_PF_CO_S_DEMAND		*Value of true power factor coincident with apparent power demand*	*1.5*
5 tagQuantitySignificantDigitsID	Scalar	UINT4			Optional	Defines the number of significant digits in the data represented by this series.	1.5
5 tagQuantityResolutionID	Scalar	REAL8			Optional	Contains a double indicating the scaled distance between two values of the quantity represented by this series (e.g. scaled A/D	1.5
5 tagStorageMethodID	Scalar	UINT4			Required	The legal values for this entry are masks, since they are OR-able.	1.0

Element tag	Element	Physical	Count	*VALID CONTENTS*	Required?	Description	Ver
				ID_SERIES_METHOD_VALUES = 0x01		*The data in tagSeriesValues are a straight array of data points.*	*1.0*
				ID_SERIES_METHOD_SCALED = 0x02		*All values in tagSeriesValues will be multiplied by tagSeriesScale.*	*1.0*
				ID_SERIES_METHOD_INCREMENT = 0x04		*The data in tagSeriesValues consists of a special sequence to indicate the contents of a regular rate series (see main documentation for details). The vector contains: #rates, numpts1, rate1 ... numptsN, rateN.*	*1.0*
5 tagValueTypeName	Vector	CHAR1	*		Optional	Arbitrary string	1.0
5 tagHintGreekPrefixID	Scalar	UINT4			Optional		1.0
				ID_GREEK_DONTCARE = 0			*1.0*
				ID_GREEK_FEMTO = 1			*1.0*
				ID_GREEK_PICO = 2			*1.0*
				ID_GREEK_NANO = 3			*1.0*
				ID_GREEK_MICRO = 4			*1.0*
				ID_GREEK_MILLI = 5			*1.0*
				ID_GREEK_NONE = 6			*1.0*
				ID_GREEK_KILO = 7			*1.0*
				ID_GREEK_MEGA = 8			*1.0*
				ID_GREEK_TERA = 10			*1.0*
				ID_GREEK_GIGA = 9			*1.0*
5 tagHintPreferredUnitsID	Scalar	UINT4			Optional		1.0
				ID_PREFER_ENG = 1			*1.0*
				ID_PREFER_PCT = 2			*1.0*
				ID_PREFER_PU = 3			*1.0*
5 tagHintDefaultDisplayID	Scalar	UINT4			Optional		1.0
				ID_DEFAULT_DONTCARE = 0			*1.0*
				ID_DEFAULT_MAG = 1			*1.0*
				ID_DEFAULT_ANG = 2			*1.0*
				ID_DEFAULT_REAL = 3			*1.0*
				ID_DEFAULT_IMAG = 4			*1.0*
				ID_DEFAULT_RX = 5			*1.0*
5 tagProbInterval	Scalar	REAL8			Optional	For a probability series definition, this specifies its time interval (in seconds; >0).	1.5
5 tagProbPercentile	Scalar	REAL8			Optional	For a probability series definition, this specifies its probability percentile (in percent; 0-100).	1.5
5 tagSeriesNominalQuantity	Scalar	REAL8			Optional	Contains the default nominal base voltage or any or any other necessary normalizing quantity. Display programs may use this value or the tagSeriesBaseQuantity in the series instance for displaying data in percent or per	1.5
1 tagEffective	Scalar	TIMESTAMP			Required	Time that this data source record became effective	1.5

Element tag	Element	Physical	Count	*VALID CONTENTS*	Required?	Description	Ver
tagRecMonitorSettings	Collection		*		Optional	Record-level tag which identifies a set of configuration parameters.	
1 tagEffective	Scalar	TIMESTAMP			Required	The time that these settings become effective.	1.0
1 tagTimeInstalled	Scalar	TIMESTAMP			Required		1.0
1 tagTimeRemoved	Scalar	TIMESTAMP			Optional		1.0
1 tagUseCalibration	Scalar	BOOL4			Required	If TRUE, the calibration adjustments *must* be applied to the series data before using. Otherwise the data is for informative use only.	1.0
1 tagUseTransducer	Scalar	BOOL4			Required	If TRUE, the transducer adjustments *must* be applied to the series data before using. Otherwise the data is for informative use only.	1.0
1 tagChannelSettingsArray	Collection	(n/a)	# chan		Required	Channel specific monitor settings stuff	1.0
2 tagOneChannelSetting	Collection	(n/a)	*		Required	One of these collections per channel.	1.0
3 tagChannelDefnIdx	Scalar	UINT4			Required	The channel definition which these settings apply to. Index into tagChannelDefns collection of the matching data source record.	1.0
3 tagTriggerTypeID	Scalar	UINT4			Optional	Integer ID representing which trigger fields are used.	1.0
				ID_TRIG_NONE = 0x00			*1.0*
				ID_TRIG_LOW = 0x01			*1.0*
				ID_TRIG_HIGH = 0x02			*1.0*
				ID_TRIG_RATE = 0x04			*1.0*
				ID_TRIG_SHAPE = 0x08			*1.0*
				ID_TRIG_OTHER = 0x10			*1.0*
3 tagFullScale	Scalar	REAL8			Optional	Full scale range for this instrument channel	1.0
3 tagNoiseFloor	Scalar	REAL8			Optional	Noise floor for this instrument channel	1.0
3 tagTriggerLow	Scalar	REAL8			Optional	Low trigger for this channel	1.0
3 tagTriggerHigh	Scalar	REAL8			Optional	High trigger for this channel	1.0
3 tagTriggerRate	Scalar	REAL8			Optional	Rate of change trigger for this channel	1.0
3 tagTriggerShapeParam	Vector	REAL8	3		Optional	Parameters for shape based triggering algorithms for this channel	1.0
3 tagXDTransformerTypeID	Scalar	UINT4			Optional	PT or CT	1.0
				ID_XFORMER_TYPE_CT = 2			*1.0*
				ID_XFORMER_TYPE_PT = 1			*1.0*
3 tagXDSystemSideRatio	Scalar	REAL8			Optional	System side part of ratio	1.0
3 tagXDMonitorSideRatio	Scalar	REAL8			Optional	Monitor side part of ratio	1.0
3 tagXDFrequencyResponse	Vector	REAL8	# freq		Optional	Transducer frequency response	1.0
3 tagCalTimeSkew	Scalar	REAL8			Optional	Chanel time skew	1.0
3 tagCalOffset	Scalar	REAL8			Optional	Channel DC offset error	1.0
3 tagCalRatio	Scalar	REAL8			Optional	Channel ratio error	1.0
3 tagCalMustUseARCal	Scalar	BOOL4			Optional	Flag indicating that the applied/recordedt calibration arrays must be used to correct data.	1.0
3 tagCalApplied	Vector	REAL8	# cal		Optional	Array of applied signals for this channel	1.0
3 tagCalRecorded	Vector	REAL8	# cal		Optional	Array of recorded actual values for the applied signal	1.0
1 tagNominalFrequency	Scalar	REAL8			Optional	Nominal power system frequency for this instrument in Hertz	1.5

Element tag	Element	Physical	Count	*VALID CONTENTS*	Required?	Description	Ver
tagRecObservation	Collection		*		Required	Record-level tag which identifies an observation -- an event, measurement, etc.	
1 tagObservationName	Vector	CHAR1	*		Required	Name of the observation	1.0
1 tagTimeCreate	Scalar	TIMESTAMP			Required	Time this observation was created	1.0
1 tagTimeStart	Scalar	TIMESTAMP			Required	The start time of the observation -- the zero point where the .	1.0
1 tagTriggerMethodID	Scalar	UINT4			Required	Type of trigger which caused the observation.	1.0
				ID_TRIGGER_METH_NONE = 0			*1.0*
				ID_TRIGGER_METH_CHANNEL = 1		*A specific channel (or channels) caused the trigger; should be used with tagChannelTriggerIdx to specify which channels.*	*1.0*
				ID_TRIGGER_METH_PERIODIC = 2			*1.0*
				ID_TRIGGER_METH_EXTERNAL = 3			*1.0*
1 tagTimeTriggered	Scalar	TIMESTAMP			Optional	Tme this observation was triggered if	1.0
1 tagChannelTriggerIdx	Vector	UINT4			Optional	Index into tagChannelInstances collection within this record. This specifies which channel(s) initiated the observation.	1.0
1 tagObservationSerial	Scalar	UINT4			Optional	The serial number of the observation (if generated by an instrument, for example).	1.0
1 tagObservationAggregationSerial	Scalar	UINT4			Optional	Serial number -- of specific cycle, for example -- that can be used to correlate observations.	1.0
1 tagDisturbanceCategoryID	Scalar	GUID			Optional	Currently uses the IEEE 1159 disturbance categories, but others could be used as well.	1.0
				ID_DISTURB_1159_NONE		*No IEEE 1159 definition applicable or desired*	*1.0*
				ID_DISTURB_1159_TRANSIENT		*IEEE 1159 Transient*	*1.0*
				ID_DISTURB_1159_TRANSIENT_IMPULSIVE		*IEEE 1159 Impulsive Transient*	*1.5*
				ID_DISTURB_1159_TRANSIENT_IMPULSIVE_NANO		*IEEE 1159 Impulsive Transient - nanosecond duration*	*1.5*
				ID_DISTURB_1159_TRANSIENT_IMPULSIVE_MIC RO		*1159 Impulsive Transient - microsecond duration*	*IEEE 1.5*
				ID_DISTURB_1159_TRANSIENT_IMPULSIVE_MIL LI		*1159 Impulsive Transient - millisecond duration*	*IEEE 1.5*
				ID_DISTURB_1159_TRANSIENT_OSCILLATORY		*IEEE 1159 Oscillatory Transient*	*1.5*
				ID_DISTURB_1159_TRANSIENT_OSCILLATORY_ LOWFREQ		*1159 Oscillatory Transient - Low Frequency*	*IEEE 1.5*
				ID_DISTURB_1159_TRANSIENT_OSCILLATORY_ MEDFREQ		*1159 Oscillatory Transient - Medium Frequency*	*IEEE 1.5*
				ID_DISTURB_1159_TRANSIENT_OSCILLATORY_ HIGHFREQ		*1159 Oscillatory Transient - High Frequency*	*IEEE 1.5*
				ID_DISTURB_1159_SHORTDUR		*IEEE 1159 Short Duration RMS Variation*	*1.0*
				ID_DISTURB_1159_SHORTDUR_INSTANT		*IEEE 1159 Short Duration RMS Variation - Instantaneous duration*	*1.5*
				ID_DISTURB_1159_SHORTDUR_INSTANT_SAG		*IEEE 1159 Short Duration RMS Variation - Instantaneous Sag*	*1.5*
				ID_DISTURB_1159_SHORTDUR_INSTANT_SWELL		*IEEE 1159 Short Duration RMS Variation - Instantaneous Swell*	*1.5*
				ID_DISTURB_1159_SHORTDUR_MOMENT		*IEEE 1159 Short Duration RMS Variation - Momentary Duration*	*1.5*
				ID_DISTURB_1159_SHORTDUR_MOMENT_INTERRUPT		*IEEE 1159 Short Duration RMS Variation - Momentary Interruption*	*1.5*
				ID_DISTURB_1159_SHORTDUR_MOMENT_SAG		*IEEE 1159 Short Duration RMS Variation - Momentary Sag*	*1.5*
				ID_DISTURB_1159_SHORTDUR_MOMENT_SWELL		*IEEE 1159 Short Duration RMS Variation - Momentary Swell*	*1.5*
				ID_DISTURB_1159_SHORTDUR_TEMP		*IEEE 1159 Short Duration RMS Variation - Temporary Duration*	*1.5*
				ID_DISTURB_1159_SHORTDUR_TEMP_INTERRUPT		*IEEE 1159 Short Duration RMS Variation - Temporary Interruption*	*1.5*

ID_DISTURB_1159_SHORTDUR_TEMP_SAG	*IEEE 1159 Short Duration RMS Variation - Temporary Sag*	*1.5*
ID_DISTURB_1159_SHORTDUR_TEMP_SWELL	*IEEE 1159 Short Duration RMS Variation - Temporary Swell*	*1.5*

Element tag	Element	Physical	Count	*VALID CONTENTS*	Required?	Description	Ver
				ID_DISTURB_1159_LONGDUR		*IEEE 1159 Long Duration RMS Variation*	*1.0*
				ID_DISTURB_1159_LONGDUR_INTERRUPT		*IEEE 1159 Long Duration RMS Variation - Interruption*	*1.5*
				ID_DISTURB_1159_LONGDUR_SAG		*IEEE 1159 Long Duration RMS Variation - Undervoltage*	*1.5*
				ID_DISTURB_1159_LONGDUR_SWELL		*IEEE 1159 Long Duration RMS Variation - Overvoltage*	*1.5*
				ID_DISTURB_1159_IMBALANCE		*IEEE 1159 Imbalance*	*1.5*
				ID_DISTURB_1159_POWERFREQVARIATION		*IEEE 1159 Power Frequency Variation*	*1.5*
				ID_DISTURB_1159_FLICKER		*IEEE 1159 Light Flicker due to Voltage Fluctuations*	
				ID_DISTURB_1159_VOLTAGEFLUCTUATION		*IEEE 1159 Voltage Fluctuation (causes light flicker)*	*1.5*
				ID_DISTURB_1159_WAVEDISTORT		*IEEE 1159 Waveform Distortion*	*1.5*
				ID_DISTURB_1159_HARMONIC		*IEEE 1159 Waveform Distortion*	
				ID_DISTURB_1159_WAVEDISTORT_DCOFFSET		*DC offset of voltage or current waveform*	*1.5*
				ID_DISTURB_1159_WAVEDISTORT_HARMONIC		*IEEE 1159 Waveform Harmonics Present*	*1.5*
				ID_DISTURB_1159_WAVEDISTORT_INTERHARMONIC		*IEEE 1159 Waveform Interharmonics Present*	*1.5*
				ID_DISTURB_1159_NOTCH		*IEEE 1159 Notching - waveforms exhibiting power electronic commutation notches*	
				ID_DISTURB_1159_WAVEDISTORT_NOTCHING		*IEEE 1159 Waveform Notching Present*	*1.5*
				ID_DISTURB_1159_NOISE		*IEEE 1159 Noise - broadband noise signals*	
				ID_DISTURB_1159_WAVEDISTORT_NOISE		*IEEE 1159 Waveform Noise Present*	*1.5*
1 tagChannelInstances	Collection	(n/a)	# chan		Required	This collection contains a set of channel instances. It is not required to contain the same number of channels as there are channel instances. This can be determined on an observation-by-observation basis.	1.0
2 tagOneChannelInst	Collection	(n/a)	*		Required	One of these collections per channel instance.	1.0
3 tagChannelDefnIdx	Scalar	UINT4			Required	Specifies which of the available channel definitions this instance belongs to. Index into tagChannelDefns collection of the matching data source record.	1.0
3 tagSeriesInstances	Collection	(n/a)	# sers		Required	This collection must contain the exact number of series which were defined for the specified channel definition.	1.0
4 tagOneSeriesInstance	Collection	(n/a)	*		Required	One of these collections per series instance.	1.0
5 tagSeriesBaseQuantity	Scalar	REAL8			Optional	Contains the nominal base voltage, or any other necessary normalizing quantity.	1.0
5 tagSeriesScale	Scalar	(any type)			Optional	If not present, assumed to be 1. The physical type should match that of tagSeriesValues.	1.0
5 tagSeriesOffset	Scalar	(any type)			Optional	If not present, assumed to be 0. Generally used as a starting point when the ID_SERIES_METHOD_INCREMENT storage method is used. The physical type should match that of tagSeriesValues.	1.0
5 tagSeriesShareChannelIdx	Scalar	UINT4			Optional	Identifies the channel which owns the series to be shared. An index into the tagChannelInstances collection.	1.0
5 tagSeriesShareSeriesIdx	Scalar	UINT4			Optional	Identifies the series to be shared. An index into the tagSeriesInstances collection. The tagSeriesValues vector from this series is used. This must be present if tagSeriesShareChannelIdx is used.	1.0
5 tagSeriesValues	Vector	(any type)	*		Required	Contains the actual data points of the series. Required unless the data series is shared, in	1.0

Element tag	Element	Physical	Count	*VALID CONTENTS*	Required?	Description	Ver
						which case both tagSeriesShareChannelIdx and tagSeriesShareSeriesIdx should be present.	
