Communicator EXT™ 4.0 Software Log Database Application Guide for EIG Meters

Application Guide
V.11
January 28, 2016

Electro Industries/GaugeTech™
The Leader In Power Monitoring and Smart Grid Solutions™

www.electroind.com

1800 Shames Drive, Westbury, NY 11590 Tel: (516) 334-0870 Fax: (516) 338-4741 Email: sales@electroind.com

Communicator EXT™ 4.0 Software

Log Database Application Guide for EIG Meters

Published by:

Electro Industries/GaugeTech

1800 Shames Drive

Westbury, NY 11590

Copyright Notice

Customer Service and Support

Customer support is available 9:00 am to 4:30 pm, Eastern Standard Time, Monday through Friday. For customer or technical assistance, phone 516-334-0870 or fax 516-338-4741.

Disclaimer

About Electro Industries/GaugeTech (EIG)

Founded in 1975 by engineer and inventor Dr. Samuel Kagan, Electro Industries/ GaugeTech changed the face of power monitoring forever with its first breakthrough innovation: an affordable, easy-to-use AC power meter.

More than forty years since its founding, Electro Industries/GaugeTech, the leader in power monitoring and control, continues to revolutionize the industry with the highest quality, cutting edge power monitoring and control technology on the market today. An ISO 9001:2000 certified company, EIG sets the industry standard for advanced power quality and reporting, revenue metering and substation data acquisition and control. EIG products can be found on site at virtually all of today's leading manufacturers, industrial giants and utilities.

EIG products are primarily designed, manufactured, tested and calibrated at our facility in Westbury, New York.

Table of Contents

This page intentionally left blank.

1: Introduction

The purpose of this manual is as follows:

- To describe the log databases used by Communicator EXT^{TM} 4.0 software for storing all recorded values from EIG meters, once they are retrieved by the user (during log retrieval).
- To show the structure of the database used to store data by the Communicator EXT^{TM} 4.0 software package, as well as to show the interaction between tables in the databases.
- To provide templates, which contain characteristics of devices to be polled and the types of data available.
- To provide usage examples for log types.

Note that this manual is intended for use by persons familiar with databases and their programming. The examples shown are in the programming language C#.

NOTE: Communicator EXT^{TM} 3.0 software doesn't use the same database structure. Please upgrade your system to Communicator EXT 4.0 software to be compatible with this document. The Communicator EXT^{TM} 4.0 application can be purchased from EIG's webstore: https://electroind.com/product/communicator-ext/.

This page intentionally left blank.

2: Logs

This chapter describes the different types of logs that can be retrieved.

2.1: Historical Logs

Historical log tables are used to store historical data from the meters. Historical logs can either be stored as hour tables or channel tables:

- When stored as hour tables, there will be two tables for each hour of data: one for the timestamps, and one for the stored values. When hour tables are used, the data will be stored in the HistLog_N1_N2 tables (where N1 is the Index number from the AllHistoricalLogs, and N2 is the hour: 0-23). There will also be a HistLog-TimeIndex_N1_N2 table, which will be linked to the HistLog tables with the same N index numbers. This table will be linked to its HistLog table using the DateTimeIn-dex to determine which part of the hour the data is from (see 3.18: Hist-Log_N1_N2, on page 3-24 and 3.19: HistLogTimeIndex_N1_N2, on page 3-25).

- When channel tables are used, the data will be stored in the channel tables (Data_ID, where ID is the ID number from the EIGNameList.DB). The srcid will need to be used to determine which log the actual data comes from, as the logs may be configured to store the same readings in multiple logs (see 7.1: Source ID (srcid), on page 7-1).

The determination of channel tables versus hour tables can be done by looking at the DB_SETTINGS table (an example is shown on the next page):

- If the dbset_value for dbset_key is equal to "CHANNEL," then channel tables are used.

- If the dbset_value for dbset_key is equal to "HOUR," then hour tables are used.

If you find that "HOUR" tables are being used, see , on page 2-2; if you find that "CHANNEL" tables are being used see 2.1.2: Channel Tables, on page 2-5.

Example:

```
public string GetTableFormat()
{
    string tableFormat = "";
    //Get the channel table type from the settings table.
    tableFormat = query("SELECT dbset_value
                         FROM DB_SETTINGS
                         WHERE dbset_key = 'channel.table.type'");
    switch(tableFormat)
    {
    case "CHANNEL":
        return "CHANNEL";
        //Process channel tables.
        break;
    case "HOUR":
        return "HOUR";
        //Process hour tables.
        break;
    default:
        return "HOUR";
        //Unknown format, assume hour tables.
        break;
     }
}
```

2.1.1: Hour Tables

Hour tables are the original method for storing historical data in the database, and databases created with Communicator EXT^{TM} 3.0 software use it. Databases created with Communicator EXT^{TM} 4.0 software use the newer channel table method, except for the MP200^{TM} metering system and any meter's Flicker logs, which always use the hour tables.

With the hour tables, the data in the database splits log data into a time/value pair of tables for every hour to be stored. For each log, there are 24 tables (one per hour). A master AllHistoricalLogs table is then used to determine what days and hours exist, and what the indexes for those tables are.

Once you have determined that "HOUR" tables are being used, query the AllHistorical-Logs table using a time range, to learn which HistLog tables exist for that time range.

Example 1:

```
string tableFormat = GetTableFormat();//See 2.1: Historical Logs, on page 2-1.
string dataId = 201000;
List<double> VANvalues = new List<double>();
if(tableFormat.Equals("HOUR"))
{
    string startTime = Input Time;
    string endTime = Input Time;
    nums = query("SELECT *
              FROM AllHistoricalLogs
              WHERE DateTimeValue >= " + startTime + "
              AND DateTimeValue <= " + endTime);
    for(int i =0; i < nums.rows.count; i++)
    {
        for(int j = 0; j < 23; j++)
        {
             //below gets the Hour_ field value
             bool hourTableExist = sqlStr.fields("Hour_" + j);
             if(hourTableExists)
             {
                 date = query("SELECT DateTimeIndex,DateTimeValue
                     FROM HistLogTimeIndex_" + i + "_" + j);
                 string[] timeIndexQ = date.fields["DateTimeIndex"];
                 foreach(string timeIndex in timeIndexQ)
                 {
                     data = query("SELECT ItemValue
                       FROM HistLog_" + i +"_" + j + "
                       WHERE DateTimeIndex = " + timeIndex + "
                       AND DataID = " + dataId);
                       VANvalues.Add(data.fields["ItemValue"]);
                 }
             }
        }
    }
}
```

Example 2:

```
string tableFormat = GetTableFormat();//See 2.1: Historical Logs, on page 2-1.
List<double> VANvalues, VBNvalues, VCNvalues = new List<double>();
if(tableFormat.Equals("HOUR"))
{
    string startTime = "#04/22/2015#";
    string endTime = "#05/22/2015#";
    allHistTableQuery = query("SELECT *
                          FROM AllHistoricalLogs
                          WHERE DateTimeValue >= " + startTime + "
                          AND DatetimeValue <= " + endTime);
    for(int i =0; i < allHistTableQuery.rows.count; i++)
    {
        for(int j = 0; j < 23; j++)
        {
            //below gets the Hour_[j] field value
            bool hourTableExist = sqlStr.fields("Hour_" + j);
            if(hourTableExists)
            {
                dateQuery = query("SELECT DateTimeIndex,DateTimeValue
                         FROM HistLogTimeIndex_" + i + "_" + j);
                string[] timeIndexHolder = dateQuery.DateTimeIndex;
                foreach(string timeIndex in timeIndexHolder)
                {
                    data = query("SELECT ItemValue,DataID
                                 FROM HistLog_" + i +"_" + j +
                                 " WHERE DateTimeIndex = " + timeIndex);
                    if(data.fields["DataID"]  == 201000)
                    { VANvalues.Add(data.fields["ItemValue"]); }
                    elseif(dataQuery.DataID == 201002)
                    { VBNvalues.Add(data.fields["ItemValue"]); }
                    elseif(dataQuery.DataID == 201004)
                    { VCNvalues.Add(data.fields["ItemValue"]); }
                }
            }
        }
    }
}
```

2.1.2: Channel Tables

Channel tables were created to improve on the hour table method. Data channels are now stored in a single channel per table, identified from the EIGNameList.DB. This table combines the data channel value with the log, timestamp, and profileIndex, which were formerly split between two tables in the hour table format. This method allows for the values for any particular data channel to be retrieved with a single query, and simplifies finding the data.

Another change to the record format is that the timestamp is now OLE (object linking and embedding) encoded as a double, as opposed to the hour tables configuration, in which the timestamp is a date and time. This change significantly improves the query time, since it does not require conversion for every comparison. It also allows millisecond values to be encoded directly into the timestamp, rather than storing them as a separate field. This results in simpler and faster queries.

Since the channels are organized in a flat, large table, the tables can now be split into their own linked databases. This greatly increases the storage space for the database, (by as much as 2 GB per channel), which, with the current table format, results in 50 million available records.

Once you have determined that channel tables are being used, query the chan_dist table using the time range you want to filter by, so that you can find which data channels are available.

Once you have a list of all channels for that time range, query the actual channel tables using the same time range. Since we are using channel tables, we need to use a source id (srcid) to only collect historical data and not data from other logs. Example:

```
string tableFormat = GetTableFormat();
List<double> VANvalues, VBNvalues, VCNvalues = new List<double>();
if(tableFromat.Equals("CHANNEL")
{
    //Query below will return a list of channels that exist for the given
    //time range
    channel = query("SELECT DISTINCT chandist_channel
                FROM chan_dist
                WHERE chandist_day >= 42100
                AND chandist_day <= 42106");
    List<long> channels = channel;
    foreach(long id in channels)
    {
        //Loop through all channels in the list
        //Query below shows how to get interval (historical) 1 logs
        data = query("SELECT data_value,dataID
                FROM data_" + id +
                " WHERE data_timestamp >=42100
                AND data_timestamp <= 42106
                AND data_srcid = 2"); //see 7.1
        foreach(row val in dataTableQuery.fields["data_value"])
        {
            if(dataQuery.dataID  == 201000)
            {VANvalues.Add(data.fields["ItemValue"]);}
            elseif(dataQuery.dataID == 201002)
            {VBNvalues.Add(data.fields["ItemValue"]);}
            elseif(dataQuery.dataID == 201004)
            {VCNvalues.Add(data.fields["ItemValue"]);}
         }
    }
}
```

NOTE: To get all historical logs, just add all of the srcid's for historical data to the query, except for the Nexus® 1500/1500+ meter. If using a Nexus® 1500/1500+ meter, you will need to query the other Interval databases.

Examples:

```
OR data_srcid = 3
OR data_srcid = 18
OR data_srcid = 19
OR data_srcid = 20
OR data_srcid = 21
OR data_srcid = 22
OR data_srcid = 23
```

You can query historical 2 logs by a specific channel:

```
string tableFormat = GetTableFormat();
bool chanExists = false;
List<double> values = new List<double>();
if(tableFormat.Equals("CHANNEL")
{
    //First check if the data point exists, the query below will return an
    //array of channels
    channel = query("SELECT DISTINCT chandist_channel
                FROM chan_dist
                WHERE chandist_channel = 201000");
    long[] channels = channel.rows.count;
    if(channels.Count > 0)
    {
        chanExists = true;
    }
    if(chanExists)
    {
         //Channel exists, now you can query the table
         data = query("SELECT data_value
                  FROM data_201000
                  WHERE data_timestamp >= + startTime +
                  " AND data_timestamp <= " + endTime +
                  " AND data_profileindex = 1
                  " AND data_srcid = 2");
         foreach(row val in data.fields["data_value"])
         {
             values.Add(val)
         }
     }
}
```

2.1.3: Identifying the Data

Before you use the Data_ID's to query the data, you must find the Data_ID for the value(s) you want.

- Data_IDs vary based on the meter.
- Each meter retrieves the Data_ID differently.

Follow this procedure:

1. Connect to the EIGNameList.DB and search for the Data_ID based on the meter type.

   a. For Shark® Series meters, use the Modbus map’s starting address to find the data ID's in the EIGNameList.DB (see 5.4.2: Start Address (Shark® Meters), on page 5-7).

b. For Nexus® Series meters, use the Modbus map’s line and point numbers to find the data ID's in the EIGNameList.DB (see 5.4.1: Line and Point (Nexus® Meters), on page 5-6).

2. Once you have found the Data_ID(s), you need to get the description from the EIGNameList.DB (see 5.3.1: Queries, on page 5-3).

   a. For Shark® 200 meters, use DeviceModbusMap table searching for Description.

   b. For the MP200™ metering system, use ShrkMF200 table searching for Description.

   c. For Nexus® Series meters, use DeviceProtocol_3_1 table searching for Description1 and Description2.

3. You now have the descriptions as well as the values. The last thing to do before displaying the data, is to scale the values using the ScaledFormats table, if you are using scaled energy (see 3.26: ScaledFormats, on page 3-43).

   a. For Shark® Series meters, query the ScaledFormats table using the correct ProfileIndex, so that you use the correct row from the table (see , on page 3-44).

   b. For Nexus® Series Meters, connect to the EIGNameList.DB, again, and search for the appropriate SEID matching the FormatID in the scaled format table (see 5.5: Scaled Energy ID (SEID), on page 5-8).

2.2: Limits Logs (Alarm Logs/Sequence of Events Logs)

Limits are threshold settings that are configured by a user to trigger a record in a log when an alarm condition is met. The user can then control a relay or send a warning email on alarm. Limits/alarms are configured in the meter's Device Profile, using Communicator EXT™ software. Refer to the *Communicator EXT™ 4.0 and MeterManager EXT Software User Manual* for instructions.

When the value for the reading type rises above, or falls below, the specified limit value, a limit record is created. Limit records provide magnitude of the event, and also the duration of the event. This is because Limit records recorded in the meter are paired with an out record and a return to normal record. Using these two records, a duration of the event can be computed. The meter keeps track of these limit records in its internal storage. When logs are retrieved, the meter sends the limit records to be stored in the database.

Limit records can be stored in two types of logs: Event logs and Snapshot logs.

- Limit events are stored in the LimitLog_N table, where N is the index number from the AllLimitsLogs table pointing to the day of the log (see 3.23: LimitsLogs_N, on page 3-31). The Limit Events log records an entry every time limit values monitored by the meter change their state. The log records information about the limits; for example, which limits are currently exceeded, and which limits have just changed.

- Limit Snapshots are stored in the historical log tables (see 2.1: Historical Logs, on page 2-1), using the source id 4.

The LimitsLogs tables are linked to the LimitsLogDataItem table by their LinkedIndex fields. LimitsLogs tables are also linked to the AllLimitsLogs table by their index fields, which is used to find limits in time. For each LogTablesIndex in the AllLimitsLogs table, there will exist a LimitLog_N table (where N is the index number of the day from the AllLimitsLogs table pointing to the day of the log).

To find all limit records from a specific time (the limits log tables also have linked data in the LimitsLogDataItem table, the example below shows how to pull this data as well).

