IEEE Std 1159.3™-2003

IEEE Standards

1159.3™

IEEE Recommended Practice for the Transfer of Power Quality Data

IEEE Standards Coordinating Committee 21

Sponsored by the
IEEE Standards Coordinating Committee 21
on Power Quality

IEEE

Published by
The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

12 January 2004

Print: SH95085
PDF: SS95085

Recognized as an
American National Standard (ANSI)

IEEE Std 1159.3™-2003

IEEE Recommended Practice for the Transfer of Power Quality Data

Sponsor

IEEE Standards Coordinating Committee 22
on Power Quality

Approved 7 August 2003

American National Standards Institute

Approved 20 March 2003

IEEE-SA Standards Board

Abstract: This recommended practice defines a file format suitable for exchanging power quality related measurement and simulation data in a vendor independent manner. The format is designed to represent all power quality phenomena identified in IEEE Std 1159™-1995, IEEE Recommended Practice on Monitoring Electric Power Quality, other power related measurement data, and is extensible to other data types as well. The recommended file format utilizes a highly compressed storage scheme to minimize disk space and transmission times. The utilization of globally unique identifiers (GUID) to represent each element in the file permits the format to be extensible without the need for a central registration authority.
Keywords: data interchange, file format, measurement, monitoring, power quality, power quality data interchange format (PQDIF)

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

Print: ISBN 0-7381-3578-X SH95085
PDF: ISBN 0-7381-3579-9 SS95085

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

Use of an IEEE Standard is wholly voluntary. The IEEE disclaims liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEEE Standard document.

The IEEE does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEEE Standards documents are supplied "**AS IS**."

The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

In publishing and making this document available, the IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other IEEE Standards document, should rely upon the advice of a competent professional in determining the exercise of reasonable care in any given circumstances.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE-SA Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
USA

Note: Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Introduction

(This introduction is not part of IEEE Std 1159.3-2003, IEEE Recommended Practice for the Transfer of Power Quality Data.)

This recommended practice evolved over 10 years and includes the work of private companies and research institutions who kindly contributed their earlier work, source code, and staff to the 1159.3 task force when it was formed. This earlier work made it's way into commercial applications whose vendors have continued to upgrade throughout the standards development process. The feedback from those vendors and their customers have provided invaluable experience and feedback that have made this recommended practice a much better document in its initial release.

The primary user of this standard will be engineers and software developers who are called upon to create software to encode and decode files utilized for power quality data interchange using the guidelines in this recommended practice. Although the document is thorough and complete, detailed examples and source code can be quite helpful when implementing the format described herein. Therefore, reference and supporting documentation (header files, sample code, etc.) related to this standard can be found at the 1159.3 task force web site which at the time of this writing could be found at the following URL:

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

Patents

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Interpretations and errata

Interpretations and errata associated with this recommended practice may be found at one of the following Internet locations:

— http://standards.ieee.org/reading/ieee/interp
— http://standards.ieee.org/reading/ieee/updates/errata/

Acknowledgments

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

Core Design and Documentation

Bill Dabbs
Erich W. Gunther
Jack King
Robert F. Scott

Independent Review

Dr. Mesut E. Baran
Daniel Brooks

Example Software and Maintenance

Erich W. Gunther
Jack King
Robert F. Scott

Participants

At the time this recommended practice was completed, the working group on Power Quality Data Interchange Format had the following membership.

Scott Peele, *Chair*

David Kreiss, *Secretary Emeritus*

Richard Bingham, *Secretary*

Erich W. Gunther, *Technical Editor*

Rajaie Abu-Hashim
Vladimir Basch
Roger Bergeron
Mike Bridgewood
Daniel Brooks
Richard Brown
Randy Collins
Larry Conrad
James Crane
John Csomay
Behnam Danai
Andy Detloff
Joe Enk
Stan Fabinwski
Gil Hensley
Mark Kempker
Ajay Koliwad
Jeff Lamoree
Alex McEachern
Mark McGranaghan
Larry Morgan
Allen Morinec
David Mueller
Ram Mukerji
Dan Nordel
Greg Olsen
Pragasen Pillay
V. Rajagopalan
Greg Rauch
Dan Sabin
Andrew Sagl
Surya Santosa
Peter Shah
George Simard
Linh Tu
David Vannoy
Marek Waclawiak
Cheri Warren
Steve Whisenant
James Wikston

The following members of the balloting committee voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

Gregory Ardrey
Richard Bingham
Math Bollen
Richard Brown
Tommy Cooper
Andrew Dettloff
David Gilmer
Thomas Grebe
Randall Groves
Erich W. Gunther
Gilbert Hensley
John Kennedy
David Kreiss
Maurice Linker
Alex McEachern
Mark McGranaghan
Christopher Melhorn
Robert L Morgan
Charles Perry
Paulo Ribeiro
James Ruggieri
Daniel Sabin
Jordan Shikoski
Robert Smith
Dehn Stevens
Cassio Vinhal

When the IEEE-SA Standards Board approved this standard on 20 March 2003, it had the following membership:

Don Wright, *Chair*
Howard M. Frazier, *Vice Chair*
Judith Gorman, *Secretary*

H. Stephen Berger
Joe Bruder
Bob Davis
Richard DeBlasio
Julian Forster*
Toshio Fukuda
Arnold M. Greenspan
Raymond Hapeman
Donald M. Heirman
Laura Hitchcock
Richard H. Hulett
Anant Jain
Lowell G. Johnson
Joseph L. Koepfinger*
Tom McGean
Steve Mills
Daleep C. Mohla
William J. Moylan
Paul Nikolich
Gary Robinson
Malcolm V. Thaden
Geoffrey O. Thompson
Doug Topping
Howard L. Wolfman

*Member Emeritus

Also included are the following nonvoting IEEE-SA Standards Board liaisons:

Alan Cookson, *NIST Representative*
Satish K. Aggarwal, *NRC Representative*

Michelle Turner
IEEE Standards Project Editor

Contents

IEEE Recommended Practice for the Transfer of Power Quality Data

1. Overview

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 (SCC-22) 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 and import from to allow the end user maximum flexibility in choice of tool and vendor.

There are two “layers” to this Power Quality Data Interchange Format (PQDIF): 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 tagged image file format (TIFF), 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
- Little endian byte ordering (least significant byte stored first, ordering used in PCs)
- Tags, using globally unique identifiers (GUIDs), 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

- 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.)

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 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. The IEEE 1159 working group web site (http://grouper.ieee.org/groups/1159/3/doc.html) has public domain example data files, programs, and source code. Annex C documents a commonly implemented disk directory structure for containing multiple PQDIF files.

2. References

This recommended practice shall be used in conjunction with the following publications. When the following publications are superseded by an approved revision, the revision shall apply.

IEEE Std 1159™-1995 (Reaff 2001), IEEE Recommended Practice for Monitoring Electric Power Quality.[1]

RFC 1950, ZLIB Compressed Data Format Specification, Version 3.3.[2]

RFC 1951, DEFLATE Compressed Data Format Specification, Version 1.3.

3. Definitions

For the purposes of this recommended practice, the following terms and definitions apply. *The Authoritative Dictionary of IEEE Standards Terms* should be referenced for terms not defined in this clause.

3.1 globally unique identifier (GUID): A 16 byte value that when randomly generated is large enough to be considered globally unique (infinitesimal chance of another party randomly generating the same value). It is used to uniquely identify an object.

4. Physical structure

4.1 Overall file structure

4.1.1 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.

[1]IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA (http://standards.ieee.org/).

[2]RFC documents are available from Requests for Comments Editor, http://www.rfc-editor.org/.

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

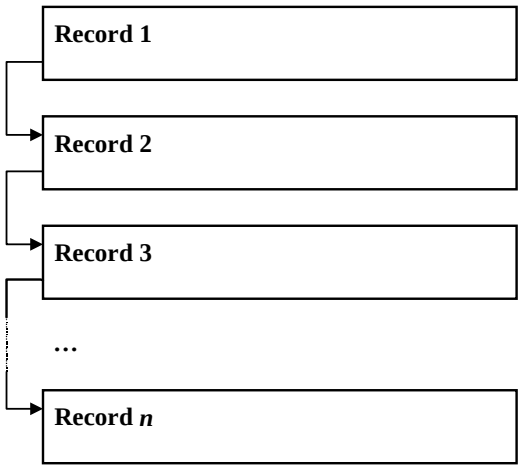

Figure 1—Standard record linkages

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

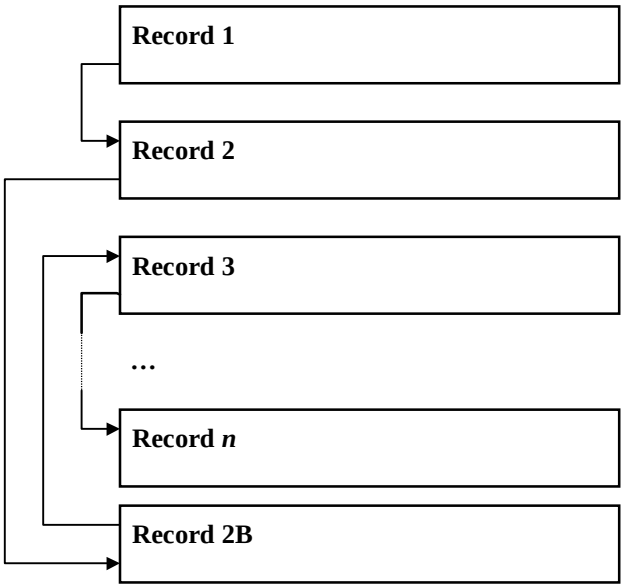

Figure 2—Inserting a new record

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

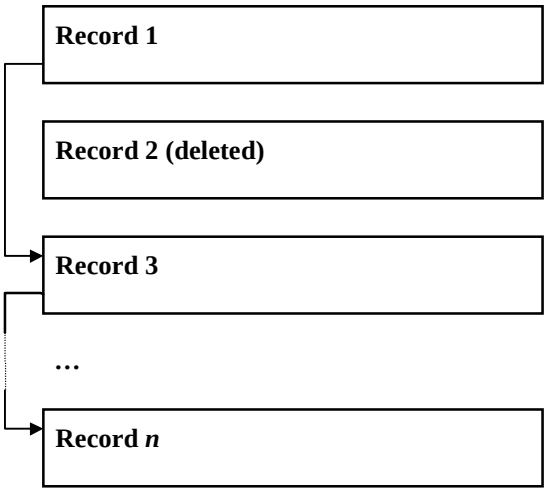

Figure 3—Deleting a record

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

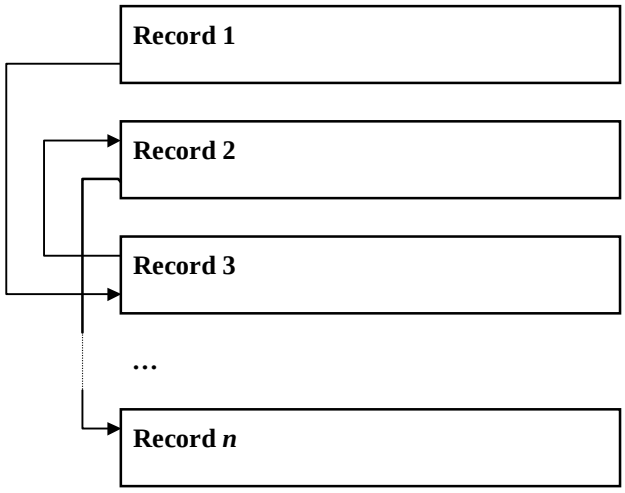

Figure 4—Reordering records

4.1.2 Alternative 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, because the primary focus is the standard flat file that will be used for interchanging data.

4.2 Internal record structure

Each record type in a PQDIF file has the same basic structure consisting of a header and then the record body. Table 1 illustrates the structure of a record.

Table 1—Record structure

Record header *Tag: PQDIF signature *Tag: Type of record *Size of record header *Size of record body	{4a111440-e49f-11cf-9900-505144494600} tag container 64 bytes 512 bytes
Record body *Starts with a collection *Links are relative file reference and point to elements within the body of the record. They are relative to the first byte of the record header.	Collection *Count: 12 Element: 0 *Tag: tagFileName *Type: Vector *Physical type: CHAR1 *Link *Size (16 bytes - padded from 13) _____ Vector *Count: 13 (includes the NULL terminal) *Data: “FILENAMEPQD”

4.2.1 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).

4.2.2 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.

5. Logical structure

5.1 Logical record structure

The basic structure is made up of a logical hierarchy of “records” consisting of a single container, followed 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 shown in Figure 5.

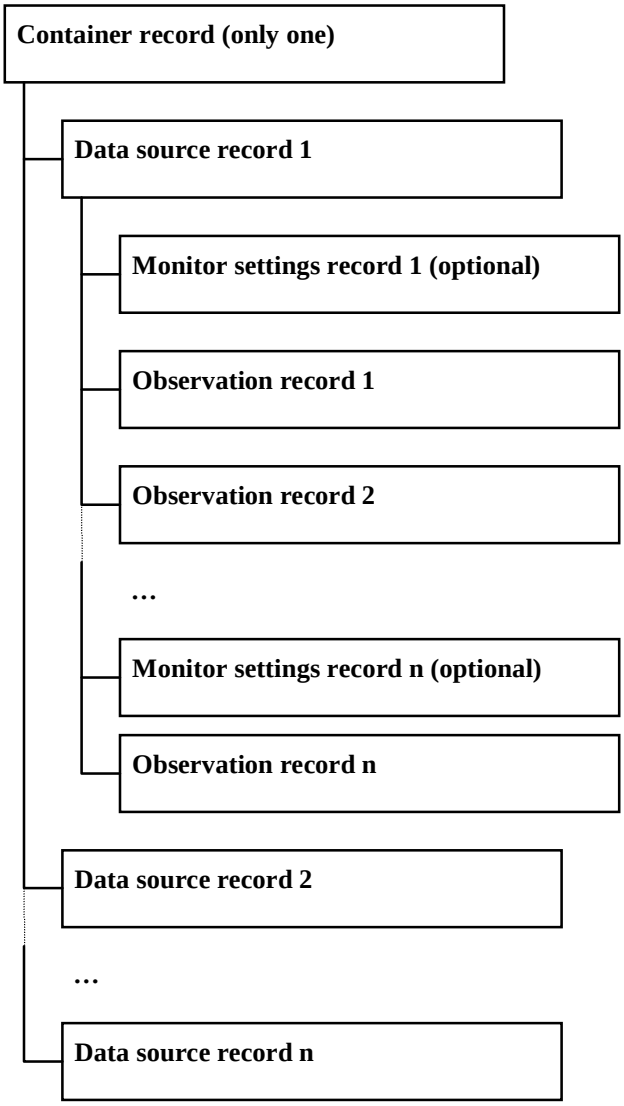

Figure 5—Logical record structure

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. Consequently, 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 prior to the subject observation record are used.

5.2 Logical element structure

5.2.1 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 it 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 it 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 Annex B for full documentation of all tags. See the C language header files (Table 2) included in these documents for the actual tag definitions and for the lists of IDs.

Table 2—C language implementation header files

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).

5.3 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 that are particularly important when interpreting the contents of the file:

— Version

— Compression

5.3.1 Version

Version information is used to control and track the evolution of the format. At some point, major changes may need to be made that 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, as denoted in Table 3.

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.

Table 3—Version-related items

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	The version of the logical structure being used by the writer of the file. This is for a reader to determine.
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	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.

5.3.2 Compression

The container record contains an element indicating whether 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, which provides more information on compression.

5.4 Logical structures of data sources and observations

5.4.1 Structure

The internal structure of the *data source* and *observation records* parallel each other, and in so doing, they are implicitly linked together (Figure 6). 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):

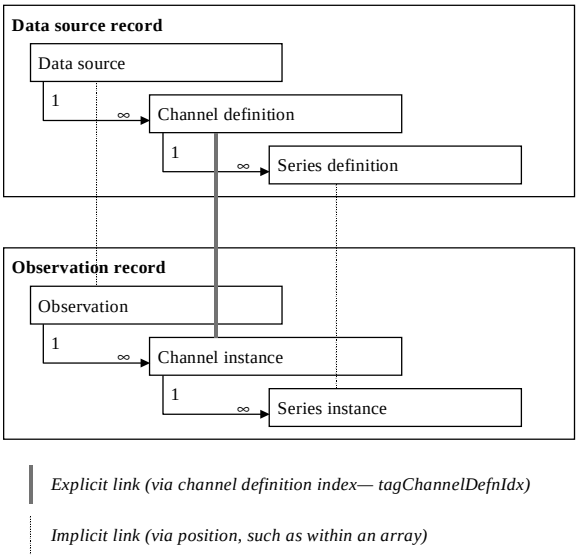

Figure 6—Data source and observation record relationship

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 it serves the same purpose. "Static" information common to all "instances" is kept at the definition (or class) level.

5.4.2 Definitions

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

- 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 observations, as follows:

- 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.)

5.4.3 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 that 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.

5.5 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.

5.5.1 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.

5.5.2 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, as follows:

```
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 as follows:

```
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 as follows:

```
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 that still contains 1792 points, but has two different sampling rates, as follows:

```
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— For time series, the first value should always be 0.

5.5.3 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.

6. Compression

6.1 Compression algorithm—zlib

As 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.gzip.org/zlib/.

The data format used by the zlib library is described by RFC 1950 and RFC 1951.

6.2 Record-level compression

In record-level compression, all record headers are uncompressed (Figure 7). 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.

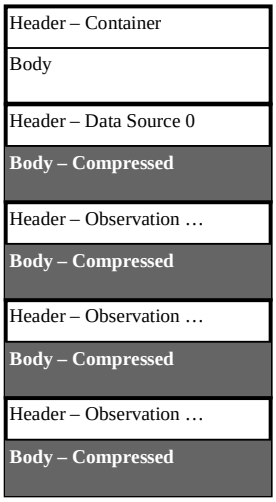

Figure 7—Compressed record layout

As 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.

7. Logical device modeling

Clause 4 and Clause 5 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 its 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 as follows:

— 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.