				For storage method _INCREMENT		*The vector holds numbers which specify "instructions" on how to reconstruct the *real* array from points at constant rates. Vector [0] contains the number of rates. Vector [1] contains the number of points for the first rate. Vector [2] contains the first rate. Vector [3] (if applicable) contains the number of points for the second rate, etc.*	*1.5*
				For storage method _VALUES		*The vector holds a straight array of data points which correspond (after scaling, etc.) to the actual samples, time series, etc.*	*1.5*
3 tagCharactMagnitude	Scalar	REAL8			Optional	Simple characterization value: magnitude of disturbance (percent: 100%=nominal)	1.5
3 tagCharactFrequency	Scalar	REAL8			Optional	Simple characterization value: frequency	1.5
3 tagChanTriggerModuleInfo	Scalar	UINT4			Optional	Contains a 32 bit integer that represents module spefic information related to the trigger	1.5
3 tagChanTriggerModuleName	Vector	CHAR1	*		Optional	Contains the name of a device specific code or hardware module, algorithm or rule not necessarily channel based that caused this channel to be recorded	1.5
3 tagCrossTriggerDeviceName	Vector	CHAR1	*		Optional	Contains the name of the device involved in an external cross trigger scenario.	1.5
3 tagCrossTriggerChanIdx	Scalar	UINT4			Optional	Contains the channel definition index of the channel that triggered in a cross trigger	1.5
3 tagChanTriggerTypeID	Scalar	UINT4			Optional	Integer ID representing the trigger type for this channel instance. Used only with type ID_QT_VALUELOG wit a trigger method of channel.	1.5
				ID_CTT_NONE = 0		*No transition - should not happen*	*1.5*
				ID_CTT_NORMAL_TO_LO = 1		*Normal to low transition*	*1.5*
				ID_CTT_NORMAL_TO_LO_LO = 2		*Normal to low low transition*	*1.5*
				ID_CTT_NORMAL_TO_HI = 3		*Normal to High transition*	*1.5*
				ID_CTT_NORMAL_TO_HI_HI = 4		*Normal to High High transition*	*1.5*
				ID_CTT_LO_LO_TO_LO = 5		*Low Low to Lo transition*	*1.5*
				ID_CTT_LO_LO_TO_NORMAL = 6		*Low Low to Normal transition*	*1.5*
				ID_CTT_LO_LO_TO_HI = 7		*Low Low to High transition*	*1.5*
				ID_CTT_LO_LO_TO_HI_HI = 8		*Low Low to High High transition*	*1.5*
				ID_CTT_LO_TO_LO_LO = 9		*Low to Low Low transition*	*1.5*
				ID_CTT_LO_TO_NORMAL = 10		*Low to Normal transition*	*1.5*
				ID_CTT_LO_TO_HI = 11		*Low to High transition*	*1.5*
				ID_CTT_LO_TO_HI_HI = 12		*Low to High High transition*	*1.5*
				ID_CTT_HI_TO_LO_LO = 13		*High to Low Low transition*	*1.5*
				ID_CTT_HI_TO_LO = 14		*High to Low transition*	*1.5*
				ID_CTT_HI_TO_NORMAL = 15		*High to Normal transition*	*1.5*
				ID_CTT_HI_TO_HI_HI = 16		*High to High High transition*	*1.5*
				ID_CTT_HI_HI_TO_LO_LO = 17		*High High to Low Low transition*	*1.5*
				ID_CTT_HI_HI_TO_LO = 18		*High High to Low transition*	*1.5*
				ID_CTT_HI_HI_TO_NORMAL = 19		*High High to Normal transition*	*1.5*
				ID_CTT_HI_HI_TO_HI = 20		*High High to High transition*	*1.5*
				ID_CTT_DB_LO = 21		*Deadband transition lower*	*1.5*
				ID_CTT_DB_HI = 22		*Deadband transition higher*	*1.5*
				ID_CTT_PERIODIC = 23		*Hardware initiated trigger based on periodic trigger rule*	*1.5*
				ID_CTT_MANUAL = 24		*User commanded sample - button was pushed*	*1.5*

Element tag	Element	Physical	Count	*VALID CONTENTS*	Required?	Description	Ver
				ID_CTT_INT_CROSS_TRIG = 25		*Channel triggered because of internal cross-trigger rule. tagCrossTriggerChanIdx is index of channel that triggered.*	*1.5*
				ID_CTT_EXT_CROSS_TRIG = 26		*Channel triggered because of external cross-trigger rule. tagCrossTriggerChanIdx is index of channel that triggered on external device. tagCrossTriggerDeviceName is the name of the external device that initiated the cross trigger.*	*1.5*
				ID_CTT_MODULE = 27		*Channel triggered because of hardware or software module, rule or algorithm*	*1.5*
				ID_CTT_RATE = 28		*Rate of change threshold exceeded (dV/dt or dI/dt)*	*1.5*
3 tagChannelFrequency	Scalar	REAL8			Optional	For a channel which contains multiple instances to represent a sparse log of time-stamped frequency-domain information, this specifies the frequency for which this channel instance applies (in Hertz). If not present, the channel characteristics are frequency independent unless further	1.5
2 tagCharactDuration	Scalar	REAL8			Optional	Simple characterization value: duration of disturbance (seconds)	1.5
1 tagCharactDisturbDirection	Scalar	UINT4			Optional	Direction of disturbance represented by the data in this observation. Value of 0 means don't know, 1 means originated from load side, 2 means originated from source side.	1.5
1 tagCharactDisturbDirectionQuality	Scalar	UINT4			Optional	Quality of the direction result given in another tag. Range is from 0 (no confidence), to 100 (darn sure).	1.5

The following text lists all of the logical tags and ID's:

```
/*

**  PQDIF - Power Quality Data Interchange Format

**  Version 1.5

**
**  File name:          $Workfile: pqdif_lg.h $
**  Last modified:      $Modtime: 4/26/99 5:02p $
**  Last modified by:   $Author: Erich $
**
**  VCS archive path:   $Archive: /PQDIF/Document/Version15/Working/pqdif_lg.h $
**  VCS revision:       $Revision: 42 $
**
**  LOGICAL FORMAT DEFINITIONS
**  ==========================================================================
**  This file contains the complete specifications for the logical
**  format of a PQDIF file. It is based on the _physical_ structure of
**  the file, which is defined in PQDIF_PH.H
**
**  ==========================================================================
**  The current version of this file and related information
**  can be found at URL:
**
**  http://www.pqnet.electrotek.com/pqnet/main/measure/pqdif/pqdif.htm
**
**  This site can be freely accessed after registration at root URL
**  (http://www.pqnet.e.com)
**  ==========================================================================
**
**  LOGICAL HIERARCHY OF RECORDS
**  ==========================================================================
**  The records that make up PQDIF are currently of four different kinds:
**  Container, Data Source, Monitor Settings, and Observation.
**
**  There are absolute links from one record to another (these are
**  different from links within a record, which are relative within the
**  record). When these links are followed, the records form a logical
**  hierarchy:
**
**    +-----------+
**    | Container |
**    +-----------+
**        |
**        |   +---------------+
**        +---| Data Source 1 |
**        |   +---------------+
**        |       |
```


```
**  The first record in a PQDIF file must be of the Container type.
**  This data in this record describes attributes of the items
**  contained within the PQDIF file.
**
**  The container record is then followed by a Data Source
**  record which can be followed by one or more Data Source
**  records or Observation records.  A Data Source record
**  describes the source of the data that is contained in the
**  Observation records that follow it.
**
**  Note that Monitor Settings records are optional; in their absence,
**  Observation records fall directly under the appropriate
**  Data Source record.
*/
#ifndef PQDIF_LG_H
#define PQDIF_LG_H
```

```
/*
**  RECORD HEADER
**  ============================================================================
**  The first item in a PQDIF File (and in each record) is a 128 bit
**  GUID which serves as a unique signature for both the file as a whole
**  and each record.
**
**  Every record must have this GUID.
*/
const GUID guidRecordSignaturePQDIF = { /* 4a111440-e49f-11cf-9900-505144494600 */
    0x4a111440,
    0xe49f,
    0x11cf,
    {0x99, 0x00, 0x50, 0x51, 0x44, 0x49, 0x46, 0x00}
  };


/*
**  TAG FOR ANY RECORD
**  ============================================================================
**  The following tag can be used to leave space in a collection. To leave
**  a collection item blank, specify it as a scalar of less than 8 bytes
**  (such as a UINT4), and specify it as embedded.
*/
const GUID tagBlank =  //
    { 0x89738618, 0xf1c3, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };
    // {89738618-F1C3-11cf-9D89-0080C72E70A3}


//  ============================================================================
//  Do not modify anything after the following comment:
//  {{{{ AUTO-GENERATED CONSTANTS }}}}

//  Description:   Record-level tag which identifies the container record (always  the first one in the file, and there must be only
one per file).
//  Element type:  Collection [ * ]
//  Physical type: (n/a)
//  Required/opt:  Required
//  Version:       1.0
const GUID tagContainer = { 0x89738606, 0xf1c3, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Record-level tag which identifies a data source (an instrument, etc.).
//  Element type:  Collection [ * ]
//  Physical type: (n/a)
//  Required/opt:  Required
//  Version:       1.0
const GUID tagRecDataSource = { 0x89738619, 0xf1c3, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Record-level tag which identifies a set of configuration parameters.
//  Element type:  Collection [ * ]
//  Physical type: (n/a)
```

```
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagRecMonitorSettings = { 0xb48d858c, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Record-level tag which identifies an observation -- an event, measurement, etc.
//  Element type:  Collection [ * ]
//  Physical type: (n/a)
//  Required/opt:  Required
//  Version:       1.0
const GUID tagRecObservation = { 0x8973861a, 0xf1c3, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Specifies the format version for read/write compatibility. The four required numbers in the vector are described
below.