Example:

```
//first query the AllLimitsLogs table
int[] logTables = query("SELECT LogTablesIndex
                      FROM AllLimitsLogs
                      WHERE DateTimeValue >= #04/22/2015#
                      AND DateTimeValue <= #05/22/2015#");
for(int i = 0; i < logTables.Count; i++)
{
    //To find the limit records for Volts A-N where the limit is set to above:
    limTable = query("SELECT *
                  FROM LimitsLogs_" + tables[i] +
                  "WHERE DataId = 201000 //See 5.0 to see how to determine dataID
                  AND DescriptionCode = 1");//See 3.23.1 for description codes
    //LimitsLogDataItem table has linked data using the LinkedIndex field.
    //You can query this table to find programmable settings information on
    //the limits you just queried.
    limDataIndex = query("SELECT DISTINCT LinkIndex
                    FROM LimitsLog_" + i);
    int index = limDataIndex;
    dataItem = query("SELECT *
                FROM LimitsLogDataItem
                WHERE LinkIndex = " + index +
                "AND DataID = 201000");
}
```

NOTE: DataID will vary depending on the meter (see 5.1: NameList Lookup, on page 5-1).

2.3: Power Quality Logs

The synchronization of the voltage frequency and phase allows electrical systems to function in their intended manner, without significant loss of performance or life. Power Quality (PQ) determines the fitness of electrical power for consumer devices.

The PQ Logs record data in response to surges and sags of programmed High Speed Limit Triggers on enabled High Speed channels. The information this provides allows the calculation of duration and magnitude of the surges and sags, as well as information for locating the start and end of the surge or sag in the Waveform captures. PQ records can be triggered from Voltage & Current RMS channels, transient channels and digital input channels, and the Power Quality log records information regarding the meter's trigger, condition at the time of the trigger, and duration of the event. PQ log records contain:

- Timestamp (in the record header).
- Event identification, i.e., present state and change flags for each of the 9 potential events.
- Capture number and cycle causing the event. Capture number will be zero if the event cycle isn't in a capture.
- Additional flags to indicate whether or not the cycle is in a capture, and if the cycle is only partially in the capture.
- RMS values for each potential event. For departure events, RMS values will be zero. For return to normal events, it will be the worst-excursion RMS values for the channels that returned to normal. Other channels will be zero.

Example:

```
//First query the AllPQLogs table.
int[] allPQTable = query("SELECT LogTablesIndex
                        FROM AllPQLogs
                        WHERE DateTimeValue >= #04/22/2015#
                        AND DateTimeValue <= #05/22/2015#");
//Use dates below to query a range of time in a day.
string startTime = "2015/04/15 09:30:00.0000";
string endTime = "2015/04/15 17:45:00.0000";
for(int i = 0; i < allPQTable.Count; i++)
{
    //Below will query the PQ table during a range of time in a day.
    pqTableTime = query("SELECT *
                    FROM PQLog_" + tables[i] +
                    " WHERE StartDateTimeValue >= #" + startTime +
                    " AND EndDateTimeValue <= #" + endTime + "#");
    //Below will query the PQ table by magnitude.
    pqTableMag = query("SELECT *
                  FROM PQLog_" + tables[i] +
                  " WHERE PQValue >= 110");
    //Below will query the PQ table by duration in milliseconds.
    pqTableDur = query("SELECT *
                  FROM PQLog_" + tables[i] +
                  " WHERE duration > 500");
}
```

2.3.1: Linked PQ Wave Data

PQ and Waveform records may be associated together, using their table index numbers. To check if there is linked data, query the PQ table with the same index number as the waveform table you are querying, using the WaveformLink field in the PQ table (see 3.24: PQLogs_N, on page 3-34). If the WaveformLink field is true, then there is linked data for this record.

Example:

```
int[] allWaveTables = query("SELECT LogTablesIndex
                        FROM AllWaveformLogs
                        WHERE DateTimeValue >= #04/22/2015#
                        AND DateTimeValue <= #05/22/2015#");
for(int i = 0; i < allWaveTables.Length; i++)
{
    //Check if PQLog_i table exists where WaveformLog_i exists.
    waveLink = query("SELECT WaveformLink, IndexValue
                  FROM PQLog_" + waveformTables[i]);
    bool waveformLink = waveLink.fields["WaveformLink"];
    if(!waveformLink)
    {
        //If false, then table does not exist, or there is no linked PQ data.
    }
    else
    {
        //If field is true, then there is linked PQ data Next, query the
        //waveformLog_i table.
        waveLinkQuery = query("SELECT *
                        FROM WaveformLog_" + i);
    }
}
```

2.4: Relay Logs

There are two types of Relays: Digital Input and Digital Output. Data values from both of these types of relays are stored in two types of logs: snapshots and events. Snapshots are stored in Historical tables (see 2.1: Historical Logs, on page 2-1) using source id 5 for digital input, and source id 6 for digital output (see 7.1: Source ID (srcid), on page 7-1). Events are stored in the InputLogA (see 3.21: InputLogA, on page 3-27) table for digital inputs, and the RelayLogA (see 3.25: RelayLogA, on page 3-39) table for digital outputs.

2.4.1: Digital Input Log

The Digital Input log is used to document the transitions of internal and external digital inputs. The log records an entry every time the digital inputs change state, recording both the state of each input and a snapshot of the items configured in the programmable settings (parsing the snapshot items is the same as for the interval/ historical log items; see 2.1: Historical Logs, on page 2-1).

The Digital Input records only store the current state of the inputs. To determine information about overall input events, you would need to analyze the sequence of the digital input records. However, this is not always necessary, as the inputs can be used just for state changes, rather than more involved conditions.

2.4.2: Digital Output Log

The Digital Output log is used to document the changing states of digital outputs. Records are stored in four instances:

1. When the delay at the end of a Relay Logic Tree is finished, indicating that a relay needs to change state.
2. When a communication port requests to lock or unlock a relay.
3. When a command is transmitted to the external device.
4. When a response is returned from the external device.

2.5: System Events Log

The System Events log records the system changes made to the meter. It contains information on instances of the following: power up/power down, password access/ modification, change of programmable settings, change of a run time, firmware change, change of clock time by communication (Modbus or DNP), Test Mode usage, meter resets (Logs, Max/Min, Energy), and other meter occurrences. The System Events log records and timestamps each of these events. Its purpose is to provide data for security and anti-tampering measures.

The System Events log is stored using the AllSystemEventsLogs table and the SystemEventLogA table. The AllSystemEventsLogs table has one record per day, and tells you which days will be in the SystemEventLogA table. You can also use the AllSystemEventsLogs table to determine how many records exist for each day.

Example:

```
//Query system events at a given time.

SELECT *
FROM SystemEventLogA
WHERE DateTimeValue = #05/22/2015#

//Or query a time range of system events.

SELECT *
FROM SystemEventLogA
WHERE DateTimeValue >= #04/22/2015#
AND DateTimeValue <= #05/22/2015#

//Query Log Activity events only.

SELECT *
FROM SystemEventLogA
WHERE DateTimeValue >= #04/22/2015#
AND DateTimeValue <= #05/22/2015#
AND EventType = 'Log Activity'
```

2.6: Waveform Logs

Some EIG meters have the ability to record waveforms. Waveform logs store records of waveform data captures that have been taken based on a limit condition, input state change, or manually triggered by a Modbus message or software command. The meter’s device profile defines what needs to be recorded. Waveform logs store more data than other logs, which is why comma separated values (CSV) files are used for the Nexus® 1500/1500+ waveform recordings.

Each waveform capture is stored as a single record. The record holds summary information for the capture as a whole, plus all of the sample data. Waveform records contain:

1. Timestamp (in the header of the record).

2. RMS values for all channels in the trigger cycle.

3..Sample rate.

4. Trigger type (RMS, adaptive RMS, manual, etc.).

2.6.1: Waveform Capable Meters (Not Nexus® 1500/1500+ Meters)

For all meters that are capable of waveform capture (other than the Nexus® 1500/ 1500+ meter), the waveform data is stored in the database's WaveformLog_N table (where N is the index number from the AllWaveformLogs table pointing to the day; see 3.29: WaveformLog_N, on page 3-56, for table description). Unlike the Nexus® 1500/1500+ meter, these meters only record the waveform samples.

Example:

```
string startDate = "#04/22/2015#";
string endDate = "#05/22/2015#";
allWaveforms = query("SELECT LogTablesIndex
                    FROM AllWaveformLogs
                    WHERE DateTimeValue >= "+ startDate +
                    "AND DateTimeValue <= " + endDate);
//Above will return an array of index numbers where there are tables for the
//specified date; store them below. The next step will be to query the Waveform/
//Log_[i] table:
int[] waveformTables = allWaveforms;
for(int i = 0; i< waveformTables.Length; i++)
{
    //Query below will get all waveform records for the time range.
    waveformLog = query("SELECT *
                      FROM WaveformLog_" + waveformTables[i] +
                      "WHERE WSDT >= " + startDate +
                      "AND WEDT <= " endDate);
    //Query below will get all waveform records where the duration is >= 500 ms.
    waveformDurQ = query("SELECT *
                      FROM WaveformLog_" + waveformTables[i] +
                      "WHERE WDuration >= 500")
    //Store the samples (Go to 2.6.3: Waveform Samples, on page 2-24, for
    //information) on sample conversion.
}
```

2.6.2: Nexus® 1500/1500+ Meter Comma Separated Value (CSV) Files

CSV files are used to store Waveform readings for the Nexus® 1500/1500+ meter. The path to the CSV file is located in the Sample0 field in the WaveformLog table. The path is given relative to the root DML file (see 4.3: Nexus® 1500/1500 Meter, on page 4-2). Once you have queried the path to the CSV file(s) that are needed, you read the file(s) to get the readings.

The CSV files are stored in the Waveform Folder, csv sub folder:

...\Communicator_EXT\RetrievedLogs\UsersNexus1500_12345\LOGS\WAVE-FORM\csv

The file is split into four sections:

- Header.
- DSP ADC Block Factors.
- Samples Block.
- RMS Info Block.

2.6.2.1: Header

The Header section of the CSV file contains basic information: File version and File size (in bytes).

2.6.2.2: DSP ADC Block Factors

The DSP ADC Block Factors section of the CSV file contains the sample factors used to convert the values. By parsing the file by this section, you can pull the sample factor for the appropriate channel name.

For example, to find the sample factor for Volts AN:

1. Once you are in the block Factor section, search each line for Volts AN.
2. When you find Volts AN, select the next column - this will be the scale factor to use (See 2.6.3.5: Sample Conversions (Nexus® 1500/1500+ Meter Only), on page 2-33, for an example).

2.6.2.2.1: Scale Factors

The two following charts can be used to find the appropriate scale factor needed for the conversion. You must extract the correct sample factor and multiply by the value.

Channel Name	ID	Scale Factor
Volts AN	0	Logical Voltage
Volts BN	1	Logical Voltage
Volts CN	2	Logical Voltage
Volts AB	3	Logical Voltage
Volts BC	4	Logical Voltage
Volts CA	5	Logical Voltage
Volts XN	6	Logical Voltage
Volts Residual	7	Logical Voltage Residual
Current Residual	8	Logical Current Residual
Volts AE	32	Physical Voltage
Volts BE	33	Physical Voltage
Volts CE	34	Physical Voltage
Volts XE	35	Physical Voltage
Volts NE	36	Physical Voltage
Current A	37	Physical Current
Current B	38	Physical Current
Current C	39	Physical Current
Current X	40	Physical Current
Waveform Transient 0	77	Waveform Combine
Waveform Transient 1	78	Waveform Combine
Waveform Transient 2	79	Waveform Combine
High Speed Inputs	80	N/A

Factor Name	Scalar
Logical Voltage	0.098401062
Logical Voltage Residual	0.196802124
Physical Voltage	0.049200531
Logical Current Residual 20A	0.012363048
Logical Current Residual 2A	0.001960812
Physical Current 20A	0.003090762
Physical Current 2A	0.000495203
Peak Transient	14.0625
Waveform Combine	0.098401062

2.6.2.2.2: Waveform Start Time

Most of the timestamps in the waveform capture are based on the RMS block times (see 2.6.2.4.1: RMS Block Timestamp, on page 2-23). However, a capture may have multiple sample blocks before the first RMS block. To determine the exact time of the first sample (the start of the waveform), determine the number of samples before the first RMS block, and multiply by the time per sample.

Example:

First Sample Block TTC: 5105

First RMS block TTC: 5106

First RMS Block Time: 2015/05/22 17:42:16.3044

Time Per Sample: (1000ms / (60 * 1024)) = apx 0.016276 ms

Samples to Offset: 256 * (First_RMS_Block - First_Samples_Block) + 1
= 256 * (5106 - 5105 + 1)
= 512

Time of First Sample: First_RMS_Block_Time - (Samples_To_Offset * Time_Per_Sample)

= 2015/05/22 17:42:16.3044 - (512 * 0.016276)

= 304.4 - 8.333312

= 296.07

= 2012/04/13 17:42:16.29607

NOTE: We offset by two sample blocks, because the first sample block was not the same as the first RMS block. Also, as the TTC (Time Counter) can roll over, the first sample block time counter may be greater than the RMS block time counter.

2.6.2.3: Sample Block

For information on the Sample Block section of the CSV file, see 2.6.3: Waveform Samples, on page 2-24.

2.6.2.4: RMS Info Block

The RMS Info Block section of the CSV file contains all of the RMS measurement data. The RMS block section begins after the Samples section of the CSV file. Every sample has a corresponding RMS block.

The first line in each block is the RMS Transfer Time Counter, which corresponds to the sample with the same time. The other fields are as follows:

- Period count - The number of samples used to compute the cycle data.
- Cycle count - Incrementing number of cycles, used for cycle identification.
- Reserved.
- Reference Sample Index - Index of the specific sample in the matching sample.
- Absolute Sample Index - Absolute index for the sample which marks the end of the RMS block. This counter rolls over at 65536.
- High Speed Input Transitions - Bit flag which indicates the state of the high speed inputs that have changed since the last RMS block.
- High Speed Input States - Bit flag which indicates the current state of the high speed inputs.
- DSP2 Firmware Type - Firmware type of the DSP2.

- Waveform Voltage RMS Sag Flags - Bit flags which indicate if the voltage RMS was below the waveform sag threshold during the period of the RMS block.
- Waveform Voltage RMS Swell Flags - Bit flags which indicate if the voltage RMS was below the waveform swell threshold during the period of the RMS block.
- Transient Pos Overrange Flags - Indicates a positive transient which exceeded the configured threshold occurred.
- Transient Neg Overrange Flags - Indicates a negative transient which exceeded the configured threshold occurred.
- Transient Pos Max Sample Index - The sample index value of the peak positive transient.
- Transient Neg Max Sample Index - The sample index value of the peak negative transient.
- Transient Pos Max Sample Value - The peak positive transient value.
- Transient Neg Max Sample Value - The peak negative transient value.

For example:

1. Read the file, searching each line until you find the data you are looking for.
2. Store each line into an array until you get to the line consisting of ----- , which indicates the end of the section.

RMS	**Tag**	****WaveRMS********
	RMS Transfer Time Counter	8209
	Timestamp	2015/05/22 08:45:39.8077
	One Cycle RMS [0]	122.99
	One Cycle RMS [1]	122.94
	One Cycle RMS [2]	0
---------		

3. After the line indicating the section has ended (------), the next section begins. There will be no title in column 1, because the title will be the same as the last section; however, the readings in the rest of the columns will be for the item in the transfer time counter.

RMS Transfer Time Counter	**8210**
Timestamp	2015/05/22 08:45:39.8077
One Cycle RMS [0]	122.98

2.6.2.4.1: RMS Block Timestamp

The timestamp of the RMS block is located at the end of its sample block. Since the RMS block may be complete in the middle of a sample block, this time may not be the exact time of the RMS block. To compute the block's actual time, adjust the timestamp by the sample offset from the end of the block and multiply by the time per sample. Time per sample is always (1000ms / (60 * 1024)).