A waveform, or RMS trace, or value can be recorded as a result of a triggered event as defined by IEEE Std 1159-1995[3], 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 Std 1159-1995 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 Std 1159-1995 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.

[3]Information on references can be found in Clause 2.

7.1 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:

- 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
- Monitor information—general monitor information such as threshold settings, installation date, calibration information, and so on. 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.
- 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.

7.1.1 Channel specifications overview

Before the individual measurements can be translated to the PQDIF format, individual logical channels must be defined in the PQDIF file that 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 that define each channel. Table 4 lists the required element tags that must be assigned values for each channel. Table 4 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 4 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 4 has an associated set of acceptable values. The “Example Value” column of Table 4 shows the values that should be assigned to define the channel that 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 4.

7.1.2 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 exist and would be useful to other applications.

Table 5 lists the required monitor settings tags and example values for the example channel defined in Table 4. Note that only one channel is included in Table 5, 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.

Table 4—Required data source record element tags

Level	Element Tag	Example Value
0	tagRecDataSource	NA
1	tagNominalFrequency	60
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 5—Required monitor settings record element tags

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

7.1.3 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 are 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 6 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 6 are provided in PQDIF logical structure document. The “Example Value” column of Table 6 lists possible values for a phase A voltage waveform associated with the channel specified in Table 4.

Table 6—Required observation record element tags

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, ...

Clause 7.2 through 7.4 describe how to represent these common power quality data types in PQDIF by example.

7.2 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. Figure 8 illustrates a transient event captured by a power monitor, exported into the PQDIF representation described here and visualized by a PQDIF viewer program.

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 Std 1159-1995 (e.g., five cycles of waveform at the beginning of a sag and five 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.

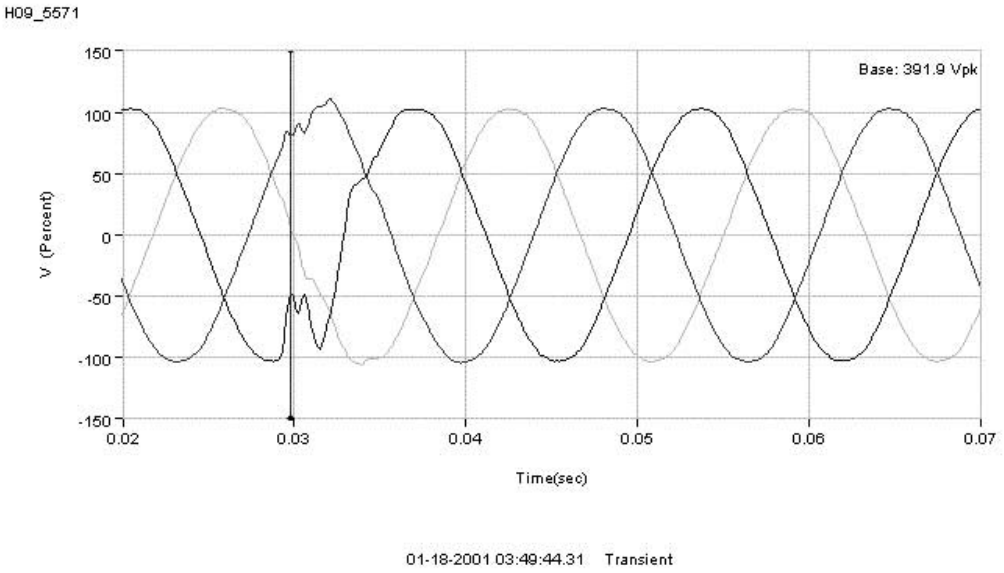

Figure 8—Low-frequency transient (capacitor switching)

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 it 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 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 information is stored in the monitor settings record for reference purposes, but the instance data already have 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 information.

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 specify an optional name “Obs,” 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. As 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 these instance data are for the fourth channel definition (zero-based index of 3) within the data source record, as shown earlier.

The instance data contain the data for both the time and the value series. The time series is first and consists of an array of three 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 1536, 2 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.

7.3 Representing an 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. Figure 9 illustrates an RMS variation event captured by a three-phase power monitor, exported to PQDIF, and then visualized in a PQDIF viewer program.

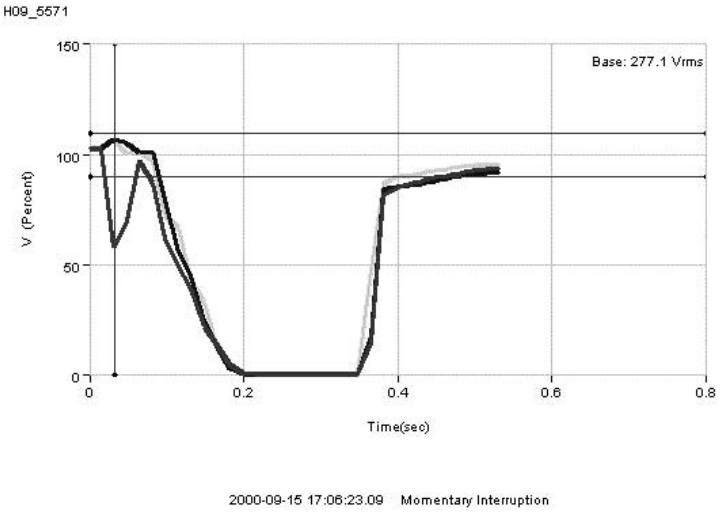

Figure 9—RMS variation showing all three phases and trigger lines

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 maximum, minimum, and average registers for a value. Instead of saving this instantaneous data, however, they save the minimum, maximum, and average values at some prescribed rate (e.g., every six cycles), and then reset the minimum, maximum, 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 voltage:

```
+-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 because 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 minimum, maximum, 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 specify an optional name "RMS Variation," 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. As 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 contain the data for the time and the value series. Each of these series is of the same length (59 entries) because 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 multipane plot, the waveform plot in one pane, the RMS trace plot in another, and the phase angle trace in yet another pane if available. Figure 10 illustrates a plot of this type from data captured by an instrument capable of simultaneous capture of the RMS envelope and waveform data.

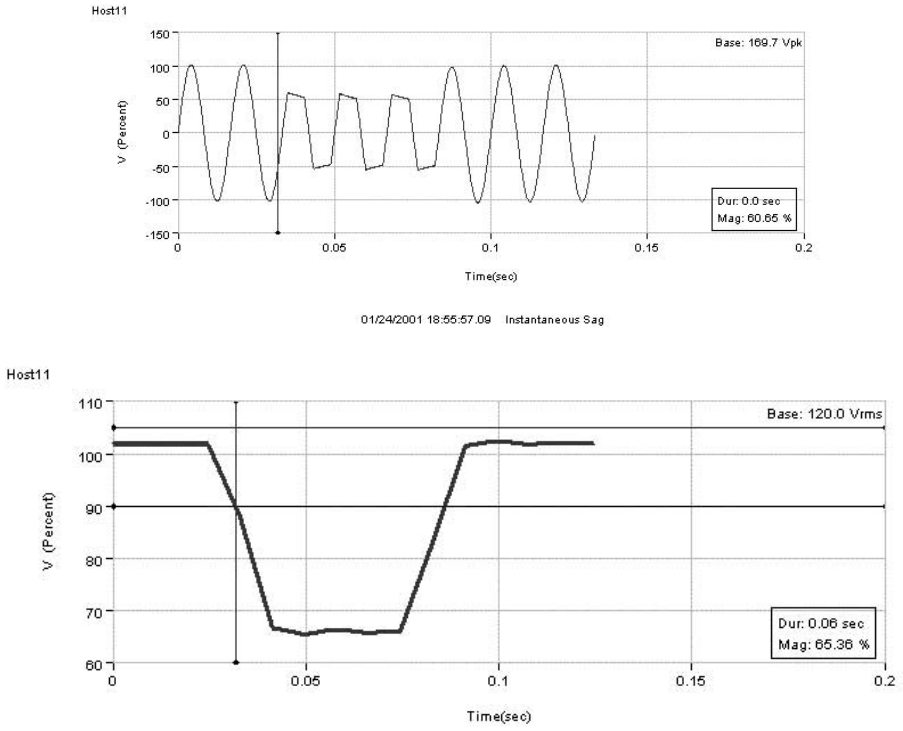

Figure 10—RMS variation represented as waveform and RMS envelope

7.4 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. These data are 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 to exceed some threshold in the steady state. These data are 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, and so on. Additionally, the minimum, maximum, and average values recorded for a characteristic during the reporting interval may also be recorded. Figure 11 illustrates a trend plot of total three-phase power at an office service entrance for a 48 hour period. The instrument used to measure the power calculates the real power value once a second; saves the minimum, maximum, and average of the one second values every five minutes; and then exports a PQDIF file once every 24 hours. A PQDIF viewer program was used to create the figure by processing two such files to create a 48 hour plot.

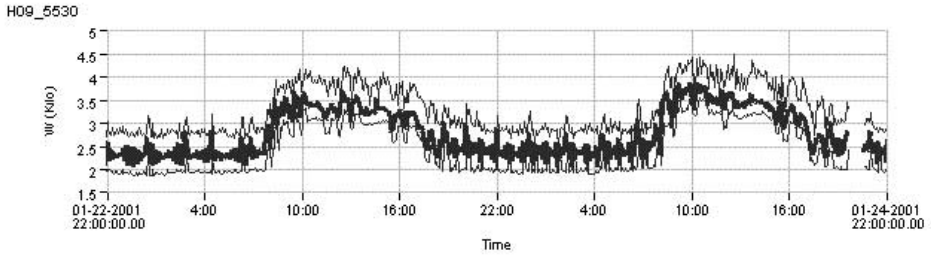

Figure 11—Three-phase real power trend over 48 hours: two PQDIF files joined

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 because we have a finite amount of storage for such logs. A typical native instrument log might have the following entries:

```
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 because it requires the user to parse variable length, nonhomogeneous 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, nonhomogeneous, variable length record log file and creating 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 that 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, the engineer 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 (tagChannelFrequency) 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.

In addition to discrete spectra, PQDIF can represent harmonic and interharmonic groups as defined in pending and current standards such as IEC 61000-4-7. This is done by using the ID_QC_SPECTRA_HGROUP or the ID_QC_SPECTRA_IGROUP quantity characteristics tags as appropriate and then using the tagChannelGroupID to specify the group number (0 based) in the channel instance data instead of the tagChannelFrequency tag as used for discrete spectra.

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 alternative time intervals because they cannot 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 are 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 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. IDs 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 Std 1159-1995 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 Std 1159-1995 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 Std 1159-1995 type and characteristic tags are used in the observation data.

Generally, a single set of channel definitions can be used for all usages 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 nontriggered values and steady state versus power quality event transitions for display purposes.

Annex A

(normative)

Physical format definitions

A.1 Physical format definitions—pqdif_ph.h

```
/*
**  PQDIF - Power Quality Data Interchange Format
**  Version 1.5
**
**  File name:          $Workfile: pqdif_ph.h $
**  Last modified:      $Modtime: 1/07/02 3:59p $
**  Last modified by:   $Author: Erich $
**
**  VCS archive path:   $Archive: /PQDIF/Document/IEEE/pqdif_ph.h $
**  VCS revision:       $Revision: 22 $
**
**  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://grouper.ieee.org/groups/1159/3/docs.html
**
**  ===========================================================================
*/
#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   image;
    } COMPLEX8;

typedef struct _complex16
    {
    REAL8   real;
    REAL8   image;
    } 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 convertible 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 )
#else
    #pragma pack()
#endif


#endif  //  PQDIF_PH_H
```

Annex B

(normative)

Logical structure documentation

The following pages document the logical structure, which tags are mandatory (M) or optional (O), 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. See Table B.1 as an example.

Table B.1—Tag listing format: record-level tag

Element Tag	**Element**	**Physical**	**Count**	***Valid Contents***	**M/O**	**Description**
tagContainer	Collection		*		M	Record-level tag that identifies the container

The other levels (1 through 5) denote various levels of hierarchy (using collections) within the records.

NOTE— 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 an * (asterisk) (see Table B.2).

Table B.2—Tag listing format: count column

Element Tag	**Element**	**Physical**	**Count**	***Valid Contents***	**M/O**	**Description**
1 tagFileName	Vector	CHAR1	*		M	Original name of the file

In some cases, a vector of a specific size is specified. The indices into the vector are usually documented as well (see Table B.3).

Table B.3—Tag listing format: vector size

Element	**Element**	**Physical**	**Count**	***Valid Contents***	**M/O**	**Description**
1 tagVersionInfo	Vector	UNIT4	4	Vector [0]- Major version	M	Specifies the format version for read/ write
						Version of the PQDIF format used for this file

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 (see Table B.4).

Table B.4—Tag listing format: external size definition

Element Tag 1 tagChannelDefns	**Element** Collection	**Physical** NA	**Count** **# defs**	***Valid Contents***	**M/O** M	**Description** The tagChannelDefns must be a

For many scalar elements (whose name ends in ID), the "Valid contents" column specifies the expected values (see Table B.5).

Table B.5—Tag listing format: valid contents

Element Tag **1 tagCompressionStyleID**	**Element** **Scalar**	**Physical** **UNIT4**	**Count**	***Valid Contents*** *ID_COMP_STYLE_NONE = 0* *ID_COMP_STYLE_RECORDLEVEL =2*	**M/O** O	**Description** Specified how the compression is applied to the file
					No compression is used *The body of each record is compressed individually. The checksums will be found in the header of each record.*	