//  Element type:  Vector [ 4 ]
//  Physical type: UINT4
//  Required/opt:  Required
//  Version:       1.0
const GUID tagVersionInfo = { 0x89738607, 0xf1c3, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Original name of the file.
//  Element type:  Vector [ * ]
//  Physical type: CHAR1
//  Required/opt:  Required
//  Version:       1.0
const GUID tagFileName = { 0x89738608, 0xf1c3, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Date/time when the file was created.
//  Element type:  Scalar
//  Physical type: TIMESTAMP
//  Required/opt:  Required
//  Version:       1.0
const GUID tagCreation = { 0x89738609, 0xf1c3, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Date/time when the file was last saved.
//  Element type:  Scalar
//  Physical type: TIMESTAMP
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagLastSaved = { 0x8973860a, 0xf1c3, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   The number of times the file has been saved/modified.
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagTimesSaved = { 0x8973860b, 0xf1c3, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   The language (English, etc.) of the file.
//  Element type:  Vector [ * ]
//  Physical type: CHAR1
//  Required/opt:  Optional
```

```
//  Version:       1.0
const GUID tagLanguage = { 0x8973860c, 0xf1c3, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Arbitrary title.
//  Element type:  Vector [ * ]
//  Physical type: CHAR1
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagTitle = { 0x8973860d, 0xf1c3, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Arbitrary subject string
//  Element type:  Vector [ * ]
//  Physical type: CHAR1
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagSubject = { 0x8973860e, 0xf1c3, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Individual/company who caused the file to be written
//  Element type:  Vector [ * ]
//  Physical type: CHAR1
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagAuthor = { 0x8973860f, 0xf1c3, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Keywords for assisting searches
//  Element type:  Vector [ * ]
//  Physical type: CHAR1
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagKeywords = { 0x89738610, 0xf1c3, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Arbitrary comments
//  Element type:  Vector [ * ]
//  Physical type: CHAR1
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagComments = { 0x89738611, 0xf1c3, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Individual/company who last wrote to file
//  Element type:  Vector [ * ]
//  Physical type: CHAR1
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagLastSavedBy = { 0x89738612, 0xf1c3, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Creating application
//  Element type:  Vector [ * ]
//  Physical type: CHAR1
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagApplication = { 0x89738623, 0xf1c3, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };
```

```
//  Description:   Security descriptor information
//  Element type:  Vector [ * ]
//  Physical type: CHAR1
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagSecurity = { 0x89738613, 0xf1c3, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Owner of file contents (This and some of the following fields are for copyright and trademark information)
//  Element type:  Vector [ * ]
//  Physical type: CHAR1
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagOwner = { 0x89738614, 0xf1c3, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Copyright notice
//  Element type:  Vector [ * ]
//  Physical type: CHAR1
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagCopyright = { 0x89738615, 0xf1c3, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Trademark notice
//  Element type:  Vector [ * ]
//  Physical type: CHAR1
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagTrademarks = { 0x89738616, 0xf1c3, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   RTF formatted notes associated with this file (This corresponds to the IEEE COMTRADE .HDR file, for example).
//  Element type:  Vector [ * ]
//  Physical type: CHAR1
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagNotes = { 0x89738617, 0xf1c3, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Specified how the compression is applied to the file.
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagCompressionStyleID = { 0x8973861b, 0xf1c3, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Required if tagCompressionStyleID specifies that compression is turned on.
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagCompressionAlgorithmID = { 0x8973861c, 0xf1c3, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };
```

```
//  Description:   If compression style is _TOTALFILE, this is the checksum for the entire file. This feature has been deprecated
under 1.5.
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Optional
//  Version:       1.5 Deprecated
const GUID tagCompressionChecksum = { 0x8973861d, 0xf1c3, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   General contact information (all optional)
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagName = { 0xb48d85a2, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   General contact information (all optional)
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagAddress1 = { 0xb48d85a3, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   General contact information (all optional)
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagAddress2 = { 0xb48d85a4, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   General contact information (all optional)
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagCity = { 0xb48d85a5, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   General contact information (all optional)
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagState = { 0xb48d85a6, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   General contact information (all optional)
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagPostalCode = { 0xb48d85a7, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   General contact information (all optional)
```

```
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagCountry = { 0xb48d85a8, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   General contact information (all optional)
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagPhoneVoice = { 0xb48d85a9, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   General contact information (all optional)
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagPhoneFAX = { 0x3d786f80, 0xf76e, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   General contact information (all optional)
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagEMail = { 0x3d786f81, 0xf76e, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Since this ID is a GUID, you can generate a custom ID if a standard ID is not defined.
//  Element type:  Scalar
//  Physical type: GUID
//  Required/opt:  Required
//  Version:       1.0
const GUID tagDataSourceTypeID = { 0xb48d8581, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Since this ID is a GUID, you can generate a custom ID if a standard ID is not defined.
//  Element type:  Scalar
//  Physical type: GUID
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagVendorID = { 0xb48d8582, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Since this ID is a GUID, you can generate a custom ID if a standard ID is not defined.
//  Element type:  Scalar
//  Physical type: GUID
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagEquipmentID = { 0xb48d8583, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   This collection can include the standard name, address & telephone number tags -- these apply to the vendor -- as
well as tags about the instrument itself.
//  Element type:  Collection
```

```
//  Physical type: (n/a)
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagCustomSourceInfo = { 0xb48d8584, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Arbitrary string to hold data souce (instrument) serial number
//  Element type:  Vector [ * ]
//  Physical type: CHAR1
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagSerialNumberDS = { 0xb48d8585, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Arbitrary string to hold data souce (instrument) version number (if applicable)
//  Element type:  Vector [ * ]
//  Physical type: CHAR1
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagVersionDS = { 0xb48d8586, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Arbitrary string to hold the name of the data souce (instrument)
//  Element type:  Vector [ * ]
//  Physical type: CHAR1
//  Required/opt:  Required
//  Version:       1.0
const GUID tagNameDS = { 0xb48d8587, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Arbitrary string to hold data souce (instrument) owner name
//  Element type:  Vector [ * ]
//  Physical type: CHAR1
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagOwnerDS = { 0xb48d8588, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Arbitrary string to hold data souce (instrument) location information
//  Element type:  Vector [ * ]
//  Physical type: CHAR1
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagLocationDS = { 0xb48d8589, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   EST, CST, etc.
//  Element type:  Vector [ * ]
//  Physical type: CHAR1
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagTimeZoneDS = { 0xb48d858a, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Longitude/latitude
//  Element type:  Vector [ 2 ]
//  Physical type: UINT4
//  Required/opt:  Optional
```

```
//  Version:       1.0
const GUID tagCoordinatesDS = { 0xb48d858b, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   The tagChannelDefns collection must be a collection where the count = the number of channel definitions. Each entry
must be another collection, each having the tagOneChannelDefn tag.
//  Element type:  Collection [ # defs ]
//  Physical type: (n/a)
//  Required/opt:  Required
//  Version:       1.0
const GUID tagChannelDefns = { 0xb48d858d, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Element type:  Scalar
//  Physical type: GUID
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagInstrumentTypeID = { 0x3d786f82, 0xf76e, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Arbitrary string
//  Element type:  Vector [ * ]
//  Physical type: CHAR1
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagInstrumentModelName = { 0x3d786f83, 0xf76e, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Arbitrary string
//  Element type:  Vector [ * ]
//  Physical type: CHAR1
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagInstrumentModelNumber = { 0x3d786f84, 0xf76e, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Must have one or more channel definitions
//  Element type:  Collection [ * ]
//  Physical type: (n/a)
//  Required/opt:  Required
//  Version:       1.0
const GUID tagOneChannelDefn = { 0xb48d858e, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Arbitrary string
//  Element type:  Vector [ * ]
//  Physical type: CHAR1
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagChannelName = { 0xb48d8590, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Phase identifier
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Required
//  Version:       1.0
const GUID tagPhaseID = { 0xb48d8591, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };
```

```
//  Description:   Arbitrary string
//  Element type:  Vector [ * ]
//  Physical type: CHAR1
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagOtherChannelIdentifier = { 0xb48d8593, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   This tag can be repeated if there are multiple groupings. The first one should be the highest-level group (example:
a bus), and the next one should be a lower group (example: a feeder).
//  Element type:  Vector [ * ]
//  Physical type: CHAR1
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagGroupName = { 0xb48d8594, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   The high-level description of the type of quantity which is being captured by this channel. In order to guarantee
reader compatibility, the following series value types (in order) should be used (ID_SERIES_VALUE_TYPE_VAL, etc.).
//  Element type:  Scalar
//  Physical type: GUID
//  Required/opt:  Required
//  Version:       1.0
const GUID tagQuantityTypeID = { 0xb48d8592, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Identifies the physical quantity under measurement -- Voltage, Current, Power, etc. In general, there is a one-to-
one correspondence between this and the units of the series, but not always.
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Required
//  Version:       1.5
const GUID tagQuantityMeasuredID = { 0xc690e872, 0xf755, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   The instrument physical channel that this channel definition is associated with.
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagPhysicalChannel = { 0x89738622, 0xf1c3, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Additional quantity information.
//  Element type:  Vector [ * ]
//  Physical type: CHAR1
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagQuantityName = { 0xb48d8595, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Identifies the series which will be the primary. Index into the tagSeriesDefns collection.
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Optional
//  Version:       1.0
```

```
const GUID tagPrimarySeriesIdx = { 0xb48d8596, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   The tagSeriesDefns collection must be a collection where the count = the number of series definitions. Each entry
must be another collection, each having the tagOneSeriesDefn tag.
//  Element type:  Collection [ # sers ]
//  Physical type: (n/a)
//  Required/opt:  Required
//  Version:       1.0
const GUID tagSeriesDefns = { 0xb48d8598, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   One of these collections per series.
//  Element type:  Collection [ * ]
//  Physical type: (n/a)
//  Required/opt:  Required
//  Version:       1.0
const GUID tagOneSeriesDefn = { 0xb48d859a, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   This specifies the meaning of the series data.
//  Element type:  Scalar
//  Physical type: GUID
//  Required/opt:  Required
//  Version:       1.0
const GUID tagValueTypeID = { 0xb48d859c, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   This specifies the units of the data in this series. The expected physical type for the tagSeriesValues vector is
REAL4 or REAL8 (except as noted).
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Required
//  Version:       1.0
const GUID tagQuantityUnitsID = { 0xb48d859b, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   This specifies additional detail about the meaning of the series data.
//  Element type:  Scalar
//  Physical type: GUID
//  Required/opt:  Required
//  Version:       1.5
const GUID tagQuantityCharacteristicID = { 0x3d786f9e, 0xf76e, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Defines the number of significant digits in the data represented by this series.
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Optional
//  Version:       1.5
const GUID tagQuantitySignificantDigitsID = { 0xa112f421, 0xb111, 0x11d2, { 0x9b, 0x37, 0x0, 0x40, 0x5, 0x2c, 0x2d, 0x28 } };

//  Description:   The legal values for this entry are masks, since they are OR-able.
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Required
//  Version:       1.0
```

```
const GUID tagStorageMethodID = { 0xb48d85a1, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Arbitrary string
//  Element type:  Vector [ * ]
//  Physical type: CHAR1
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagValueTypeName = { 0xb48d859d, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagHintGreekPrefixID = { 0xb48d859e, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagHintPreferredUnitsID = { 0xb48d859f, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagHintDefaultDisplayID = { 0xb48d85a0, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   For a probability series definition, this specifies its time interval (in seconds; >0).
//  Element type:  Scalar
//  Physical type: REAL8
//  Required/opt:  Optional
//  Version:       1.5
const GUID tagProbInterval = { 0x2747d441, 0x2bd0, 0x11d2, { 0xae, 0x42, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   For a probability series definition, this specifies its probability percentile (in percent; 0-100).
//  Element type:  Scalar
//  Physical type: REAL8
//  Required/opt:  Optional
//  Version:       1.5
const GUID tagProbPercentile = { 0x2747d440, 0x2bd0, 0x11d2, { 0xae, 0x42, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   Contains the default nominal base voltage or any or any other necessary normalizing quantity.  Display programs may
use this value or the tagSeriesBaseQuantity in the series instance for displaying data in percent or per unit.
//  Element type:  Scalar
//  Physical type: REAL8
//  Required/opt:  Optional
//  Version:       1.5
const GUID tagSeriesNominalQuantity = { 0xfa118c8, 0xcb4a, 0x11d2, { 0xb3, 0xb, 0xfe, 0x25, 0xcb, 0x9a, 0x17, 0x60 } };

//  Description:   The time that these settings become effective.
