

1159.3™

IEEE Recommended Practice for the Transfer of Power Quality Data

IEEE Standards Coordinating Committee 22

Sponsored by the
IEEE Standards Coordinating Committee 22
on Power Quality



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

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

Reaffirmed 20 January 2009

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

Copyright © 2004 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 12 January 2004. Printed in the United States of America.

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

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

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.

Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

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 its 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

1. Overview.....	1
1.1 Scope.....	1
1.2 Purpose.....	1
2. References.....	2
3. Definitions	2
4. Physical structure	2
4.1 Overall file structure	2
4.2 Internal record structure.....	4
5. Logical structure	5
5.1 Logical record structure	5
5.2 Logical element structure.....	7
5.3 Logical structure of container record.....	7
5.4 Logical structures of data sources and observations.....	9
5.5 How to put together a series	10
6. Compression	12
6.1 Compression algorithm—zlib.....	12
6.2 Record-level compression.....	12
7. Logical device modeling.....	13
7.1 General procedures	14
7.2 Representing a waveform recording	16
7.3 Representing an RMS variation recording.....	20
7.4 Representing periodically recorded steady state values	25
Annex A (normative) Physical format definitions	31
Annex B (normative) Logical structure documentation.....	39
Annex C (informative) File directory structure	117

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 1159TM-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.

¹IEEE 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/>).

²RFC 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 .