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
```

Table-B.6—Logical structure tag definitions

Level	Element Tag *Valid Contents of element*	Element	Physical Type	Count	M/O	Description
tagContainer		Collection		*	M	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 [0] - Major version* *Vector [1] - Minor version* *Vector [2] - Major version compatible* *Vector [3] - Minor version compatible*	Vector	UINT4	4	M	Specifies the format version for read/write compatibility. The four required numbers in the vector are described below. *Version of the PQDIF format used for this file.* *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	tagFileName	Vector	CHAR1	*	M	Original name of the file.
1	tagCreation	Scalar	TIMESTAMP		M	Date/time when the file was created.
1	tagLastSaved	Scalar	TIMESTAMP		O	Date/time when the file was last saved.
1	tagTimesSaved	Scalar	UINT4		O	The number of times the file has been saved/modified.
1	tagLanguage	Vector	CHAR1	*	O	The language (English, etc.) of the file.
1	tagTitle	Vector	CHAR1	*	O	Arbitrary title. *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	tagSubject	Vector	CHAR1	*	O	Arbitrary subject string
1	tagAuthor	Vector	CHAR1	*	O	Individual/company who caused the file to be written
1	tagKeywords	Vector	CHAR1	*	O	Keywords for assisting searches
1	tagComments	Vector	CHAR1	*	O	Arbitrary comments
1	tagLastSavedBy	Vector	CHAR1	*	O	Individual/company who last wrote to file
1	tagApplication	Vector	CHAR1	*	O	Creating application
1	tagSecurity	Vector	CHAR1	*	O	Security descriptor information
1	tagOwner	Vector	CHAR1	*	O	Owner of file contents (This and some of the following fields are for copyright and trademark information)
1	tagCopyright	Vector	CHAR1	*	O	Copyright notice
1	tagTrademarks	Vector	CHAR1	*	O	Trademark notice
1	tagNotes	Vector	CHAR1	*	O	RTF formatted notes associated with this file (This corresponds to the IEEE COMTRADE .HDR file, for example).
1	tagCompressionStyleID *ID_COMP_STYLE_NONE = 0* *ID_COMP_STYLE_RECORDLEVEL = 2*	Scalar	UINT4		O	Specified how the compression is applied to the file. *No compression is used.* *The body of each record is compressed individually. The checksums will be found in the header of each record.*
1	tagCompressionAlgorithmID *ID_COMP_ALG_NONE = 0* *ID_COMP_ALG_ZLIB = 1*	Scalar	UINT4		O	Required if tagCompressionStyleID specifies that compression is turned on. *No compression algorithm is used.* *A standard compression algorithm -- ZLIB -- standardized by the IETF (Internet Engineering Task Force). See http://quest.jpl.nasa.gov/zlib/ for details.*
1	tagCompressionChecksum	Scalar	UINT4		O	If compression style is _TOTALFILE, this is the checksum for the entire file. This feature has been deprecated under 1.5.
1	tagName	Scalar	UINT4		O	General contact information (all optional)
1	tagAddress1	Scalar	UINT4		O	General contact information (all optional)
1	tagAddress2	Scalar	UINT4		O	General contact information (all optional)
1	tagCity	Scalar	UINT4		O	General contact information (all optional)
1	tagState	Scalar	UINT4		O	General contact information (all optional)
1	tagPostalCode	Scalar	UINT4		O	General contact information (all optional)
1	tagCountry	Scalar	UINT4		O	General contact information (all optional)
1	tagPhoneVoice	Scalar	UINT4		O	General contact information (all optional)
1	tagPhoneFAX	Scalar	UINT4		O	General contact information (all optional)
1	tagEMail	Scalar	UINT4		O	General contact information (all optional)

Table B.6—Logical structure tag definitions *(continued)*

Level	Element Tag *Valid Contents of element*	Element	Physical Type	Count	M/O	Description
tagRecDataSource		Collection		*	M	Record-level tag which identifies a data source (an instrument, etc.).
1	tagDataSourceTypeID	Scalar	GUID		M	Since this ID is a GUID, you can generate a custom ID if a standard ID is not defined.
	ID_DS_TYPE_MEASURE					
	ID_DS_TYPE_MANUAL					
	ID_DS_TYPE_SIMULATE					
	ID_DS_TYPE_BENCHMARK					
	ID_DS_TYPE_DEBUG					
1	tagVendorID	Scalar	GUID		O	Since this ID is a GUID, you can generate a custom ID if a standard ID is not defined.
	ID_VENDOR_WPT					
	ID_VENDOR_SATEC					
	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					
	ID_VENDOR_PMI					
	ID_VENDOR_METONE					
	ID_VENDOR_TRINERGI					*Trinergi*
	ID_VENDOR_GE					*General Electric US*
	ID_VENDOR_LEM					
	ID_VENDOR_ACTL					
	ID_VENDOR_ADVANTECH					*Advantech*
	ID_VENDOR_ELCOM					*Elcom*
1	tagEquipmentID	Scalar	GUID		O	Since this ID is a GUID, you can generate a custom ID if a standard ID is not defined.
	ID_EQUIP_WPT_5540					
	ID_EQUIP_WPT_5530					
	ID_EQUIP_BMI_3100					
	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					
	ID_EQUIP_FLUKE_CUR					
	ID_EQUIP_IEEE_COMTRADE					
	ID_EQUIP_FLUKE_F41					
	ID_EQUIP_PUBLIC_ATP					
	ID_EQUIP_METROSONIC_M1					
	ID_EQUIP_SQD_SMS					

Table B.6—Logical structure tag definitions *(continued)*

Level	Element Tag	Element	Physical Type	Count	M/O	Description
	Valid Contents of element					
	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					
	ID_EQUIP_PMI_SCANNER					
	ID_EQUIP_METONE_ELT15					
	ID_EQUIP_ADVANTECH_ADAM4017					*Advantech ADAM 4017*
	ID_EQUIP_ETK_DSS					*Electrotek DSS*
	ID_EQUIP_ADVANTECH_ADAM4018					*Advantech ADAM 4018*
	ID_EQUIP_ADVANTECH_ADAM4018M					*Advantech ADAM 4018M*
	ID_EQUIP_ADVANTECH_ADAM4052					*Advantech ADAM 4052*
	ID_EQUIP_BMI_8800					*BMI 8800*
	ID_EQUIP_TRINERGI_PQM					*Trinergi PQM*
	ID_EQUIP_MEDCAL					*Medcal*
	ID_EQUIP_GE_KV					*GE kV Energy Meter*
	ID_EQUIP_GE_KV2					*GE kV2 Energy Meter*
	ID_EQUIP_ACUMENTRICS_CONTROL					
	ID_EQUIP_ETK_TEXTPQDIF					
	ID_EQUIP_ETK_PQWEB					
	ID_EQUIP_QWAVE_POWER_DISTRIBUTION					
	ID_EQUIP_QWAVE_POWER_TRANSMISSION					
	ID_EQUIP_QWAVE_MICRO					
	ID_EQUIP_QWAVE_TWIN					
	ID_EQUIP_QWAVE_PREMIUM					
	ID_EQUIP_QWAVE_LIGHT					
	ID_EQUIP_QWAVE_NOMAD					
	ID_EQUIP_EWON_4000					
	ID_EQUIP_QUALIMETRE					
	ID_EQUIP_LEM_ANALYST3Q					
	ID_EQUIP_LEM_ANALYST1Q					
	ID_EQUIP_LEM_ANALYST2050					
	ID_EQUIP_LEM_ANALYST2060					
	ID_EQUIP_LEM_MIDGET200					
	ID_EQUIP_LEM_MBX300					
	ID_EQUIP_LEM_MBX800					
	ID_EQUIP_LEM_MBX601					
	ID_EQUIP_LEM_MBX602					
	ID_EQUIP_LEM_MBX603					
	ID_EQUIP_LEM_MBX686					
	ID_EQUIP_LEM_PERMA701					
	ID_EQUIP_LEM_PERMA702					
	ID_EQUIP_LEM_PERMA705					
	ID_EQUIP_LEM_PERMA706					
	ID_EQUIP_LEM_QWAVEMICRO					
	ID_EQUIP_LEM_QWAVENOMAD					
	ID_EQUIP_LEM_QWAVELIGHT					
	ID_EQUIP_LEM_QWAVETWIN					
	ID_EQUIP_LEM_QWAVEPOWER_DISTRIBUTION					
	ID_EQUIP_LEM_QWAVEPREMIUM					
	ID_EQUIP_LEM_QWAVEPOWER_TRANSPORT					
	ID_EQUIP_LEM_TOPASLT					
	ID_EQUIP_LEM_TOPAS1000					
	ID_EQUIP_LEM_TOPAS1019					
	ID_EQUIP_LEM_TOPAS1020					
	ID_EQUIP_LEM_TOPAS1040					
	ID_EQUIP_LEM_BEN5000					
	ID_EQUIP_LEM_BEN6000					
	ID_EQUIP_LEM_EWAVE					
	ID_EQUIP_WPT_5510					
	ID_EQUIP_LEM_EWON4000					
	ID_EQUIP_WPT_5520					
	ID_EQUIP_WPT_5530T					
	ID_EQUIP_WPT_5560					
	ID_EQUIP_WPT_5590					
	ID_EQUIP_ETK_NODECENTER					
	ID_EQUIP_WPT_DRANVIEW					
	ID_EQUIP_ADVANTECH_ADAM5017					
	ID_EQUIP_ADVANTECH_ADAM5018					
	ID_EQUIP_ADVANTECH_ADAM5080					
	ID_EQUIP_ADVANTECH_ADAM5052					

Table B.6—Logical structure tag definitions *(continued)*

Level	Element Tag	Element	Physical Type	Count	M/O	Description
	Valid Contents of element					
	ID_EQUIP_ADVANTECH_ADAM4050					
	ID_EQUIP_ADVANTECH_ADAM4053					
	ID_EQUIP_ADVANTECH_ADAM4080					
	ID_EQUIP_ADVANTECH_ADAM5050					
	ID_EQUIP_ADVANTECH_ADAM5051					
	ID_EQUIP_ELCOM_BK550					*ELCOM_BK550*
1	tagCustomSourceInfo	Collection	(n/a)		O	This collection can include the standard name, address & telephone number tags -- these apply to the vendor -- as well as tags about the instrument itself.
2	tagInstrumentTypeID	Scalar	GUID		O	
	ID_INSTR_TYPE_SCOPE					
	ID_INSTR_TYPE_FR					*Frequency recorder.*
	ID_INSTR_TYPE_PQM					*Power quality meter.*
	ID_INSTR_TYPE_VR					*Voltage recorder.*
	ID_INSTR_TYPE_SA					
2	tagInstrumentModelName	Vector	CHAR1	*	O	Arbitrary string
2	tagInstrumentModelNumber	Vector	CHAR1	*	O	Arbitrary string
1	tagSerialNumberDS	Vector	CHAR1	*	O	Arbitrary string to hold data souce (instrument) serial number
1	tagVersionDS	Vector	CHAR1	*	O	Arbitrary string to hold data souce (instrument) version number (if applicable)
1	tagNameDS	Vector	CHAR1	*	M	Arbitrary string to hold the name of the data souce (instrument)
1	tagOwnerDS	Vector	CHAR1	*	O	Arbitrary string to hold data souce (instrument) owner name
1	tagLocationDS	Vector	CHAR1	*	O	Arbitrary string to hold data souce (instrument) location information
1	tagTimeZoneDS	Vector	CHAR1	*	O	EST, CST, etc.
1	tagCoordinatesDS	Vector	UINT4	2	O	Longitude/latitude
1	tagChannelDefns	Collection	(n/a)	# defs	M	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.
2	tagOneChannelDefn	Collection	(n/a)	*	M	Must have one or more channel definitions
3	tagChannelName	Vector	CHAR1	*	O	Arbitrary string
3	tagPhaseID	Scalar	UINT4		M	Phase identifier
	ID_PHASE_NONE = 0					*Phase is not applicable.*
	ID_PHASE_AN = 1					*A-to-neutral.*
	ID_PHASE_BN = 2					*B-to-neutral.*
	ID_PHASE_CN = 3					*C-to-neutral.*
	ID_PHASE_NG = 4					*Neutral-to-ground.*
	ID_PHASE_AB = 5					*A-to-B.*
	ID_PHASE_BC = 6					*B-to-C.*
	ID_PHASE_CA = 7					*C-to-A.*
	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.*
	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.*
	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.*
	ID_PHASE_LN_AVE = 14					*The value representing average of 3 line-neutral values*
	ID_PHASE_LL_AVE = 15					*The value representing average of 3 line-line values*
	ID_PHASE_WORST = 16					*The value representing the "worst" of the 3 phases*
	ID_PHASE_PLUS = 17					*DC Positive*
	ID_PHASE_MINUS = 18					*DC Negative*
	ID_PHASE_GENERAL_1 = 19					*Generic Phase 1*
	ID_PHASE_GENERAL_2 = 20					*Generic Phase 2*
	ID_PHASE_GENERAL_3 = 21					*Generic Phase 3*
	ID_PHASE_GENERAL_4 = 22					*Generic Phase 4*
	ID_PHASE_GENERAL_5 = 23					*Generic Phase 5*
	ID_PHASE_GENERAL_6 = 24					*Generic Phase 6*
	ID_PHASE_GENERAL_7 = 25					*Generic Phase 7*

Table B.6—Logical structure tag definitions *(continued)*

Level	Element Tag	Element	Physical Type	Count	M/O	Description
	Valid Contents of element					
	ID_PHASE_GENERAL_8 = 26					*Generic Phase 8*
	ID_PHASE_GENERAL_9 = 27					*Generic Phase 9*
	ID_PHASE_GENERAL_10 = 28					*Generic Phase 10*
	ID_PHASE_GENERAL_11 = 29					*Generic Phase 11*
	ID_PHASE_GENERAL_12 = 30					*Generic Phase 12*
	ID_PHASE_GENERAL_13 = 31					*Generic Phase 13*
	ID_PHASE_GENERAL_14 = 32					*Generic Phase 14*
	ID_PHASE_GENERAL_15 = 33					*Generic Phase 15*
	ID_PHASE_GENERAL_16 = 34					*Generic Phase 16*
3	tagOtherChannelIdentifier	Vector	CHAR1	*	O	Arbitrary string
3	tagGroupName	Vector	CHAR1	*	O	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).
3	tagQuantityTypeID	Scalar	GUID		M	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.).
	ID_QT_WAVEFORM					*TIME, VAL / For point-on-wave measurements,*
	ID_QT_VALUELOG					*TIME, MIN, MAX, AVG, INST, VAL … / For time-based logged entries.*
	ID_QT_PHASOR					*TIME, MIN, MAX, AVG, INST, VAL, PHASEANGLE … / For time-domain measurements including magnitudes and (optionally) phase angle.*
	ID_QT_RESPONSE					*VAL (FREQUENCY), VAL, PHASEANGLE / For frequency-domain measurements including magnitude and (optionally) phase angle.*
	ID_QT_FLASH					*TIME, LAT, LON, VAL, POLARITY, ELLIPSE*
	ID_QT_HISTOGRAM					*BINLOW, BINHIGH, BINID, COUNT*
	ID_QT_HISTOGRAM3D					*XBINLOW, XBINHIGH, YBINLOW, YBINHIGH, BINID, COUNT*
	ID_QT_CPF					*PROB, VAL. (Note that the specific P1 value types have been deprecated.)*
	ID_QT_XY					*VAL, VAL*
	ID_QT_MAGDUR					*VAL, DUR*
	ID_QT_XYZ					*VAL, VAL, VAL*
	ID_QT_MAGDURTIME					*TIME, VAL, DUR*
	ID_QT_MAGDURCOUNT					*TIME, VAL, DUR, COUNT*
	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		M	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.
	ID_QM_NONE = 0					*None or not applicable.*
	ID_QM_VOLTAGE = 1					*Voltage.*
	ID_QM_CURRENT = 2					*Current.*
	ID_QM_POWER = 3					*Power - includes all data for a quantity or characteristic derived from multiplying voltage and current components.*
	ID_QM_ENERGY = 4					*Energy - includes all data from an integration of a quantity or characteristic derived from multiplying voltage and current components together.*
	ID_QM_TEMPERATURE = 5					*Temperature.*
	ID_QM_PRESSURE = 6					*Pressure.*
	ID_QM_CHARGE = 7					*Charge.*
	ID_QM_EFIELD = 8					*Electrical field.*
	ID_QM_MFIELD = 9					*Magnetic field.*
	ID_QM_VELOCITY = 10					*Velocity*
	ID_QM_BEARING = 11					*Compass Bearing*
	ID_QM_FORCE = 12					*Applied Force, Electrical, Mechanical, etc.*
	ID_QM_TORQUE = 13					*Torque*
	ID_QM_POSITION = 14					*Spacial Position*
	ID_QM_FLUXLINKAGE = 15					*Flux linkage Weber Turns*
	ID_QM_FLUXDENSITY = 16					*Magnetic field Density*
	ID_QM_STATUS = 17					*Status Data*
3	tagPhysicalChannel	Scalar	UINT4		O	The instrument physical channel that this channel definition is associated with.

Table B.6—Logical structure tag definitions *(continued)*

Level	Element Tag *Valid Contents of element*	Element	Physical Type	Count	M/O	Description
3	tagQuantityName	Vector	CHAR1	*	O	Additional quantity information.
3	tagPrimarySeriesIdx	Scalar	UINT4		O	Identifies the series which will be the primary. Index into the tagSeriesDefns collection.
3	tagSeriesDefns	Collection	(n/a)	# sers	M	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
4	tagOneSeriesDefn	Collection	(n/a)	*	M	One of these collections per series.
5	tagValueTypeID	Scalar	GUID		M	This specifies the meaning of the series data.
	ID_SERIES_VALUE_TYPE_VAL					*This should be the default value type for a measurement -- a value.*
	ID_SERIES_VALUE_TYPE_TIME					*Time.*
	ID_SERIES_VALUE_TYPE_MIN					*Minimum.*
	ID_SERIES_VALUE_TYPE_MAX					*Maximum.*
	ID_SERIES_VALUE_TYPE_AVG					*Average.*
	ID_SERIES_VALUE_TYPE_PHASEANGLE					*Phase angle (used for a _VAL series or when it applies to all).*
	ID_SERIES_VALUE_TYPE_PHASEANGLE_MIN					*Phase angle which corresponds to a _MIN series (completing a complex pair).*
	ID_SERIES_VALUE_TYPE_PHASEANGLE_MAX					*Phase angle which corresponds to a _MAX series.*
	ID_SERIES_VALUE_TYPE_PHASEANGLE_AVG					*Phase angle which corresponds to an _AVG series.*
	ID_SERIES_VALUE_TYPE_AREA					*Area under the signal, usually an rms voltage, current or other quantity.*
	ID_SERIES_VALUE_TYPE_LATITUDE					*Latitude.*
	ID_SERIES_VALUE_TYPE_DURATION					*Duration.*
	ID_SERIES_VALUE_TYPE_LONGITUDE					*Longitude.*
	ID_SERIES_VALUE_TYPE_POLARITY					*Polarity.*
	ID_SERIES_VALUE_TYPE_ELLIPSE					*Ellipse (for lightning flash density).*
	ID_SERIES_VALUE_TYPE_BINID					
	ID_SERIES_VALUE_TYPE_BINHIGH					
	ID_SERIES_VALUE_TYPE_BINLOW					
	ID_SERIES_VALUE_TYPE_XBINHIGH					
	ID_SERIES_VALUE_TYPE_XBINLOW					
	ID_SERIES_VALUE_TYPE_YBINHIGH					
	ID_SERIES_VALUE_TYPE_YBINLOW					
	ID_SERIES_VALUE_TYPE_COUNT					
	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.*
	ID_SERIES_VALUE_TYPE_PROB					*Cumulative probability in percent.*
	ID_SERIES_VALUE_TYPE_INTERVAL					*Interval data*
	ID_SERIES_VALUE_TYPE_STATUS					*Status Data*
5	tagQuantityUnitsID	Scalar	UINT4		M	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).
	ID_QU_NONE = 0					*Unitless.*
	ID_QU_SECONDS = 2					*Seconds -- relative from the beginning time of the observation (using tagTimeStart as the beginning time).*
	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.*
	ID_QU_CYCLES = 3					*The timestamps are in cycles, relative to tagTimeStart.*
	ID_QU_VOLTS = 6					*Volts.*
	ID_QU_AMPS = 7					*Amperes.*
	ID_QU_VA = 8					*Volt-amperes.*
	ID_QU_WATTS = 9					*Watts.*
	ID_QU_VARS = 10					*Volt-amperes reactive.*
	ID_QU_OHMS = 11					*Ohms.*
	ID_QU_SIEMENS = 12					*Siemens.*
	ID_QU_VOLTSPERAMP = 13					*Volts per amp.*
	ID_QU_JOULES = 14					*Joules.*
	ID_QU_HERTZ = 15					*Hertz.*
	ID_QU_CELCIUS = 16					*Celcius.*
	ID_QU_DEGREES = 17					*Degrees of arc.*
	ID_QU_DB = 18					*Decibels.*
	ID_QU_PERCENT = 19					*Percent.*
	ID_QU_PERUNIT = 20					*Per-unit.*
	ID_QU_SAMPLES = 21					*Number of counts or samples*
	ID_QU_VARHOURS = 22					*Energy - var-hours*
	ID_QU_WATTHOURS = 23					*Energy - Watt-hours*
	ID_QU_VAHOURS = 24					*Energy - VA-hours*

Table B.6—Logical structure tag definitions *(continued)*

Level	Element Tag	Element	Physical Type	Count	M/O	Description
	Valid Contents of element					
	ID_QU_MPS = 25					*Meters/Second*
	ID_QU_MPH = 26					*Miles/Hr*
	ID_QU_BARS = 27					*Pressure, Bars*
	ID_QU_PASCALS = 28					*Pressure, Pascals*
	ID_QU_NEWTONS = 29					*Force, Newtons*
	ID_QU_NEWTONMETERS = 30					*Torque, Newton Meters*
	ID_QU_RPM = 31					*Revolutions/minute*
	ID_QU_RADPERSEC = 32					*Radians/Second*
	ID_QU_METERS = 33					*Meters*
	ID_QU_WEBERTURNS = 34					*Flux Linkage - Weber Turns*
	ID_QU_TESLAS = 35					*Flux Density - Teslas*
	ID_QU_WEBERS = 36					*Magnetic Field - Webers*
	ID_QU_VOLTSPERVOLT = 37					*Volts/Volts transfer function*
	ID_QU_AMPSPERAMP = 38					*Amps/Amps transfer function*
	ID_QU_AMPSPERVOLT = 39					*Impedance Transfer Funtion*
5	tagQuantityCharacteristicID	Scalar	GUID		M	This specifies additional detail about the meaning of the series data.
	General characteristics:					
	ID_QC_NONE					
	ID_QC_INSTANTANEOUS					*Instantaneous f(t)*
	ID_QC_SPECTRA					*Spectra F(F)*
	ID_QC_PEAK					*Peak value*
	ID_QC_RMS					*RMS value*
	ID_QC_HRMS					*Harmonic RMS*
	ID_QC_FREQUENCY					*Frequency*
	ID_QC_TOTAL_THD					*Total harmonic distortion (%)*
	ID_QC_EVEN_THD					*Even harmonic distortion (%)*
	ID_QC_ODD_THD					*Odd harmonic distortion (%)*
	ID_QC_CREST_FACTOR					*Crest factor*
	ID_QC_FORM_FACTOR					*Form factor*
	ID_QC_ARITH_SUM					*Arithmetic sum*
	ID_QC_S0S1					*Zero sequence component unbalance (%)*
	ID_QC_S2S1					*Negative sequence component unbalance (%)*
	ID_QC_SPOS					*Positive sequence component*
	ID_QC_SNEG					*Negative sequence component*
	ID_QC_SZERO					*Zero sequence component*
	ID_QC_AVG_IMBAL					*Imbalance by max deviation from average*
	ID_QC_TOTAL_THD_RMS					*Total THD normalized to RMS*
	ID_QC_ODD_THD_RMS					*Odd THD normalized to RMS*
	ID_QC_EVEN_THD_RMS					*Even THD normalized to RMS*
	ID_QC_TID					*Total Interharmonic Distortion*
	ID_QC_TID_RMS					*Total Interharmonic Distortion Normalized to RMS*
	ID_QC_IHRMS					*Interharmonic RMS*
	ID_QC_SPECTRA_HGROUP					*Spectra by Harmonic Group index*
	ID_QC_SPECTRA_IGROUP					*Spectra by Interharmonic Group Index*
	Voltage characteristics:					
	ID_QC_TIF					*TIF*
	ID_QC_FLKR_MAG_AVG					*Flicker average RMS value*
	ID_QC_FLKR_MAX_DVV					*dV/V base*
	ID_QC_FLKR_FREQ_MAX					*Frequence of maximum flicker harmonic*
	ID_QC_FLKR_MAG_MAX					*Magnitude of maximum flicker harmonic*
	ID_QC_FLKR_WGT_AVG					*Spectrum weighted average*
	ID_QC_FLKR_SPECTRUM					*Flicker spectrum VRMS(F)*
	ID_QC_FLKR_PST					*Short Term Flicker*
	ID_QC_FLKR_PLT					*Long Term Flicker*
	ID_QC_TIF_RMS					*TIF normalized to RMS*
	ID_QC_FLKR_PLTSLIDE					*Sliding PLT*
	ID_QC_FLKR_PILPF					
	ID_QC_FLKR_PIMAX					
	ID_QC_FLKR_PIROOT					
	ID_QC_FLKR_PIROOTLPF					
	Current characteristics:					
	ID_QC_IT					*IT*
	ID_QC_RMS_DEMAND					*RMS value of current for a demand interval*
	ID_QC_ANSI_TDF					*Transformer Derating Factor*
	ID_QC_K_FACTOR					*Transformer K Factor*
	ID_QC_TDD					*Total Demand Distortion*
	ID_QC_RMS_PEAK_DEMAND					*Peak Demand Current*
	Power characteristics:					
	ID_QC_P					*Real power (watts)*
	ID_QC_Q					*Reactive power (VAR)*
	ID_QC_S					*Apparent power (VA)*
	ID_QC_PF					*True Power Factor - (Vrms * Irms) / P.*

Table B.6—Logical structure tag definitions *(continued)*

Level	Element Tag	Element	Physical Type	Count	M/O	Description
	Valid Contents of element					
	ID_QC_DF					*Displacement Factor - Cosine of the phase angle between fundamental frequency voltage and current phasors.*
	ID_QC_P_DEMAND					*Value of active power for a demand interval*
	ID_QC_Q_DEMAND					*Value of reactive power for a demand interval*
	ID_QC_S_DEMAND					*Value of apparent power for a demand interval*
	ID_QC_DF_DEMAND					*Value of displacement power factor for a demand interval*
	ID_QC_PF_DEMAND					*Value of true power factor for a demand interval*
	ID_QC_P_PRED_DEMAND					*Predicted value of active power for current demand interval*
	ID_QC_Q_PRED_DEMAND					*Predicted value of reactive power for current demand interval*
	ID_QC_S_PRED_DEMAND					*Pedicted value of apparent power for current demand interval*
	ID_QC_P_CO_Q_DEMAND					*Value of active power coincident with reactive power demand*
	ID_QC_P_CO_S_DEMAND					*Value of active power coincident with apparent power demand*
	ID_QC_Q_CO_P_DEMAND					*Value of reative power coincident with active power demand*