//  Element type:  Scalar
```

```
//  Physical type: TIMESTAMP
//  Required/opt:  Required
//  Version:       1.0
const GUID tagEffective = { 0x62f28183, 0xf9c4, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Element type:  Scalar
//  Physical type: TIMESTAMP
//  Required/opt:  Required
//  Version:       1.0
const GUID tagTimeInstalled = { 0x3d786f85, 0xf76e, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Element type:  Scalar
//  Physical type: TIMESTAMP
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagTimeRemoved = { 0x3d786f86, 0xf76e, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   If TRUE, the calibration adjustments *must* be applied to the series data before using. Otherwise the data is for
informative use only.
//  Element type:  Scalar
//  Physical type: BOOL4
//  Required/opt:  Required
//  Version:       1.0
const GUID tagUseCalibration = { 0x62f28180, 0xf9c4, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   If TRUE, the transducer adjustments *must* be applied to the series data before using. Otherwise the data is for
informative use only.
//  Element type:  Scalar
//  Physical type: BOOL4
//  Required/opt:  Required
//  Version:       1.0
const GUID tagUseTransducer = { 0x62f28181, 0xf9c4, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Channel specific monitor settings stuff
//  Element type:  Collection [ # chan ]
//  Physical type: (n/a)
//  Required/opt:  Required
//  Version:       1.0
const GUID tagChannelSettingsArray = { 0x62f28182, 0xf9c4, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Nominal power system frequency for this instrument in Hertz
//  Element type:  Scalar
//  Physical type: REAL8
//  Required/opt:  Optional
//  Version:       1.5
const GUID tagNominalFrequency = { 0xfa118c3, 0xcb4a, 0x11d2, { 0xb3, 0xb, 0xfe, 0x25, 0xcb, 0x9a, 0x17, 0x60 } };

//  Description:   One of these collections per channel.
//  Element type:  Collection [ * ]
//  Physical type: (n/a)
//  Required/opt:  Required
```

```
//  Version:       1.0
const GUID tagOneChannelSetting = { 0x3d786f9a, 0xf76e, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   The channel definition which these settings apply to. Index into tagChannelDefns collection of the matching data
source record.
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Required
//  Version:       1.0
const GUID tagChannelDefnIdx = { 0xb48d858f, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Integer ID representing which trigger fields are used.
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagTriggerTypeID = { 0x62f28184, 0xf9c4, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Full scale range for this instrument channel
//  Element type:  Scalar
//  Physical type: REAL8
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagFullScale = { 0x3d786f87, 0xf76e, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Noise floor for this instrument channel
//  Element type:  Scalar
//  Physical type: REAL8
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagNoiseFloor = { 0x3d786f89, 0xf76e, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Low trigger for this channel
//  Element type:  Scalar
//  Physical type: REAL8
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagTriggerLow = { 0x62f28185, 0xf9c4, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   High trigger for this channel
//  Element type:  Scalar
//  Physical type: REAL8
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagTriggerHigh = { 0x62f28186, 0xf9c4, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Rate of change trigger for this channel
//  Element type:  Scalar
//  Physical type: REAL8
//  Required/opt:  Optional
//  Version:       1.0
```

```
const GUID tagTriggerRate = { 0x62f28187, 0xf9c4, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Parameters for shape based triggering algorithms for this channel
//  Element type:  Vector [ 3 ]
//  Physical type: REAL8
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagTriggerShapeParam = { 0x62f28188, 0xf9c4, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   PT or CT
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagXDTransformerTypeID = { 0x62f28189, 0xf9c4, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   System side part of ratio
//  Element type:  Scalar
//  Physical type: REAL8
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagXDSystemSideRatio = { 0x62f2818a, 0xf9c4, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Monitor side part of ratio
//  Element type:  Scalar
//  Physical type: REAL8
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagXDMonitorSideRatio = { 0x62f2818b, 0xf9c4, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Transducer frequency response
//  Element type:  Vector [ # freq ]
//  Physical type: REAL8
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagXDFrequencyResponse = { 0x62f2818c, 0xf9c4, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Chanel time skew
//  Element type:  Scalar
//  Physical type: REAL8
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagCalTimeSkew = { 0x62f2818d, 0xf9c4, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Channel DC offset error
//  Element type:  Scalar
//  Physical type: REAL8
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagCalOffset = { 0x62f2818e, 0xf9c4, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };
```

```
//  Description:   Channel ratio error
//  Element type:  Scalar
//  Physical type: REAL8
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagCalRatio = { 0x62f2818f, 0xf9c4, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Flag indicating that the applied/recordedt calibration arrays must be used to correct data.
//  Element type:  Scalar
//  Physical type: BOOL4
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagCalMustUseARCal = { 0x62f28190, 0xf9c4, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Array of applied signals for this channel
//  Element type:  Vector [ # cal ]
//  Physical type: REAL8
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagCalApplied = { 0x62f28191, 0xf9c4, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Array of recorded actual values for the applied signal
//  Element type:  Vector [ # cal ]
//  Physical type: REAL8
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagCalRecorded = { 0x62f28192, 0xf9c4, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Name of the observation
//  Element type:  Vector [ * ]
//  Physical type: CHAR1
//  Required/opt:  Required
//  Version:       1.0
const GUID tagObservationName = { 0x3d786f8a, 0xf76e, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Time this observation was created
//  Element type:  Scalar
//  Physical type: TIMESTAMP
//  Required/opt:  Required
//  Version:       1.0
const GUID tagTimeCreate = { 0x3d786f8b, 0xf76e, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   The start time of the observation -- the zero point where the .
//  Element type:  Scalar
//  Physical type: TIMESTAMP
//  Required/opt:  Required
//  Version:       1.0
const GUID tagTimeStart = { 0x3d786f8c, 0xf76e, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Type of trigger which caused the observation.
//  Element type:  Scalar
```

```
//  Physical type: UINT4
//  Required/opt:  Required
//  Version:       1.0
const GUID tagTriggerMethodID = { 0x3d786f8d, 0xf76e, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Tme this observation was triggered if appropriate
//  Element type:  Scalar
//  Physical type: TIMESTAMP
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagTimeTriggered = { 0x3d786f8e, 0xf76e, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Index into tagChannelInstances collection within this record. This specifies which channel(s) initiated the
observation.
//  Element type:  Vector
//  Physical type: UINT4
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagChannelTriggerIdx = { 0x3d786f8f, 0xf76e, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   The serial number of the observation (if generated by an instrument, for example).
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagObservationSerial = { 0x3d786f90, 0xf76e, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Serial number -- of specific cycle, for example -- that can be used to correlate observations.
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagObservationAggregationSerial = { 0x89738621, 0xf1c3, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Currently uses the IEEE 1159 disturbance categories, but others could be used as well.
//  Element type:  Scalar
//  Physical type: GUID
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagDisturbanceCategoryID = { 0xb48d8597, 0xf5f5, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   This collection contains a set of channel instances. It is not required to contain the same number of channels as
there are channel instances. This can be determined on an observation-by-observation basis.
//  Element type:  Collection [ # chan ]
//  Physical type: (n/a)
//  Required/opt:  Required
//  Version:       1.0
const GUID tagChannelInstances = { 0x3d786f91, 0xf76e, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Direction of disturbance represented by the data in this observation.  Value of 0 means don't know, 1 means
originated from load side, 2 means originated from source side.
```

```
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Optional
//  Version:       1.5
const GUID tagCharactDisturbDirection = { 0xfa118c0, 0xcb4a, 0x11d2, { 0xb3, 0xb, 0xfe, 0x25, 0xcb, 0x9a, 0x17, 0x60 } };

//  Description:   Quality of the direction result given in another tag.  Range is from 0 (no confidence), to 100 (darn sure).
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Optional
//  Version:       1.5
const GUID tagCharactDisturbDirectionQuality = { 0xfa118c1, 0xcb4a, 0x11d2, { 0xb3, 0xb, 0xfe, 0x25, 0xcb, 0x9a, 0x17, 0x60 } };

//  Description:   One of these collections per channel instance.
//  Element type:  Collection [ * ]
//  Physical type: (n/a)
//  Required/opt:  Required
//  Version:       1.0
const GUID tagOneChannelInst = { 0x3d786f92, 0xf76e, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Simple characterization value: duration of disturbance (seconds)
//  Element type:  Scalar
//  Physical type: REAL8
//  Required/opt:  Optional
//  Version:       1.5
const GUID tagCharactDuration = { 0x2747d444, 0x2bd0, 0x11d2, { 0xae, 0x42, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   This collection must contain the exact number of series which were defined for the specified channel definition.
//  Element type:  Collection [ # sers ]
//  Physical type: (n/a)
//  Required/opt:  Required
//  Version:       1.0
const GUID tagSeriesInstances = { 0x3d786f93, 0xf76e, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Simple characterization value: magnitude of disturbance (percent: 100%=nominal)
//  Element type:  Scalar
//  Physical type: REAL8
//  Required/opt:  Optional
//  Version:       1.5
const GUID tagCharactMagnitude = { 0x2747d443, 0x2bd0, 0x11d2, { 0xae, 0x42, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   Simple characterization value: frequency (Hertz)
//  Element type:  Scalar
//  Physical type: REAL8
//  Required/opt:  Optional
//  Version:       1.5
const GUID tagCharactFrequency = { 0x2747d445, 0x2bd0, 0x11d2, { 0xae, 0x42, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   Contains a 32 bit integer that represents module spefic information related to the trigger reason.
//  Element type:  Scalar
//  Physical type: UINT4
```

```
//  Required/opt:  Optional
//  Version:       1.5
const GUID tagChanTriggerModuleInfo = { 0xfa118c7, 0xcb4a, 0x11d2, { 0xb3, 0xb, 0xfe, 0x25, 0xcb, 0x9a, 0x17, 0x60 } };

//  Description:   Contains the name of a device specific code or hardware module, algorithm or rule not necessarily channel based
that caused this channel to be recorded
//  Element type:  Vector [ * ]
//  Physical type: CHAR1
//  Required/opt:  Optional
//  Version:       1.5
const GUID tagChanTriggerModuleName = { 0xfa118c6, 0xcb4a, 0x11d2, { 0xb3, 0xb, 0xfe, 0x25, 0xcb, 0x9a, 0x17, 0x60 } };

//  Description:   Contains the name of the device involved in an external cross trigger scenario.
//  Element type:  Vector [ * ]
//  Physical type: CHAR1
//  Required/opt:  Optional
//  Version:       1.5
const GUID tagCrossTriggerDeviceName = { 0xfa118c5, 0xcb4a, 0x11d2, { 0xb3, 0xb, 0xfe, 0x25, 0xcb, 0x9a, 0x17, 0x60 } };

//  Description:   Contains the channel definition index of the channel that triggered in a cross trigger scenrio.
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Optional
//  Version:       1.5
const GUID tagCrossTriggerChanIdx = { 0xfa118c4, 0xcb4a, 0x11d2, { 0xb3, 0xb, 0xfe, 0x25, 0xcb, 0x9a, 0x17, 0x60 } };

//  Description:   Integer ID representing the trigger type for this channel instance.  Used only with type ID_QT_VALUELOG wit a
trigger method of channel.
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Optional
//  Version:       1.5
const GUID tagChanTriggerTypeID = { 0xfa118c2, 0xcb4a, 0x11d2, { 0xb3, 0xb, 0xfe, 0x25, 0xcb, 0x9a, 0x17, 0x60 } };

//  Description:   One of these collections per series instance.
//  Element type:  Collection [ * ]
//  Physical type: (n/a)
//  Required/opt:  Required
//  Version:       1.0
const GUID tagOneSeriesInstance = { 0x3d786f94, 0xf76e, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Contains the nominal base voltage, or any other necessary normalizing quantity.
//  Element type:  Scalar
//  Physical type: REAL8
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagSeriesBaseQuantity = { 0x3d786f95, 0xf76e, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   If not present, assumed to be 1. The physical type should match that of tagSeriesValues.
//  Element type:  Scalar
//  Physical type: (any type)
```

```
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagSeriesScale = { 0x3d786f96, 0xf76e, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   If not present, assumed to be 0. Generally used as a starting point when the ID_SERIES_METHOD_INCREMENT storage
method is used. The physical type should match that of tagSeriesValues.
//  Element type:  Scalar
//  Physical type: (any type)
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagSeriesOffset = { 0x3d786f97, 0xf76e, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Identifies the channel which owns the series to be shared. An index into the tagChannelInstances collection.
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagSeriesShareChannelIdx = { 0x8973861f, 0xf1c3, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Identifies the series to be shared. An index into the tagSeriesInstances collection. The tagSeriesValues vector
from this series is used. This must be present if tagSeriesShareChannelIdx is used.
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Optional
//  Version:       1.0
const GUID tagSeriesShareSeriesIdx = { 0x89738620, 0xf1c3, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Contains the actual data points of the series. Required unless the data series is shared, in which case both
tagSeriesShareChannelIdx and tagSeriesShareSeriesIdx should be present.
//  Element type:  Vector [ * ]
//  Physical type: (any type)
//  Required/opt:  Required
//  Version:       1.0
const GUID tagSeriesValues = { 0x3d786f99, 0xf76e, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   For a channel instance which contains a sparse log of time-stamped frequency-domain information of type
ID_QT_VALUELOG, this specifies the frequency for which the quantity characteristic for this series applies (in Hertz).  If this tag is
not present, then the characteristic is independent of frequency unless the characteristic itself is more specific.