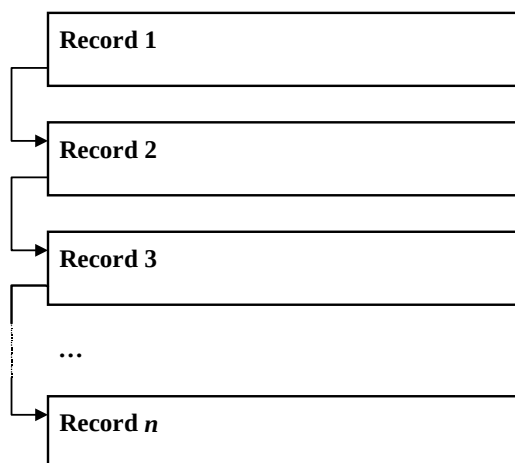


Figure 1—Standard record linkages

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

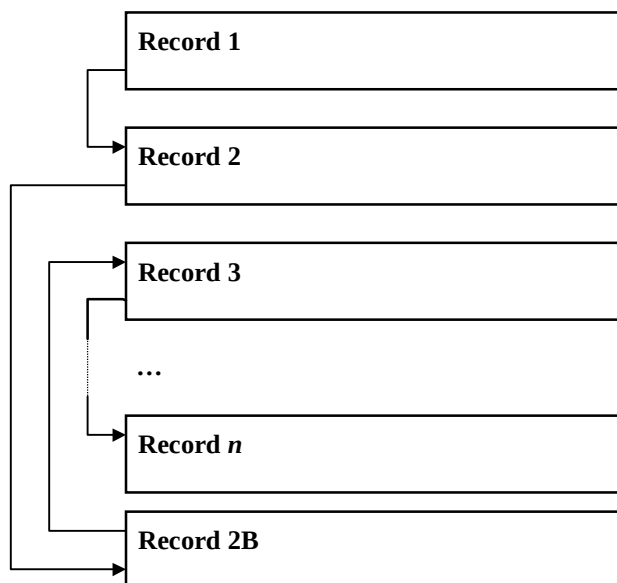


Figure 2—Inserting a new record

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

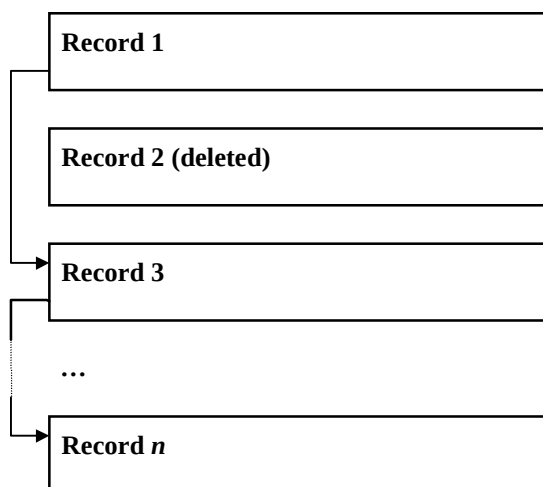


Figure 3—Deleting a record

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

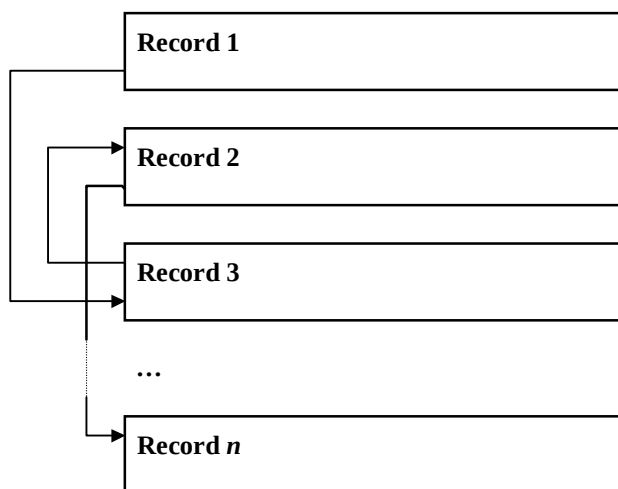


Figure 4—Reordering records

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.

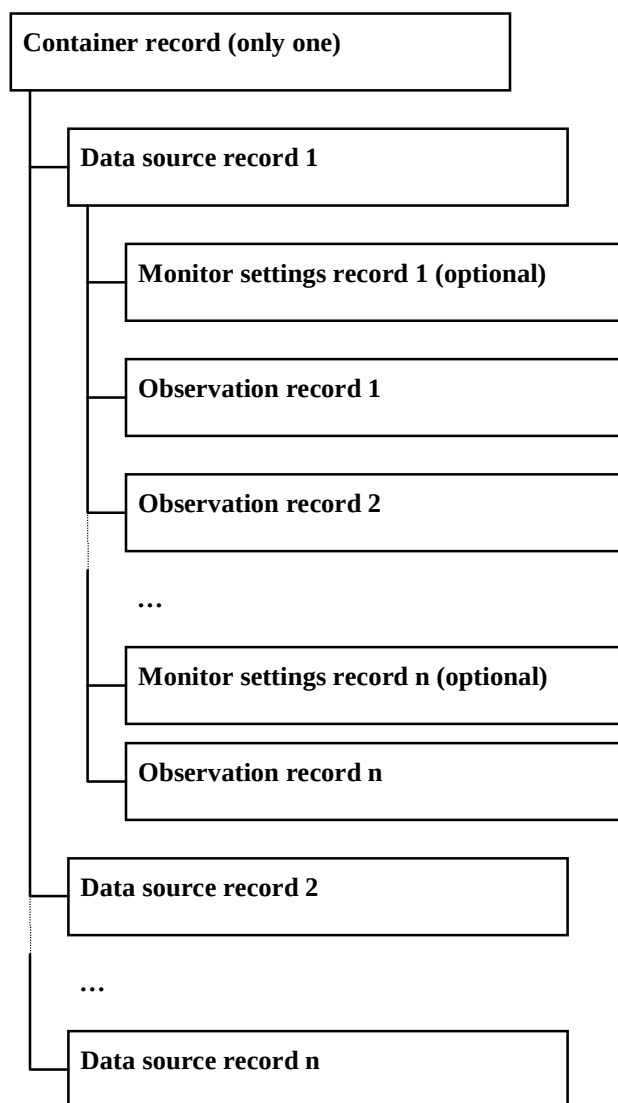


Figure 5—Logical record structure

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.