For example:

Block Timestamp: 2015/05/22 17:42:16.3044
Ref Sample Index: 218
Time Per Sample: (1000ms / (60 * 1024)) = apx 0.016276ms
True End of block Time: BlockTime - ((256 - 218) * 0.016246)
= BlockTime - (38 * 0.016246)
= 304.4 - .618488
= 303.78
Time = 2015/05/22 17:42:16.30378

Start time: You can then use the same method to determine the start time of the RMS block. To determine the time of the first sample in the capture, you need to subtract the time for 256 samples from the time of the RMS block. Using the above block as an example:

Beginning of Block Time: BlockTime - (256 * 0.016276)
= 304.4 - 4.16656
= 300.23
Time = 2015/05/22 17:42:16.30023

2.6.3: Waveform Samples

Waveform samples are recorded in response to High Speed Limit triggers. They contain samples from the requested channels at the requested sample rate, programmed in the meter's device profile. Meters vary in the way they record waveform samples.

2.6.3.1: Nexus® 1500/1500+Meter

The waveform samples are stored in the CSV file. To get the sample readings:

1. Search each line as you loop through the file for "Waveform Samples Start."

2. Read each line until the end of the section which will be indicated by a line consisting of: -------- , located in column 1.

The format of the waveform samples are similar to the chart below. The first line will show you what is being recorded in the samples section. You can use the first line of the file to find out which values are recorded and the columns they will be in:

NOTE: The first value Channel 80 [80] is High Speed Digital Inputs.

Nexus® 1500/1500+ Waveform Record				Channel 80 [80]	Volts AN [0]	Volts BN [1]	Volts CN [2]	Ia [37]	Ib [38]	Ic [39]	In [40]

The sample section is similar to the chart below.

Waveform Samples Start	**Tag**	****WaveSample******									
	Samples Transfer Time Counter	5105	S 0	255	121	115	120	4	3	1	2
			S 1	255	750	745	750	4	3	1	2
			S 2	255	1326	1320	1325	4	3	3	2
			S 3	255	1660	1657	1658	5	2	1	2
	Samples Transfer Time Counter	5106	S 0	255	1719	1717	1718	4	3	2	2
			S 1	255	1626	1629	1628	3	3	2	2
			S 2	255	1201	1206	1201	4	2	1	1
			S 3	255	577	582	577	5	3	1	2

Possible channels and ID's are:

Channel Name	ID
High Speed Inputs	80
Volts AN	0
Volts BN	1
Volts CN	2
Volts AB	3
Volts BC	4
Volts CA	5
Volts XN	6
Volts NE	36
Volts AE	32
Volts BE	33
Volts CE	34
I_A	37
I_B	38
I_C	39
I_N	40
Volts Residual	7
I Residual	8
Transient Volts AN/AB	77
Transient Volts BN/BC	78
Transient Volts CN/CA	79

Following is an example demonstrating how to pull all Volts AN sample values from the CSV samples section, and convert them.

1. You will first need to get the CT/PT ratio from the database. The CT/PT ratio is stored as a string. Follow this procedure:

   a. Parse the string at the '/'.

   b. Convert both numbers from strings to doubles.

c. Divide the numerator by the denominator.

d. Multiply by scale factor (see the scale factors chart in 2.6.2.2.1: Scale Factors, on page 2-19).

NOTE: Before pulling the full scale value, check the profileIndex of the profile you are retrieving from and add it to the query, so that the appropriate full scale values are queried.

Example:

```
int profileIndex = 1;
//Use PTCT_Ratio3 because we are searching for Volts A-N.
string str = query("SELECT PTCT_Ratio3
                  FROM FullScales
                  WHERE IndexValue = " + profileIndex);
//Query will return a string of a ratio looking similar to "5.00/1.00".
double[] nums = str.Split("/");//Parse the string at the '/' and store the
//numbers
double numerator = Convert.ToDouble(nums[0]);//Numbers are in string format so
//convert
double denominator = Convert.ToDouble(nums[1]);
double ratio = numerator / denominator;//Divide the numerator by the denominator.
List<double> vanValues = new List<double>();
bool factorSection = false;
bool samplesSection = false;
int voltsAn = 0;//This will be used to store the column number for Volts AN.
while(!reader.EndOfStream)
{
    //The first thing you could do is find the index for Volts AN on the first
    //line.
    string line = reader.ReadLine();
    if(line.Contains("Nexus 1500 Waveform Record"))
    {
        string[] split = line.Split(',');
        for(int i = 0; i < split.Length; i++)
        {
            if(split[i].Contains("Volts AN")
            {
            voltsAn = i;
            }
        }
     }
     double scaleFactor = 0;
     //First loop until you find the DSP ADC Block Factors section.
     if(line.Contains("DSP ADC") || factorSection)
     {
         factorSection = true;
         //Once you reach this section you need to search every line for
         //V A-N.
         if(line.Contains("V A-N") && factorSection)
         {
            //Now that you have found the appropriate line, you need to
            //locate and then store the sample factor.
            string[] split = line.split(',');
            for(int i = 0; i < split.length; i++)
            {
                //Loop through the array looking for V A-N.
                if(split[i].Contains("V A-N"))
                {
                    //If volts AN is there then take the next line which is the
                    //factor.
                    scaleFactor = Convert.ToDouble(split[i+1]);
                    factorSection = false;
                    break;
                }
            }
       }
     }
//Once you have the samples and scalars stored, go to sample conversion section
//2.6.3.5: Sample Conversions (Nexus® 1500/1500+ Meter Only), on page 2-33.
}
```

2.5.3.2: Nexus® 1200 Series

For the Nexus® 1200 Series meters, the samples are stored in the log database using the WaveformLog_N table (where N is the index number from the AllWaveformLogs table pointing to the day of the log.).

Depending on the sampling rate, the meter can record up to 15 channels. Decreasing the sampling rate will increase the number of cycles in a capture. Samples fields 0-6 are fixed; however the rest of the sample fields are used for High Speed inputs. To parse the fields in the database, see 2.6.3.4: Parsing the Sample Fields, on page 2-31.

Some sample rates are affected by the record format:

- Sample Rate 16, 32, 64, 128 (These rates do not depend on record format.)

Channel Name	Sample
Voltage A	0
Voltage B	1
Voltage C	2
Current A	3
Current B	4
Current C	5
Current N	6

- Sample Rate 256:

  Record Format 0 = Voltage

Channel Name	Sample
Voltage A	0
	1
Voltage B	2
	3
Voltage C	4
	5

NOTE: When sampling at the 256 rate, two sample fields are used for each voltage or current reading. This is because that rate allows for more captures. For example: Current A will be using sample0 and sample1 fields to store the captures. Both fields need to be combined to see the full reading.

- Sample Rate 512:

  Record Format 0 = Voltage A

  Record Format 1 = Voltage B

  Record Format 2 = Voltage C

Channel Name	**Sample**
Current A	0
	1
Current B	2
	3
Current C	4
	5

Record Format 4 = Current A

Record Format 5 = Current B

Record Format 6 = Current C

Record Format 7 = Current N

Channel Name	**Sample**
Current	0
	1
	2
	3

NOTE: When sampling at the 512 rate, four sample fields are used for each voltage or current reading. For example: current or voltage will be using sample0 - sample3 fields to store the captures. All four fields need to be combined to see the full reading.

2.6.3.3: Shark® 200 Meter

For the Shark® 200 Series meters, the samples are stored in the log database using the WaveformLog_N table (where N is the index number from the AllWaveformLogs table pointing to the day of the log.). The only difference with this meter series is that there are only six channels (see chart below). The number of cycles depends on the sampling rate. Decreasing the sampling rate will increase the number of cycles in a capture. A full capture includes 2048 samples for 6 channels, plus associated record overhead and trigger information. To parse the fields in the database, see 2.6.3.4: Parsing the Sample Fields, on page 2-31.

Sample fields for the Shark® 200 meter are always the same:

Channel Name	Sample
Voltage A	0
Voltage B	1
Voltage C	2
Current A	3
Current B	4
Current C	5

2.6.3.4: Parsing the Sample Fields

To parse the samples, follow the steps below:

1. Query the Sample field(s) you want to get from the database.

2. The samples are stored in the field as a tab delimited string. Split the string and store each value into an array.

Example:

The following example shows how to get the values for Sample0 and Sample1 from the first Waveform table. It can be adjusted to get all samples and search all waveform tables.

```
//Query the waveform table.
string command = Query("SELECT * FROM WaveformLog_1");

//Select the samples you want.
string sample0 = command.fields("Sample0");
string sample1 = command.fields("Sample1");

//Store the values into an array.
string[] s0 = sample0.Split('\r');
string[] s1 = sample1.Split('\r');
```

This example shows how to combine 256 sample rate fields for the Nexus® 1200 meter.

```
//Query the waveform table.
string command = Query("SELECT * FROM WaveformLog_1");

//Select the samples you want.
string sample0 = command.fields("Sample0");
string sample1 = command.fields("Sample1");

//Store the values into an array.
string[] s0 = sample0.Split('\r');
string[] s1 = sample1.Split('\r');

ArrayList combinedSamples = new ArrayList();
//Combine the two arrays.
for(int i = 0; i < s0.Length)
{
      combinedSamples.Add(s0[i]);
}
for(int i = 0; i < s1.Length)
{
      combinedSamples.Add(s1[i]);
}
```

The following example shows how to combine 512 sample rate fields for a Nexus® 1200 Series meter.

```
//Query the waveform table.
string command = Query("SELECT * FROM WaveformLog_1");
//Select the samples you want.
string sample0 = command.fields("Sample0");
string sample1 = command.fields("Sample1");
string sample2 = command.fields("Sample2");
string sample3 = command.fields("Sample3");
//Store the values into an array.
string[] s0 = sample0.Split('\r');
string[] s1 = sample1.Split('\r');
string[] s2 = sample2.Split('\r');
string[] s3 = sample3.Split('\r');
ArrayList combinedSamples = new ArrayList();
//Combine the four arrays.
for(int i = 0; i < s0.Length)
{
      combinedSamples.Add(s0[i]);
}
for(int i = 0; i < s1.Length)
{
      combinedSamples.Add(s1[i]);
}
for(int i = 0; i < s2.Length)
{
      combinedSamples.Add(s2[i]);
}
for(int i = 0; i < s3.Length)
{
      combinedSamples.Add(s3[i]);
}
```

2.6.3.5: Sample Conversions (Nexus® 1500/1500+ Meter Only)

The sample values in the CSV file for the Nexus® 1500/1500+ meter are stored as ADC values. ADC values need to be converted before being displayed:

1. Use the scale factors from 2.6.2.2.1: Scale Factors, on page 2-19, to convert to secondary.

2. Multiply the secondary value by the CT/PT Ratio to convert to primary.

Following is an example of this conversion process:

1. Convert ADC to Secondary:

   a. Take a raw ADC sample, e.g., Volts AN = 121 from the chart in 2.6.3.1: Nexus® 1500/1500+Meter, on page 2-24.

   b. Multiply the sample by its scale factor (use the chart in 2.6.2.2.1: Scale Factors, on page 2-19) which is 0.098401062:
   121 * 0.098401062 = 11.9

2. Convert the Secondary to Primary:

   a. Take the primary value from above: 11.9.

   b. Multiply by the appropriate CT/PT Ratio from the FullScales table (see 3.17: FullScales, on page 3-22). Lets say your CT/PT ratio is 1440/120:
   11.9 * (1440/120)
   11.9 * 12 = 142.8

3. The converted AN value is 142.8.

2.7: EN 50160 Log

EN 50160 is a European standard that gives the main characteristics of the voltage at the customer's supply terminals, in public low voltage and medium voltage electricity distribution systems, under normal operating conditions. This standard gives the limits, or values, within which any customer can expect the voltage characteristics to remain.

The object of this standard is to define and describe the characteristics of the supply voltage concerning:

- Frequency
- Magnitude
- Waveform
- Symmetry of the three phase voltage

The EN 50160 log is a kind of trending log. The results of the logs is stored in the AllPQTestLogs table (see 3.7: AllPQTestLogs, on page 3-7). For the Nexus® 1500/ 1500+ meters, the EN 50160 weekly and yearly data is stored in a different format than for the other logs: each week and year is stored in its own XML File, named after the week and the year it contains data for. To retrieve this log, all you must do is download the file in question.

2.8: Event Triggered Log

The Event Triggered log is used to record Historical logs when an event occurs. Once the event occurs, the meter will record as quickly as possible until it is told to stop. The Event Triggered log can be programmed in the meter's programmable settings, in which both the event that will trigger the meter to start recording, and the length of time the meter should record, are configured. The Event Triggered log is stored as a Historical log with the source id of 24 (see 7.1: Source ID (srcid), on page 7-1).

2.9: Flicker

Flicker is a visible change in electricity caused by rapid fluctuations in the voltage of the power supply. If Flicker goes beyond a certain point, it can have detrimental effects on people. For this reason, some EIG meters have the capability to measure Flicker.

The Flicker log stores records in order to document Short Term and Long Term Flicker. Flicker logs are stored in the Historical log tables of the database using "HOUR" tables. If your database is using "CHANNEL" tables and a Flicker log is retrieved, there will also be "HOUR" tables in the database, since this is where the Flicker data is stored.

This page intentionally left blank.

3: Tables

This chapter contains descriptions of tables in the database, as well as table formats, table examples, and table usage examples using queries. All table fields in this section will be using 2003 format. To see the field names for 1997 format, see 6.3: Field Differences, on page 6-3.

3.1: AllFlickerLogs

The AllFlickerLogs table is used to store the time arrangement of all flicker logs.

3.1.1: Format

- LogTablesIndex [int32] - Index number for each record.
- HTable [boolean] - Used to determine if that index has a table.
- Recs [int32] - Number of records for each log.
- DateTimeValue [Date/Time] - The date of the entry.

3.1.2: Example

LogTables Index	HTable	Recs	DateTime Value
1	Yes	22	05/22/2015
2	Yes	42	04/11/2015
3	Yes	56	03/28/2015
4	Yes	11	03/07/2015

3.2: AllHistoricalLogs

The AllHistoricalLogs table is used to store the time arrangement for all historical logs when using hour tables, as well as snapshot logs.

3.2.1: Format

- LogTablesIndex [int32] - Index number given to each entry.
- Hour_0 [boolean] - Indicates if a table exists for that hour.
- ...

- Hour_23 [boolean] - Indicates if a table exists for that hour.
- Recs_0 [int32] - The number of records for that hour.
  …
- Recs_23 [int32] - The number of records for that hour.
- DateTimeValue [Date/Time] - The date and time of the entry.

3.2.2: Example

LogTablesIndex	Hour_0	.	.	Hour_23	Recs_0	.	.	Recs_23	DateTime Value
1	Yes	Yes	Yes	Yes	22	11	20	17	04/04/2015
2	Yes	Yes	Yes	Yes	22	11	20	17	04/04/2015

3.3: AllInputLogs

The AllInputLogs table is used to store the time arrangement of all digital input logs. Digital Input log records an entry every time the digital inputs change state. The log records states of each input at the same time of the record, along with a snapshot of the items configured in the programmable settings for the limit log. These snapshot items are stored with the historical logs.