//  Element type:  Scalar
//  Physical type: REAL8
//  Required/opt:  Optional
//  Version:       1.5
const GUID tagSeriesFrequency = { 0x2747d442, 0x2bd0, 0x11d2, { 0xae, 0x42, 0x0, 0x60, 0x8, 0x3a, 0x26,
0x28 } };



#endif  //  PQDIF_LG_H
```

```
/*

**  PQDIF - Power Quality Data Interchange Format

**  Version 1.5

**

**  File name:          $Workfile: pqdif_id.h $
**  Last modified:      $Modtime: 4/20/99 3:10p $
**  Last modified by:   $Author: Jamie $
**
**  VCS archive path:   $Archive: /Hank/DMM/FirmWare/Level3/Include/pqdif_id.h $
**  VCS revision:       $Revision: 20 $
**
**  STANDARD ID DEFINITIONS
**  ======================================================================
**  This file contains the current list of standard IDs for the various
**  tags in the PQDIF standard. The IDs consist of two types: GUIDs
**  and integers (UINT4). For now, the integer uses a #define and the GUID
**  uses a const GUID ...
**  ======================================================================
**  The current version of this file and related information
**  can be found at URL:
**
**  http://www.pqnet.electrotek.com/pqnet/main/measure/pqdif/pqdif.htm
**
**  This site can be freely accessed after registration at root URL
**  (http://www.pqnet.electrotek.com)
**  ======================================================================
**
*/

#ifndef PQDIF_ID_H
#define PQDIF_ID_H

//  ======================================================================
//  Do not modify anything after the following comment:
//  {{{{ AUTO-GENERATED CONSTANTS }}}}

//  ============================================================
//  The following IDs are the legal values for
//  tagCompressionStyleID
//  ============================================================

//  Description:   No compression is used.
```

```
//  Version:       1.0
#define ID_COMP_STYLE_NONE 0

//  Description:   The body of each record is compressed individually. The checksums will be found in the header of each record.
//  Version:       1.0
#define ID_COMP_STYLE_RECORDLEVEL 2

//  Description:   Everything after the container record is compressed as a single block. This feature has been deprecated under 1.5
and should not be used.
//  Version:       1.5 Deprecated
#define ID_COMP_STYLE_TOTALFILE 1

//  ============================================================
//  The following IDs are the legal values for
//  tagCompressionAlgorithmID
//  ============================================================

//  Description:   No compression algorithm is used.
//  Version:       1.0
#define ID_COMP_ALG_NONE 0

//  Description:   A standard compression algorithm -- ZLIB -- standardized by the IETF (Internet Engineering Task Force). See
http://quest.jpl.nasa.gov/zlib/ for details.
//  Version:       1.0
#define ID_COMP_ALG_ZLIB 1

//  Description:   A commercial package, the PKZIP data compression library, was used to compress the data. This feature has been
deprecated under 1.5 and should not be used.
//  Version:       1.5 Deprecated
#define ID_COMP_ALG_PKZIPCL 64

//  ============================================================
//  The following IDs are the legal values for
//  tagDataSourceTypeID
//  ============================================================

//  Version:       1.0
const GUID ID_DS_TYPE_MEASURE = { 0xe6b51730, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Version:       1.0
const GUID ID_DS_TYPE_MANUAL = { 0xe6b51731, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Version:       1.0
const GUID ID_DS_TYPE_SIMULATE = { 0xe6b51732, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Version:       1.0
const GUID ID_DS_TYPE_BENCHMARK = { 0xe6b51733, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Version:       1.0
const GUID ID_DS_TYPE_DEBUG = { 0xe6b51734, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };
```

```
//  ============================================================
//  The following IDs are the legal values for
//  tagVendorID
//  ============================================================

const GUID ID_VENDOR_NONE = { 0xe6b51701, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_VENDOR_BMI = { 0xe6b51702, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_VENDOR_BPA = { 0xe6b51703, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_VENDOR_CESI = { 0xe6b51704, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_VENDOR_COOPER = { 0xe6b51705, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_VENDOR_DCG = { 0xe6b51706, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_VENDOR_DRANETZ = { 0xe6b51707, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_VENDOR_EDF = { 0xe6b51708, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_VENDOR_EPRI = { 0xe6b51709, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_VENDOR_ELECTROTEK = { 0xe6b5170a, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_VENDOR_FLUKE = { 0xe6b5170b, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_VENDOR_HYDROQUEBEC = { 0xe6b5170c, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_VENDOR_IEEE = { 0xe6b5170d, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_VENDOR_KREISSJOHNSON = { 0xe6b5170e, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_VENDOR_METROSONIC = { 0xe6b5170f, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_VENDOR_PML = { 0xe6b51710, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_VENDOR_PSI = { 0xe6b51711, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_VENDOR_PTI = { 0xe6b51712, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_VENDOR_PUBLICDOMAIN = { 0xe6b51713, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_VENDOR_RPM = { 0xe6b51714, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_VENDOR_SQUAREDPOWERLOGIC = { 0xe6b51715, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_VENDOR_TELOG = { 0xe6b51716, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  ============================================================
//  The following IDs are the legal values for
```

```
//  tagEquipmentID
//  ============================================================

const GUID ID_EQUIP_BMI_7100 = { 0xe6b51717, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_EQUIP_BMI_8010 = { 0xe6b51718, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_EQUIP_BMI_8020 = { 0xe6b51719, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_EQUIP_BMI_9010 = { 0xe6b5171a, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_EQUIP_COOPER_VHARM = { 0xe6b5171b, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_EQUIP_COOPER_VFLICKER = { 0xe6b5171c, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_EQUIP_DCG_EMTP = { 0xe6b5171d, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_EQUIP_DRANETZ_656 = { 0xe6b5171e, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_EQUIP_DRANETZ_658 = { 0xe6b5171f, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_EQUIP_ETK_TESTPROGRAM = { 0xe6b51721, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_EQUIP_DRANETZ_8000 = { 0xe6b51720, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_EQUIP_ETK_PQDIFEDITOR = { 0xe6b51722, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_EQUIP_ETK_PASS = { 0xe6b51723, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_EQUIP_ETK_SUPERHARM = { 0xe6b51724, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_EQUIP_ETK_SUPERTRAN = { 0xe6b51725, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_EQUIP_ETK_TOP = { 0xe6b51726, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_EQUIP_ETK_PQVIEW = { 0xe6b51727, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_EQUIP_ETK_HARMONI = { 0xe6b51728, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_EQUIP_FLUKE_CUR = { 0xe6b51729, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_EQUIP_IEEE_COMTRADE = { 0xe6b5172b, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_EQUIP_FLUKE_F41 = { 0xe6b5172a, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_EQUIP_PUBLIC_ATP = { 0xe6b5172c, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_EQUIP_METROSONIC_M1 = { 0xe6b5172d, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_EQUIP_SQD_SMS = { 0xe6b5172e, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };
```

```
const GUID ID_EQUIP_TELOG_M1 = { 0xe6b5172f, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_EQUIP_PML_3710 = { 0x85726d0, 0x1dc0, 0x11d0, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_EQUIP_PML_3720 = { 0x85726d1, 0x1dc0, 0x11d0, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_EQUIP_PML_3800 = { 0x85726d2, 0x1dc0, 0x11d0, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_EQUIP_PML_7300 = { 0x85726d3, 0x1dc0, 0x11d0, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_EQUIP_PML_7700 = { 0x85726d4, 0x1dc0, 0x11d0, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_EQUIP_PML_VIP = { 0x85726d5, 0x1dc0, 0x11d0, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

const GUID ID_EQUIP_PML_LOGSERVER = { 0x85726d6, 0x1dc0, 0x11d0, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  ============================================================
//  The following IDs are the legal values for
//  tagInstrumentTypeID
//  ============================================================

//  Version:       1.0
const GUID ID_INSTR_TYPE_SCOPE = { 0xe6b51735, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Frequency recorder.
//  Version:       1.0
const GUID ID_INSTR_TYPE_FR = { 0xe6b51736, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Power quality meter.
//  Version:       1.0
const GUID ID_INSTR_TYPE_PQM = { 0xe6b51737, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Voltage recorder.
//  Version:       1.0
const GUID ID_INSTR_TYPE_VR = { 0xe6b51738, 0xf747, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Version:       1.0
const GUID ID_INSTR_TYPE_SA = { 0xc690e871, 0xf755, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  ============================================================
//  The following IDs are the legal values for
//  tagPhaseID
//  ============================================================

//  Description:   Phase is not applicable.
//  Version:       1.0
#define ID_PHASE_NONE 0

//  Description:   A-to-neutral.
//  Version:       1.0
#define ID_PHASE_AN   1
```

```
//  Description:   B-to-neutral.
//  Version:       1.0
#define ID_PHASE_BN   2

//  Description:   C-to-neutral.
//  Version:       1.0
#define ID_PHASE_CN   3

//  Description:   Neutral-to-ground.
//  Version:       1.0
#define ID_PHASE_NG   4

//  Description:   A-to-B.
//  Version:       1.0
#define ID_PHASE_AB   5

//  Description:   B-to-C.
//  Version:       1.0
#define ID_PHASE_BC   6

//  Description:   C-to-A.
//  Version:       1.0
#define ID_PHASE_CA   7

//  Description:   Residual - the vector or point-on-wave sum of Phases A, B, and C.  Should be zero in a perfectly balanced system.
//  Version:       1.0
#define ID_PHASE_RES  8

//  Description:   Net - the vector or point-on-wave sum of Phases A, B, C and the Neutral phase.  Should be zero in a 4 wire system
with no earth return path.
//  Version:       1.0
#define ID_PHASE_NET  9

//  Description:   The value representing a total or other summarizing value in a multi-phase system.
//  Version:       1.5
#define ID_PHASE_TOTAL 13

//  ============================================================
//  The following IDs are the legal values for
//  tagQuantityTypeID
//  ============================================================

//  Description:   TIME, VAL / For point-on-wave measurements,
//  Version:       1.0
const GUID ID_QT_WAVEFORM = { 0x67f6af80, 0xf753, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   TIME, MIN, MAX, AVG, INST, VAL … / For time-based logged entries.
//  Version:       1.5
const GUID ID_QT_VALUELOG = { 0x67f6af82, 0xf753, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };
```

```
//  Description:   TIME, MIN, MAX, AVG, INST, VAL, PHASEANGLE … / For time-domain  measurements including magnitudes and (optionally)
phase angle.
//  Version:       1.5
const GUID ID_QT_PHASOR = { 0x67f6af81, 0xf753, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   VAL (FREQUENCY), VAL, PHASEANGLE / For frequency-domain measurements including magnitude and (optionally) phase
angle.
//  Version:       1.0
const GUID ID_QT_RESPONSE = { 0x67f6af85, 0xf753, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   TIME, LAT, LON, VAL, POLARITY, ELLIPSE
//  Version:       1.0
const GUID ID_QT_FLASH = { 0x67f6af83, 0xf753, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   BINLOW, BINHIGH, BINID, COUNT
//  Version:       1.0
const GUID ID_QT_HISTOGRAM = { 0x67f6af87, 0xf753, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   XBINLOW, XBINHIGH, YBINLOW, YBINHIGH, BINID, COUNT
//  Version:       1.0
const GUID ID_QT_HISTOGRAM3D = { 0x67f6af88, 0xf753, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   PROB, VAL. (Note that the specific P1 value types have been deprecated.)