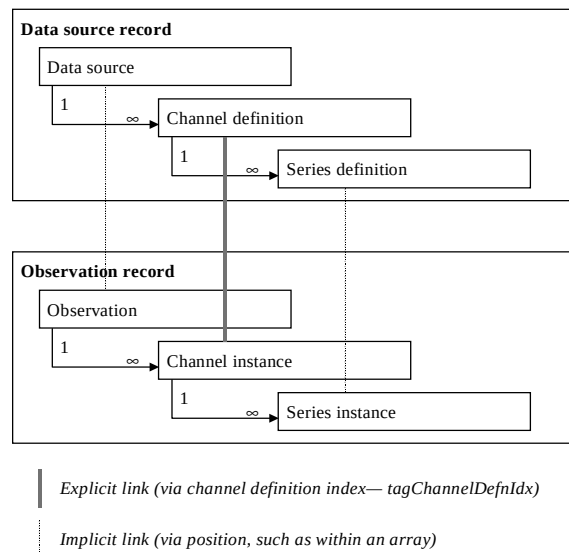
5.3.2 Compression

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.

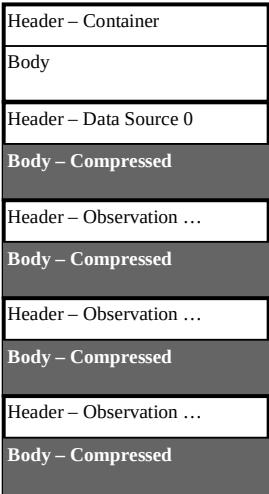


Figure 7—Compressed record layout

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 1159-1995³, 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.

³Information 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.

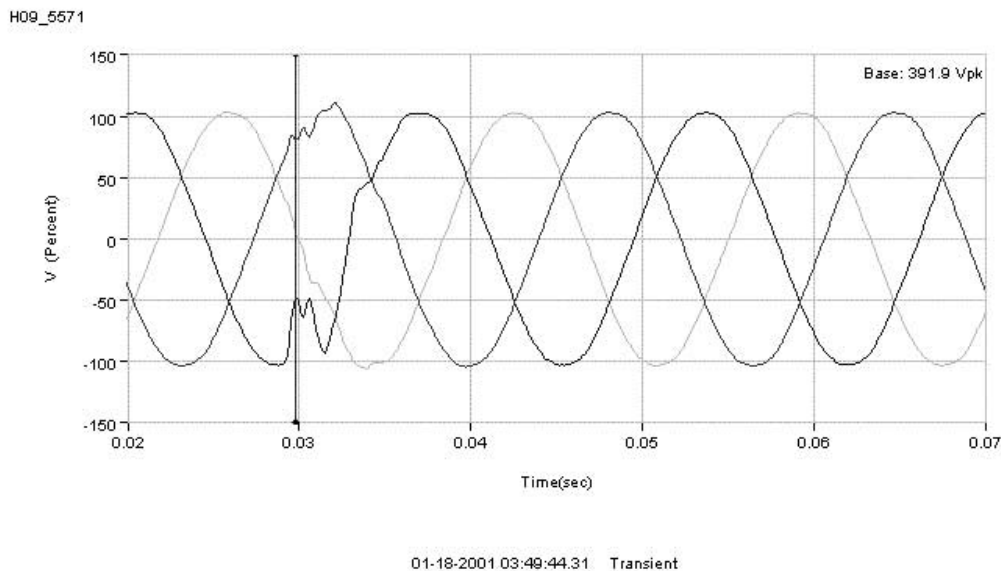


Figure 8—Low-frequency transient (capacitor switching)

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.