	ID_QC_Q_CO_S_DEMAND					*Value of reactive power coincident with apparent power demand*
	ID_QC_DF_CO_S_DEMAND					*Value of displacement power factor coincident with apparent power demand*
	ID_QC_PF_CO_S_DEMAND					*Value of true power factor coincident with apparent power demand*
	ID_QC_PF_CO_P_DEMAND					*Value of true power factor coincident with active power demand*
	ID_QC_PF_CO_Q_DEMAND					*Value of true power factor coincident with reactive power demand*
	ID_QC_ANGLE_FUND					*Value of the power angle at fundamental*
	ID_QC_Q_FUND					*Value of the reactive power at fundamental frequency*
	ID_QC_PF_VECTOR					*True Power Factor - IEEE vector calculations*
	ID_QC_DF_VECTOR					*Displacement Factor - IEEE vector calculations*
	ID_QC_S_VECTOR					*Value of apparent power - IEEE vector calculations*
	ID_QC_S_VECTOR_FUND					*Value of fundamental frequency apparent power - IEEE vector calculations*
	ID_QC_S_FUND					*Value of fundamental frequency apparent power*
	ID_QC_S_CO_P_DEMAND					*Apparent power coincident with active power demand*
	ID_QC_S_CO_Q_DEMAND					*Apparent power coincident with reactive power demand*
	ID_QC_PF_ARITH					*True Power Factor - IEEE arithmetic calculations*
	ID_QC_DF_ARITH					*Displacement Factor - IEEE Arithmetic calculations*
	ID_QC_S_ARITH					*Value of apparent power - IEEE Arithmetic calculations*
	ID_QC_S_ARITH_FUND					*Value of fundamental frequency apparent power - IEEE Arithmetic calculations*
	ID_QC_S_PEAK_DEMAND					*Peak Apparent Power Demand*
	ID_QC_Q_PEAK_DEMAND					*Peak Reactive Power Demand*
	ID_QC_P_PEAK_DEMAND					*Peak Active Power Demand*
	ID_QC_P_HARMONIC					*Net Harmonic Actve Power*
	ID_QC_P_HARMONIC_UNSIGNED					*Arithmetic sum Harmonic Actve Power*
	ID_QC_P_FUND					*Value of fundamental frequency real power*
	Energy characteristics:					
	ID_QC_P_INTG					*Value of active power integrated over time (Energy - watt-hours)*
	ID_QC_P_INTG_POS					*Value of active power integrated over time (Energy - watt-hours) in the positive direction (toward load).*
	ID_QC_P_INTG_POS_FUND					*Value of active fund. Freq. power integrated over time (Energy - watt-hours) in the positive direction (toward load).*
	ID_QC_P_INTG_NEG					*Value of active power integrated over time (Energy - watt-hours) in the negative direction (away from load).*
	ID_QC_P_INTG_NEG_FUND					*Value of active fund. Freq. power integrated over time (Energy - watt-hours) in the negative direction (away from load).*
	ID_QC_Q_INTG					*Value of reactive power integrated over time (var-hours)*
	ID_QC_Q_INTG_POS					*Value of reactive power integrated over time (Energy - watt-hours) in the positive direction (toward load).*
	ID_QC_Q_INTG_POS_FUND					*Value of fund. Freq. reactive power integrated over time (Energy - watt-hours) in the positive direction (toward load).*
	ID_QC_Q_INTG_NEG_FUND					*Value of fund. Freq. reactive power integrated over time (Energy - watt-hours) in the negative direction (away from load).*
	ID_QC_Q_INTG_NEG					*Value of reactive power integrated over time (Energy - watt-hours) in the negative direction (away from load).*
	ID_QC_S_INTG					*Value of apparent power integrated over time (VA-hours)*
	ID_QC_S_INTG_FUND					*Value of fundamental frequency apparent power integrated over time (VA-hours)*
	ID_QC_P_IVL_INTG					*Value of active power integrated over time (Energy - watt-hours)*
	ID_QC_P_IVL_INTG_POS					*Value of active power integrated over time (Energy - watt-hours) in the positive direction (toward load).*
	ID_QC_P_IVL_INTG_POS_FUND					*Value of active fund. Freq. power integrated over time (Energy - watt-hours) in the positive direction (toward load).*
	ID_QC_P_IVL_INTG_NEG					*Value of active power integrated over time (Energy - watt-hours) in the negative direction (away from load).*
	ID_QC_P_IVL_INTG_NEG_FUND					*Value of active fund. Freq. power integrated over time (Energy - watt-hours) in the negative direction (away from load).*
	ID_QC_Q_IVL_INTG					*Value of reactive power integrated over time (var-hours)*
	ID_QC_Q_IVL_INTG_POS					*Value of reactive power integrated over time (Energy - watt-hours) in the positive direction (toward load).*
	ID_QC_Q_IVL_INTG_POS_FUND					*Value of fund. Freq. reactive power integrated over time (Energy - watt-hours) in the positive direction (toward load).*

Table B.6—Logical structure tag definitions *(continued)*

Level	Element Tag	Element	Physical Type	Count	M/O	Description
	Valid Contents of element					
	ID_QC_Q_IVL_INTG_NEG_FUND					*Value of fund. Freq. reactive power integrated over time (Energy - watt-hours) in the negative direction (away from load).*
	ID_QC_Q_IVL_INTG_NEG					*Value of reactive power integrated over time (Energy - watt-hours) in the negative direction (away from load).*
	ID_QC_S_IVL_INTG					*Value of apparent power integrated over time (VA-hours)*
	ID_QC_S_IVL_INTG_FUND					*Value of fundamental frequency apparent power integrated over time (VA-hours)*
	ID_QC_DAXISFIELD					*D Axis Components*
	ID_QC_QAXIS					*Q Axis Components*
	ID_QC_ROTATIONAL					*Rotational Position*
	ID_QC_DAXIS					*D Axis Components*
	ID_QC_LINEAR					*Linear Position*
	ID_QC_TRANSFERFUNC					*Transfer function*
	ID_QC_STATUS					*Status Data*
5	tagQuantitySignificantDigitsID	Scalar	UINT4		O	Defines the number of significant digits in the data represented by this series.
5	tagQuantityResolutionID	Scalar	REAL8		O	Contains a double indicating the scaled distance between two values of the quantity represented by this series (e.g. scaled A/D
5	tagStorageMethodID	Scalar	UINT4		M	The legal values for this entry are masks, since they are OR-able.
	ID_SERIES_METHOD_VALUES = 0x01					*The data in tagSeriesValues are a straight array of data points.*
	ID_SERIES_METHOD_SCALED = 0x02					*All values in tagSeriesValues will be multiplied by tagSeriesScale.*
	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.*
5	tagValueTypeName	Vector	CHAR1	*	O	Arbitrary string
5	tagHintGreekPrefixID	Scalar	UINT4		O	
	ID_GREEK_DONTCARE = 0					
	ID_GREEK_FEMTO = 1					
	ID_GREEK_PICO = 2					
	ID_GREEK_NANO = 3					
	ID_GREEK_MICRO = 4					
	ID_GREEK_MILLI = 5					
	ID_GREEK_NONE = 6					
	ID_GREEK_KILO = 7					
	ID_GREEK_MEGA = 8					
	ID_GREEK_TERA = 10					
	ID_GREEK_GIGA = 9					
5	tagHintPreferredUnitsID	Scalar	UINT4		O	
	ID_PREFER_ENG = 1					
	ID_PREFER_PCT = 2					
	ID_PREFER_PU = 3					
5	tagHintDefaultDisplayID	Scalar	UINT4		O	
	ID_DEFAULT_DONTCARE = 0					
	ID_DEFAULT_MAG = 1					
	ID_DEFAULT_ANG = 2					
	ID_DEFAULT_REAL = 3					
	ID_DEFAULT_IMAG = 4					
	ID_DEFAULT_RX = 5					
5	tagProbInterval	Scalar	REAL8		O	For a probability series definition, this specifies its time interval (in seconds; >0).
5	tagProbPercentile	Scalar	REAL8		O	For a probability series definition, this specifies its probability percentile (in percent; 0-100).
5	tagSeriesNominalQuantity	Scalar	REAL8		O	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	tagEffective	Scalar	TIMESTAMP		M	Time that this data source record became

Table B.6—Logical structure tag definitions *(continued)*

Level	Element Tag *Valid Contents of element*	Element	Physical Type	Count	M/O	Description
	tagRecMonitorSettings	Collection		*	O	Record-level tag which identifies a set of configuration parameters.
1	tagEffective	Scalar	TIMESTAMP		M	The time that these settings become effective.
1	tagTimeInstalled	Scalar	TIMESTAMP		M	
1	tagTimeRemoved	Scalar	TIMESTAMP		O	
1	tagUseCalibration	Scalar	BOOL4		M	If TRUE, the calibration adjustments *must* be applied to the series data before using. Otherwise the data is for informative use only.
1	tagUseTransducer	Scalar	BOOL4		M	If TRUE, the transducer adjustments *must* be applied to the series data before using. Otherwise the data is for informative use only.
1	tagChannelSettingsArray	Collection	(n/a)	# chan	M	Channel specific monitor settings stuff
2	tagOneChannelSetting	Collection	(n/a)	*	M	One of these collections per channel.
3	tagChannelDefnIdx	Scalar	UINT4		M	The channel definition which these settings apply to. Index into tagChannelDefns collection of the matching data source record.
3	tagTriggerTypeID *ID_TRIG_NONE = 0x00* *ID_TRIG_LOW = 0x01* *ID_TRIG_HIGH = 0x02* *ID_TRIG_RATE = 0x04* *ID_TRIG_SHAPE = 0x08* *ID_TRIG_OTHER = 0x10*	Scalar	UINT4		O	Integer ID representing which trigger fields are used.
3	tagFullScale	Scalar	REAL8		O	Full scale range for this instrument channel
3	tagNoiseFloor	Scalar	REAL8		O	Noise floor for this instrument channel
3	tagTriggerShapeParam	Vector	REAL8	3	O	Parameters for shape based triggering algorithms for this channel
3	tagXDTransformerTypeID *ID_XFORMER_TYPE_CT = 2* *ID_XFORMER_TYPE_PT = 1*	Scalar	UINT4		O	PT or CT
3	tagXDSystemSideRatio	Scalar	REAL8		O	System side part of ratio
3	tagXDMonitorSideRatio	Scalar	REAL8		O	Monitor side part of ratio
3	tagXDFrequencyResponse	Vector	REAL8	# freq	O	Transducer frequency response
3	tagCalTimeSkew	Scalar	REAL8		O	Chanel time skew
3	tagCalOffset	Scalar	REAL8		O	Channel DC offset error
3	tagCalRatio	Scalar	REAL8		O	Channel ratio error
3	tagCalMustUseARCal	Scalar	BOOL4		O	Flag indicating that the applied/recordedt calibration arrays must be used to correct data.
3	tagCalApplied	Vector	REAL8	# cal	O	Array of applied signals for this channel
3	tagCalRecorded	Vector	REAL8	# cal	O	Array of recorded actual values for the applied signal
3	tagTriggerHighHigh	Scalar	REAL8		O	High High trigger for this channel
3	tagTriggerHigh	Scalar	REAL8		O	High trigger for this channel
3	tagTriggerLow	Scalar	REAL8		O	Low trigger for this channel
3	tagTriggerLowLow	Scalar	REAL8		O	Low Low trigger for this channel
3	tagTriggerDeadBand	Scalar	REAL8		O	Deadband trigger for this channel
3	tagTriggerRate	Scalar	REAL8		O	Rate of change trigger for this channel
1	tagNominalFrequency	Scalar	REAL8		O	Nominal power system frequency for this instrument in Hertz
1	tagSettingPhysicalConnection *ID_SINGLE_PHASE = 1* *ID_2ELEMENT_DELTA = 2* *ID_2_5ELEMENT_WYE = 3* *ID_3ELMENT_WYE = 4* *ID_3ELEMENT_DELTA = 5* *ID_SPLIT_PHASE = 6* *ID_2ELEMENT_2PHASE = 7*	Scalar	UINT4		O	Identifies the physical connection of the instrumentation or insturment transducers. *Single phase connection, 1 voltage, 1 current* *Delta Connected 2 Element Monitoring* *Wye 2 Voltages, 3 Currents* *3 Voltages, 3 Currents* *Delta Connection, 3 voltages, 3 currents* *Split Single Phase, 2 Volage, 2 Current* *2 Phase, 2 Voltages, 2 Currents*

Table B.6—Logical structure tag definitions *(continued)*

Level	Element Tag *Valid Contents of element*	Element	Physical Type	Count	M/O	Description
tagRecObservation		Collection		*	M	Record-level tag which identifies an observation -- an event, measurement, etc.
1	tagObservationName	Vector	CHAR1	*	M	Name of the observation
1	tagTimeCreate	Scalar	TIMESTAMP		M	Time this observation was created
1	tagTimeStart	Scalar	TIMESTAMP		M	The start time of the observation -- the zero point where the .
1	tagTriggerMethodID	Scalar	UINT4		M	Type of trigger which caused the observation.
	ID_TRIGGER_METH_NONE = 0					
	ID_TRIGGER_METH_CHANNEL = 1					*A specific channel (or channels) caused the trigger; should be used with tagChannelTriggerIdx to specify which channels.*
	ID_TRIGGER_METH_PERIODIC = 2					
	ID_TRIGGER_METH_EXTERNAL = 3					
	ID_TRIGGER_METH_PERIODIC_STATS = 4					*Periodic Statistical Data*
1	tagTimeTriggered	Scalar	TIMESTAMP		O	Tme this observation was triggered if appropriate
1	tagChannelTriggerIdx	Vector	UINT4		O	Index into tagChannelInstances collection within this record. This specifies which channel(s) initiated the observation.
1	tagObservationSerial	Scalar	UINT4		O	The serial number of the observation (if generated by an instrument, for example).
1	tagObservationAggregationSerial	Scalar	UINT4		O	Serial number -- of specific cycle, for example -- that can be used to correlate observations.
1	tagDisturbanceCategoryID	Scalar	GUID		O	Currently uses the IEEE 1159 disturbance categories, but others could be used as well.
	ID_DISTURB_1159_NONE					*No IEEE 1159 definition applicable or desired*
	ID_DISTURB_1159_TRANSIENT					*IEEE 1159 Transient*
	ID_DISTURB_1159_TRANSIENT_IMPULSIVE					*IEEE 1159 Impulsive Transient*
	ID_DISTURB_1159_TRANSIENT_IMPULSIVE_NANO					*IEEE 1159 Impulsive Transient - nanosecond duration*
	ID_DISTURB_1159_TRANSIENT_IMPULSIVE_MICRO					*IEEE 1159 Impulsive Transient - microsecond duration*
	ID_DISTURB_1159_TRANSIENT_IMPULSIVE_MILLI					*IEEE 1159 Impulsive Transient - millisecond duration*
	ID_DISTURB_1159_TRANSIENT_OSCILLATORY					*IEEE 1159 Oscillatory Transient*
	ID_DISTURB_1159_TRANSIENT_OSCILLATORY_LOWFREQ					*IEEE 1159 Oscillatory Transient - Low Frequency*
	ID_DISTURB_1159_TRANSIENT_OSCILLATORY_MEDFREQ					*IEEE 1159 Oscillatory Transient - Medium Frequency*
	ID_DISTURB_1159_TRANSIENT_OSCILLATORY_HIGHFREQ					*IEEE 1159 Oscillatory Transient - High Frequency*
	ID_DISTURB_1159_SHORTDUR					*IEEE 1159 Short Duration RMS Variation*
	ID_DISTURB_1159_SHORTDUR_INSTANT					*IEEE 1159 Short Duration RMS Variation - Instantaneous duration*
	ID_DISTURB_1159_SHORTDUR_INSTANT_SAG					*IEEE 1159 Short Duration RMS Variation - Instantaneous Sag*
	ID_DISTURB_1159_SHORTDUR_INSTANT_SWELL					*IEEE 1159 Short Duration RMS Variation - Instantaneous Swell*
	ID_DISTURB_1159_SHORTDUR_MOMENT					*IEEE 1159 Short Duration RMS Variation - Momentary Duration*
	ID_DISTURB_1159_SHORTDUR_MOMENT_INTERRUPT					*IEEE 1159 Short Duration RMS Variation - Momentary Interruption*
	ID_DISTURB_1159_SHORTDUR_MOMENT_SAG					*IEEE 1159 Short Duration RMS Variation - Momentary Sag*
	ID_DISTURB_1159_SHORTDUR_MOMENT_SWELL					*IEEE 1159 Short Duration RMS Variation - Momentary Swell*
	ID_DISTURB_1159_SHORTDUR_TEMP					*IEEE 1159 Short Duration RMS Variation - Temporary Duration*
	ID_DISTURB_1159_SHORTDUR_TEMP_INTERRUPT					*IEEE 1159 Short Duration RMS Variation - Temporary Interruption*
	ID_DISTURB_1159_SHORTDUR_TEMP_SAG					*IEEE 1159 Short Duration RMS Variation - Temporary Sag*
	ID_DISTURB_1159_SHORTDUR_TEMP_SWELL					*IEEE 1159 Short Duration RMS Variation - Temporary Swell*
	ID_DISTURB_1159_LONGDUR					*IEEE 1159 Long Duration RMS Variation*
	ID_DISTURB_1159_LONGDUR_INTERRUPT					*IEEE 1159 Long Duration RMS Variation - Interruption*
	ID_DISTURB_1159_LONGDUR_SAG					*IEEE 1159 Long Duration RMS Variation - Undervoltage*
	ID_DISTURB_1159_LONGDUR_SWELL					*IEEE 1159 Long Duration RMS Variation - Overvoltage*
	ID_DISTURB_1159_IMBALANCE					*IEEE 1159 Imbalance*
	ID_DISTURB_1159_POWERFREQVARIATION					*IEEE 1159 Power Frequency Variation*
	ID_DISTURB_1159_FLICKER					*IEEE 1159 Light Flicker due to Voltage Fluctuations*
	ID_DISTURB_1159_VOLTAGEFLUCTUATION					*IEEE 1159 Voltage Fluctuation (causes light flicker)*
	ID_DISTURB_1159_HARMONIC					*IEEE 1159 Waveform Distortion*
	ID_DISTURB_1159_WAVEDISTORT					*IEEE 1159 Waveform Distortion*
	ID_DISTURB_1159_WAVEDISTORT_DCOFFSET					*DC offset of voltage or current waveform*
	ID_DISTURB_1159_WAVEDISTORT_HARMONIC					*IEEE 1159 Waveform Harmonics Present*
	ID_DISTURB_1159_WAVEDISTORT_INTERHARMONIC					*IEEE 1159 Waveform Interharmonics Present*
	ID_DISTURB_1159_NOTCH					*IEEE 1159 Notching - waveforms exhibiting power electronic commutation notches*
	ID_DISTURB_1159_WAVEDISTORT_NOTCHING					*IEEE 1159 Waveform Notching Present*
	ID_DISTURB_1159_NOISE					*IEEE 1159 Noise - broadband noise signals*
	ID_DISTURB_1159_WAVEDISTORT_NOISE					*IEEE 1159 Waveform Noise Present*
1	tagChannelInstances	Collection	(n/a)	# chan	M	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.

Table B.6—Logical structure tag definitions *(continued)*

Level	Element Tag *Valid Contents of element*	Element	Physical Type	Count	M/O	Description
2	tagOneChannelInst	Collection	(n/a)	*	M	One of these collections per channel instance.
3	tagChannelDefnIdx	Scalar	UINT4		M	Specifies which of the available channel definitions this instance belongs to. Index into tagChannelDefns collection of the matching data source record.
3	tagSeriesInstances	Collection	(n/a)	# sers	M	This collection must contain the exact number of series which were defined for the specified channel definition.
4	tagOneSeriesInstance	Collection	(n/a)	*	M	One of these collections per series instance.
5	tagSeriesBaseQuantity	Scalar	REAL8		O	Contains the nominal base voltage, or any other necessary normalizing quantity.
5	tagSeriesScale	Scalar	(any type)		O	If not present, assumed to be 1. The physical type should match that of tagSeriesValues.
5	tagSeriesOffset	Scalar	(any type)		O	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.
5	tagSeriesShareChannelIdx	Scalar	UINT4		O	Identifies the channel which owns the series to be shared. An index into the tagChannelInstances collection.
5	tagSeriesShareSeriesIdx	Scalar	UINT4		O	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.
5	tagSeriesValues	Vector	(any type)	*	M	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.
	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.*
	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.*
3	tagCharactMagnitude	Scalar	REAL8		O	Simple characterization value: magnitude of disturbance (percent: 100%=nominal)
3	tagCharactFrequency	Scalar	REAL8		O	Simple characterization value: frequency (Hertz)
3	tagChanTriggerModuleInfo	Scalar	UINT4		O	Contains a 32 bit integer that represents module spefic information related to the trigger reason.
3	tagChanTriggerModuleName	Vector	CHAR1	*	O	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
3	tagCrossTriggerDeviceName	Vector	CHAR1	*	O	Contains the name of the device involved in an external cross trigger scenario.
3	tagCrossTriggerChanIdx	Scalar	UINT4		O	Contains the channel definition index of the channel that triggered in a cross trigger scenrio.
3	tagChanTriggerTypeID	Scalar	UINT4		O	Integer ID representing the trigger type for this channel instance. Used only with type ID_QT_VALUELOG wit a trigger method of channel.
	ID_CTT_NONE = 0					*No transition - should not happen*
	ID_CTT_NORMAL_TO_LO = 1					*Normal to low transition*
	ID_CTT_NORMAL_TO_LO_LO = 2					*Normal to low low transition*
	ID_CTT_NORMAL_TO_HI = 3					*Normal to High transition*
	ID_CTT_NORMAL_TO_HI_HI = 4					*Normal to High High transition*
	ID_CTT_LO_LO_TO_LO = 5					*Low Low to Lo transition*
	ID_CTT_LO_LO_TO_NORMAL = 6					*Low Low to Normal transition*
	ID_CTT_LO_LO_TO_HI = 7					*Low Low to High transition*
	ID_CTT_LO_LO_TO_HI_HI = 8					*Low Low to High High transition*
	ID_CTT_LO_TO_LO_LO = 9					*Low to Low Low transition*
	ID_CTT_LO_TO_NORMAL = 10					*Low to Normal transition*
	ID_CTT_LO_TO_HI = 11					*Low to High transition*
	ID_CTT_LO_TO_HI_HI = 12					*Low to High High transition*
	ID_CTT_HI_TO_LO_LO = 13					*High to Low Low transition*

Table B.6—Logical structure tag definitions *(continued)*

Level	Element Tag	Element	Physical Type	Count	M/O	Description
	Valid Contents of element					
	ID_CTT_HI_TO_LO = 14					*High to Low transition*
	ID_CTT_HI_TO_NORMAL = 15					*High to Normal transition*
	ID_CTT_HI_TO_HI_HI = 16					*High to High High transition*
	ID_CTT_HI_HI_TO_LO_LO = 17					*High High to Low Low transition*
	ID_CTT_HI_HI_TO_LO = 18					*High High to Low transition*
	ID_CTT_HI_HI_TO_NORMAL = 19					*High High to Normal transition*
	ID_CTT_HI_HI_TO_HI = 20					*High High to High transition*
	ID_CTT_DB_LO = 21					*Deadband transition lower*
	ID_CTT_DB_HI = 22					*Deadband transition higher*
	ID_CTT_PERIODIC = 23					*Hardware initiated trigger based on periodic trigger rule*
	ID_CTT_MANUAL = 24					*User commanded sample - button was pushed*
	ID_CTT_INT_CROSS_TRIG = 25					*Channel triggered because of internal cross-trigger rule. tagCrossTriggerChanIdx is index of channel that triggered.*
	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.*
	ID_CTT_MODULE = 27					*Channel triggered because of hardware or software module, rule or algorithm*
	ID_CTT_RATE = 28					*Rate of change threshold exceeded (dV/dt or dI/dt)*
3	tagChannelGroupID	Scalar	INT2		O	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 Index). The index referes to a harmonic or interharmonic group index.
2	tagCharactDuration	Scalar	REAL8		O	Simple characterization value: duration of disturbance (seconds)
1	tagCharactDisturbDirection	Scalar	UINT4		O	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	tagCharactDisturbDirectionQuality	Scalar	UINT4		O	Quality of the direction result given in another tag. Range is from 0 (no confidence), to 100 (darn sure).

B.1 Logical format constant definitions—pqdif_lg.h

The following listing contains all of the GUID constants that define the logical structure of a PQDIF file.