3.3.1: Format

- LogTablesIndex [int32] - Index number for each record.
- HTable [boolean] - Used to determine if that index has a table.
- Recs [int32] - Number of records for each log.
- DateTimeValue [Date/Time] - The date of the entry.

3.3.2: Example

LogTables Index	HTable	Recs	DateTime Value
1	Yes	22	05/22/2015
2	Yes	42	04/11/2015
3	Yes	56	03/28/2015
4	Yes	11	03/07/2015

3.4: AllLimitsLogs

The AllLimitsLogs table is used to store the time arrangement of all limits logs. The table is used to determine how many tables there are for the log type. These snapshot items are stored with the historical logs.

3.4.1: Format

- LogTablesIndex [int32] - Index number for each record.
- HTable [boolean] - Used to determine if that index has a table.
- Recs [int32] - Number of records for each log.
- DateTimeValue [Date/Time] - The date of the entry.

3.4.2: Example

LogTables Index	HTable	Recs	DateTime Value
1	Yes	22	05/22/2015
2	Yes	42	04/11/2015
3	Yes	56	03/28/2015
4	Yes	11	03/07/2015

3.5: AllLogs

The AllLogs table holds information about each time that logs are retrieved. A new record is created for each log download.

3.5.1: Format

- LogOptionID1 [int32] - The enumerations of the Log Option Names for the record added together (see 3.5.3: Log ID and Name, on page 3-5).
- LogOptionID2 [int32] - (Reserved for internal use.)
- LogOptionName [text] - Names of the logs retrieved for the specific retrieval.
- ProFileIndex [int32] - Index number of the profile (foreign key to AllProfiles table).
- LogConverterVersion [text] - Version number of log converter.
- LogConverterInfo [text] - The build date of the log converter.
- DownloadSoftwareInfo [text] - The software which downloaded the logs from the meter.
- FirmwareInfo1 [text] - Firmware version, dependent on the meter type.
- FirmwareInfo2 [text] - Firmware version, dependent on the meter type.
- FirmwareInfo3 [text] - Firmware version, dependent on the meter type.
- FirmwareInfo4 [text] - Firmware version, dependent on the meter type.
- FirmwareInfo5 [text] - Firmware version, dependent on the meter type.
- EntryDateTimeValue [Date/Time] - The date and time of the log download.
- IndexValue [int32] - Index number for the entry.

3.5.2: Example

Log Option ID	Log Option ID2	Log Option Name	Profile Index	Log Converter Version	Log Converter Info	Down load Soft ware Info	Firm ware Info 1	.	.	Firm ware Info 5	Entry Date/ Time	Index Value
193	0	Program-ming File, Log 1, Log 2	1	v4.0.23	11/03/ 2014 15:49:00	CEXT	FC02	N/ A	N/ A	N/A	03/31/ 15 3:02:37 PM	1
4801	0	Program-ming File, Log 1, Log 2, Limit Log, Flicker Log	1	v4.0.23	11/03/ 2014 15:49:00	CEXT	FC02	N/ A	N/ A	N/A	03/31/ 15 3:05:58 PM	2
6849	0	Program-ming File, Log 1, Log 2, Limit Log, Flicker Log, Relay Log	1	v4.0.23	11/03/ 2014 15:49:00	CEXT	FC02	N/ A	N/ A	N/A	04/01/ 15 8:21:58 AM	3
57537	0	Program-ming File, Log 1, Log 2, Wave-form Log, System Event Log, Power Quality	2	v4.0.23	11/03/ 2014 15:49:00	CEXT	FC02	N/ A	N/ A	N/A	04/01/ 15 8:23:37 AM	4

3.5.3: Log ID and Name

Each possibility that can be in log option name has an enumeration, for all of the logs in a download, their enumerations are added together to get the LogOptionID.

For example:

If a record in this table contains Programming File, and Limit Log, the LogOptionID for this record will be:
1 + 512 = 513.

If a record in this table contains Programming File, Flicker Log, the LogOptionID for this record will be:
1 + 4096 = 4097

The chart below shows each LogOptionID with its name.

LogOptionID	LogOptionName
0	No Data Available
1	Programming File
2	Historical Log Profile
4	Event Log Profile
8	Waveform Log Profile
16	Programming Block
32	Historical Log
64	Log 1
128	Log 2
256	Energy/Demand Log
512	Limit Log
1024	Input Log
2048	Relay Log
4096	Flicker Log
8192	Waveform Log
16384	Power Quality Log
32768	System Event Log
65536	Log 3
131072	Log 4
262144	Log 5
524288	Log 6
1048576	Log 7
2097152	Log 8
4194304	Event Triggered
8388608	Transient Waveform

3.6: AllPQLogs

The AllPQLogs table is used to store the time arrangements for all PQ logs. The table is used to determine how many PQ tables exist. For each record in this table there is a PQ table for that day.

3.6.1: Format

- LogTablesIndex [int32] - Index number for each record.
- HTable [boolean] - Used to determine if that index has a table.
- Recs [int32] - Number of records for each log.
- DateTimeValue [Date/Time] - The date of the entry.

3.6.2: Example

LogTables Index	HTable	Recs	DateTime Value
1	Yes	22	04/11/2015
2	Yes	42	03/28/2015
3	Yes	56	03/07/2015
4	Yes	11	02/22/2015

3.7: AllPQTestLogs

The AllPQTestLogs table is an EN 50160 feature that is used to summarize test results.

3.7.1: Format

- ID_PQTest [int32] - Index number of the record.
- StartTime [Date/Time] - Start time of the event.
- Endtime [Date/Time] - End time of the event.
- Freq [double] - Frequency.
- S_Freq [int32] - Frequency Fluctuations (for test descriptions see 3.7.3 #1).
- S_LSV [int32] - Low Speed Voltage (for test descriptions see 3.7.3 #2).

- S_FVF [int32] - Fast Voltage Fluctuations (for test descriptions see 3.7.3 #3).
- S_Flicker [int32] - Flicker (for test descriptions see 3.7.3 #4).
- S_SVU [int32] - Supply Voltage Unbalance (for test descriptions see 3.7.3 #5).
- S_Harmonic [int32] - Harmonic Magnitude (for test descriptions see 3.7.3 #6).
- S_THD [int32] - Total Harmonic Distortion (for test descriptions see 3.7.3 #7)
- P_Freq [single] - Frequency Fluctuations (for test descriptions see 3.7.3 #1).
- P_LSV [single] - Low Speed Voltage (for test descriptions see 3.7.3 #2)
- P_FVF [single] - Low Speed Voltage (for test descriptions see 3.7.3 #3)
- P_Flicker [single] - Fast Voltage Fluctuations (for test descriptions see 3.7.3 #4)
- P_SVU [single] - Supply Voltage Unbalance (for test descriptions see 3.7.3 #5)
- P_Harmonic [single] - Harmonic Magnitude (for test descriptions see 3.7.3 #6)
- P_THD [single] - Total Harmonic Distortion (for test descriptions see 3.7.3 #7)
- bin_1 [double] - Bins used to generate the tests (reserved for internal use)
- …
- bin_170 [double] - Bins used to generate the tests (reserved for internal use).
- NominalVoltage [double] - The voltage which the device is designed to operate.
- TimeRange [int32] - 0=none,1=week,2=year.
- XMLFileName [text] - Name of the XML File storing the records (Nexus® 1500/ 1500+ meter).

3.7.2: Example

NOTE: P fields and bin fields are not in the table, due to space limitations.

ID_ PQ Test	Start Time	End Time	Freq	S_ Freq	S_ LSV	S_ FVF	S_ Flicker	S_ SVU	S_ Harmonic	S_ THD	Nominal Voltage	Time Range	XML File Name
1	12/ 31/12 7:41: 19 PM	12/ 31/12 7:57: 11 PM	0	-1	-1	-1	-1	-1	-1	-1	120	2	2012. XML
2	01/ 01/13 1:10: 02 AM	03/ 16/13 1:38: 46 AM	0	-1	-1	-1	-1	-1	-1	-1	120	2	2013. XML
3	12/ 31/12 7:41: 19 PM	01/ 04/13 1:10: 03 AM	0	-1	-1	-1	-1	-1	-1	-1	120	1	2013_ W01. XML
4	01/ 14/13 9:15: 48 PM	01/ 15/13 10:43 :22 PM	0	-1	-1	-1	-1	-1	-1	-1	120	1	2013_ W03. XML

3.7.3: Tests

1. Frequency Fluctuations: Data taken from the EN 50160 Week Block, defined as any deviation of the 10 second average fundamental frequency from nominal (60Hz or 50Hz) for greater then 10 seconds. Two ranges are defined for synchronized systems: 1. 95% of the week, the mean fundamental frequency must be within ±1% of the nominal; 2. 100% of the week, the mean fundamental frequency must be within +4%-6% of the nominal. Two ranges are defined for non-synchronized systems: 1. 95% of the week, the mean fundamental frequency must be within ±2% of the nominal; 2. 100% of the week, the mean fundamental frequency must be within ±15% of the nominal. The test uses two totalized percentages for the 10 second average frequency.

2. Low Speed Voltage Fluctuations: Data taken from the EN 50160 Week Block, defined as any deviation of the 10 minute average of the supply voltage from the nominal, where the 10 minute average of the supply voltage must remain within ±10% of the nominal for 95% of the week. The test uses three totalized percentages of the 10 minute average bins for the individual phase supply voltages.

3. Fast Voltage Fluctuations: Data taken from the EN 50160 Week Block, defined as any deviation of the instantaneous supply voltage from the nominal, where the supply voltage must remain within ±5% of the nominal. Deviations of up to ±10% are allowed several times per day, programmable by the user (FVF per day, in the meter's programmable settings). The test uses three totalized percentages of the instantaneous bins for the individual phase supply voltages

4. Flicker: Data taken from the EN 50160 Week Block, defined as any deviation of the Long Term Flicker above 1, where the Long Term Flicker must stay below 1 for 95% of the week. The test uses a bin of counts of PLTs greater then 1 for each phase, compared to the total count.

5. Supply Voltage Unbalance: Data taken from the EN 50160 Week Block, defined as the percentage of 10 minute mean RMS values of the negative phase sequence compared to the 10 minute mean RMA values of the positive phase sequence, where the negative phase sequence must be within 0% to 2% of the positive phase sequence. The test uses a bin of counts of negative phase sequence greater then 2% of the positive phase sequence, compared to the total count.

6. Harmonic Magnitude: Data taken from the EN 50160 Week Block, defined as any deviation of the individual voltage harmonics for each phase from a defined percentage, where all harmonics must be below their defined limit for 95% of the week. The test uses a bin of counts of harmonics which exceed their defined limit, compared to the total count.

7. %THD: Data taken from the EN 50160 Week Block, defined as any deviation of the THD of the supply voltage, where the THD must remain below 8% at all times. The test uses a bin of counts of THD which have exceeded 8%.

3.8: AllProfiles

The AllProfiles table is used to hold the programmable settings for each device profile.

3.8.1: Format

- ProfileIndex [int32] - Index number of the entry and the profile.
- ms [int16] - The milliseconds to be added to the DateTimeValue to determine the timestamp down to the millisecond.
- DTmode [int16] - Used to store the last daylight savings time mode (see 7.2: Daylight Savings Time (DST), on page 7-2).
- Profile [text] - The device's programmable settings (reserved for internal use).
- DateTimeValue [Date/Time] - The date of the entry.

3.8.2: Example

ProfileIndex	ms	DTmode	Profile	DateTimeValue
1	0	0		05/22/2015
2	0	0		04/28/2015
3	0	0		03/07/2015
4	0	0		02/28/2015

3.9: AllRelayLogs

The AllRelayLogs table holds the time arrangements of all relay logs. The table is used to determine the time of the record, and the number of records in the table.

3.9.1: Format

- LogTablesIndex [int32] - Index number for each record.
- HTable [boolean] - Used to determine if that index has a table.
- Recs [int32] - Number of records for each log.
- DateTimeValue [Date/Time] - The date of the entry.

3.9.2: Example

LogTables Index	HTable	Recs	DateTimeValue
1	True	5	05/22/2015

3.10: AllSystemEventLogs

The AllSystemEventLogs table holds the time arrangements of all events which occur in the meter, such as time changes, log downloads, profile updates, meter power losses and log resets.

3.10.1: Format

- LogTablesIndex [int32] - Index number for each record.
- HTable [boolean] - Used to determine if that index has a table.
- Recs [int32] - Number of records for each log.
- DateTimeValue [Date/Time] - The date of the entry.

3.10.2: Example

LogTables Index	HTable	Recs	DateTimeValue
1	Yes	22	04/11/2015
2	Yes	42	03/28/2015
3	Yes	56	03/07/2015
4	Yes	11	02/22/2015

3.11: AllWaveformLogs

The AllWaveformLogs table stores records for all waveform logs. The table is used to determine how many days of data there are available for the waveform log. For example: There are four records in the table below, therefore the database will have four waveform tables.

3.11.1: Format

- LogTablesIndex [int32] - Index number for each record.
- HTable [boolean] - Used to determine if that index has a table.
- Recs [int32] - Number of records for each log.
- DateTimeValue [Date/Time] - The date of the entry.

3.11.2: Example

LogTables Index	HTable	Recs	DateTimeValue
1	Yes	22	04/11/2015
2	Yes	42	03/28/2015
3	Yes	56	03/07/2015
4	Yes	11	02/22/2015

3.12: chan_dist

The chan_dist table contains the number of records for each day, for each channel. This allows you to determine the overall time range of any channel, including gaps in the center, as well as determining how many records are available in that time range. The table's primary purpose is to allow the software to limit the time range, based on the number of records it queries for any channel, without directly querying a potentially huge channel data table. There should only be one entry for each channel/day pair. By querying this table for any particular channel, the software can artificially perform the limitation ahead of time, and limit the time range, rather than the number of records.

3.12.1: Format

- chandist_channel [int32] - The channel id of the channel, matching the id used by the data_id table. The channel id comes from the EIGNameList.DB.
- chandist_day [double] - The OLE-formatted double timestamp, encoding just the date portion.
- chandist_reccount [int32] - The number of records in a single day, for a given channel.

3.12.2: Example

chandist_channel	chandist_day	chandist_reccount
201000	42146	100
201002	42146	100
201004	42146	100
201012	42146	100
201000	42145	100
201002	42145	100
201004	42145	100
201012	42145	100

3.12.3: Usage

To find out how many records exist for a day you can query this table:

Example:

```
list<long> records = query("SELECT chandist_reccount
                            FROM chan_dist
                            WHERE chandist_day = 42146");
long recordNumsi = 0;
foreach(long record in records)
{
    recordNums += record;
}
//If we go by the table above there will be 400 records for this day.
```

To get record counts for a time range just alter the query:

```
list<long> records = query("SELECT chandist_reccount
                        FROM chan_dist
                        WHERE chandist_day >= 42145
                        AND chandist_day <= 42146");
list<long> records = new list<long>();
long recordNums = 0;
foreach(long record in records)
{
    recordNums += record;
}
//If we go by the table above there will be 800 records for this time range.
```

3.12.3.1: Time Usage

This section shows how to get the actual date using the chandist_day timestamps, which are used to represent an actual day.

- Double to date:

  chandist_day = 42146

  a. Take the units portion: 42146 which means 42146 days since 1899/11/30, which is: May 22, 2015

  b. You end up with the day May 22, 2015.