//  Version:       1.0
const GUID ID_QT_CPF = { 0x67f6af89, 0xf753, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   VAL, VAL
//  Version:       1.0
const GUID ID_QT_XY = { 0x67f6af8a, 0xf753, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   VAL, DUR
//  Version:       1.0
const GUID ID_QT_MAGDUR = { 0x67f6af8b, 0xf753, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   VAL, VAL, VAL
//  Version:       1.0
const GUID ID_QT_XYZ = { 0x67f6af8c, 0xf753, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   TIME, VAL, DUR
//  Version:       1.0
const GUID ID_QT_MAGDURTIME = { 0x67f6af8d, 0xf753, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   TIME, VAL, DUR, COUNT
//  Version:       1.0
const GUID ID_QT_MAGDURCOUNT = { 0x67f6af8e, 0xf753, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  ============================================================
//  The following IDs are the legal values for
//  tagDisturbanceCategoryID
//  ============================================================
```

```
//  Description:   No IEEE 1159 definition applicable or desired
//  Version:       1.0
const GUID ID_DISTURB_1159_NONE = { 0x67f6af8f, 0xf753, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   IEEE 1159 Transient
//  Version:       1.0
const GUID ID_DISTURB_1159_TRANSIENT = { 0x67f6af90, 0xf753, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   IEEE 1159 Impulsive Transient
//  Version:       1.5
const GUID ID_DISTURB_1159_TRANSIENT_IMPULSIVE = { 0xdd56ef60, 0x7edd, 0x11d2, { 0xb3, 0xa, 0x0, 0x60, 0x97, 0x89, 0xd1, 0x93 } };

//  Description:   IEEE 1159 Impulsive Transient - nanosecond duration
//  Version:       1.5
const GUID ID_DISTURB_1159_TRANSIENT_IMPULSIVE_NANO = { 0xdd56ef61, 0x7edd, 0x11d2, { 0xb3, 0xa, 0x0, 0x60, 0x97, 0x89, 0xd1, 0x93 }
};

//  Description:   IEEE 1159 Impulsive Transient - microsecond duration
//  Version:       1.5
const GUID ID_DISTURB_1159_TRANSIENT_IMPULSIVE_MICRO = { 0xdd56ef63, 0x7edd, 0x11d2, { 0xb3, 0xa, 0x0, 0x60, 0x97, 0x89, 0xd1, 0x93 }
};

//  Description:   IEEE 1159 Impulsive Transient - millisecond duration
//  Version:       1.5
const GUID ID_DISTURB_1159_TRANSIENT_IMPULSIVE_MILLI = { 0xdd56ef64, 0x7edd, 0x11d2, { 0xb3, 0xa, 0x0, 0x60, 0x97, 0x89, 0xd1, 0x93 }
};

//  Description:   IEEE 1159 Oscillatory Transient
//  Version:       1.5
const GUID ID_DISTURB_1159_TRANSIENT_OSCILLATORY = { 0xdd56ef65, 0x7edd, 0x11d2, { 0xb3, 0xa, 0x0, 0x60, 0x97, 0x89, 0xd1, 0x93 } };

//  Description:   IEEE 1159 Oscillatory Transient - Low Frequency
//  Version:       1.5
const GUID ID_DISTURB_1159_TRANSIENT_OSCILLATORY_LOWFREQ = { 0xdd56ef66, 0x7edd, 0x11d2, { 0xb3, 0xa, 0x0, 0x60, 0x97, 0x89, 0xd1,
0x93 } };

//  Description:   IEEE 1159 Oscillatory Transient - Medium Frequency
//  Version:       1.5
const GUID ID_DISTURB_1159_TRANSIENT_OSCILLATORY_MEDFREQ = { 0xdd56ef67, 0x7edd, 0x11d2, { 0xb3, 0xa, 0x0, 0x60, 0x97, 0x89, 0xd1,
0x93 } };

//  Description:   IEEE 1159 Oscillatory Transient - High Frequency
//  Version:       1.5
const GUID ID_DISTURB_1159_TRANSIENT_OSCILLATORY_HIGHFREQ = { 0xdd56ef68, 0x7edd, 0x11d2, { 0xb3, 0xa, 0x0, 0x60, 0x97, 0x89, 0xd1,
0x93 } };

//  Description:   IEEE 1159 Short Duration RMS Variation
//  Version:       1.0
const GUID ID_DISTURB_1159_SHORTDUR = { 0x67f6af91, 0xf753, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   IEEE 1159 Short Duration RMS Variation - Instantaneous duration
```

```
//  Version:       1.5
const GUID ID_DISTURB_1159_SHORTDUR_INSTANT = { 0xdd56ef69, 0x7edd, 0x11d2, { 0xb3, 0xa, 0x0, 0x60, 0x97, 0x89, 0xd1, 0x93 } };

//  Description:   IEEE 1159 Short Duration RMS Variation - Instantaneous Sag
//  Version:       1.5
const GUID ID_DISTURB_1159_SHORTDUR_INSTANT_SAG = { 0xdd56ef6a, 0x7edd, 0x11d2, { 0xb3, 0xa, 0x0, 0x60, 0x97, 0x89, 0xd1, 0x93 } };

//  Description:   IEEE 1159 Short Duration RMS Variation - Instantaneous Swell
//  Version:       1.5
const GUID ID_DISTURB_1159_SHORTDUR_INSTANT_SWELL = { 0xdd56ef6b, 0x7edd, 0x11d2, { 0xb3, 0xa, 0x0, 0x60, 0x97, 0x89, 0xd1, 0x93 } };

//  Description:   IEEE 1159 Short Duration RMS Variation - Momentary Duration
//  Version:       1.5
const GUID ID_DISTURB_1159_SHORTDUR_MOMENT = { 0xdd56ef6c, 0x7edd, 0x11d2, { 0xb3, 0xa, 0x0, 0x60, 0x97, 0x89, 0xd1, 0x93 } };

//  Description:   IEEE 1159 Short Duration RMS Variation - Momentary Interruption
//  Version:       1.5
const GUID ID_DISTURB_1159_SHORTDUR_MOMENT_INTERRUPT = { 0xdd56ef6d, 0x7edd, 0x11d2, { 0xb3, 0xa, 0x0, 0x60, 0x97, 0x89, 0xd1, 0x93 }
};

//  Description:   IEEE 1159 Short Duration RMS Variation - Momentary Sag
//  Version:       1.5
const GUID ID_DISTURB_1159_SHORTDUR_MOMENT_SAG = { 0xdd56ef6e, 0x7edd, 0x11d2, { 0xb3, 0xa, 0x0, 0x60, 0x97, 0x89, 0xd1, 0x93 } };

//  Description:   IEEE 1159 Short Duration RMS Variation - Momentary Swell
//  Version:       1.5
const GUID ID_DISTURB_1159_SHORTDUR_MOMENT_SWELL = { 0xdd56ef6f, 0x7edd, 0x11d2, { 0xb3, 0xa, 0x0, 0x60, 0x97, 0x89, 0xd1, 0x93 } };

//  Description:   IEEE 1159 Short Duration RMS Variation - Temporary Duration
//  Version:       1.5
const GUID ID_DISTURB_1159_SHORTDUR_TEMP = { 0xdd56ef70, 0x7edd, 0x11d2, { 0xb3, 0xa, 0x0, 0x60, 0x97, 0x89, 0xd1, 0x93 } };

//  Description:   IEEE 1159 Short Duration RMS Variation - Temporary Interruption
//  Version:       1.5
const GUID ID_DISTURB_1159_SHORTDUR_TEMP_INTERRUPT = { 0xdd56ef71, 0x7edd, 0x11d2, { 0xb3, 0xa, 0x0, 0x60, 0x97, 0x89, 0xd1, 0x93 } };

//  Description:   IEEE 1159 Short Duration RMS Variation - Temporary Sag
//  Version:       1.5
const GUID ID_DISTURB_1159_SHORTDUR_TEMP_SAG = { 0xdd56ef72, 0x7edd, 0x11d2, { 0xb3, 0xa, 0x0, 0x60, 0x97, 0x89, 0xd1, 0x93 } };

//  Description:   IEEE 1159 Short Duration RMS Variation - Temporary Swell
//  Version:       1.5
const GUID ID_DISTURB_1159_SHORTDUR_TEMP_SWELL = { 0xdd56ef73, 0x7edd, 0x11d2, { 0xb3, 0xa, 0x0, 0x60, 0x97, 0x89, 0xd1, 0x93 } };

//  Description:   IEEE 1159 Long Duration RMS Variation
//  Version:       1.0
const GUID ID_DISTURB_1159_LONGDUR = { 0x67f6af92, 0xf753, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   IEEE 1159 Long Duration RMS Variation - Interruption
//  Version:       1.5
const GUID ID_DISTURB_1159_LONGDUR_INTERRUPT = { 0xdd56ef74, 0x7edd, 0x11d2, { 0xb3, 0xa, 0x0, 0x60, 0x97, 0x89, 0xd1, 0x93 } };
```

```
//  Description:   IEEE 1159 Long Duration RMS Variation - Undervoltage
//  Version:       1.5
const GUID ID_DISTURB_1159_LONGDUR_SAG = { 0xdd56ef75, 0x7edd, 0x11d2, { 0xb3, 0xa, 0x0, 0x60, 0x97, 0x89, 0xd1, 0x93 } };

//  Description:   IEEE 1159 Long Duration RMS Variation - Overvoltage
//  Version:       1.5
const GUID ID_DISTURB_1159_LONGDUR_SWELL = { 0xdd56ef76, 0x7edd, 0x11d2, { 0xb3, 0xa, 0x0, 0x60, 0x97, 0x89, 0xd1, 0x93 } };

//  Description:   IEEE 1159 Imbalance
//  Version:       1.5
const GUID ID_DISTURB_1159_IMBALANCE = { 0xdd56ef77, 0x7edd, 0x11d2, { 0xb3, 0xa, 0x0, 0x60, 0x97, 0x89, 0xd1, 0x93 } };

//  Description:   IEEE 1159 Power Frequency Variation
//  Version:       1.5
const GUID ID_DISTURB_1159_POWERFREQVARIATION = { 0xdd56ef7e, 0x7edd, 0x11d2, { 0xb3, 0xa, 0x0, 0x60, 0x97, 0x89, 0xd1, 0x93 } };

//  Description:   IEEE 1159 Voltage Fluctuation (causes light flicker)
//  Version:       1.5
const GUID ID_DISTURB_1159_VOLTAGEFLUCTUATION = { 0x67f6af93, 0xf753, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   IEEE 1159 Waveform Distortion
//  Version:       1.5
const GUID ID_DISTURB_1159_WAVEDISTORT = { 0x67f6af94, 0xf753, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   DC offset of voltage or current waveform
//  Version:       1.5
const GUID ID_DISTURB_1159_WAVEDISTORT_DCOFFSET = { 0xdd56ef78, 0x7edd, 0x11d2, { 0xb3, 0xa, 0x0, 0x60, 0x97, 0x89, 0xd1, 0x93 } };

//  Description:   IEEE 1159 Waveform Harmonics Present
//  Version:       1.5
const GUID ID_DISTURB_1159_WAVEDISTORT_HARMONIC = { 0xdd56ef79, 0x7edd, 0x11d2, { 0xb3, 0xa, 0x0, 0x60, 0x97, 0x89, 0xd1, 0x93 } };

//  Description:   IEEE 1159 Waveform Interharmonics Present
//  Version:       1.5
const GUID ID_DISTURB_1159_WAVEDISTORT_INTERHARMONIC = { 0xdd56ef7a, 0x7edd, 0x11d2, { 0xb3, 0xa, 0x0, 0x60, 0x97, 0x89, 0xd1, 0x93 }
};

//  Description:   IEEE 1159 Waveform Notching Present
//  Version:       1.5
const GUID ID_DISTURB_1159_WAVEDISTORT_NOTCHING = { 0x67f6af95, 0xf753, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   IEEE 1159 Waveform Noise Present
//  Version:       1.5
const GUID ID_DISTURB_1159_WAVEDISTORT_NOISE = { 0x67f6af96, 0xf753, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  ============================================================
//  The following IDs are the legal values for
//  tagQuantityUnitsID
//  ============================================================
```

```
//  Description:   Unitless.
//  Version:       1.0
#define ID_QU_NONE        0

//  Description:   Seconds -- relative from the beginning time of the observation (using tagTimeStart as the beginning time).
//  Version:       1.0
#define ID_QU_SECONDS     2

//  Description:   Absolute time. Each timestamp in the series must be in absolute time using the TIMESTAMPPQDIF physical type. This
is generally *not* recommended, but is acceptible when _VALUELOG is used.
//  Version:       1.0
#define ID_QU_TIMESTAMP   1

//  Description:   The timestamps are in cycles, relative to tagTimeStart.
//  Version:       1.0
#define ID_QU_CYCLES      3

//  Description:   Volts.
//  Version:       1.0
#define ID_QU_VOLTS       6

//  Description:   Amperes.
//  Version:       1.0
#define ID_QU_AMPS        7

//  Description:   Volt-amperes.
//  Version:       1.0
#define ID_QU_VA          8

//  Description:   Watts.
//  Version:       1.0
#define ID_QU_WATTS       9

//  Description:   Volt-amperes reactive.
//  Version:       1.0
#define ID_QU_VARS        10

//  Description:   Ohms.
//  Version:       1.0
#define ID_QU_OHMS        11

//  Description:   Siemens.
//  Version:       1.0
#define ID_QU_SIEMENS     12

//  Description:   Volts per amp.
//  Version:       1.0
#define ID_QU_VOLTSPERAMP 13

//  Description:   Joules.
//  Version:       1.0
```

```
#define ID_QU_JOULES      14

//  Description:   Hertz.
//  Version:       1.0
#define ID_QU_HERTZ       15

//  Description:   Celcius.
//  Version:       1.0
#define ID_QU_CELCIUS     16

//  Description:   Degrees of arc.
//  Version:       1.0
#define ID_QU_DEGREES     17

//  Description:   Decibels.
//  Version:       1.0
#define ID_QU_DB          18

//  Description:   Percent.
//  Version:       1.0
#define ID_QU_PERCENT     19

//  Description:   Per-unit.