```
/*
**  PQDIF - Power Quality Data Interchange Format
**  Version 1.5
**
**  File name:          $Workfile: pqdif_lg.h $
**  Last modified:      $Modtime: 1/17/02 2:04p $
**  Last modified by:   $Author: Jack $
**
**  VCS archive path:   $Archive: /PQDIF/Document/Version15/pqdif_lg.h $
**  VCS revision:       $Revision: 54 $
**
**  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://grouper.ieee.org/groups/1159/3/docs.html
**
**  ==========================================================================
**
**  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 |
**        |   +---------------+
**        |       |
**        |       |   +--------------------+
**        |       +---| Monitor Settings 1 |
**        |       |   +--------------------+
**        |       |       |
**        |       |       |   +---------------+
**        |       |       +---| Observation 1 |
**        |       |       |   +---------------+
**        |       |       |   +---------------+
**        |       |       +---| Observation 2 |
**        |       |       |   +---------------+
**        |       |      ...
**        |       |       |   +---------------+
```



```
**        |       |       +---| Observation n |
**        |       |           +---------------+
**        |       |
**        |       |   +--------------------+
**        |       +---| Monitor Settings 2 |
**        |       |   +--------------------+
**        |      ...
**        |       |   +--------------------+
**        |       +---| Monitor Settings n |
**        |       |   +--------------------+
**        |      ...
**        |   +---------------+
**        +---| Data Source 2 |
**        |   +---------------+
**       ...
**        |   +---------------+
**        +---| Data Source n |
**        |   +---------------+
**       ...
**
**  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 num-
ber 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 cap-
tured 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:   Contains a double indicating the scaled distance between two values of
the quantity represented by this series (e.g. scaled A/D resolution)
//  Element type:  Scalar
//  Physical type: REAL8
//  Required/opt:  Optional
//  Version:       1.5
const GUID tagQuantityResolutionID = { 0xfb228ee0, 0xfc8d, 0x11d2, { 0xb4, 0x9a, 0x0, 0x60,
0x8, 0xb3, 0x71, 0x83 } };

//  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:   Identifies the physical connection of the instrumentation or instrument
transducers.
```

```
//  Element type:  Scalar
//  Physical type: UINT4
//  Required/opt:  Optional
//  Version:       1.5
const GUID tagSettingPhysicalConnection = { 0x9f256ee0, 0x803b, 0x11d3, { 0xb9, 0x2f, 0x0,
0x50, 0xda, 0x2b, 0x1f, 0x4d } };

//  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:   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:   High High trigger for this channel
//  Element type:  Scalar
//  Physical type: REAL8
//  Required/opt:  Optional
//  Version:       1.5
const GUID tagTriggerHighHigh = { 0x5b12f431, 0xff54, 0x11d3, { 0xb9, 0x68, 0x0, 0x50,
0xda, 0x2b, 0x1f, 0x4d } };

//  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:   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:   Low Low trigger for this channel
//  Element type:  Scalar
//  Physical type: REAL8
//  Required/opt:  Optional
//  Version:       1.5
const GUID tagTriggerLowLow = { 0x5b12f430, 0xff54, 0x11d3, { 0xb9, 0x68, 0x0, 0x50, 0xda,
0x2b, 0x1f, 0x4d } };

//  Description:   Deadband trigger for this channel
//  Element type:  Scalar
//  Physical type: REAL8
//  Required/opt:  Optional
//  Version:       1.5
```



```
const GUID tagTriggerDeadBand = { 0x5b12f432, 0xff54, 0x11d3, { 0xb9, 0x68, 0x0, 0x50,
0xda, 0x2b, 0x1f, 0x4d } };

//  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:   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:   Time 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 deter-
mined 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 specific 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, algo-
rithm 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 scenario.
//  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:   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 specified by the quantity characteristic.
//  Element type:  Scalar
//  Physical type: REAL8
//  Required/opt:  Optional
//  Version:       1.5
const GUID tagChannelFrequency = { 0x2747d442, 0x2bd0, 0x11d2, { 0xae, 0x42, 0x0, 0x60,
0x8, 0x3a, 0x26, 0x28 } };

//  Description:   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 Index). The index refers to a harmonic or interharmonic
group index.
//  Element type:  Scalar
//  Physical type: INT2
//  Required/opt:  Optional
//  Version:       1.5
const GUID tagChannelGroupID = { 0xf90de218, 0xe67b, 0x4cf1, { 0xa2, 0x95, 0xb0, 0x21,
0xa2, 0xd4, 0x67, 0x67 } };

//  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 } };
```

```
#endif  //  PQDIF_LG_H
```

B.2 Constant IDs and GUIDs

The following listing details constant GUID's and integer ID's used in the format.