  Programmatically:

```
int d1 = 42146;
DateTime date1 = DateTime.FromOADate(d1);
//date1 will be 05/22/15 12:00:00.
```

- Date to double:

  Use the date above May 22, 2015.

  a. Get the days since 1899/11/30 = 42146.

  b. You end up with the number: 42146

  Programmatically:

```
DateTime date1 = new DateTime(2015,5,22);
int d1 = (int)date1.ToOADate();
//Cast to int because ToOADate returns a double: d1 will be 42146.
```

3.13: data_ID

The Data_ID (the id number from the EIGNameList.DB) in the tables, contains the actual channel data in a flat, timestamp/value pair. There is one table for each channel id. The channel ID comes from the EIGNameList.DB. Duplicate timestamps are allowed, as long as they have different profile ids and log types. This table is paired with the chan_dist table to determine the distribution of records in time. For the Nexus® 1500/1500+ meter, each channel table is stored in an external, linked database. These tables will be used when the database format is "CHANNEL" and channel tables are being used (see 6.2: Identify, on page 6-1).

3.13.1: Format

- data_timestamp [double] - The time that the value was recorded. The timestamp is stored as an OLE formatted double, which encodes the days since 1899/11/30 as the integer portion, and the fraction of the day in the fraction portion, which allows for approximately a 10 microseconds accuracy.

- data_value [double] - The value of the record as a double.

- data_dst [int32] - Index indicating if the timestamp represents DST (Daylight Savings Time) time. Only relevant during the overlap period (see 7.2: Daylight Savings Time (DST), on page 7-2).

- data_profileindex [int32] - Foreign key to the AllProfiles table. Used to look up information such as Scaled Energy formats.

- data_srcid [int32] - Integer code used to point to the log type that the data came from (see 7.1: Source ID (srcid), on page 7-1).

3.13.2: Example

data_timestamp	data_value	data_dst	data_profileindex	data_srcid
42146.615972	122.8779068	-1	1	3
42146.616667	122.87591533	-1	1	2
42146.616667	122.87591533	-1	1	2
42146.617361	123.14221191	-1	1	3

3.13.4: Usage

The following sections elaborate on the timestamp usage and on the table’s usage.

3.13.4.1: Time Usage

This section shows how the OLE encoded double timestamps are used to represent an actual date and time.

- Decimal to date:
  data_timestamp = 42146.615972
  a. Take the decimal portion: 0615972, which is 61.5972% of the day, so it's 14:46 and 59.980 seconds (14:46:59.980).

  b. Take the units portion: 42146 which means 42146 days since 1899/11/30 which is May 22, 2015.

  c. You end up with the date time of: May 22, 2015 2:46.59.980 PM.

Programmatically:

```
double oaDate = 42146.615972;
DateTime date = DateTime.FromOADate(oaDate);
//Date will be 05/22/15 2:46:59.
```

- Date to decimal:
  Use the date above May 22, 2015 2:46.59 PM.
  a.Get the decimals for 14:46.59.980. 86400 seconds in the day / 53219.980 seconds so far = .615972 or 61.5972% of the day.

  b. Get the days since 1899/11/30 = 42146.

  c. You end up with the number: 42146.615972.

Programmatically:

```
//2015/05/22 14:46:59.980
short year = 2015;
short month = 5;
short day = 22;
short hour = 14;
short minutes = 46;
short seconds = 59;
short ms = 980;
DateTime date = new DateTime(year,month,day,hour,minutes,seconds,ms);
double oaDate = date.ToOADate();
//oaDate will be 42146.615972
```

3.13.4.2: Table Usage

This section shows how to use the table.

NOTE: If you want to query a specific log, you would need to get the data_srcid (see 7.1: Source ID (srcid), on page 7-1).

Example:

To query all values of the same Data_ID between a certain timeframe you can use:

```
SELECT data_value
    FROM data_201500//See section 5.4 to see how to get the data ID's.
    WHERE data_timestamp >= 42100.5
    AND data_timestamp <= 42110.5
    AND data_srcid = 3 //Historical Log 2
To query all data types between a certain timeframe you can use:
//Store all data points in the database.
List<long> dataPoints = GetDataPoints();
//Loop through each data point in the collection.
foreach(long id in datapoints)
{
    value  = query("SELECT data_value
                FROM data_" + id +
                "WHERE data_timestamp >= 42100
                AND data_timestamp <= 42110
                AND data_srcid = 3");
}
```

NOTE: The src_id is used to determine which log the data is coming from. To find out which ID number you need, see 7.1: Source ID (srcid), on page 7-1.

3.14: DataPoints

The DataPoints table stores all of the DataID's used in the interval logs, limit logs, and flicker logs.

- Every reading has an ID number. For example; Volts A-N can have multiple records in the database; however its ID number will only be in the DataPoints table once. This ID is matched in the EIGNameList.DB to find its description (see 5.1: NameList Lookup, on page 5-1).
- When using channel tables, each data point will have its own table in the database.

3.14.1: Format

DataID [int32] - The ID number from the EIGNameList.DB.

3.14.2: Example

DataID
201000
201002
201004
201012

3.15: DB_Settings

The DB_SETTINGS table contains extra settings and configurations about the database, in a tag/value pair format. The key is unique. The purpose of this table is so that the software can determine whether the database should use the old hour tables or the new channel tables. If this table does not exist in the database, then the database is using the 97 format (for more information on table format see 6.1: Usage, on page 6-1).

3.15.1: Format

- dbset_key [text] - The lookup key for the setting tag/pair value.
- dbset_value [text] - The value of the dbset_key.

3.15.2: Example

The example below indicates the table is using the Channel tables.

dbset_key	dbset_value
channel.table.type	CHANNEL
db.format	2003

The example below indicates the table is using the Hour tables.

dbset_key	dbset_value
channel.table.type	HOUR
db.format	2003

3.16: DeviceInformation

The DeviceInformation table stores information about the meter from which the logs came.

3.16.1: Format

- DeviceTypeID [int32] - ID number of the meter designated type (3 = Nexus® 1200 Series, 4 = Shark® 200 meter, 5 = Nexus® 1500/1500+meter, 6 =MP 200™ metering system).
- DeviceType [short text] - Meter name.
- HardwareID [short text] - Meter’s serial number.
- HardwareName [short text] - User configurable meter designation.
- TimeZone [short text] - The time zone that the meter is using.
- HardwareTypeName [short text] -The native hardware name.
- VSwitch [int32] - The V-Switch™ key currently installed in the meter (if applicable). Each V-Switch™ key unlocks different features in the meter.

3.16.2: Example

DeviceType ID	DeviceType	Address	HardwareID	Hardware Name	TimeZone	Hardware TypeName	VSwitch
4	Shark 200	0	0123456789	UsersShark200	N/A	Shark 200	2

3.17: FullScales

The FullScales table stores the PT and CT Ratios which are used by the meter for the voltage and current readings. Values are stored in the database as primary values. This table can be used to convert voltage and current readings to and from secondary and primary values. The table also holds the Full Scale values. A full scale is the value which represents 100% of a value (for information on conversions, see 2.6.3.5: Sample Conversions (Nexus® 1500/1500+ Meter Only), on page 2-33).

3.17.1: Format

- PTCT_Ratio1 [text] - Current A, B, C ratio.
- PTCT_Ratio2 [text] - Current N ratio.
- PTCT_Ratio3 [text] - Voltage A, B, C ratio.
- PTCT_Ratio4 [text] - Voltage Aux ratio.
- FullScale1 [text] - Current A, B, C Full Scale.
- FullScale2 [text] - Current N Full Scale.
- FullScale3 [text] - Voltage AN, BN, CN Full Scale.
- FullScale4 [text] - Voltage Aux Full Scale.
- FullScale5 [text] - Voltage AB, BC, CA Full Scale.
- FullScale6 [text] - Power - phase Full Scale.
- FullScale7 [text] - Power - total Full Scale.
- FullScale8 [text] - Frequency Full Scale.
- IndexValue [int32] - Index number which indicates the profile the record belongs to: foreign key to AllProfiles table.

3.17.2: Example

PTCT_ Ratio1	PTCT_ Ratio1	PTCT_ Ratio1	PTCT_ Ratio1	FullScale 1	.	.	.	FullScale8	Index Value
5.00/ 5.00	5.00/ 5.00	120.00/ 120.00	120.00/ 120.00	5.000	5.000	120.000	600.000	60.000	1
5.00/ 5.00	5.00/ 5.00	120.00/ 120.00	120.00/ 120.00	5.000	5.000	120.000	600.000	60.000	2
5.00/ 5.00	5.00/ 5.00	120.00/ 120.00	120.00/ 120.00	5.000	5.000	120.000	600.000	60.000	3
5.00/ 5.00	5.00/ 500	120.00/ 120.00	120.00/ 120.00	5.000	5.000	120.000	600.000	60.000	4

3.17.3: Relation

The index value is very important in this table. You need to choose the ratio with the same index value as the ProfileIndex value from the AllProfiles table. This needs to be done so that you multiply by the correct ratios from the appropriate profile.

For example: If we have a secondary value of 10.22, and the CTPT_Ratio for this value is 1400/120, we multiply the two values together to get its primary value.

= 10.22 * (1400/120)

= 10.22 * 12

= 122.64

3.18: HistLog_N1_N2

This table stores the actual data values of the historical log. The name of the table is constructed from the day and hour it describes, where N1 is the LogTablesIndex number from AllHistoricalLogs, and N2 is the hour 0-23. There is a table for each index in the AllHistoricalLogs table and for each hour of the day: HistLog_1_0 - HistLog_1_23 through HistLog_N_0 - HistLog_N_23.

3.18.1: Format

- DataID [int32] - The ID number from the EIGNameList.DB.
- ItemValue [double] - The value of the data.
- DateTimeIndex [int32] - Index number given to a specific date/time. Go to the corresponding HistLogTimeIndex table with the same N1 and N2 values. The matching DateTimeIndex in this table will give you the date and time of the reading.

3.18.2: Example

DataID	ItemValue	DateTimeIndex
201000	124.03603363	1
201002	124.05510712	1
201004	124.10174561	1
201000	122.00333463	2
201002	123.05523412	2
201004	122.10132161	2

3.18.3: Usage

Historical logs can be queried either by time range, by an individual data id or by a specific time (see examples in , on page 2-2).

3.19: HistLogTimeIndex_N1_N2

Stores the timestamps of the historical log records. The name of the table is constructed from the day and hour it describes, where N1 is the LogTablesIndex number from AllHistoricalLogs, and N2 is the hour 0-23. There is a a HistLogTimeIndex table for each HistLog table.

3.19.1: Format

- DateTimeValue [Date/Time] - The date and time of the entry.
- DateTimeIndex [int32] - Index number associated with a date and time.
- ms [int16] - The milliseconds which get added to the timestamp; used to determine the dateTime down to the millisecond.
- DTMode [int16] - The daylight savings time mode (see 7.2: Daylight Savings Time (DST), on page 7-2).
- TypeID [int16] - The type of log the data came from, i.e., the srcId.
- ProfileIndex [int32] - Index number of the profile the data came from. Foreign key to AllProfiles & AllLogs.

3.19.2: Example

ms	DTMode	TypeID	ProfileIndex	DateTimeValue	DateTimeIndex
0	-1	3	1	04/04/15 11:00:00 AM	1
0	-1	3	1	04/04/15 11:00:00 AM	2
0	-1	3	1	04/04/15 11:00:00 AM	3

3.19.3: Usage

Historical logs can be queried either by time range, by an individual Data_ID or by a specific time (see examples in , on page 2-2).

3.20: ID_TABLE

The ID_TABLE stores information pertaining to the database itself. (For more information on this table see 6.1: Usage, on page 6-1.)

3.20.1: Format

- Name [text] - Name of the database.
- Version [double] - Version number of the database.
- KeyName [text] - The database’s key name.
- Note [text] - A note about the last database update, version date and time.

3.20.2: Example

Name	Version	KeyName	Note
EI Data Log File	9.2	Master DB 2003	Database Updated to V9.2 on 2013-08-30 11:57:00

3.21: InputLogA

The InputLogA table is used to store event logs for digital inputs.

3.21.1: Format

- IndexValue [int32] - The index of the input event.
- DT1 [Date/Time] - Date and time of the start of the limit.
- ms1 [int16] - The milliseconds which get added to DT1 timestamp; used to determine the time down to the millisecond.
- DT1mode [int16] - Daylight savings time mode of DT1 (see 7.2).
- DT2 [Date/Time] - The date and time of the end of the limit.
- ms2 [int16] - The milliseconds which get added to DT2 timestamp; used to determine the time down to the millisecond.
- DT2mode [int16] - Daylight savings time mode of DT2 (see 7.2: Daylight Savings Time (DST), on page 7-2).
- ModuleIndex [int16] - Option board 1 or 2.
- InputIndex [int16] - The index of the record.
- Status [byte] - The state of the relay: 0=Closed, 1=Open.
- DescriptionCode [int16] - An integer code: -2=End with no start, -1=Start with no end, 0=Normal end.
- Duration [double] - Duration in milliseconds from DT1 to DT2.

3.21.2: Example

Index Value	DT1	ms1	DT1 mode	DT2	ms2	DT2 mode	Module Index	Input Index	Status	Description Code	Duration
1	2015/05/22 15:49:00.0000	870	1	2015/05/22 15:49:00.0000	870	1	0	1	1	-2	0
1	2015/05/22 15:49:00.0000	870	1	2015/05/22 15:49:00.0000	870	1	0	2	1	-2	0
1	2015/05/22 15:49:00.0000	870	1	2015/05/22 15:49:00.0000	870	1	0	3	1	-2	0
1	2015/05/22 15:49:00.0000	870	1	2015/05/22 15:49:00.0000	870	1	0	4	1	-2	0
1	2015/05/22 15:49:00.0000	870	1	2015/05/22 15:49:00.0000	870	1	0	5	1	-2	0

3.22: LastTimeStamps

The LastTimeStamps table stores data about the last log retrieval. It is useful for partial log retrievals.

3.22.1: Format

- LogID [int32] - The enumerator of the log (see 3.22.3: LogID and Name Chart, on page 3-30).
- LogName [text] - The name of the log (see 3.22.3: LogID and Name Chart, on page 3-30).
- LastDateTime [Date/Time] - The date/time of the last record.
- Lastms [int16] - The milliseconds to be added to the LastDateTime timestamp to determine the time down to the milliseconds.
- LastDTmode [int16] - Used to store the last daylight savings time mode (see 7.2: Daylight Savings Time (DST), on page 7-2).
- LastRecNum [int32] - Stores the last record position number. For programmable settings, this is the checksum.

3.22.2: Example

LogID	LogName	LastDateTime	Lastms	LastDT mode	LastRec Num	LastRec OffSet
1	Log 1	04/01/15 9:06:00 AM	0	-1	0	1
4	Limit SS	03/24/15 9:22:00 AM	0	-1	0	1
10	Wave Trig	07/09/14 11:23:41 AM	0	-1	0	1
11	Sys Event	4/01/15 9:06:26 AM	0	-1	0	1

3.22.3: LogID and Name Chart

This table contains the LogID and LogName. See the chart, below.