//  Version:       1.0
#define ID_QU_PERUNIT     20

//  Description:   Number of counts or samples
//  Version:       1.0
#define ID_QU_SAMPLES     21

//  Description:   Energy - var-hours
//  Version:       1.5
#define ID_QU_VARHOURS 22

//  Description:   Energy - Watt-hours
//  Version:       1.5
#define ID_QU_WATTHOURS 23

//  Description:   Energy - VA-hours
//  Version:       1.5
#define ID_QU_VAHOURS 24

//  ============================================================
//  The following IDs are the legal values for
//  tagValueTypeID
//  ============================================================

//  Description:   This should be the default value type for a measurement -- a value.
//  Version:       1.0
const GUID ID_SERIES_VALUE_TYPE_VAL = { 0x67f6af97, 0xf753, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };
```

```
//  Description:   Time.
//  Version:       1.0
const GUID ID_SERIES_VALUE_TYPE_TIME = { 0xc690e862, 0xf755, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Minimum.
//  Version:       1.0
const GUID ID_SERIES_VALUE_TYPE_MIN = { 0x67f6af98, 0xf753, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Maximum.
//  Version:       1.0
const GUID ID_SERIES_VALUE_TYPE_MAX = { 0x67f6af99, 0xf753, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Average.
//  Version:       1.0
const GUID ID_SERIES_VALUE_TYPE_AVG = { 0x67f6af9a, 0xf753, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Instantaneous. (depricated - use VAL instead)
//  Version:       1.5 Deprecated
const GUID ID_SERIES_VALUE_TYPE_INST = { 0x67f6af9b, 0xf753, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Phase angle (used for a _VAL series or when it applies to all).
//  Version:       1.0
const GUID ID_SERIES_VALUE_TYPE_PHASEANGLE = { 0x3d786f9d, 0xf76e, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Phase angle which corresponds to a _MIN series (completing a complex pair).
//  Version:       1.5
const GUID ID_SERIES_VALUE_TYPE_PHASEANGLE_MIN = { 0xdc762340, 0x3c56, 0x11d2, { 0xae, 0x44, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   Phase angle which corresponds to a _MAX series.
//  Version:       1.5
const GUID ID_SERIES_VALUE_TYPE_PHASEANGLE_MAX = { 0xdc762341, 0x3c56, 0x11d2, { 0xae, 0x44, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   Phase angle which corresponds to an _AVG series.
//  Version:       1.5
const GUID ID_SERIES_VALUE_TYPE_PHASEANGLE_AVG = { 0xdc762342, 0x3c56, 0x11d2, { 0xae, 0x44, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   Latitude.
//  Version:       1.0
const GUID ID_SERIES_VALUE_TYPE_LATITUDE = { 0xc690e864, 0xf755, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Duration.
//  Version:       1.0
const GUID ID_SERIES_VALUE_TYPE_DURATION = { 0xc690e863, 0xf755, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Longitude.
//  Version:       1.0
const GUID ID_SERIES_VALUE_TYPE_LONGITUDE = { 0xc690e865, 0xf755, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Polarity.
//  Version:       1.0
const GUID ID_SERIES_VALUE_TYPE_POLARITY = { 0xc690e866, 0xf755, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };
```

```
//  Description:   Ellipse (for lightning flash density).
//  Version:       1.0
const GUID ID_SERIES_VALUE_TYPE_ELLIPSE = { 0xc690e867, 0xf755, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Version:       1.0
const GUID ID_SERIES_VALUE_TYPE_BINID = { 0xc690e869, 0xf755, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Version:       1.0
const GUID ID_SERIES_VALUE_TYPE_BINHIGH = { 0xc690e86a, 0xf755, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Version:       1.0
const GUID ID_SERIES_VALUE_TYPE_BINLOW = { 0xc690e86b, 0xf755, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Version:       1.0
const GUID ID_SERIES_VALUE_TYPE_XBINHIGH = { 0xc690e86c, 0xf755, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Version:       1.0
const GUID ID_SERIES_VALUE_TYPE_XBINLOW = { 0xc690e86d, 0xf755, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Version:       1.0
const GUID ID_SERIES_VALUE_TYPE_YBINHIGH = { 0xc690e86e, 0xf755, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Version:       1.0
const GUID ID_SERIES_VALUE_TYPE_YBINLOW = { 0xc690e86f, 0xf755, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Version:       1.0
const GUID ID_SERIES_VALUE_TYPE_COUNT = { 0xc690e870, 0xf755, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Transition event code series.  This series contains codes cooresponding to values in a value series that indicates
what kind of transition caused the event to be recorded.  Used only with VALUELOG data.
//  Version:       1.5
const GUID ID_SERIES_VALUE_TYPE_TRANSITION = { 0x5369c260, 0xc347, 0x11d2, { 0x92, 0x3f, 0x0, 0x10, 0x4b, 0x2b, 0x84, 0xb1 } };

//  Description:   Probability: 1%. This has been deprecated under 1.5 and should not be used.
//  Version:       1.5 Deprecated
const GUID ID_SERIES_VALUE_TYPE_P1 = { 0x67f6af9c, 0xf753, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Probability: 5%. This has been deprecated under 1.5 and should not be used.
//  Version:       1.5 Deprecated
const GUID ID_SERIES_VALUE_TYPE_P5 = { 0x67f6af9d, 0xf753, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Probability: 10%. This has been deprecated under 1.5 and should not be used.
//  Version:       1.5 Deprecated
const GUID ID_SERIES_VALUE_TYPE_P10 = { 0x67f6af9e, 0xf753, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Probability: 90%. This has been deprecated under 1.5 and should not be used.
//  Version:       1.5 Deprecated
const GUID ID_SERIES_VALUE_TYPE_P90 = { 0x67f6af9f, 0xf753, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Probability: 95%. This has been deprecated under 1.5 and should not be used.
```

```
//  Version:       1.5 Deprecated
const GUID ID_SERIES_VALUE_TYPE_P95 = { 0xc690e860, 0xf755, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Probability: 99%. This has been deprecated under 1.5 and should not be used.
//  Version:       1.5 Deprecated
const GUID ID_SERIES_VALUE_TYPE_P99 = { 0xc690e861, 0xf755, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  Description:   Frequency. This has been deprecated under 1.5 and should not be used. (It is now a characteristic instead of a
value type.)
//  Version:       1.5 Deprecated
const GUID ID_SERIES_VALUE_TYPE_FREQUENCY = { 0xc690e868, 0xf755, 0x11cf, { 0x9d, 0x89, 0x0, 0x80, 0xc7, 0x2e, 0x70, 0xa3 } };

//  ============================================================
//  The following IDs are the legal values for
//  tagStorageMethodID
//  ============================================================

//  Description:   The data in tagSeriesValues are a straight array of data points.
//  Version:       1.0
#define ID_SERIES_METHOD_VALUES    0x01

//  Description:   All values in tagSeriesValues will be multiplied by tagSeriesScale.
//  Version:       1.0
#define ID_SERIES_METHOD_SCALED 0x02

//  Description:   The data in tagSeriesValues consists of a special sequence to indicate the contents of a regular rate series (see
main documentation for details). The vector contains: #rates, numpts1, rate1 ... numptsN, rateN.
//  Version:       1.0
#define ID_SERIES_METHOD_INCREMENT 0x04

//  ============================================================
//  The following IDs are the legal values for
//  tagHintGreekPrefixID
//  ============================================================

//  Version:       1.0
#define ID_GREEK_DONTCARE 0

//  Version:       1.0
#define ID_GREEK_FEMTO    1

//  Version:       1.0
#define ID_GREEK_PICO     2

//  Version:       1.0
#define ID_GREEK_NANO     3

//  Version:       1.0
#define ID_GREEK_MICRO    4

//  Version:       1.0
```

```
#define ID_GREEK_MILLI    5

//  Version:       1.0
#define ID_GREEK_NONE     6

//  Version:       1.0
#define ID_GREEK_KILO     7

//  Version:       1.0
#define ID_GREEK_MEGA     8

//  Version:       1.0
#define ID_GREEK_TERA 10

//  Version:       1.0
#define ID_GREEK_GIGA     9

//  ============================================================
//  The following IDs are the legal values for
//  tagHintPreferredUnitsID
//  ============================================================

//  Version:       1.0
#define ID_PREFER_ENG 1

//  Version:       1.0
#define ID_PREFER_PCT 2

//  Version:       1.0
#define ID_PREFER_PU  3

//  ============================================================
//  The following IDs are the legal values for
//  tagHintDefaultDisplayID
//  ============================================================

//  Version:       1.0
#define ID_DEFAULT_DONTCARE 0

//  Version:       1.0
#define ID_DEFAULT_MAG      1

//  Version:       1.0
#define ID_DEFAULT_ANG      2

//  Version:       1.0
#define ID_DEFAULT_REAL     3

//  Version:       1.0
#define ID_DEFAULT_IMAG 4
```

```
//  Version:       1.0
#define ID_DEFAULT_RX 5

//  ============================================================
//  The following IDs are the legal values for
//  tagTriggerTypeID
//  ============================================================

//  Version:       1.0
#define ID_TRIG_NONE  0x00

//  Version:       1.0
#define ID_TRIG_LOW   0x01

//  Version:       1.0
#define ID_TRIG_HIGH  0x02

//  Version:       1.0
#define ID_TRIG_RATE  0x04

//  Version:       1.0
#define ID_TRIG_SHAPE 0x08

//  Version:       1.0
#define ID_TRIG_OTHER 0x10

//  ============================================================
//  The following IDs are the legal values for
//  tagXDTransformerTypeID
//  ============================================================

//  Version:       1.0
#define ID_XFORMER_TYPE_CT 2

//  Version:       1.0
#define ID_XFORMER_TYPE_PT 1

//  ============================================================
//  The following IDs are the legal values for
//  tagTriggerMethodID
//  ============================================================

//  Version:       1.0
#define ID_TRIGGER_METH_NONE 0

//  Description:   A specific channel (or channels) caused the trigger; should be used with tagChannelTriggerIdx to specify which
channels.
//  Version:       1.0
#define ID_TRIGGER_METH_CHANNEL 1

//  Version:       1.0
```

```
#define ID_TRIGGER_METH_PERIODIC 2

//  Version:       1.0
#define ID_TRIGGER_METH_EXTERNAL 3

//  ============================================================
//  The following IDs are the legal values for
//  tagQuantityCharacteristicID
//  ============================================================

//  Version:       1.5
const GUID ID_QC_NONE = { 0xa6b31adf, 0xb451, 0x11d1, { 0xae, 0x17, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   Instantaneous f(t)
//  Version:       1.5
const GUID ID_QC_INSTANTANEOUS = { 0xa6b31add, 0xb451, 0x11d1, { 0xae, 0x17, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   Spectra F(F)
//  Version:       1.5
const GUID ID_QC_SPECTRA = { 0xa6b31ae9, 0xb451, 0x11d1, { 0xae, 0x17, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   Peak value
//  Version:       1.5
const GUID ID_QC_PEAK = { 0xa6b31ae2, 0xb451, 0x11d1, { 0xae, 0x17, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   RMS value
//  Version:       1.5
const GUID ID_QC_RMS = { 0xa6b31ae5, 0xb451, 0x11d1, { 0xae, 0x17, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   Harmonic RMS
//  Version:       1.5
const GUID ID_QC_HRMS = { 0xa6b31adc, 0xb451, 0x11d1, { 0xae, 0x17, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   Frequency
//  Version:       1.5
const GUID ID_QC_FREQUENCY = { 0x7ef68af, 0x9ff5, 0x11d2, { 0xb3, 0xb, 0x0, 0x60, 0x8, 0xb3, 0x71, 0x83 } };

//  Description:   Total harmonic distortion (%)
//  Version:       1.5
const GUID ID_QC_TOTAL_THD = { 0xa6b31aec, 0xb451, 0x11d1, { 0xae, 0x17, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   Even harmonic distortion (%)
//  Version:       1.5
const GUID ID_QC_EVEN_THD = { 0xa6b31ad4, 0xb451, 0x11d1, { 0xae, 0x17, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   Odd harmonic distortion (%)
//  Version:       1.5
const GUID ID_QC_ODD_THD = { 0xa6b31ae0, 0xb451, 0x11d1, { 0xae, 0x17, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   Crest factor
//  Version:       1.5
```

```
const GUID ID_QC_CREST_FACTOR = { 0xa6b31ad2, 0xb451, 0x11d1, { 0xae, 0x17, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   Form factor
//  Version:       1.5
const GUID ID_QC_FORM_FACTOR = { 0xa6b31adb, 0xb451, 0x11d1, { 0xae, 0x17, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   Arithmetic sum
//  Version:       1.5
const GUID ID_QC_ARITH_SUM = { 0xa6b31ad0, 0xb451, 0x11d1, { 0xae, 0x17, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   Zero sequence component unbalance (%)
//  Version:       1.5
const GUID ID_QC_S0S1 = { 0xa6b31ae7, 0xb451, 0x11d1, { 0xae, 0x17, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   Negative sequence component unbalance (%)
//  Version:       1.5
const GUID ID_QC_S2S1 = { 0xa6b31ae8, 0xb451, 0x11d1, { 0xae, 0x17, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   Positive sequence component
//  Version:       1.5
const GUID ID_QC_SPOS = { 0xa6b31aea, 0xb451, 0x11d1, { 0xae, 0x17, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   Imbalance by max deviation from average
//  Version:       1.5
const GUID ID_QC_AVG_IMBAL = { 0xa6b31ad1, 0xb451, 0x11d1, { 0xae, 0x17, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   TIF
//  Version:       1.5
const GUID ID_QC_TIF = { 0xa6b31aeb, 0xb451, 0x11d1, { 0xae, 0x17, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   Flicker average RMS value
//  Version:       1.5
const GUID ID_QC_FLKR_MAG_AVG = { 0xa6b31ad6, 0xb451, 0x11d1, { 0xae, 0x17, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   dV/V base
//  Version:       1.5
const GUID ID_QC_FLKR_MAX_CVV = { 0xa6b31ad8, 0xb451, 0x11d1, { 0xae, 0x17, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   Frequence of maximum flicker harmonic
//  Version:       1.5
const GUID ID_QC_FLKR_FREQ_MAX = { 0xa6b31ad5, 0xb451, 0x11d1, { 0xae, 0x17, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   Magnitude of maximum flicker harmonic
//  Version:       1.5
const GUID ID_QC_FLKR_MAG_MAX = { 0xa6b31ad7, 0xb451, 0x11d1, { 0xae, 0x17, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   Spectrum weighted average
//  Version:       1.5
const GUID ID_QC_FLKR_WGT_AVG = { 0xa6b31ada, 0xb451, 0x11d1, { 0xae, 0x17, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   Flicker spectrum VRMS(F)
```

```
//  Version:       1.5
const GUID ID_QC_FLKR_SPECTRUM = { 0xa6b31ad9, 0xb451, 0x11d1, { 0xae, 0x17, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   IT
//  Version:       1.5
const GUID ID_QC_IT = { 0xa6b31ade, 0xb451, 0x11d1, { 0xae, 0x17, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   RMS value of current for a demand interval
//  Version:       1.5
const GUID ID_QC_RMS_DEMAND = { 0x7ef68a0, 0x9ff5, 0x11d2, { 0xb3, 0xb, 0x0, 0x60, 0x8, 0xb3, 0x71, 0x83 } };

//  Description:   Real power (watts)
//  Version:       1.5
const GUID ID_QC_P = { 0xa6b31ae1, 0xb451, 0x11d1, { 0xae, 0x17, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   Reactive power (VAR)
//  Version:       1.5
const GUID ID_QC_Q = { 0xa6b31ae4, 0xb451, 0x11d1, { 0xae, 0x17, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   Apparent power (VA)
//  Version:       1.5
const GUID ID_QC_S = { 0xa6b31ae6, 0xb451, 0x11d1, { 0xae, 0x17, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   True Power Factor - (Vrms * Irms) / P.