```
/*
**  PQDIF - Power Quality Data Interchange Format
**  Version 1.5
**
**  File name:          $Workfile: pqdif_id.h $
**  Last modified:      $Modtime: 2/01/02 1:40p $
**  Last modified by:   $Author: Jack $
**
**  VCS archive path:   $Archive: /PQDIF/Document/Version15/pqdif_id.h $
**  VCS revision:       $Revision: 48 $
**
**  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://grouper.ieee.org/groups/1159/3/docs.html
**
**  ======================================================================
**
*/

#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_SATEC = { 0xe2da5081, 0x7fdb, 0x11d3, { 0x9b, 0x39, 0x0, 0x40, 0x5,
0x2c, 0x2d, 0x28 } };

const GUID ID_VENDOR_WPT = { 0xe2da5082, 0x7fdb, 0x11d3, { 0x9b, 0x39, 0x0, 0x40, 0x5,
0x2c, 0x2d, 0x28 } };
```

```
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 } };

//  Version:       1.5
const GUID ID_VENDOR_PMI = { 0x609acec0, 0x993d, 0x11d4, { 0xa4, 0xb3, 0x44, 0x45, 0x53,
0x54, 0x0, 0x0 } };

//  Version:       1.5
const GUID ID_VENDOR_METONE = { 0xb5b5da61, 0xe2e1, 0x11d4, { 0x82, 0xd9, 0x0, 0xe0, 0x98,
0x72, 0xa0, 0x94 } };

//  Description:   Trinergi
//  Version:       1.5
const GUID ID_VENDOR_TRINERGI = { 0xfd5a3a8, 0xd73a, 0x11d2, { 0xac, 0x3e, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  Description:   General Electric US
//  Version:       1.5
const GUID ID_VENDOR_GE = { 0x5202bd00, 0x245c, 0x11d5, { 0xa4, 0xb3, 0x44, 0x45, 0x53,
0x54, 0x0, 0x0 } };

//  Version:       1.5
const GUID ID_VENDOR_LEM = { 0x80c4a722, 0x2816, 0x11d4, { 0x8a, 0xb4, 0x0,0x40, 0x5, 0x69,
0x8d, 0x26 } };

//  Version:       1.5
const GUID ID_VENDOR_ACTL = { 0x80c4a761, 0x2816, 0x11d4, { 0x8a, 0xb4,0x0,0x40, 0x5, 0x69,
0x8d, 0x26 } };

//  Description:   Advantech
//  Version:       1.5
const GUID ID_VENDOR_ADVANTECH = { 0x650f988f, 0x378c, 0x47b8, { 0xba, 0xed, 0xcc, 0xcb,
0x3f, 0x95, 0x9a, 0xd7 } };

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

const GUID ID_EQUIP_WPT_5530 = { 0xe2da5083, 0x7fdb, 0x11d3, { 0x9b, 0x39, 0x0, 0x40, 0x5,
0x2c, 0x2d, 0x28 } };

const GUID ID_EQUIP_WPT_5540 = { 0xe2da5084, 0x7fdb, 0x11d3, { 0x9b, 0x39, 0x0, 0x40, 0x5,
0x2c, 0x2d, 0x28 } };

const GUID ID_EQUIP_BMI_3100 = { 0xf1c04780, 0x50fb, 0x11d3, { 0xac, 0x3e, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

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 } };

//  Version:       1.5
const GUID ID_EQUIP_METONE_ELT15 = { 0xb5b5da62, 0xe2e1, 0x11d4, { 0x82, 0xd9, 0x0, 0xe0,
0x98, 0x72, 0xa0, 0x94 } };

//  Version:       1.5
const GUID ID_EQUIP_PMI_SCANNER = { 0x609acec1, 0x993d, 0x11d4, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  Description:   Advantech ADAM 4017
//  Version:       1.5
const GUID ID_EQUIP_ADVANTECH_ADAM4017 = { 0x92b7977b, 0xc02, 0x4766, { 0x95, 0xcf, 0xdd,
0x37, 0x9c, 0xae, 0xb4, 0x17 } };

//  Description:   Electrotek DSS
//  Version:       1.5
const GUID ID_EQUIP_ETK_DSS = { 0xd347ba66, 0xe34c, 0x11d4, { 0x82, 0xd9, 0x0, 0xe0, 0x98,
0x72, 0xa0, 0x94 } };

//  Description:   Advantech ADAM 4018
//  Version:       1.5
const GUID ID_EQUIP_ADVANTECH_ADAM4018 = { 0x3008151e, 0x2317, 0x4405, { 0xa5, 0x9e, 0xe7,
0xb3, 0xb2, 0x6, 0x67, 0xa9 } };

//  Description:   Advantech ADAM 4018M
```

```
//  Version:       1.5
const GUID ID_EQUIP_ADVANTECH_ADAM4018M = { 0x3a1af807, 0x1347, 0x45f8, { 0x96, 0x6a, 0xf4,
0x81, 0xc6, 0xae, 0x20, 0x8e } };

//  Description:   Advantech ADAM 4052
//  Version:       1.5
const GUID ID_EQUIP_ADVANTECH_ADAM4052 = { 0x8bba416b, 0xa7ec, 0x4616, { 0x8b, 0x8f, 0x59,
0xfe, 0xd7, 0x49, 0x32, 0x3d } };

//  Description:   BMI 8800
//  Version:       1.5
const GUID ID_EQUIP_BMI_8800 = { 0xe77d1a81, 0x1235, 0x11d5, { 0xa3, 0x90, 0x0, 0x10, 0xa4,
0x92, 0x4e, 0xcc } };

//  Description:   Trinergi PQM
//  Version:       1.5
const GUID ID_EQUIP_TRINERGI_PQM = { 0xfd5a3aa, 0xd73a, 0x11d2, { 0xac, 0x3e, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  Description:   Medcal
//  Version:       1.5
const GUID ID_EQUIP_MEDCAL = { 0xf3bfa0a1, 0xeb87, 0x11d2, { 0xac, 0x3e, 0x44, 0x45, 0x53,
0x54, 0x0, 0x0 } };

//  Description:   GE kV Energy Meter
//  Version:       1.5
const GUID ID_EQUIP_GE_KV = { 0x5202bd01, 0x245c, 0x11d5, { 0xa4, 0xb3, 0x44, 0x45, 0x53,
0x54, 0x0, 0x0 } };

//  Description:   GE kV2 Energy Meter
//  Version:       1.5
const GUID ID_EQUIP_GE_KV2 = { 0x5202bd03, 0x245c, 0x11d5, { 0xa4, 0xb3, 0x44, 0x45, 0x53,
0x54, 0x0, 0x0 } };

//  Version:       1.5
const GUID ID_EQUIP_ACUMENTRICS_CONTROL = { 0x5202bd04, 0x245c, 0x11d5, { 0xa4, 0xb3, 0x44,
0x45, 0x53, 0x54, 0x0, 0x0 } };

//  Version:       1.5
const GUID ID_EQUIP_ETK_TEXTPQDIF = { 0x5202bd05, 0x245c, 0x11d5, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  Version:       1.5
const GUID ID_EQUIP_ETK_PQWEB = { 0x5202bd06, 0x245c, 0x11d5, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  Version:       1.5
const GUID ID_EQUIP_QWAVE_POWER_DISTRIBUTION = { 0x80c4a723, 0x2816,0x11d4,{ 0x8a, 0xb4,
0x0, 0x40, 0x5, 0x69, 0x8d, 0x26 } };

//  Version:       1.5
const GUID ID_EQUIP_QWAVE_POWER_TRANSMISSION = { 0x80c4a725, 0x2816,0x11d4,{ 0x8a, 0xb4,
0x0, 0x40, 0x5, 0x69, 0x8d, 0x26 } };

//  Version:       1.5
const GUID ID_EQUIP_QWAVE_MICRO = { 0x80c4a727, 0x2816, 0x11d4, { 0x8a,0xb4, 0x0, 0x40,
0x5, 0x69, 0x8d, 0x26 } };
```



```
//  Version:       1.5
const GUID ID_EQUIP_QWAVE_TWIN = { 0x80c4a728, 0x2816, 0x11d4, { 0x8a,0xb4,0x0, 0x40, 0x5,
0x69, 0x8d, 0x26 } };

//  Version:       1.5
const GUID ID_EQUIP_QWAVE_PREMIUM = { 0x80c4a729, 0x2816, 0x11d4, { 0x8a,0xb4, 0x0, 0x40,
0x5, 0x69, 0x8d, 0x26 } };

//  Version:       1.5
const GUID ID_EQUIP_QWAVE_LIGHT = { 0x80c4a72a, 0x2816, 0x11d4, { 0x8a,0xb4, 0x0, 0x40,
0x5, 0x69, 0x8d, 0x26 } };

//  Version:       1.5
const GUID ID_EQUIP_QWAVE_NOMAD = { 0x80c4a72b, 0x2816, 0x11d4, { 0x8a,0xb4, 0x0, 0x40,
0x5, 0x69, 0x8d, 0x26 } };

//  Version:       1.5
const GUID ID_EQUIP_EWON_4000 = { 0x80c4a762, 0x2816, 0x11d4, { 0x8a, 0xb4,0x0, 0x40, 0x5,
0x69, 0x8d, 0x26 } };

//  Version:       1.5
const GUID ID_EQUIP_QUALIMETRE = { 0x80c4a764, 0x2816, 0x11d4, { 0x8a,0xb4,0x0, 0x40, 0x5,
0x69, 0x8d, 0x26 } };

//  Version:       1.5
const GUID ID_EQUIP_LEM_ANALYST3Q = { 0xd567cb71, 0xbcc0, 0x41ee, { 0x8e, 0x8c, 0x35, 0x85,
0x15, 0x53, 0xf6, 0x55 } };

//  Version:       1.5
const GUID ID_EQUIP_LEM_ANALYST1Q = { 0x477ecb3b, 0x917f, 0x4915, { 0xaf, 0x99, 0xa6, 0xc2,
0x9a, 0xc1, 0x87, 0x64 } };

//  Version:       1.5
const GUID ID_EQUIP_LEM_ANALYST2050 = { 0x9878ccab, 0xa842, 0x4cac, { 0x95, 0xf, 0x6d,
0x47, 0x21, 0x5b, 0xff, 0xdf } };

//  Version:       1.5
const GUID ID_EQUIP_LEM_ANALYST2060 = { 0x312471a2, 0xb586, 0x491c, { 0x85, 0x5a, 0xca,
0x5, 0x45, 0x9a, 0x7e, 0x20 } };

//  Version:       1.5
const GUID ID_EQUIP_LEM_MIDGET200 = { 0x8449f6b9, 0x10f4, 0x40a7, { 0xa1, 0xc3, 0xbe, 0x33,
0x8e, 0xb9, 0x74, 0x22 } };

//  Version:       1.5
const GUID ID_EQUIP_LEM_MBX300 = { 0xd4578d61, 0xdf2b, 0x4218, { 0xa7, 0xb1, 0x5e, 0xf1,
0xa9, 0xbb, 0x85, 0xfa } };

//  Version:       1.5
const GUID ID_EQUIP_LEM_MBX800 = { 0x1c14b57a, 0xba25, 0x47fb, { 0x88, 0xfa, 0x5f, 0xe5,
0xce, 0xc9, 0x9e, 0x6a } };

//  Version:       1.5
const GUID ID_EQUIP_LEM_MBX601 = { 0x1f3cda7b, 0x2ce1, 0x4030, { 0xa3, 0x90, 0xe3, 0xd4,
0x9c, 0x56, 0x15, 0xd2 } };

//  Version:       1.5
const GUID ID_EQUIP_LEM_MBX602 = { 0x4a157756, 0x414a, 0x427b, { 0x99, 0x32, 0x55, 0x76,
```

```
0xe, 0xd5, 0xf7, 0x7 } };

//  Version:       1.5
const GUID ID_EQUIP_LEM_MBX603 = { 0xf7b4677b, 0xb277, 0x45b5, { 0xaa, 0xae, 0x5f, 0xb3,
0x93, 0x41, 0xb3, 0x90 } };

//  Version:       1.5
const GUID ID_EQUIP_LEM_MBX686 = { 0x40004266, 0xa978, 0x4991, { 0x9e, 0xd6, 0xc1, 0xcd,
0x73, 0xf, 0x5b, 0xf5 } };

//  Version:       1.5
const GUID ID_EQUIP_LEM_PERMA701 = { 0x9b0dfd9d, 0xd4e9, 0x419d, { 0xba, 0x10, 0xc1, 0xce,
0xe6, 0xcf, 0x8f, 0x93 } };

//  Version:       1.5
const GUID ID_EQUIP_LEM_PERMA702 = { 0x7f5d62ac, 0x9fab, 0x400f, { 0xb5, 0x1a, 0xf0, 0xf3,
0x94, 0x1f, 0xb5, 0xaa } };

//  Version:       1.5
const GUID ID_EQUIP_LEM_PERMA705 = { 0xd85fea9c, 0x14d5, 0x45eb, { 0x83, 0x1f, 0xe0, 0x39,
0x73, 0x9, 0x2b, 0xd8 } };

//  Version:       1.5
const GUID ID_EQUIP_LEM_PERMA706 = { 0x16d6bbfc, 0xb5a, 0x4cf0, { 0x81, 0xcf, 0x48, 0xa3,
0x10, 0x5e, 0xff, 0x4f } };

//  Version:       1.5
const GUID ID_EQUIP_LEM_QWAVEMICRO = { 0xe0380e52, 0xc205, 0x43a0, { 0x9f, 0xf4, 0x76,
0xfb, 0xd6, 0x76, 0x5f, 0x37 } };

//  Version:       1.5
const GUID ID_EQUIP_LEM_QWAVENOMAD = { 0x165f145d, 0x90c3, 0x4591, { 0x95, 0x9a, 0x33,
0xb1, 0x1, 0xd4, 0xbf, 0x8b } };

//  Version:       1.5
const GUID ID_EQUIP_LEM_QWAVELIGHT = { 0x5198ceb9, 0x4b4e, 0x439c, { 0xa1, 0xc0, 0x21,
0x8c, 0x96, 0x3d, 0x6a, 0x9c } };

//  Version:       1.5
const GUID ID_EQUIP_LEM_QWAVETWIN = { 0x67a42a2d, 0xb831, 0x4222, { 0x80, 0x5e, 0xd5, 0xfd,
0xeb, 0xdd, 0x3a, 0x46 } };

//  Version:       1.5
const GUID ID_EQUIP_LEM_QWAVEPOWER_DISTRIBUTION = { 0x2401bf48, 0x9db2, 0x46ec, { 0xac,
0xde, 0x5d, 0xed, 0xde, 0x25, 0xe5, 0x4e } };

//  Version:       1.5
const GUID ID_EQUIP_LEM_QWAVEPREMIUM = { 0x6b609a29, 0x4a64, 0x4d1c, { 0xa6, 0xe3, 0xca,
0xef, 0x94, 0xfa, 0x56, 0xa0 } };

//  Version:       1.5
const GUID ID_EQUIP_LEM_QWAVEPOWER_TRANSPORT = { 0xd4422eeb, 0xb1cd, 0x4ba9, { 0xa7, 0xc8,
0x5d, 0x14, 0x1d, 0xf4, 0x5, 0x18 } };

//  Version:       1.5
const GUID ID_EQUIP_LEM_TOPASLT = { 0x9c46483a, 0x541e, 0x4d66, { 0x9c, 0x10, 0xf9, 0x43,
0xab, 0xfc, 0x34, 0x8a } };
```



```
//  Version:       1.5
const GUID ID_EQUIP_LEM_TOPAS1000 = { 0x459b8614, 0x6724, 0x48fb, { 0xb5, 0xd4, 0xf1, 0x49,
0xed, 0xc, 0x62, 0xf5 } };

//  Version:       1.5
const GUID ID_EQUIP_LEM_TOPAS1019 = { 0x7b11408b, 0x9d2c, 0x407c, { 0x84, 0xa5, 0x89, 0x44,
0x2, 0x18, 0xdc, 0xf8 } };

//  Version:       1.5
const GUID ID_EQUIP_LEM_TOPAS1020 = { 0xd1def77d, 0x990f, 0x484e, { 0xa1, 0x66, 0xf7, 0x92,
0x11, 0x70, 0xa6, 0x4b } };

//  Version:       1.5
const GUID ID_EQUIP_LEM_TOPAS1040 = { 0xd3cc1de2, 0x6e6b, 0x4b6e, { 0xad, 0x90, 0x10, 0xd6,
0x58, 0x5f, 0x8f, 0xa2 } };

//  Version:       1.5
const GUID ID_EQUIP_LEM_BEN5000 = { 0xa70e32b1, 0x2f1a, 0x4543, { 0xa6, 0x84, 0x78, 0xa4,
0xb5, 0xbe, 0x34, 0xbb } };

//  Version:       1.5
const GUID ID_EQUIP_LEM_BEN6000 = { 0x5a4c1b5, 0x6681, 0x47e6, { 0x9f, 0x64, 0x8d, 0xa1,
0x25, 0xdb, 0xec, 0x32 } };

//  Version:       1.5
const GUID ID_EQUIP_LEM_EWAVE = { 0xe46981d5, 0x708d, 0x4822, { 0x97, 0xaa, 0xfd, 0xb6,
0xf7, 0x3b, 0x3a, 0xf2 } };

//  Version:       1.5
const GUID ID_EQUIP_LEM_EWON4000 = { 0xd4c0895c, 0xfd48, 0x4981, { 0x99, 0x7c, 0x9e, 0x70,
0xd8, 0xe, 0xfb, 0x6 } };

//  Version:       1.5
const GUID ID_EQUIP_WPT_5510 = { 0x752871de, 0x583, 0x4d44, { 0xa9, 0xae, 0xc5, 0xfa, 0xdc,
0x1, 0x44, 0xac } };

//  Version:       1.5
const GUID ID_EQUIP_WPT_5520 = { 0xb72d289, 0x7645, 0x40b8, { 0x94, 0x6e, 0xc3, 0xce, 0x4f,
0x1b, 0xcd, 0x37 } };

//  Version:       1.5
const GUID ID_EQUIP_WPT_5530T = { 0x8f88ea9e, 0x1007, 0x4569, { 0xab, 0x47, 0x75, 0x6f,
0x29, 0x2a, 0x23, 0xed } };

//  Version:       1.5
const GUID ID_EQUIP_WPT_5560 = { 0x5fd9c0ff, 0x4432, 0x41b5, { 0x9a, 0x9e, 0x9a, 0x32,
0xba, 0x2c, 0xf0, 0x5 } };

//  Version:       1.5
const GUID ID_EQUIP_WPT_5590 = { 0x2861d5ca, 0x23ac, 0x4a51, { 0xa5, 0xa0, 0x49, 0x8d,
0xa6, 0x1d, 0x26, 0xdd } };

//  Version:       1.5
const GUID ID_EQUIP_ETK_NODECENTER = { 0xc52e8460, 0x58b4, 0x4f1a, { 0x84, 0x69, 0x69,
0xca, 0x3f, 0xef, 0x9f, 0xf1 } };

//  Version:       1.5
const GUID ID_EQUIP_WPT_DRANVIEW = { 0x8d97aa1, 0x1719, 0x11d6, { 0xa4, 0xb3, 0x44, 0x45,
```

```
0x53, 0x54, 0x0, 0x0 } };

//  Version:       1.5
const GUID ID_EQUIP_ADVANTECH_ADAM5017 = { 0x2f46263c, 0x92ac, 0x4717, { 0x8a, 0x8, 0xa6,
0x17, 0x7d, 0xf3, 0xf6, 0x11 } };

//  Version:       1.5
const GUID ID_EQUIP_ADVANTECH_ADAM5018 = { 0xcc2d3247, 0xfe65, 0x4db6, { 0x82, 0x6, 0x50,
0xa, 0x23, 0x15, 0x1b, 0xb2 } };