LogID	LogName
0	Profile
1	Log 1
2	Log 2
3	Limits
4	Limit SS
5	Input
6	Input SS
7	Relay
9	Flicker
10	Wave Trig
11	System Event
12	Wave Sample
13	PQ
16	Interval 1
17	Interval 2
18	Interval 3
19	Interval 4
20	Interval 5
21	Interval 6
22	Interval 7
23	Interval 8

3.23: LimitsLogs_N

The purpose of these tables is to record limit events which occur for each limit set in the programmable settings, when the log is retrieved. There is one table for each day that contains records. Each day will be assigned an index number in the AllLimitsLogs table - that index number is the N index.

3.23.1: Format

- LinkIndex [int32] - Link index to the LimitsLogDataItem table.
- Startms [int16] - The milliseconds of the limit start time: combine the ms to the StartDateT timestamp to get the milliseconds attached to the time.
- SDTMode [int16] - Start daylight savings time mode (see 7.2: Daylight Savings Time (DST), on page 7-2).
- Endms [int16] - The milliseconds of the limit end time: combine the ms to the End-DateTime timestamp to get the milliseconds attached to the time.
- EDTMode [int16] - End daylight savings time mode (see 7.2: Daylight Savings Time (DST), on page 7-2).
- LState [text] - Limit conditions: Above or Below.
- Duration [double] - Duration in seconds.
- LValue [double] - Snapshot value of the Limit.
- DescriptionCode [int16] - An integer code: 1 = Above, 2 = Below, 0 = Normal, -1 = Start With No End, -2 = End With No Start.
- IndexValue [int32] - The profile index.
- DataID [int32] - The DataID from the EIGNameList.DB that the limit is set to record.
- StartDateTimeValue [Date/Time] - Start date and time of the limit.
- EndDateTimeValue [Date/Time] - End date and time of the limit.

3.23.2: Example

Each time a limits log is retrieved, a new table is created. See the following example.

Link Index	Start ms	SDT Mode	End ms	EDT Mode	LState	Duration	LValue	Descrip-tionCode	Index Value	Data ID	Start Date Time	End Date Time
18	120	-1	240	-1	Above	1	97.92	1	6	201000	07/05/14 2:22:30 PM	07/05/14 2:22:31 PM
20	220	-1	440	-1	Below	1	97.92	2	6	201002	07/05/14 2:22:30 PM	07/05/14 2:22:31 PM

3.23.3: Time Usage

Time Usage- for record 1:

StartDateTime is 07/05/15 2:22:30 and start ms is 120; combine the two and you get 07/05/15 2:22:30.120.

EndDatetime is 07/05/15 2:22:30 and end ms is 240; combine the two and you get 07/05/15 2:22:30.240.

(For table usage example see 2.2: Limits Logs (Alarm Logs/Sequence of Events Logs), on page 2-8.)

3.23.4: LimitsLogsDataItem

The LimitsLogsDataItem table is used to store Limit information from the programmable settings.

3.23.4.1: Format

- LinkIndex [int32] - Index number of the record linked to the index number of the corresponding LimitLog_N, table where N is the index number of the table from the AllLimitsLog table pointing to the day of the log.
- IndexValue [int32] - The profile index. Each download of data is given a unique index number.
- DataID [int32] - The DataID of the limit being measured.
- DataIndex [int16] - Index number given to each alarm. Entries that belong to the same limit measurement will have the same DataIndex. For example, the above and below rows for Volts A-N will have the same data index. This is done so that the readings can be linked together.
- LimitID [int32] - An id number given to Above or Below values: 1 = Above, 2 = Below, 3 = AND (both above and below).
- LSet [double] - The decimal value that the reading was over/under its limit.
- LSAB [Text] - Above or Below the limit setting.
- LSet_Percent [double] - The percent value that the reading was over/under its limit.

3.23.4.2: Example

Link Index	IndexValue	DataID	DataIndex	LimitID	LSet	LSAB	LSet_ Percent
1	2	201000	1	1	1.1	Above	110
2	2	201000	1	2	.9	Below	90
3	2	201002	2	1	1.1	Above	110
4	2	201002	2	2	.9	Below	90
5	2	202052	3	1	1.1	Above	110
6	2	202052	3	2	.9	Below	90
7	2	206002	4	1	1.1	Above	110
8	2	206002	4	2	.9	Below	90

3.23.4.3: Usage

The LimitsLogsDataItem is linked to the LimitLog table by the LinkIndex field. For usage example, see 2.2: Limits Logs (Alarm Logs/Sequence of Events Logs), on page 2-8.

3.24: PQLogs_N

The PQLogs_N table stores all of the PQ event records when they are downloaded from the meter.

3.24.1: Format

- Startms [double] - The milliseconds which get added to the StartDate timestamp; used to determine the start time down to the millisecond.
- SDTMode [int16] - The start daylight savings time mode (see 7.2: Daylight Savings Time (DST), on page 7-2).
- SWNumber [int16] - The start PQ waveform's unadjusted waveform number.
- SSample [int16] - The start PQ unadjusted waveform sample point.
- Endms [double] - The milliseconds which get added to the EndDate timestamp: used to determine the end time down to the millisecond.
- EDTMode [int16] - The end daylight savings time mode (see 7.2: Daylight Savings Time (DST), on page 7-2).

- EWNumber [int16] - The end PQ waveform's unadjusted waveform number.
- ESample [int16] - The end PQ unadjusted waveform sample point.
- Duration [double] - The duration in milliseconds of the event from Startms to Endms.
  NOTE:For the Shark® 200 meter, this will only show durations in increments of 1000 ms.
- ChannelID [int16] - An integer code to determine the channel (see Part 2 of Chart, on page 3-36).
- PQValue [double] - The RMS value.
- ChannelName [text] - The channel name of the recorded value (see 3.24.3: Channel ID and Channel Name Chart, on page 3-37).
- ConditionCode [int16] - An integer code to determine the condition: 0=normal, 1=surge, 2=sag,-1=Start Point, -2=End Point, 5=Positive Transient, 6=Negative Transient.
- ConditionName [text] - A textual description of the condition.
- WaveformLink [boolean] - A value indicating whether the record has a linked waveform record.
- IndexValue [int32] - The index number of the entry.
- PQ_Percent [double] - A percentage of the PQValue: PQ_Percent = (PQValue / FullScale) * 100.
- EndDateTime [Date/Time] - The PQ end date and time.
- Startdatetime [Date/Time] - The PQ start date and time.

NOTE: If there are waveform records associated with a PQ record, the value for the "WaveformLink" field in this table will be set to TRUE.

3.24.2: Example

Part 1 of Chart

Start ms	SDT Mode	SWNumber	SSample	Endms	EDTMode	EWNumber	ESample	Duration
11.9	0	0	0	15.9	0	0	0	4
11.9	0	0	0	15.9	0	0	0	4
11.9	0	0	0	15.9	0	0	0	4
22.2	0	0	0	30.2	0	0	0	8
22.2	0	0	0	30.2	0	0	0	8
22.2	0	0	0	30.2	0	0	0	8

Part 2 of Chart

Channel ID	PQ Value	Channel Name	Cond. Code	Cond. Name	Wave-form Link	Index Value	PQ Percent	End Date Time Value	Start Date Time Value
0	107.141	Van	1	Surge	False	1	89.28	10/10/14 10:13:52 AM	10/10/14 10:13:52 AM
0	107.085	Vbn	1	Surge	False	1	89.24	10/10/14 10:13:52 AM	10/10/14 10:13:52 AM
0	107.141	Vcn	1	Surge	False	1	89.28	10/10/14 10:13:52 AM	10/10/14 10:13:52 AM
0	62.008	Van	2	Sag	False	1	51.67	10/10/14 17:13:52 PM	10/10/14 17:13:52 PM
0	62.009	Vbn	2	Sag	False	1	51.67	10/10/14 17:13:52 PM	10/10/14 17:13:52 PM
0	62.009	Vcn	2	Sag	False	1	51.66	10/10/14 17:13:52 PM	10/10/14 17:13:52 PM

3.24.3: Channel ID and Channel Name Chart

Channel ID	Channel Name
0	Van
1	Vbn
2	Vcn
3	Vaux
4	Ia
5	Ib
6	Ic
7	Iaux
8	In
9	Vab
10	Vbc
11	Vca
12	In1
13	In2
14	In3
15	In4
16	In5
17	In6
18	In7
19	In8

3.24.4: PQWaveLogDataItem

The PQWaveLogDataItem table is used to store programmable settings information for PQ and waveforms.

3.24.4.1: Format

- SampleRate [int16] - The sampling rate speed.
- CaptureNum [int16] - The number of records captured when a waveform occurs.
- Mode [int16] - Capture mode.
- HSI [int16] - High Speed Inputs.
- C0_L1 [text] - Channel 0, Limit 1 setting.
- ...
- C11_L1 [text] - Channel 11, Limit 1 setting.
- C0_L2 [text] - Channel 0, Limit 2 setting.
- …
- C11_L2 [text] - Channel 11, Limit 2 setting.
- C0_L1SAB [text] - Channel 0, Limit 1 set: Surge or Sag.
- …
- C11_L1SAB [text] - Channel 11, Limit 1 set: Surge or Sag.
- C0_L2SAB [text] - Channel 0, Limit 2 set: Surge or Sag.
- …
- C11_L2SAB [text] - Channel 11, Limit 2 set: Surge or Sag.
- WaveEnables [int32] - Code used to determine if waveform capture is enabled.
- WaveEnables_Input [int32] - Waveform capture trigger: 0=Triggers disabled, 1=Triggers enabled.
- PQEnables [int32] - Code used to determine if PQ capture is enabled.
- PQEnables_Input [byte] - PQ capture trigger: 0=Triggers disabled, 1=Triggers enabled.

- HookUp [int32] - PT/CT hook up mode:
    - 0=Wye.
    - 1=Delta, 2 CTs (Shark® 200 meter); Delta, 3 CTs (all other meters).
    - 2 = 2.5 Element (Shark® 200 meter); Delta, 2 CTs (all other meters).
    - 3= 2.5 Element.
    - 4= 4-Wire Delta.
    - 5=Open Delta.
    - 6=45S, Wye, 2 CTs.
- RecordFormat [int32] - Reserved for internal use.
- IndexValue [int32] - Profile index.

3.25: RelayLogA

The RelayLogA table stores all relay event records when they are downloaded from the meter.

3.25.1: Format

- IndexValue [int32] - The index of the record.
- ModuleIndex [int16] - Option board 1 or 2.
- RelayIndex [int16] - The index of the relay: 1,2 = Relay Out card 1; 3,4 = Relay Out card 2.
- DT1 [Date/Time] - Date and time of the event (Trigger Time).
- ms1 [int16] - The milliseconds which get added to DT1 timestamp; used to determine the time down to the millisecond.
- DT1mode [int16] - Daylight savings time mode of DT1 (see 7.2: Daylight Savings Time (DST), on page 7-2).

- DT2 [Date/Time] - The date and time the command is given to the meter (Command Time).
- ms2 [int16] - The milliseconds which get added to DT2 timestamp; used to determine the time down to the millisecond.
- DT2mode [int16] - Daylight savings time mode of DT2 (see 7.2: Daylight Savings Time (DST), on page 7-2).
- DT3 [Date/Time] - The date and time the meter receives the command to switch (Acknowledge Time).
- ms3 [int16] - The milliseconds which get added to DT3 timestamp; used to determine the time down to the millisecond.
- DT3mode [int16] - Daylight savings time mode of DT3 (see 7.2: Daylight Savings Time (DST), on page 7-2).
- DT1CK [int16] - Reserved for internal use.
- DT2CK [int16] - Reserved for internal use.
- DT3CK [int16] - Reserved for internal use.
- Duration12 [double] - Duration in milliseconds from DT1 to DT2.
- Duration23 [double] - Duration in milliseconds from DT2 to DT2.
- Duration13 [double] - Total Duration in milliseconds from DT1 to DT3.
- Condition0 [int16] - Previous condition.
- Condition1 [int16] - Code for condition, IE: NO; NO = 1; NC,NC = 0.
- Condition2 [int16] - Code for condition, IE: NO,NO = 1; NC,NC = 0.
- Condition3 [int16] - Code for condition, IE: NO,NO = 1; NC,NC = 0.
- Condition0Text [text] - Condition 0 description (previous condition).
- Condition1Text [text] - Condition 1 description: NO - Normally Open; NC - Normally Closed.

- Condition2Text [text] - Condition 2 description: NO - Normally Open; NC - Normally Closed.
- Condition3Text [text] - Condition 3 description: NO - Normally Open; NC - Normally Closed.
- InputInfo11 [text] - Reserved for internal use.
- …
- InputInfo54 [text] - Reserved for internal use.
- InputStatus [int32] - Reserved for internal use.
- GateStatus [int32] - Reserved for internal use.
- InputStatusB [int32] - Reserved for internal use.

3.25.2: Example

Part 1 of Chart

Mod Index	R Index	DT1	ms1	DT1 Mode	DT2	ms2	DT2 Mode	DT3	ms3	DT3 Mode
1	4	2014/08/20 15:49:07.0000	980	1	2014/08/20 15:49:08.0000	340	1	2014/08/20 15:49:08.0000	440	1
1	4	2014/08/20 15:49:44.0000	750	1	2014/08/20 15:49:45.0000	860	1	2014/08/20 15:49:45.0000	440	1
1	1	2014/08/21 11:09:30.0000	90	1	2014/08/21 11:09:31.0000	300	1	2014/08/21 11:09:31.0000	440	1
1	1	2014/08/21 11:09:38.0000	130	1	2014/08/21 11:09:39.0000	290	1	2014/08/21 11:09:39.0000	440	1
1	1	2014/08/21 11:22:08.0000	160	1	2014/08/21 11:22:09.0000	410	1	2014/08/21 11:22:09.0000	440	1
1	1	2014/08/21 11:22:12.0000	250	1	2014/08/21 11:22:13.0000	490	1	2014/08/21 11:22:13.0000	440	1

Part 2 of Chart

DT1CK - DT3CK	Dur 12	Dur 23	Dur13	Con0-Con3	Con0 Text-Con3 Text	Input Status	Gate Status	Input Info ...	...
-1	360	220	580	0	NC, N24	0	81	1	
-1	110	80	190	1	NO, No24	1	81	1	
-1	210	140	350	0	NC, N21	1	81	1	
-1	160	120	280	1	NO, No21	0	81	1	
-1	250	210	460	0	NC, N21	1	81	1	
-1	240	170	410	1	NO, No21	0	81	1	

3.26: ScaledFormats

The ScaledFormats table stores data which is used to format and scale energy values. Note that all energy values in the database have already been scaled to unit (1) for consistency, and this table is only needed for consistent display formatting.

3.26.1: Format

- ProfileIndex [int32] - The profile index number.
- FormatID [int32] - The Scaled Format ID (Nexus® Series meters only). Some Data_ID's have Scaled Energy ID's stored in the EIGNameList.DB (see 5.1: NameList Lookup, on page 5-1).
- Digits [int16] - Value 0 to 7 offset by 2, representing from 2 to 9 displayable digits.
- Units [int16] - Value 0 to 3, representing units of Wh, k, M and G. You also need to divide the value based on the Units value: 1 - divide by 1000, 2 - divide by 1,000,000, 3 - divide by 1,000,000,000.
- Decimals [int16] - Value 0 to 7, representing number of decimal places.

3.26.2: Example

ProfileIndex	FormatID	Digits	Units	Decimals
1	1	3	2	3
2	1	3	2	3

3.26.3: Usage

The Data_ID table has a profile index field which points to this table. Whatever the profile index is, that is the row in the ScaledFormats table to use. This table is used to format the data.