//  Version:       1.5
const GUID ID_QC_PF = { 0xa6b31ae3, 0xb451, 0x11d1, { 0xae, 0x17, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   Displacement Factor - Cosine of the phase angle between fundamental frequency voltage and current phasors.
//  Version:       1.5
const GUID ID_QC_DF = { 0xa6b31ad3, 0xb451, 0x11d1, { 0xae, 0x17, 0x0, 0x60, 0x8, 0x3a, 0x26, 0x28 } };

//  Description:   Value of active power for a demand interval
//  Version:       1.5
const GUID ID_QC_P_DEMAND = { 0x7ef68a1, 0x9ff5, 0x11d2, { 0xb3, 0xb, 0x0, 0x60, 0x8, 0xb3, 0x71, 0x83 } };

//  Description:   Value of reactive power for a demand interval
//  Version:       1.5
const GUID ID_QC_Q_DEMAND = { 0x7ef68a2, 0x9ff5, 0x11d2, { 0xb3, 0xb, 0x0, 0x60, 0x8, 0xb3, 0x71, 0x83 } };

//  Description:   Value of apparent power for a demand interval
//  Version:       1.5
const GUID ID_QC_S_DEMAND = { 0x7ef68a3, 0x9ff5, 0x11d2, { 0xb3, 0xb, 0x0, 0x60, 0x8, 0xb3, 0x71, 0x83 } };

//  Description:   Value of displacement power factor for a demand interval
//  Version:       1.5
const GUID ID_QC_DF_DEMAND = { 0x7ef68a4, 0x9ff5, 0x11d2, { 0xb3, 0xb, 0x0, 0x60, 0x8, 0xb3, 0x71, 0x83 } };

//  Description:   Value of true power factor for a demand interval
//  Version:       1.5
const GUID ID_QC_PF_DEMAND = { 0x7ef68a5, 0x9ff5, 0x11d2, { 0xb3, 0xb, 0x0, 0x60, 0x8, 0xb3, 0x71, 0x83 } };
```

```
//  Description:   Value of active power integrated over time (Energy - watt-hours)
//  Version:       1.5
const GUID ID_QC_P_INTG = { 0x7ef68a6, 0x9ff5, 0x11d2, { 0xb3, 0xb, 0x0, 0x60, 0x8, 0xb3, 0x71, 0x83 } };

//  Description:   Value of active power integrated over time (Energy - watt-hours) in the positive direction (toward load).
//  Version:       1.5
const GUID ID_QC_P_INTG_POS = { 0x7ef68a7, 0x9ff5, 0x11d2, { 0xb3, 0xb, 0x0, 0x60, 0x8, 0xb3, 0x71, 0x83 } };

//  Description:   Value of active power integrated over time (Energy - watt-hours) in the negative direction (away from load).
//  Version:       1.5
const GUID ID_QC_P_INTG_NEG = { 0x7ef68a8, 0x9ff5, 0x11d2, { 0xb3, 0xb, 0x0, 0x60, 0x8, 0xb3, 0x71, 0x83 } };

//  Description:   Value of reactive power integrated over time (var-hours)
//  Version:       1.5
const GUID ID_QC_Q_INTG = { 0x7ef68a9, 0x9ff5, 0x11d2, { 0xb3, 0xb, 0x0, 0x60, 0x8, 0xb3, 0x71, 0x83 } };

//  Description:   Value of reactive power integrated over time (Energy - watt-hours) in the positive direction (toward load).
//  Version:       1.5
const GUID ID_QC_Q_INTG_POS = { 0x7ef68aa, 0x9ff5, 0x11d2, { 0xb3, 0xb, 0x0, 0x60, 0x8, 0xb3, 0x71, 0x83 } };

//  Description:   Value of reactive power integrated over time (Energy - watt-hours) in the negative direction (away from load).
//  Version:       1.5
const GUID ID_QC_Q_INTG_NEG = { 0x7ef68ab, 0x9ff5, 0x11d2, { 0xb3, 0xb, 0x0, 0x60, 0x8, 0xb3, 0x71, 0x83 } };

//  Description:   Value of apparent power integrated over time (VA-hours)
//  Version:       1.5
const GUID ID_QC_S_INTG = { 0x7ef68ac, 0x9ff5, 0x11d2, { 0xb3, 0xb, 0x0, 0x60, 0x8, 0xb3, 0x71, 0x83 } };

//  Description:   Value of displacement power factor coincident with apparent power demand
//  Version:       1.5
const GUID ID_QC_DF_CO_S_DEMAND = { 0x7ef68ad, 0x9ff5, 0x11d2, { 0xb3, 0xb, 0x0, 0x60, 0x8, 0xb3, 0x71, 0x83 } };

//  Description:   Value of true power factor coincident with apparent power demand
//  Version:       1.5
const GUID ID_QC_PF_CO_S_DEMAND = { 0x7ef68ae, 0x9ff5, 0x11d2, { 0xb3, 0xb, 0x0, 0x60, 0x8, 0xb3, 0x71, 0x83 } };

//  ============================================================
//  The following IDs are the legal values for
//  tagQuantityMeasuredID
//  ============================================================

//  Description:   None or not applicable.
//  Version:       1.5
#define ID_QM_NONE 0

//  Description:   Voltage.
//  Version:       1.5
#define ID_QM_VOLTAGE 1

//  Description:   Current.
//  Version:       1.5
```

```
#define ID_QM_CURRENT 2

//  Description:   Power - includes all data for a quantity or characteristic derived from multiplying voltage and current components.
//  Version:       1.5
#define ID_QM_POWER 3

//  Description:   Energy - includes all data from an integration of a quantity or characteristic derived from multiplying voltage and
current components together.
//  Version:       1.5
#define ID_QM_ENERGY 4

//  Description:   Temperature.
//  Version:       1.5
#define ID_QM_TEMPERATURE 5

//  Description:   Pressure.
//  Version:       1.5
#define ID_QM_PRESSURE 6

//  Description:   Charge.
//  Version:       1.5
#define ID_QM_CHARGE 7

//  Description:   Electrical field.
//  Version:       1.5
#define ID_QM_EFIELD 8

//  Description:   Magnetic field.
//  Version:       1.5
#define ID_QM_MFIELD 9

//  ============================================================
//  The following IDs are the legal values for
//  tagChanTriggerTypeID
//  ============================================================

//  Description:   No transition - should not happen
//  Version:       1.5
#define ID_CTT_NONE 0

//  Description:   Normal to low transition
//  Version:       1.5
#define ID_CTT_NORMAL_TO_LO 1

//  Description:   Normal to low low transition
//  Version:       1.5
#define ID_CTT_NORMAL_TO_LO_LO 2

//  Description:   Normal to High transition
//  Version:       1.5
#define ID_CTT_NORMAL_TO_HI 3
```

```
//  Description:   Normal to High High transition
//  Version:       1.5
#define ID_CTT_NORMAL_TO_HI_HI 4

//  Description:   Low Low to Lo transition
//  Version:       1.5
#define ID_CTT_LO_LO_TO_LO 5

//  Description:   Low Low to Normal transition
//  Version:       1.5
#define ID_CTT_LO_LO_TO_NORMAL 6

//  Description:   Low Low to High transition
//  Version:       1.5
#define ID_CTT_LO_LO_TO_HI 7

//  Description:   Low Low to High High transition
//  Version:       1.5
#define ID_CTT_LO_LO_TO_HI_HI 8

//  Description:   Low to Low Low transition
//  Version:       1.5
#define ID_CTT_LO_TO_LO_LO 9

//  Description:   Low to Normal transition
//  Version:       1.5
#define ID_CTT_LO_TO_NORMAL 10

//  Description:   Low to High transition
//  Version:       1.5
#define ID_CTT_LO_TO_HI 11

//  Description:   Low to High High transition
//  Version:       1.5
#define ID_CTT_LO_TO_HI_HI 12

//  Description:   High to Low Low transition
//  Version:       1.5
#define ID_CTT_HI_TO_LO_LO 13

//  Description:   High to Low transition
//  Version:       1.5
#define ID_CTT_HI_TO_LO 14

//  Description:   High to Normal transition
//  Version:       1.5
#define ID_CTT_HI_TO_NORMAL 15

//  Description:   High to High High transition
//  Version:       1.5
```

```
#define ID_CTT_HI_TO_HI_HI 16

//  Description:   High High to Low Low transition
//  Version:       1.5
#define ID_CTT_HI_HI_TO_LO_LO 17

//  Description:   High High to Low transition
//  Version:       1.5
#define ID_CTT_HI_HI_TO_LO 18

//  Description:   High High to Normal transition
//  Version:       1.5
#define ID_CTT_HI_HI_TO_NORMAL 19

//  Description:   High High to High transition
//  Version:       1.5
#define ID_CTT_HI_HI_TO_HI 20

//  Description:   Deadband transition lower
//  Version:       1.5
#define ID_CTT_DB_LO 21

//  Description:   Deadband transition higher
//  Version:       1.5
#define ID_CTT_DB_HI 22

//  Description:   Hardware initiated trigger based on periodic trigger rule
//  Version:       1.5
#define ID_CTT_PERIODIC 23

//  Description:   User commanded sample - button was pushed
//  Version:       1.5
#define ID_CTT_MANUAL 24

//  Description:   Channel triggered because of internal cross-trigger rule.  tagCrossTriggerChanIdx is index of channel that
triggered.
//  Version:       1.5
#define ID_CTT_INT_CROSS_TRIG 25

//  Description:   Channel triggered because of external cross-trigger rule.  tagCrossTriggerChanIdx is index of channel that
triggered on external device.  tagCrossTriggerDeviceName is the name of the external device that initiated the cross trigger.
//  Version:       1.5
#define ID_CTT_EXT_CROSS_TRIG 26

//  Description:   Channel triggered because of hardware or software module, rule or algorithm
//  Version:       1.5
#define ID_CTT_MODULE 27

//  Description:   Rate of change threshold exceeded (dV/dt or dI/dt)
//  Version:       1.5
#define ID_CTT_RATE 28
```

```
#endif  //  PQDIF_ID_H
```