//  Version:       1.5
const GUID ID_EQUIP_ADVANTECH_ADAM5080 = { 0x6c37b63c, 0xe770, 0x4b85, { 0xbd, 0x32, 0x47,
0x39, 0xd6, 0xeb, 0x98, 0x46 } };

//  Version:       1.5
const GUID ID_EQUIP_ADVANTECH_ADAM5052 = { 0xe9261dfe, 0x3d44, 0x47e3, { 0xac, 0x36, 0x3b,
0x9, 0x7f, 0xaa, 0x8c, 0xda } };

//  Version:       1.5
const GUID ID_EQUIP_ADVANTECH_ADAM4050 = { 0x9212066d, 0xea65, 0x477e, { 0xbf, 0x95, 0xe4,
0xa0, 0x6, 0x6d, 0x25, 0xce } };

//  Version:       1.5
const GUID ID_EQUIP_ADVANTECH_ADAM4053 = { 0xdc29b83f, 0xbebe, 0x4cf3, { 0xb3, 0xfb, 0x0,
0xdc, 0x63, 0x62, 0x6d, 0xd9 } };

//  Version:       1.5
const GUID ID_EQUIP_ADVANTECH_ADAM4080 = { 0x64fc42c6, 0x3c90, 0x4633, { 0x99, 0xdf, 0x2c,
0x60, 0x58, 0x21, 0x4b, 0x72 } };

//  Version:       1.5
const GUID ID_EQUIP_ADVANTECH_ADAM5050 = { 0xc950a2e3, 0x7a35, 0x440c, { 0x86, 0x60, 0x63,
0xf6, 0x11, 0x97, 0x25, 0x19 } };

//  Version:       1.5
const GUID ID_EQUIP_ADVANTECH_ADAM5051 = { 0xc8f92334, 0xa69b, 0x4856, { 0xb2, 0x53, 0xec,
0x24, 0x71, 0xd1, 0x37, 0xd6 } };

//  ============================================================
//  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

//  Description:   The value representing average of 3 line-neutral values
//  Version:       1.5
#define ID_PHASE_LN_AVE 14

//  Description:   The value representing average of 3 line-line values
//  Version:       1.5
#define ID_PHASE_LL_AVE 15

//  Description:   The value representing the "worst" of the 3 phases
//  Version:       1.5
#define ID_PHASE_WORST 16

//  Description:   DC Positive
//  Version:       1.5
#define ID_PHASE_PLUS 17

//  Description:   DC Negative
//  Version:       1.5
#define ID_PHASE_MINUS 18

//  Description:   Generic Phase 1
//  Version:       1.5
#define ID_PHASE_GENERAL_1 19

//  Description:   Generic Phase 2
//  Version:       1.5
#define ID_PHASE_GENERAL_2 20

//  Description:   Generic Phase 3
//  Version:       1.5
#define ID_PHASE_GENERAL_3 21

//  Description:   Generic Phase 4
//  Version:       1.5
#define ID_PHASE_GENERAL_4 22

//  Description:   Generic Phase 5
//  Version:       1.5
#define ID_PHASE_GENERAL_5 23

//  Description:   Generic Phase 6
//  Version:       1.5
#define ID_PHASE_GENERAL_6 24

//  Description:   Generic Phase 7
//  Version:       1.5
#define ID_PHASE_GENERAL_7 25

//  Description:   Generic Phase 8
//  Version:       1.5
#define ID_PHASE_GENERAL_8 26

//  Description:   Generic Phase 9
//  Version:       1.5
#define ID_PHASE_GENERAL_9 27
```



```
//  Description:   Generic Phase 10
//  Version:       1.5
#define ID_PHASE_GENERAL_10 28

//  Description:   Generic Phase 11
//  Version:       1.5
#define ID_PHASE_GENERAL_11 29

//  Description:   Generic Phase 12
//  Version:       1.5
#define ID_PHASE_GENERAL_12 30

//  Description:   Generic Phase 13
//  Version:       1.5
#define ID_PHASE_GENERAL_13 31

//  Description:   Generic Phase 14
//  Version:       1.5
#define ID_PHASE_GENERAL_14 32

//  Description:   Generic Phase 15
//  Version:       1.5
#define ID_PHASE_GENERAL_15 33

//  Description:   Generic Phase 16
//  Version:       1.5
#define ID_PHASE_GENERAL_16 34

//  ============================================================
//  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 accep-
tible 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

//  Description:   Meters/Second
//  Version:       1.5
#define ID_QU_MPS 25

//  Description:   Miles/Hr
//  Version:       1.5
#define ID_QU_MPH 26

//  Description:   Pressure, Bars
//  Version:       1.5
#define ID_QU_BARS 27

//  Description:   Pressure, Pascals
//  Version:       1.5
#define ID_QU_PASCALS 28

//  Description:   Force, Newtons
//  Version:       1.5
#define ID_QU_NEWTONS 29

//  Description:   Torque, Newton Meters
//  Version:       1.5
#define ID_QU_NEWTONMETERS 30

//  Description:   Revolutions/minute
//  Version:       1.5
```

```
#define ID_QU_RPM 31

//  Description:   Radians/Second
//  Version:       1.5
#define ID_QU_RADPERSEC 32

//  Description:   Meters
//  Version:       1.5
#define ID_QU_METERS 33

//  Description:   Flux Linkage - Weber Turns
//  Version:       1.5
#define ID_QU_WEBERTURNS 34

//  Description:   Flux Density - Teslas
//  Version:       1.5
#define ID_QU_TESLAS 35

//  Description:   Magnetic Field - Webers
//  Version:       1.5
#define ID_QU_WEBERS 36

//  Description:   Volts/Volts transfer function
//  Version:       1.5
#define ID_QU_VOLTSPERVOLT 37

//  Description:   Amps/Amps transfer function
//  Version:       1.5
#define ID_QU_AMPSPERAMP 38

//  Description:   Impedance Transfer Funtion
//  Version:       1.5
#define ID_QU_AMPSPERVOLT 39

//  ============================================================
//  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:   Area under the signal, usually an rms voltage, current or other
quantity.
//  Version:       1.5
const GUID ID_SERIES_VALUE_TYPE_AREA = { 0xc7825ce0, 0x8ace, 0x11d3, { 0xb9, 0x2f, 0x0,
0x50, 0xda, 0x2b, 0x1f, 0x4d } };

//  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:   Cumulative probability in percent.
//  Version:       1.5
const GUID ID_SERIES_VALUE_TYPE_PROB = { 0x6763cc71, 0x17d6, 0x11d4, { 0x9f, 0x1c, 0x0,
0x20, 0x78, 0xe0, 0xb7, 0x23 } };

//  Description:   Interval data
//  Version:       1.5
const GUID ID_SERIES_VALUE_TYPE_INTERVAL = { 0x72e82a40, 0x336c, 0x11d5, { 0xa4, 0xb3,
0x44, 0x45, 0x53, 0x54, 0x0, 0x0 } };

//  Description:   Status Data
//  Version:       1.5
const GUID ID_SERIES_VALUE_TYPE_STATUS = { 0xb82b5c82, 0x55c7, 0x11d5, { 0xa4, 0xb3, 0x44,
0x45, 0x53, 0x54, 0x0, 0x0 } };
```



```
//  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 con-
```

```
tains: #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

//  Description:   Periodic Statistical Data
//  Version:       1.5
#define ID_TRIGGER_METH_PERIODIC_STATS 4

//  ============================================================
//  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:   Negative sequence component
//  Version:       1.5
const GUID ID_QC_SNEG = { 0xd71a4b91, 0x3c92, 0x11d4, { 0x9f, 0x2c, 0x0, 0x20, 0x78, 0xe0,
0xb7, 0x23 } };

//  Description:   Zero sequence component
//  Version:       1.5
const GUID ID_QC_SZERO = { 0xd71a4b92, 0x3c92, 0x11d4, { 0x9f, 0x2c, 0x0, 0x20, 0x78, 0xe0,
0xb7, 0x23 } };

//  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:   Total THD normalized to RMS
//  Version:       1.5
const GUID ID_QC_TOTAL_THD_RMS = { 0xf3d216e0, 0x2aa5, 0x11d5, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  Description:   Odd THD normalized to RMS
//  Version:       1.5
const GUID ID_QC_ODD_THD_RMS = { 0xf3d216e1, 0x2aa5, 0x11d5, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  Description:   Even THD normalized to RMS
//  Version:       1.5
const GUID ID_QC_EVEN_THD_RMS = { 0xf3d216e2, 0x2aa5, 0x11d5, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };;

//  Description:   Total Interharmonic Distortion
//  Version:       1.5
const GUID ID_QC_TID = { 0xf3d216e3, 0x2aa5, 0x11d5, { 0xa4, 0xb3, 0x44, 0x45, 0x53, 0x54,
0x0, 0x0 } };

//  Description:   Total Interharmonic Distortion Normalized to RMS
//  Version:       1.5
const GUID ID_QC_TID_RMS = { 0xf3d216e4, 0x2aa5, 0x11d5, { 0xa4, 0xb3, 0x44, 0x45, 0x53,
0x54, 0x0, 0x0 } };

//  Description:    Interharmonic RMS
//  Version:       1.5
const GUID ID_QC_IHRMS = { 0xf3d216e5, 0x2aa5, 0x11d5, { 0xa4, 0xb3, 0x44, 0x45, 0x53,
0x54, 0x0, 0x0 } };

//  Description:   Spectra by Harmonic Group index
//  Version:       1.5
const GUID ID_QC_SPECTRA_HGROUP = { 0x53be6ba8, 0x789, 0x455b, { 0x9a, 0x95, 0xda, 0x12,
0x86, 0x83, 0xdd, 0xa7 } };

//  Description:   Spectra by Interharmonic Group Index
//  Version:       1.5
const GUID ID_QC_SPECTRA_IGROUP = { 0x5e51e006, 0x9c95, 0x4c5e, { 0x87, 0x8f, 0x7c, 0xa8,
0x7c, 0xd, 0x2a, 0xe } };

//  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_DVV = { 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:   Short Term Flicker
//  Version:       1.5
const GUID ID_QC_FLKR_PST = { 0x515bf320, 0x71ca, 0x11d4, { 0xa4, 0xb3, 0x44, 0x45, 0x53,
0x54, 0x0, 0x0 } };

//  Description:   Long Term Flicker
//  Version:       1.5
const GUID ID_QC_FLKR_PLT = { 0x515bf321, 0x71ca, 0x11d4, { 0xa4, 0xb3, 0x44, 0x45, 0x53,
0x54, 0x0, 0x0 } };

//  Description:   TIF normalized to RMS
//  Version:       1.5
const GUID ID_QC_TIF_RMS = { 0xf3d216e6, 0x2aa5, 0x11d5, { 0xa4, 0xb3, 0x44, 0x45, 0x53,
0x54, 0x0, 0x0 } };

//  Description:   Sliding PLT
//  Version:       1.5
const GUID ID_QC_FLKR_PLTSLIDE = { 0x2257ec05, 0x6ea, 0x4709, { 0xb4, 0x3a, 0xc, 0x0, 0x53,
0x4d, 0x55, 0x4a } };

//  Version:       1.5
const GUID ID_QC_FLKR_PILPF = { 0x4d693eec, 0x5d1d, 0x4531, { 0x99, 0x3a, 0x79, 0x3b, 0x53,
0x56, 0xc6, 0x3d } };

//  Version:       1.5
const GUID ID_QC_FLKR_PIMAX = { 0x126de61c, 0x6691, 0x4d16, { 0x8f, 0xdf, 0x46, 0x48, 0x2b,
0xca, 0x46, 0x94 } };

//  Version:       1.5
const GUID ID_QC_FLKR_PIROOT = { 0xe065b621, 0xffdb, 0x4598, { 0x93, 0x30, 0x4d, 0x9, 0x35,
0x39, 0x88, 0xb6 } };;

//  Version:       1.5
const GUID ID_QC_FLKR_PIROOTLPF = { 0x7d11f283, 0x1ce7, 0x4e58, { 0x8a, 0xf0, 0x79, 0x4,
0x87, 0x93, 0xb8, 0xa7 } };

//  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:   Transformer Derating Factor
//  Version:       1.5
const GUID ID_QC_ANSI_TDF = { 0x8786ca10, 0x9113, 0x11d3, { 0xb9, 0x30, 0x0, 0x50, 0xda,
0x2b, 0x1f, 0x4d } };

//  Description:   Transformer K Factor
//  Version:       1.5
const GUID ID_QC_K_FACTOR = { 0x8786ca11, 0x9113, 0x11d3, { 0xb9, 0x30, 0x0, 0x50, 0xda,
0x2b, 0x1f, 0x4d } };

//  Description:   Total Demand Distortion
//  Version:       1.5
const GUID ID_QC_TDD = { 0xf3d216e7, 0x2aa5, 0x11d5, { 0xa4, 0xb3, 0x44, 0x45, 0x53, 0x54,
0x0, 0x0 } };

//  Description:   Peak Demand Current
//  Version:       1.5
const GUID ID_QC_RMS_PEAK_DEMAND = { 0x72e82a44, 0x336c, 0x11d5, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  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:   Predicted value of active power for current demand interval
//  Version:       1.5
const GUID ID_QC_P_PRED_DEMAND = { 0x672d0305, 0x7810, 0x11d4, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  Description:   Predicted value of reactive power for current demand interval
//  Version:       1.5
const GUID ID_QC_Q_PRED_DEMAND = { 0x672d0306, 0x7810, 0x11d4, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  Description:   Predicted value of apparent power for current demand interval
//  Version:       1.5
const GUID ID_QC_S_PRED_DEMAND = { 0x672d0307, 0x7810, 0x11d4, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  Description:   Value of active power coincident with reactive power demand
//  Version:       1.5
const GUID ID_QC_P_CO_Q_DEMAND = { 0x672d030a, 0x7810, 0x11d4, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  Description:   Value of active power coincident with apparent power demand
//  Version:       1.5
const GUID ID_QC_P_CO_S_DEMAND = { 0x672d030b, 0x7810, 0x11d4, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  Description:   Value of reactive power coincident with active power demand
//  Version:       1.5
const GUID ID_QC_Q_CO_P_DEMAND = { 0x672d030d, 0x7810, 0x11d4, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  Description:   Value of reactive power coincident with apparent power demand
//  Version:       1.5
const GUID ID_QC_Q_CO_S_DEMAND = { 0x672d030e, 0x7810, 0x11d4, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  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 } };

//  Description:   Value of true power factor coincident with active power demand
//  Version:       1.5
const GUID ID_QC_PF_CO_P_DEMAND = { 0x672d0308, 0x7810, 0x11d4, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  Description:   Value of true power factor coincident with reactive power demand
//  Version:       1.5
const GUID ID_QC_PF_CO_Q_DEMAND = { 0x672d0309, 0x7810, 0x11d4, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  Description:   Value of the power angle at fundamental
//  Version:       1.5
const GUID ID_QC_ANGLE_FUND = { 0x672d030f, 0x7810, 0x11d4, { 0xa4, 0xb3, 0x44, 0x45, 0x53,
0x54, 0x0, 0x0 } };

//  Description:   Value of the reactive power at fundamental frequency
//  Version:       1.5
const GUID ID_QC_Q_FUND = { 0x672d0310, 0x7810, 0x11d4, { 0xa4, 0xb3, 0x44, 0x45, 0x53,
0x54, 0x0, 0x0 } };

//  Description:   True Power Factor - IEEE vector calculations
//  Version:       1.5
const GUID ID_QC_PF_VECTOR = { 0x672d0311, 0x7810, 0x11d4, { 0xa4, 0xb3, 0x44, 0x45, 0x53,
0x54, 0x0, 0x0 } };

//  Description:   Displacement Factor - IEEE vector calculations
//  Version:       1.5
const GUID ID_QC_DF_VECTOR = { 0x672d0312, 0x7810, 0x11d4, { 0xa4, 0xb3, 0x44, 0x45, 0x53,
0x54, 0x0, 0x0 } };

//  Description:   Value of apparent power - IEEE vector calculations
//  Version:       1.5
const GUID ID_QC_S_VECTOR = { 0x672d0314, 0x7810, 0x11d4, { 0xa4, 0xb3, 0x44, 0x45, 0x53,
0x54, 0x0, 0x0 } };

//  Description:   Value of fundamental frequency apparent power - IEEE vector calculations
//  Version:       1.5
const GUID ID_QC_S_VECTOR_FUND = { 0x672d0315, 0x7810, 0x11d4, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  Description:   Value of fundamental frequency apparent power
//  Version:       1.5
const GUID ID_QC_S_FUND = { 0x672d0316, 0x7810, 0x11d4, { 0xa4, 0xb3, 0x44, 0x45, 0x53,
0x54, 0x0, 0x0 } };

//  Description:   Apparent power coincident with active power demand
//  Version:       1.5
const GUID ID_QC_S_CO_P_DEMAND = { 0x672d0317, 0x7810, 0x11d4, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };
```



```
//  Description:   Apparent power coincident with reactive power demand
//  Version:       1.5
const GUID ID_QC_S_CO_Q_DEMAND = { 0x672d0318, 0x7810, 0x11d4, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  Description:   True Power Factor - IEEE arithmetic calculations
//  Version:       1.5
const GUID ID_QC_PF_ARITH = { 0x1c39fb00, 0xa6aa, 0x11d4, { 0xa4, 0xb3, 0x44, 0x45, 0x53,
0x54, 0x0, 0x0 } };

//  Description:   Displacement Factor - IEEE Arithmetic calculations
//  Version:       1.5
const GUID ID_QC_DF_ARITH = { 0x1c39fb01, 0xa6aa, 0x11d4, { 0xa4, 0xb3, 0x44, 0x45, 0x53,
0x54, 0x0, 0x0 } };

//  Description:   Value of apparent power - IEEE Arithmetic calculations
//  Version:       1.5
const GUID ID_QC_S_ARITH = { 0x1c39fb02, 0xa6aa, 0x11d4, { 0xa4, 0xb3, 0x44, 0x45, 0x53,
0x54, 0x0, 0x0 } };