Example:

```
//First, select the profile index from the AllProfiles table. Let's say for this
//instance the profileIndex is 1
int profileIndex = 1;
//For Shark® meters:
scaledForm = query("SELECT *
                    FROM ScaledFormats
                    WHERE ProfileIndex = " + profileIndex);
//For Nexus® meters:
//With a Nexus® meter, you need to get the correct SEID from the NameList.DB (see
//, on page 3-44).
long dataID = 28823;
nameListSEID = query("SELECT SEID
                      FROM DeviceProtocol_3_1
                      WHERE Index = " + dataID);
int formatID = nameListSEID;
scaledFormatTable = query("SELECT *
                           FROM ScaledFormats
                           WHERE ProfileIndex = " + profileIndex
                           " AND FormatID = " + format ID);
```

3.26.4: How to Use the Table

When you get the results from the ScaledFormats table query, use them to format the energy values.

Example 1 (Shark® Series Meters):

If we see a value in one of the data_value fields is 74257891000.

data_timestamp	data_value	data_profileindex
42096.615972	7425789000	1

And by going to the ScaledFormats table we get this:

ProfileIndex	FormatID	Digits	Units	Decimals
1	1	3	2	2

NOTE: The Shark® 200 meter FormatID is always 1.

Follow the steps below to scale the value:

1. Take the value of the Units and divide by 1,000,000:
   Units = 2:742578900 / 1000000 = 742.5789

2. Take the Digits value, which is 3, (actually, 5, since digits are offset by 2):
   00742.5789.

3. Take the Decimals value, which is 2, and remove all numbers after 2 decimal places:
   00742.57.

4. Now that you have scaled the value, go back to the Units. Units has a value of 2, which means Mega (Watts, Watt Hours, varies on what is being measured). You end up with:
   00742.57 M.

Example 2 (Nexus® Series Meters):

The Nexus® Series meters can have more than one row in the ScaledFormats table with the same ProfileIndex. In that case, both the FormatID and the ProfileIndex are used to find the correct row to scale the values. To find the correct FormatID, we need to use the EIGNameList.DB to look for the SEID (Scaled Energy ID) using the Data_ID or the actual name of the Data_ID.

ProfileIndex	FormatID	Digits	Units	Decimals
1	1	5	2	2
1	11	6	2	0
1	22	5	0	2

Take the Data_ID: in this case we will use 28823, from the data_28823 table.

Go to the DeviceProtocol_3_1 table (see 5.2.3: DeviceProtocol_3_1 (Nexus® Series meters), on page 5-2) and query the SEID using the Data_ID.

```
SELECT SEID
FROM DeviceProtocol_3_1
WHERE Index = 28823
```

If you know the name of the Data_ID, you can query this way:

```
SELECT SEID
FROM DeviceProtocol_3_1
WHERE Description1 = 'Scaled Energy'
AND Description2 = 'Q2Wh'
```

The queries above will both return 22, so we need to use the third row.
dataValue = 527.891

Follow the steps below to scale the value:

1. Take the value of the Units: since Units = 0, we do not need to divide: 527.891.

2. Take the Digits value, which is 5, (actually, 7, since digits are offset by 2): 0000527.891.

3. Take the Decimals value, which is 2, and remove all numbers after 2 decimal places:
   0000527.89.

4. Now that you have scaled the value, go back to the Units. Units has a value of 0, which means Wh (varies on what is being measured). You end up with: 00527.89 Wh.

Example 3 (Giga):

dataValue = 756821639526

ProfileIndex	FormatID	Digits	Units	Decimals
1	1	5	3	3

Follow the steps below to scale the value:

1. Take the value of the Units: since Units = 3, we need to divide by 1,000,000,000: 756821639526/1000000000 = 756.821639526.

2. Take the Digits value, which is 5, (actually, 7, since digits are offset by 2): 0000756.821639526.

3. Take the Decimals value, which is 3, and remove all numbers after 3 decimal places:
00756.821.

4. Now that you have scaled the value, go back to the Units. Units has a value of 3, which means G (GW, GWh, varies on what is being measured). You end up with: 00756.821 G.

For more information on SEID see 5.5: Scaled Energy ID (SEID), on page 5-8.

3.26.5: Relation

The ScaledFormats table uses its ProfileIndex field to relate to other tables. The ProfileIndex is a foreign key to the AllProfiles table. The profile index is taken from other tables and this table is then queried to find where the profile index matches. This table also uses its FormatID to relate to the DeviceProtocol_3_1 tables SEID field in the EIGNameList.DB (not a foreign key, see 5.5: Scaled Energy ID (SEID), on page 5-8).

3.27: SystemEventLogA

The SystemEventLogA table stores the data pertaining to the system events. The System Events log stores events which happen in, and to, the meter. It contains records of actions performed by the device which change the way the device operates, i.e., any event that affects the system.

3.27.1: Format

- IndexValue [int32] - The profile index.
- ms [int16] - The milliseconds of the DateTimeValue; used to add to the timestamp to determine the time down to the millisecond (see 3.27.3: Time Usage, on page 3-49).
- DTmode [int16] - The date time index for daylight savings (see chart in 7.2: Daylight Savings Time (DST), on page 7-2).
- InTestmode [boolean] - Value to determine test mode status if applicable to the meter.
- EV0 [int16] - One of eight codes used to determine the description of the event and the event type.*
- EV1 [int16] - One of eight codes used to determine the description of the event and the event type.*
- EV2 [int16] - One of eight codes used to determine the description of the event and the event type.*
- EV3 [int16] - One of eight codes used to determine the description of the event and the event type.*
- EV4 [int16] - One of eight codes used to determine the description of the event and the event type.*
- EV5 [int16] - One of eight codes used to determine the description of the event and the event type.*
- EV6 [int16] - One of eight codes used to determine the description of the event and the event type.*

- EV7 [int16] - One of eight codes used to determine the description of the event and the event type.*
- EventType [text] - The event type based on the meter.*

  * See 3.27.4: Event Type Codes, on page 3-50.

- Description [text] - Description of the event, created by using the codes provided by EV0 - EV8
- Sequence [int32] - Integer used to determine the sequence of events.
- DateTimeValue [Date/Time] - The date and time that the event occurred.

3.27.2: Example

Index Value	ms	DT mode	InTest Mode	EV0	...	EV7	EventType	Description	Sequence	DateTime Value
1	520	0	No	1	2	255	Log Activity	Log Retrieval Begin Historical Log 2 by Option Card 1 Port	0	05/22/14 9:33:40 PM
1	220	0	No	1	3	255	Log Activity	Log Retrieval End Historical Log 2 by Option Card 1 Port	1	05/22/14 9:33:40 PM
1	440	0	No	5	2	255	Test Mode	Test Mode Started set from Internal Use:	2	05/22/14 9:33:40 PM
1	650	0	No	5	2	255	Test Mode	Test Mode Ended set from Internal Use:	3	05/22/14 9:33:40 PM

3.27.3: Time Usage

For example, if DateTimeValue field has a value of 05/22/2015 11:11:22:

1. You need to take the ms value, and add it to the timestamp to determine the time-stamp down to the millisecond: e.g., ms = 200.

2. Concatenate the ms value to its date: e.g., 05/22/2015 11:11:22.200.

3.27.4: Event Type Codes

The System Event types and type codes are as follows:

Nexus® Series Meters

EV0	EV1	Event Type
0		Run Time Status
	0	Run Time was stopped (power loss, boot mode, etc.)
	1	Run Time has started
	2	Run Time is active (features were initialized & enabled)
	3	DSP2 Stop Reset Counter = 0
	4	DSP2 Start
1		Password
	0	Password Protection was Enabled
	1	Password Protection was Disabled
	2	The Level 1 Password was changed
	3	The Level 2 Password was changed
	4	Level 1 access was granted
	5	Level 2 access was granted
	6	An invalid password was supplied
2		Change Programmable Settings
3		Change Firmware
	0	Comm Run Time
	1	DSP 1 Run Time
	2	Comm boot
	3	FPGA
	4	DSP2 Run Time
4		Change Time
	0	Old Time - The timestamp is the old time of the meter
	1	New Time - The timestamp is the new time of the meter
	2	Old Time Auto correction - The timestamp before correction was made; Internally used, so next byte should be 0
	3	New Time Auto correction - The corrected timestamp; Internally used, so next byte should be 0

EV0	EV1	Event Type
5		Test Mode
	0	Enter Test Mode
	1	Exit Test Mode
6		Log Download
	0	Retrieval Started
	1	Retrieval Started, log paused while downloading
	2	Retrieval Ended
	3	Retrieval Ended, dropped records
7		Feature Reset
	0	All Logs Reset
	1	Maximum Reset
	2	Minimum Reset
	3	Energy Reset
	4	Time of Use Current Month
	5	Internal Input Accumulations and Aggregations
	6	KYZ Output Accumulations
	7	Cumulative Demand
	8	Interval 1 Log Reset
	9	Interval 2 Log Reset
	10	Limit Log Reset
	11	Digital Input Log Reset
	12	Digital Output Log Reset
	13	Flicker Log Reset
	14	Waveform Log Reset
	15	PQ Log Reset
	16	System Event Log Reset
	17	Total Average Power Factor Reset
	18	Time of Use Active Registers
	19	Test Mode - NOT SUPPORTED
	20	Interval 3 Log Reset (Nexus® 1500/1500+, only)
	21	Interval 4 Log Reset (Nexus® 1500/1500+, only)

EV0	EV1	Event Type
	22	Interval 5 Log Reset (Nexus® 1500/1500+, only)
	23	Interval 6 Log Reset (Nexus® 1500/1500+, only)
	24	Interval 7 Log Reset (Nexus® 1500/1500+, only)
	25	Interval 8 Log Reset (Nexus® 1500/1500+, only)
	26	Event Triggered Log Reset (Nexus® 1500/1500+, only)
	27	Transient Log Reset
	28	EN 50160 Data Reset
	29	EN 50160 Report Reset
	30	EN 50160 Log Reset
	31	EN50160 Data Preset
	32	Flicker Data Reset
8		System Initialization Problem
	0	Log Initialization
9		Serial Number Change
10		Bio Byte
11		V-Switch™ Key Changed
	0	V-Switch™ Key Changed to
12		Security
	0	Sealing switch enabled
	1	Sealing switch disabled
	2	Network username/password changed
	3	Network privileges changed
13		SCL File Change
	0	CID File
	1	ICD File

MP200^{TM} Metering System

EV0	EV1	Event Type
0		Device Start Up
	0	Firmware v
1		Log Activity
	1	Log Reset
	2	Log Retrieval Begin
	3	Log Retrieval End
2		Clock Activity
	1	Clock Changed
	2	Daylight Savings On
	3	Daylight savings Off
3		System Resets
	1	Max Min Resets
	2	Energy Resets
	3	Reset Relay Board
4		Settings Activity
	1	Password Changed
	2	V-Switch^{TM} Key Changed
	3	Programmable Settings Changed
	4	Measurement Stopped by
5		Boot Activity
	1	Exit to Boot by

Shark® 200 Meter

EV0	EV1	Event Type
0		Firmware Version
1		Log Activity
	1	Reset
	2	Log Retrieval Begin
	3	Log Retrieval End
2		Clock Activity
	1	Clock Changed
	2	Daylight Time On
	3	Daylight Time Off
3		System Resets
	1	Max & Min Reset
	2	Energy Reset
4		Settings Activity
		Password Changed
		V-Switch™ Key Changed
		Programmable Settings Changed
5		Boot Activity
	1	Exit to Boot

3.28: User_Defined_Item_Labels

The User_Defined_Item_Labels table is used to store custom changes the user makes to the meter's programmable settings in Communicator EXT^{TM} software.

3.28.1: Format

- ProfileIndex [int32] - The index of the Device Profile the labels came from.
- Item_Index [int32] - The DataID of the item's label to be overridden from the EIGNameList.DB (see 5.1: NameList Lookup, on page 5-1).
- Label [text] - The label to use in place of the one obtained from the EIGNameList.DB.

3.28.2: Example

ProfileIndex	Item_Index	Label
1	26602	Input 1
1	26603	Input 2

3.28.3: Relation

The User_Defined_Item_Labels tables ProfileIndex field, is used to connect to the AllProfiles and AllLogs tables using their ProfileIndex. All three tables contain a ProfileIndex field used to find out the name of the log.

Example:

1. The ProfileIndex above is 1. Query the AllLogs table where the index is equal to 1 and get the name of the log. You can also use this ProfileIndex to query the AllProfiles table and find the binary value of the devices programming setup information.

2. Now that you have the profile index, query the User_Defined_Item_Labels table using the previously queried index.

```
//First, select the profile index from the AllProfiles table, let's say for this
//instance, the profileIndex is 1.
int profileIndex = 1;
itemLabel = query("SELECT *
                  FROM User_Defined_Item_Labels
                  WHERE ItemIndex = 1");
```

3.29: WaveformLog_N

The purpose of these tables is to store waveform data of captures that have been taken based on a limit condition Input state change or manually triggered by a Modbus message or software command.

3.29.1: Format

- ms [double] - The milliseconds which get added to the timestamp; used to determine the timestamp down to the millisecond.
- DTMode [int16] - The date time index for daylight savings (see 7.2: Daylight Savings Time (DST), on page 7-2).
- WaveformNumber [int16] - The adjusted waveform number.
- TriggeredBy1 [text] - What caused the waveform to trigger.
- TriggeredBy2 [text] - What caused the waveform to trigger.
- TriggerCode [int32] - A unique code used to determine what caused the waveform to trigger.
- ConditionCode_Input [int32] - Input conditions.
- V0 [double] - Voltage AN RMS.
- V1 [double] - Voltage BN RMS.
- V2 [double] - Voltage CN RMS.
- V3 [double] - Voltage Aux RMS.
- V4 [double] - Current A RMS.
- V5 [double] - Current B RMS.
- V6 [double] - Current C RMS.
- V7 [double] - Current Aux RMS.
- V8 [double] - Current N RMS.

- V9 [double] - Voltage AB RMS.
- V10 [double] - Voltage BC RMS.
- V11 [double] - Voltage CA RMS.
- WaveformTriggerNumber [int16] - The adjusted waveform trigger number.
- Contiguous [boolean] - A value that indicates that this waveform is contiguous with the previous record. This will often happen when a waveform trigger occurs while a previous waveform is still being recorded.
- CaptureSequence [int16] - An integer: 0 = first; 1 or more = additional to the first capture sequence.
- SampleInTriggerCyc [int16] - The number of samples in the triggering cycle: 16, 32, 64, 128, 256, 512, 1024.
- EndTriggerCycPT [int16] - The end point of the triggering cycle.
- TimingPT [int16] - A sample point with a timestamp value in the Date/Time field.
- Sample0 [text] - Sample readings vary based on the meter (see 2.6.2.3: Sample Block, on page 2-21).
- …
- Sample14 [text] - Sample readings vary based on the meter (see 2.6.2.3: Sample Block, on page 2-21).
- WTms [double] - The milliseconds which get added to the WTDateTimeValue; used to determine the timestamp down to the millisecond.
- WTDTmode [int16] - The date time mode of WTDateTimeValue; used to determine daylight savings time (see 7.2: Daylight Savings Time (DST), on page 7-2).
- OrigFirstSample [int32] - The unadjusted first sample index within the waveform capture.
- Origfirstwave [int32] - The unadjusted first waveform number.