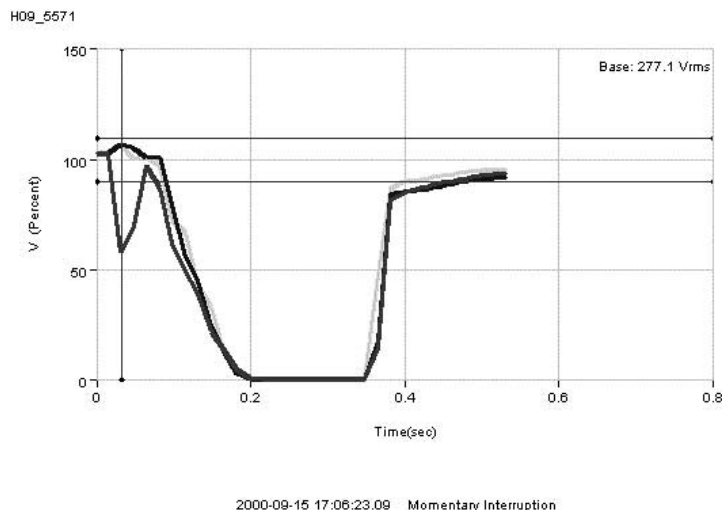


Figure 9—RMS variation showing all three phases and trigger lines

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)
|   +- (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.

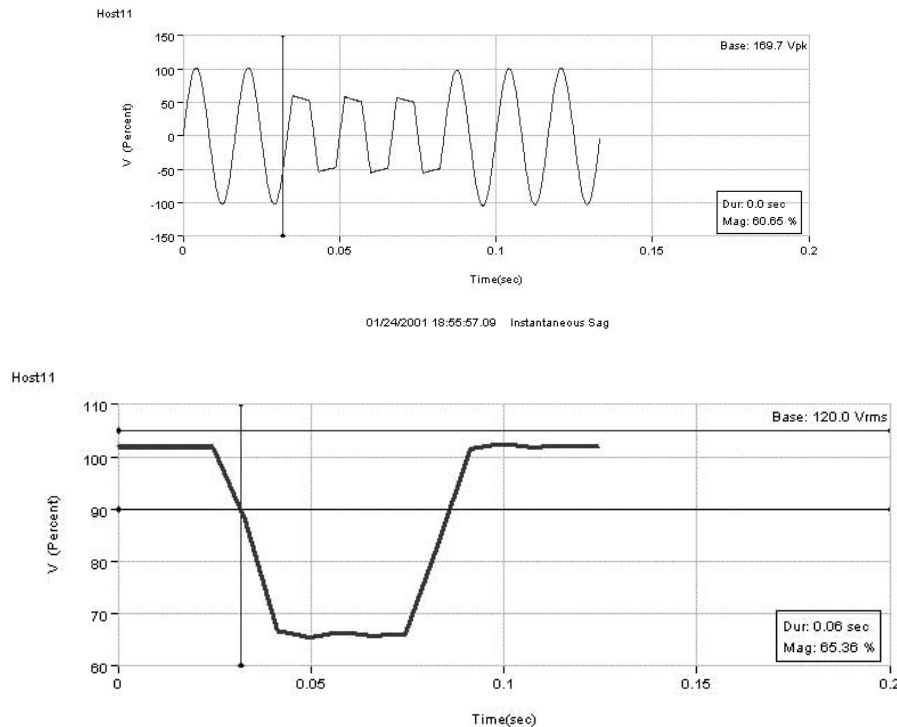


Figure 10—RMS variation represented as waveform and RMS envelope

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.

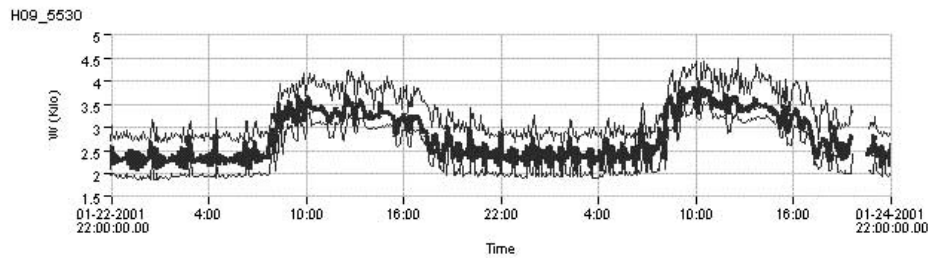


Figure 11—Three-phase real power trend over 48 hours: two PQDIF files joined

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.
** =====
*/

```