//  Description:   Value of fundamental frequency apparent power - IEEE Arithmetic calcula-
tions
//  Version:       1.5
const GUID ID_QC_S_ARITH_FUND = { 0x1c39fb03, 0xa6aa, 0x11d4, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  Description:   Peak Apparent Power Demand
//  Version:       1.5
const GUID ID_QC_S_PEAK_DEMAND = { 0x72e82a43, 0x336c, 0x11d5, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  Description:   Peak Reactive Power Demand
//  Version:       1.5
const GUID ID_QC_Q_PEAK_DEMAND = { 0x72e82a42, 0x336c, 0x11d5, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  Description:   Peak Active Power Demand
//  Version:       1.5
const GUID ID_QC_P_PEAK_DEMAND = { 0x72e82a41, 0x336c, 0x11d5, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  Description:   Net Harmonic Actve Power
//  Version:       1.5
const GUID ID_QC_P_HARMONIC = { 0xb82b5c80, 0x55c7, 0x11d5, { 0xa4, 0xb3, 0x44, 0x45, 0x53,
0x54, 0x0, 0x0 } };

//  Description:   Arithmetic sum Harmonic Active Power
//  Version:       1.5
const GUID ID_QC_P_HARMONIC_UNSIGNED = { 0xb82b5c81, 0x55c7, 0x11d5, { 0xa4, 0xb3, 0x44,
0x45, 0x53, 0x54, 0x0, 0x0 } };

//  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 fund. Freq. power integrated over time (Energy - watt-
hours) in the positive direction (toward load).
//  Version:       1.5
const GUID ID_QC_P_INTG_POS_FUND = { 0x672d0300, 0x7810, 0x11d4, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  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 active fund. Freq. power integrated over time (Energy - watt-
hours) in the negative direction (away from load).
//  Version:       1.5
const GUID ID_QC_P_INTG_NEG_FUND = { 0x672d0301, 0x7810, 0x11d4, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  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 fund. Freq. reactive power integrated over time (Energy - watt-
hours) in the positive direction (toward load).
//  Version:       1.5
const GUID ID_QC_Q_INTG_POS_FUND = { 0x672d0303, 0x7810, 0x11d4, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  Description:   Value of fund. Freq. 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_FUND = { 0x672d0304, 0x7810, 0x11d4, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  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 fundamental frequency apparent power integrated over time (VA-
hours)
```



```
//  Version:       1.5
const GUID ID_QC_S_INTG_FUND = { 0x672d0313, 0x7810, 0x11d4, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  Description:   Value of active power integrated over time (Energy - watt-hours)
//  Version:       1.5
const GUID ID_QC_P_IVL_INTG = { 0xf098a9a0, 0x3ee4, 0x11d5, { 0xa4, 0xb3, 0x44, 0x45, 0x53,
0x54, 0x0, 0x0 } };

//  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_IVL_INTG_POS = { 0xf098a9a1, 0x3ee4, 0x11d5, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  Description:   Value of active fund. Freq. power integrated over time (Energy - watt-
hours) in the positive direction (toward load).
//  Version:       1.5
const GUID ID_QC_P_IVL_INTG_POS_FUND = { 0xf098a9a2, 0x3ee4, 0x11d5, { 0xa4, 0xb3, 0x44,
0x45, 0x53, 0x54, 0x0, 0x0 } };

//  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_IVL_INTG_NEG = { 0xf098a9a3, 0x3ee4, 0x11d5, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  Description:   Value of active fund. Freq. power integrated over time (Energy - watt-
hours) in the negative direction (away from load).
//  Version:       1.5
const GUID ID_QC_P_IVL_INTG_NEG_FUND = { 0xf098a9a4, 0x3ee4, 0x11d5, { 0xa4, 0xb3, 0x44,
0x45, 0x53, 0x54, 0x0, 0x0 } };

//  Description:   Value of reactive power integrated over time (var-hours)
//  Version:       1.5
const GUID ID_QC_Q_IVL_INTG = { 0xf098a9a5, 0x3ee4, 0x11d5, { 0xa4, 0xb3, 0x44, 0x45, 0x53,
0x54, 0x0, 0x0 } };

//  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_IVL_INTG_POS = { 0xf098a9a6, 0x3ee4, 0x11d5, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  Description:   Value of fund. Freq. reactive power integrated over time (Energy - watt-
hours) in the positive direction (toward load).
//  Version:       1.5
const GUID ID_QC_Q_IVL_INTG_POS_FUND = { 0xf098a9a7, 0x3ee4, 0x11d5, { 0xa4, 0xb3, 0x44,
0x45, 0x53, 0x54, 0x0, 0x0 } };

//  Description:   Value of fund. Freq. reactive power integrated over time (Energy - watt-
hours) in the negative direction (away from load).
//  Version:       1.5
const GUID ID_QC_Q_IVL_INTG_NEG_FUND = { 0xf098a9a8, 0x3ee4, 0x11d5, { 0xa4, 0xb3, 0x44,
0x45, 0x53, 0x54, 0x0, 0x0 } };

//  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_IVL_INTG_NEG = { 0xf098a9a9, 0x3ee4, 0x11d5, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  Description:   Value of apparent power integrated over time (VA-hours)
//  Version:       1.5
const GUID ID_QC_S_IVL_INTG = { 0xf098a9aa, 0x3ee4, 0x11d5, { 0xa4, 0xb3, 0x44, 0x45, 0x53,
0x54, 0x0, 0x0 } };

//  Description:   Value of fundamental frequency apparent power integrated over time (VA-
hours)
//  Version:       1.5
const GUID ID_QC_S_IVL_INTG_FUND = { 0xf098a9ab, 0x3ee4, 0x11d5, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  Description:   D Axis Components
//  Version:       1.5
const GUID ID_QC_DAXISFIELD = { 0xd347ba65, 0xe34c, 0x11d4, { 0x82, 0xd9, 0x0, 0xe0, 0x98,
0x72, 0xa0, 0x94 } };

//  Description:   Q Axis Components
//  Version:       1.5
const GUID ID_QC_QAXIS = { 0xd347ba64, 0xe34c, 0x11d4, { 0x82, 0xd9, 0x0, 0xe0, 0x98, 0x72,
0xa0, 0x94 } };

//  Description:   Rotational Position
//  Version:       1.5
const GUID ID_QC_ROTATIONAL = { 0xd347ba62, 0xe34c, 0x11d4, { 0x82, 0xd9, 0x0, 0xe0, 0x98,
0x72, 0xa0, 0x94 } };

//  Description:   D Axis Components
//  Version:       1.5
const GUID ID_QC_DAXIS = { 0xd347ba63, 0xe34c, 0x11d4, { 0x82, 0xd9, 0x0, 0xe0, 0x98, 0x72,
0xa0, 0x94 } };

//  Description:   Linear Position
//  Version:       1.5
const GUID ID_QC_LINEAR = { 0xd347ba61, 0xe34c, 0x11d4, { 0x82, 0xd9, 0x0, 0xe0, 0x98,
0x72, 0xa0, 0x94 } };

//  Description:   Transfer function
//  Version:       1.5
const GUID ID_QC_TRANSFERFUNC = { 0x5202bd07, 0x245c, 0x11d5, { 0xa4, 0xb3, 0x44, 0x45,
0x53, 0x54, 0x0, 0x0 } };

//  Description:   Status Data
//  Version:       1.5
const GUID ID_QC_STATUS = { 0xb82b5c83, 0x55c7, 0x11d5, { 0xa4, 0xb3, 0x44, 0x45, 0x53,
0x54, 0x0, 0x0 } };

//  ============================================================
//  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 charac-
teristic 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

//  Description:   Velocity
//  Version:       1.5
#define ID_QM_VELOCITY 10

//  Description:   Compass Bearing
//  Version:       1.5
#define ID_QM_BEARING 11

//  Description:   Applied Force, Electrical, Mechanical, etc.
//  Version:       1.5
#define ID_QM_FORCE 12

//  Description:   Torque
//  Version:       1.5
#define ID_QM_TORQUE 13

//  Description:   Spacial Position
//  Version:       1.5
#define ID_QM_POSITION 14
```

```
//  Description:   Flux linkage Weber Turns
//  Version:       1.5
#define ID_QM_FLUXLINKAGE 15

//  Description:   Magnetic field Density
//  Version:       1.5
#define ID_QM_FLUXDENSITY 16

//  Description:   Status Data
//  Version:       1.5
#define ID_QM_STATUS 17

//  ============================================================
//  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

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

//  Description:   Single phase connection, 1 voltage, 1 current
//  Version:       1.5
#define ID_SINGLE_PHASE 1

//  Description:   Delta Connected 2 Element Monitoring
//  Version:       1.5
#define ID_2ELEMENT_DELTA 2

//  Description:   Wye 2 Voltages, 3 Currents
//  Version:       1.5
#define ID_2_5ELEMENT_WYE 3

//  Description:   3 Voltages, 3 Currents
//  Version:       1.5
#define ID_3ELMENT_WYE 4

//  Description:   Delta Connection, 3 Voltages, 3 currents
//  Version:       1.5
#define ID_3ELEMENT_DELTA 5

//  Description:   Split Single Phase, 2 Voltage, 2 Current
//  Version:       1.5
#define ID_SPLIT_PHASE 6

//  Description:   2 Phase, 2 Voltages, 2 Currents
//  Version:       1.5
#define ID_2ELEMENT_2PHASE 7


#endif  //  PQDIF_ID_H
```


Annex C

(informative)

File directory structure

This recommended practice does not require any specific directory structure or memory organization for holding, grouping, or otherwise containing a collection of PQDIF file or memory images. However, over the course of the evolution of PQDIF technology prior to and throughout the standardization process, recommendations on file structure and naming conventions have been made and applications written to accommodate those recommendations. This informative annex documents those conventions. These conventions are most useful when working with software that downloads/transfers multiple PQDIF files from multiple data sources onto a PC hard disk.

The downloaded data are stored in a collection of PQDIF files (sometimes called a PQDIF Pile of Files or PQDIF-POF) stored in a hierarchical file structure. The root folder where the hierarchy begins is totally arbitrary and specified by the user. A PQDIF visualization or analysis program would be pointed to this root directory.

Folders corresponding to individual data sources (individual instruments) are contained in the root PQDIF-POF folder (Figure C.1). The folders are created manually by the user or automatically by a download program upon discovery of new instruments.

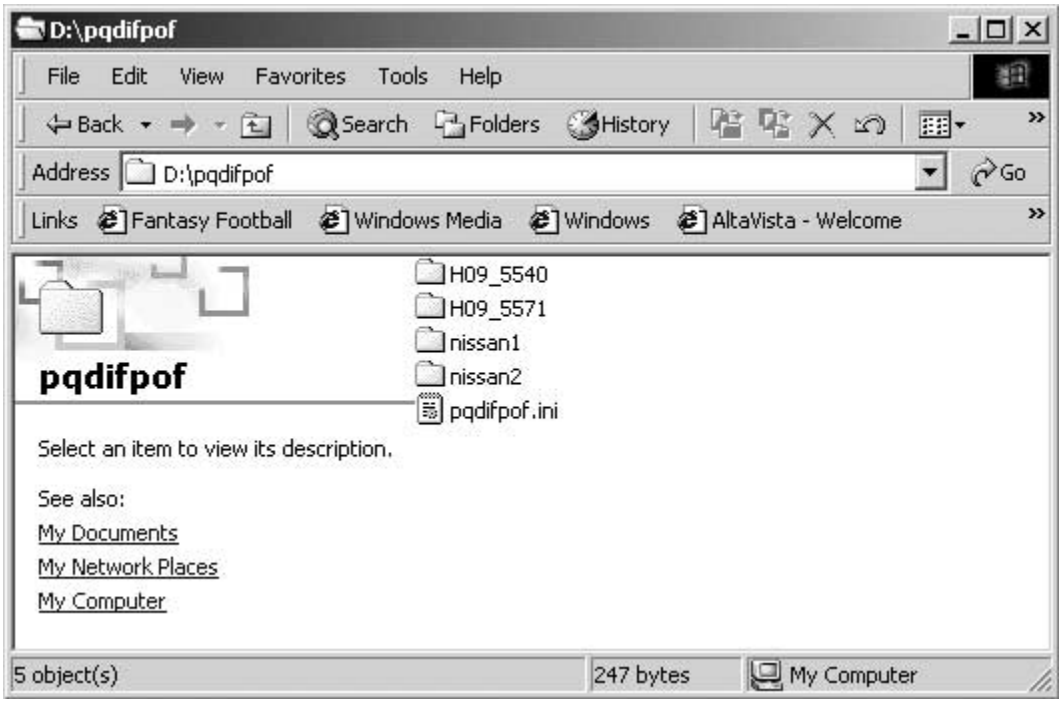

Figure C.1—PQDIF-POF root directory

Inside these folders are the actual PQDIF files containing downloaded data for the specific instrument (Figure C.2). The PQDIF files are named with a UTC timestamp corresponding to the download date and time, and they have a file extension of PQD (e.g., 19990812T015901.PQD). The format of the file name follows ISO standard 8601 for time stamps (with optional T separator between date and time parts to make it a bit more readable for humans). Each download of data generates a new PQDIF file.

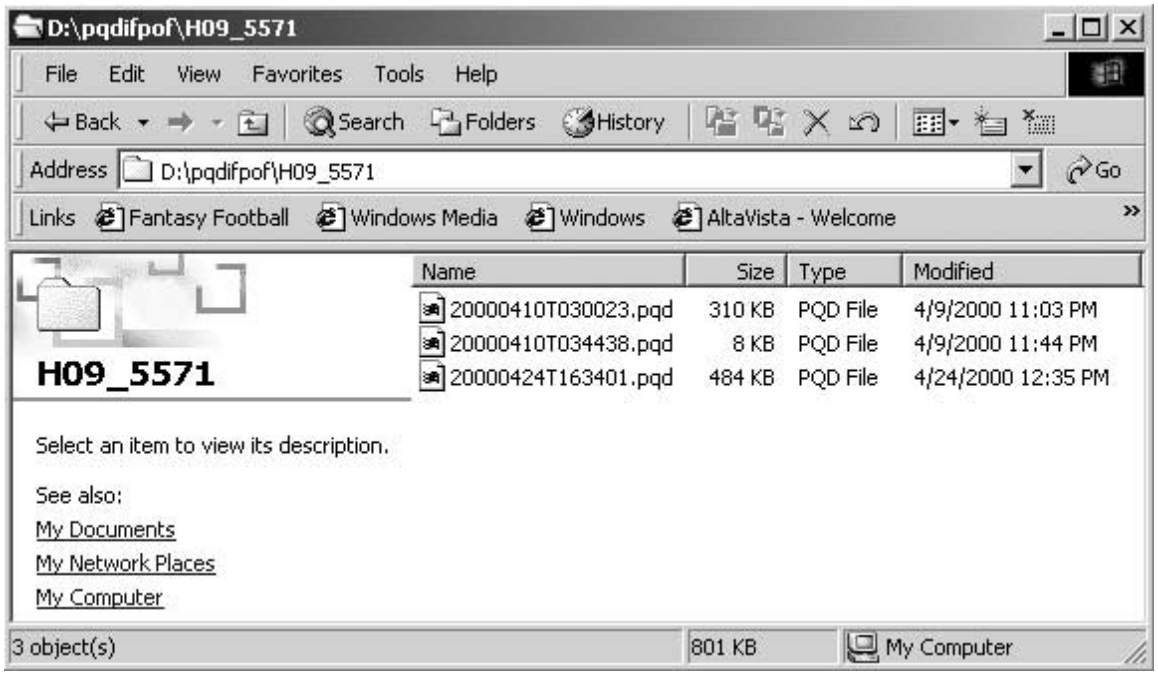

Figure C.2—Downloaded PQDIF files for specific instrument (H09_5571)

The root PQDIF-POF folder optionally contains a configuration settings file named PQDIFPOF.INI. This file can be created and maintained by the user and/or the downloader program or analysis software. The mere presence of this file indicates that the folder is the root folder of a PQDIF pile of files. The PQDIFPOF.INI file is an ASCII configuration settings file. It contains three sections maintained by downloader, visualization, and/or analysis programs.

All sections are completely optional (there are PQDIF analysis applications that use an empty PQDIFPOF.INI file simply to validate that this is a PQDIF-POF directory) and can be in any order. This .INI file name extension and syntax of its contents follow Microsoft's conventions for such configuration files such that common WIN32 programming APIs can be used to manage the file contents.

The first section, labeled DIRECTORYMAP, maps user-defined instrument names to the folders containing the PQDIF files. This is useful for download programs that autogenerate folder names for the downloaded instrument, but the user wants to specify a more user-friendly name for analysis program import purposes. This also permits renaming an instrument logically while leaving the folder name where the files reside unchanged.

The second section, labeled GATEWAYMAP, maps instrument names to a gateway name. This is often done by download programs to accommodate instrument gateways that manage multiple downloadable instruments. Analysis programs may need to maintain this information for reporting and display purposes.

The third section, labeled LASTIMPORT, contains the UTC timestamp (in ISO 8601 format without the optional T separator between the date and time parts) of the last PQDIF file imported into an application program (typically a database application used for long-term storage of the power quality data) that "owns" this section of the PQDIFPOF.INI file. Because the PQDIF files are named by the download program based on the UTC download time, a database application program can assume that any file with a name "greater than" this date-time has not been imported into the database.

C.1 Example PQDIFPOF.INI file

An example PQDIFPOF.INI file:

```
[DIRECTORYMAP]
5571 at Station B=ACKA
5540 at Service Entrance=BHAM
5571 at Service Entrance=BHAM
5571 at Commissary=CAFB
5571 at Alert Shack=CAFB

[GATEWAYMAP]
5571 at Station B=Ackerman 5501
5540 at Service Entrance=First 5502 at BHAM
5571 at Service Entrance=Second 5502 at BHAM
5571 at Commissary=Columbus AFB 5502
5571 at Alert Shack=Columbus AFB 5502

[LASTIMPORT]
5571 at Station B=19990816181920
5540 at Service Entrance=19990815070809
5571 at Service Entrance=19990816080910
5571 at Commissary=19990814171819
5571 at Alert Shack=19990810222324
```