- OrigWNumber [int16] - The unadjusted waveform number.
- WSDT [Date/Time] - The date and time of the start of the waveform.
- WSms [double] - The milliseconds which get added to the WSDT; used to determine the start time of the waveform down to the millisecond.
- WSDTmode [int32] - Waveform start date time mode; used to determine daylight savings time (see 7.2: Daylight Savings Time (DST), on page 7-2).
- WEDT [Date/Time] - The date and time of the end of the waveform.
- WEms [double] - The milliseconds which get added to the WEDT; used to determine the end time of the waveform down to the millisecond.
- WEDTmode [int32] - Waveform end date time mode; used to determine daylight savings time (see 7.2: Daylight Savings Time (DST), on page 7-2).
- WDuration [double] - Waveform duration in milliseconds from WSms to WEms.
- DateTimeValue [Date/Time] - The date and time of the record.
- WTDateTimeValue [Date/Time] - The date and time of the trigger.
- IndexValue [int32] - The index number of the record.

NOTE: For the Nexus® 1500 meter, Waveform data is stored in a CSV File (see 2.6.2: Nexus® 1500/1500+ Meter Comma Separated Value (CSV) Files, on page 2-18), not in this table. For all other meters, the waveform data is stored in this table.

3.29.2: Time Usage

Example:

WSDT has a value of 05/22/2015 11:11:22.0000 and WEDT has the same value 05/22/2015 11:11:22.0000.

1. You need to take the ms values, and add them to these timestamps to determine the duration of the waveform. Let's say WSms = 200 and WEms = 440. We can tell that the waveform has a duration of 240 ms.

2. Concatenate these ms values to their dates. Waveform Start = 05/22/2015 11:11:22.200; Waveform End = 05/22/2015 11:11:22.440.

NOTE: By using subtraction you can determine the duration of the waveform to be 240 ms, if you want to double check that the WDuration field is correct. (For table usage examples, see 2.6: Waveform Logs, on page 2-16.)

This page intentionally left blank.

4: Meters

This chapter explains how each meter uses logs and where its logs are stored. For instance, the Nexus® 1500/1500+ meter is unlike all the other meters because it has a separate database for each log type: Flicker, PQ, Waveform, etc. Each of these have their own databases to store all their data from the Nexus® 1500/1500+ meter. For all other meters, each of these logs is stored in the same database file. The Nexus® 1500/1500+ meter's method allows 2 GB of data to be stored for each log in each database, instead of all logs sharing one database of 2 GB.

4.1: MP200™ Metering System

The MP200™ metering system uses the "HOUR" tables for Historical logs (see , on page 2-2, for an example). All other logs are stored as described.

Path:

The device database is inside the RetrievedLogs folder:

…\Communicator_EXT\RetrievedLogs\MeterName.DB

4.2: Shark® 200 and Nexus® Series Meters

The Shark® 200 and Nexus® 1200 series can use either Channel tables or Hour tables for historical logs, as described in Section 2.1: Historical Logs, on page 2-1. All other logs are stored as described.

Path:

The device database is inside the RetrievedLogs folder:

…\Communicator_EXT\RetrievedLogs\MeterName.DB

4.3: Nexus® 1500/1500 Meter

The Nexus® 1500/1500+ meters can use either Channel tables or Hour tables for Historical logs, as described in section 2.1: Historical Logs, on page 2-1. All other logs are stored as described. However, each log type has its own database file instead of all logs being in one database.

Path:

The device databases are inside of the LOGS folder:

…\Communicator_EXT\RetrievedLogs\MeterName\LOGS\

- Inside the MeterName folder there is a DML file which is used to store the path to the meter's database files. For example, the Waveform table uses this file to show the path to its CSV files.
- Inside the LOGS folder are the individual log folders for each log type. Inside each folder is a .DB file, which is the log database.

Querying historical data is done the same way as described in 2.1.2: Channel Tables, on page 2-5. The only difference is that you must query the database holding the historical data, since each log type has its own database. The same applies to all other logs as well.

Example:

If you want historical 1 logs, you must query the database named Interval_1.

5: NameList.DB

The EIGNameList.DB is a separate database that stores data used to get the DataID for each meter. The database uses DataIds which are linked to descriptions in the NameList. This database also stores the FormatIDs for the Nexus® meters (see 5.5: Scaled Energy ID (SEID), on page 5-8). The meters have their DataIDs stored in their Modbus registers (see 5.4: Finding the DataID, on page 5-6). These registers are used to get the DataID from the EIGNameList.DB.

5.1: NameList Lookup

Once you have the DataId, you need to go to the EIGNameList.DB to find the description.

5.2: Tables

Each meter uses different tables in the NameList database (see 5.3: Usage, on page 5-3 and 5.4: Finding the DataID, on page 5-6, for details for each meter type.) The Shark® 200 meter uses the DeviceModbusMap table, the MP200^{TM} metering system uses the ShrkMF200 table, and the Nexus® Series meters use the DeviceProtocol_3_1 table to store their descriptions.

5.2.1: DeviceModbusMap (Shark® 200) Meter

- Description [Text] - The actual name of the value to which the global id is assigned.
- GlobalID [int32] - The number used by the software to help retrieve the labels.
- Device_ID [int32] - Linked to the Devices table. The index number in this field is used to determine the device.
- StartAddress [int32] - Modbus register used for storing the Global ID; add 200,000 to this number to get the GlobalID.
- EndAddress [int32] - The Modbus register's start address incremented by 1. Used by the meter to determine when the current data point ends and when the next begins.

5.2.2: ShrkMF200 (MP200 Metering System)

- Description [Text] - The actual name of the value to which the global id is assigned.
- GlobalID [int32] - The number used by the software to help retrieve the labels.
- Device_ID [int32] - Linked to the Devices table. The index number in this field is used to determine the device.
- StartAddress [int32] - Modbus register used for storing the Global ID: add 300,000 to this number to get the GlobalID.
- EndAddress [int32] - The Modbus registers start address incremented by 1. Used by the meter to determine when the current data point ends and when the next begins

5.2.3: DeviceProtocol_3_1 (Nexus® Series meters)

- Description1 [Text] - The actual name of the value to which the Index number is assigned.
- Description2 [Text] - The actual name of the value to which the global id is assigned.
- VIndex [int32] - point (see 5.4: Finding the DataID, on page 5-6).
- Pointer [int32] - line (see 5.4: Finding the DataID, on page 5-6).
- SEID [int16] - Scaled energy ID number (same as the FormatID from the ScaledEn-ergy table in the Log Database; see 5.5: Scaled Energy ID (SEID), on page 5-8).

5.2.4: ID_Table

This table is used to store information about the NameList.DB

- Version [double] - The database version.
- Date/Time Created [Date/Time] - The data/time the database was created.
- Name [Text] - The database name.
- Description [Text] - Comments on the database.
- Last Modification Date/Time By User [Date/Time] - The date/time of the last modification to the database.

5.3: Usage

Below are the queries needed for each meter to get their description(s), and a function showing how to get the description(s) from the EIGNameList.DB based on the meter type.

5.3.1: Queries

- Shark 200 = "SELECT Description FROM DeviceModbusMap WHERE GlobalID = "
- MP200 ™ metering system= "SELECT Description FROM ShrkMF200 WHERE GlobalID = "
- Nexus® Series meters= "SELECT Description1, Description2 FROM DeviceProtocol_3_1 WHERE Index = "

5.3.1.1: Examples

(These examples use the DataID; to find out how to find the DataID, see 5.4: Finding the DataID, on page 5-6.)

Nexus® meter:

```
long id = 20202;

nexus = query("SELECT Description1,Description2
              FROM DeviceProtocol_3_1
              WHERE Index = " + id);

string description = nexus.fields["Description1"] + " " +
nexus.fields.["Description2"];
//The description string above will be 1 Cycle Volts AN.
```

Shark® 200 meter:

```
long id = 201000;

shark200Q = query("SELECT Description
                 FROM DeviceModbusMap
                 WHERE GlobalID = " + id);

string description = shark200Q.fields["Description"];
//The description string above will be Volts AN.
```

MP200™ metering system:

```
long id = 301000;

mp200Q = query("SELECT Description
                FROM ShrkMF200
                WHERE GlobalID = " + id);

string description = mp200Q.fields["Description"];
//The description string above will be Voltage AN.
```

The queries shown can be combined to create a function which returns descriptions paired with DataID's:

```
private SortedList<long,string> GetDescriptions(ICollection<long> dataIds,
string device)
{//This function passes a collection of DataID's and the device type through its
//parameters, and returns a collection storing the description paired with its
//DataID

      SortedList<long,string> dataIdAndDescriptions = new
      SortedList<long,string>();

      if(device.equals("Shark 200"))
      {

                nameListQ = query("SELECT Description
                          FROM DeviceModbusMap
                          WHERE GlobalID = ");
       }

       elseif(device.equals("MP 200"))
       {

                nameListQ = query("SELECT Description
                          FROM ShrkMF200
                          WHERE GlobalID = ");
        }

        elseif(device.Contains("Nexus"))
        {

                nameListQ = query("SELECT Description1,Description2
                          FROM DeviceProtocol_3_1
                          WHERE Index = ");
          }

          //Loop through each ID in the collection.
          foreach(string id in dataIDs)
          {

        //Substring is needed to avoid the query from concatenating the DataID's
       // together with each loop.
       nameListQ = nameListQ.Substring(0,query.LastIndexOf("=")) + " = " + id;
       //Now query database.

       string desc = nameListQ.fields[0];//This will be the first field queried
       //try
       {//Try to get Description2. If it exists, concatenate it to
       // Description1
                 string desc2 = nameListQ.fields["Description2"];
                 dataIdAndDescriptions.Add(id,desc1 + " " + desc2)
                 continue;//If compiler reaches this line continue looping.
       }
       catch{}
       //If desc2 fails to add, add only desc and the ID to the collection.
       dataIdAndDescriptions.Add(id,desc);
   }
   return dataIdAndDescriptions;
}
```

5.4: Finding the DataID

This section shows how to get the DataID using information from the Modbus map.

5.4.1: Line and Point (Nexus® Meters)

Line and point are two numbers that the Nexus® Series meters use to find the DataIDs. The DeviceProtocol_3_1 table has two fields; pointer and VIndex, they are used by the meter to get the DataID that the software is referring to. This is done by using the two specific numbers: Pointer (int32) = line and VIndex (int16) = point. Using these two numbers, you are able to get the correct DataID.

5.4.1.1: Example

```
//Query the NameList.DB using the line and point numbers to retrieve the Data_ID

nameListQ = query("SELECT Index
                   FROM DeviceProtocol_3_1
                   WHERE Pointer = 18
                   AND VIndex = 0";

long dataID = nameListQ;//NameListQ returned 20402

//Now that the data Id has been retrieved you can get the descriptions from the
//table

descQ = query("SELECT Description1,Description2
               FROM DeviceProtocol_3_1
               WHERE Index = " + dataID);

//query will return, "High Speed Volts AN"
string description = descQ.fields["Description1"] + " " + descQ.fields["Descrip-
tion2"];
```

5.4.2: Start Address (Shark® Meters)

The start address is used by the Shark® 200 meter and the MP200™ metering system to determine the DataID. Inside the Modbus registers, every value has a number assigned to it. To find the global ID:

- Shark® 200 meter: add 200,000 to this number to get the DataID.
- MP200™ metering system: add 300,000 to this number to get the DataID.

5.4.2.1: Example

- Shark® 200 meter - Inside the Modbus map, Volts AN is assigned the address 1000. By adding 200,000 to this, you get 201000, which is the GlobalId for Shark 200 Volts AN readings.
- MP200 ™ metering system- Inside the Modbus map, Volts AN also is assigned the address 1000. By adding 300,000 to this, you get 301000, which is the GlobalID for MP200 Volts AN readings.

5.5: Scaled Energy ID (SEID)

The EIGNameList.DB also contains a table with fields needed for scaling energy. The Nexus® Series meters have a Scaled Energy ID (SEID) linked to their Data_ID (currently, only the Nexus® Series meters use this table).

The name of the table is DeviceProtocol_3_1. This is the same table where the Nexus® meter DataIDs are stored. Inside this table is a field named SEID. This is the scaled energy ID associated with the FormatID in the ScaledFormats table. Some DataIDs have the same SEID (the SEID is not unique to each entry), which is why you need to query the table using the DataID number.

Example:

Queries that may be used (for usage example, see 3.26.3: Usage, on page 3-44):

```
SELECT SEID FROM DeviceProtocol_3_1 WHERE Index =

SELECT SEID FROM DeviceProtocol_3_1 WHERE Description1 = ' ' AND
Description2 = ' '
```

6: Identification

Not all databases use the same format. As mentioned in Chapter 2, some databases use "HOUR" table format and some use "CHANNEL" table format. There are also earlier database formats which use different field names.

6.1: Usage

- The ID_TABLE is used to identify the format of the database (97 and 2003).
- The format can be found in the KeyName field.
- You will either see the value "Master DB" (which means the database is using 97 format), or "Master DB 2003" (which means the database is using 2003 format).
- Differences in these database formats are the field names (see 6.3: Field Differences, on page 6-3).
- There is no DB_SETTINGS table in the 97 format.

6.2: Identify

To identify which format the database is using, you need to query the ID_TABLE and DB_SETTINGS table. This section shows you how to determine the database format, and what the software should do when it detects new or old formats.

Example:

```
string KeyName = query("SELECT KeyName
                        FROM ID_TABLE");
if(keyName.Equals("Master DB") && userWantsNewDataBaseFormat)
{
    UpgradeDatabase();//Upgrade database before anything else
    string format = GetTableFormat();//See 2.1: Historical Logs, on page 2-1
    if(format.Equals("CHANNEL"))
    {
        ChannelTableWork();
    }
    else
    {
        HourTableWork();
    }
}
else if(keyName.Equals("Master DB") && !userWantsNewDataBaseFormat)
{
    HourTableWork();
}
else if(keyName.Contains("2003"))
{
    string format = GetTableFormat();//See 2.1: Historical Logs, on page 2-1
    if(format.Equals("CHANNEL"))
    {
        ChannelTableWork();
    }
    else
    {
        HourTableWork();
    }
}
private void HourTableWork()
{
    //Process Hour Tables
}private void ChannelTableWork()
{
    //Process Channel Tables
}
```

6.3: Field Differences

97	2003
Date/Time	DateTimeValue
Date/TimeIndex	DateTimeIndex
StartDate/Time	StartDatetimeValue
EndDate/Time	EndDateTimeValue
WDate/Time	WDateTimeValue
Value	ItemValue
Index	IndexValue
PQ%	PQ_Percent
LSet%	LSet_Percent

This page intentionally left blank.

7: Charts

This section contains charts which are used by multiple tables.

7.1: Source ID (srcid)

The chart below contains all of the srcids used. The left side is the srcid index, and the right side of this chart is the actual log type.

-1	Unknown/Ignore
0	Out of DST
1	In DST
2	Interval 1
3	Interval 2
4	Limit Log
5	Digital In
6	Digital Out
7	Flicker
16	Interval 3 (Shark® meter)
18	Interval 3
19	Interval 4
20	Interval 5
21	Interval 6
22	Interval 7
23	Interval 8
24	Event Triggered

7.2: Daylight Savings Time (DST)

The chart below contains the index numbers indicating whether or not the timestamp represents daylight savings time.

-1	Unknown/Ignore
0	Out of DST
1	In DST