AcquiSuite XML-Upload User's Guide

by

Mark N. Shepard

Applies to: AcquiSuite firmware v2.11.0808 or later

Document revised: 8/12/2011

(changes from 7/29/2011 revision are marked in red)

Copyright Information

Limited Warranty

OBVIUS IS PROVIDING THIS WARRANTY IN LIEU OF ALL OTHER EXPRESS OR IMPLIED WARRANTIES, INCLUDING ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. THIS WARRANTY IS BUYER'S EXCLUSIVE REMEDY FOR ALL CLAIMS AGAINST OBVIUS. OBVIUS SHALL NOT BE LIABLE FOR ANY CONSEQUENTIAL OR INCIDENTAL DAMAGES. OBVIUS'S TOTAL LIABILITY FOR ALL CLAIMS SHALL BE LIMITED TO THE PRICE PAID FOR ITS PRODUCT.
Obvius promises buyer that any standard product manufactured by Obvius shall be free from all material defects in design, material, or manufacturing for a period of 2 years from the manufacture date; provided, however, that the warranty shall not extend to ordinary wear and tear or to normally replaceable components (e.g., batteries). During the warranty period, Obvius may repair or replace (in its sole discretion) any product suffering from a warranty defect and returned freight prepaid by buyer, with no charge to buyer for any warranty repair or replacement. The warranty shall remain in full force and effect for such 2 year period, provided that the product: (1) was installed, operated, and maintained properly; (2) has not been abused or misused; (3) has not been repaired, altered, or modified outside of Obvius's authorized facilities; (4) has not been sold subject to other warranty terms specified at the time of sale; and (5) is still owned by the original purchaser. This warranty provides specific legal rights that may be varied by state law. Obvius's products are not designed for life or safety applications.

Product Application Limitation

Obvius products are not intended for use in critical applications such as nuclear facilities, human implantable devices or life support. Obvius is not liable, in whole or in part, for any claims or damages arising from such uses.
Obvius strongly believes in continuous improvement, therefore we must reserve the right to change specifications and product offerings without notice. Where possible, we will substitute products with equivalent functionality when necessary.

Obvius
3300 NW 211th Terrace
Hillsboro, OR 97124
ph: 503-601-2099

Table of Contents

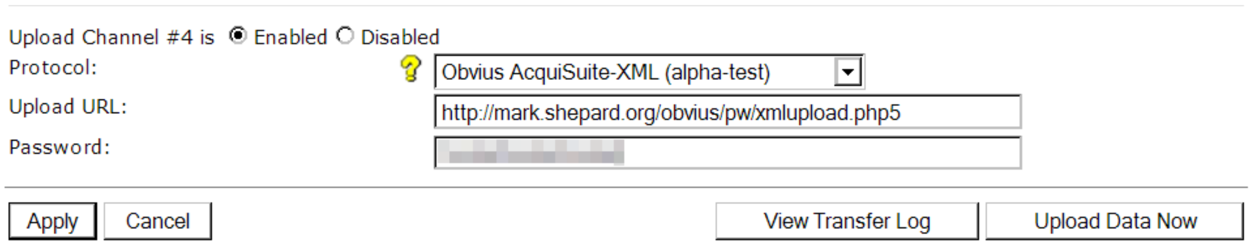
1 Overview of the AcquiSuite-XML Protocol

The 'AcquiSuite-XML' protocol is intended to be simple above all else.

Features:

- Will be part of the standard AcquiSuite firmware on Obvius' Data Acquisition Server models: A8812, A8810 and A7810.
- Simple, stateless client-server protocol based on XML and HTTP.
- AcquiSuite sends plain-text XML messages to your server, and expects an XML-formatted reply.
- Two types of XML messages: STATUS, showing health of AcquiSuite, and LOGFILEUPLOAD, containing meter readings.
- Flexible interpretation of common HTTP error messages for easy integration with existing web-app frameworks with minimal customization.
- Compatible with NAT (Network Address Translation); the AcquiSuite only makes outgoing HTTP connections, just like a web browser.
- Supports HTTP Proxies if the AcquiSuite is so configured.
- Supports HTTP Basic Authentication.
- Does not allow remote administration of the AcquiSuite; if this is required, please use the Obvius BuildingManagerOnline protocol.
- Does not allow any type of HTTP 3xx Redirects from the server.
- The XML format used when uploading data to your server is the same as the XML format used to query real-time meter data from the AcquiSuite. This lets you use the same parsing and rendering code to display historical and real-time meter data.

In this protocol, the AcquiSuite acts as an HTTP client. The AcquiSuite is configured to use the 'AcquiSuite-XML' protocol by enabling an upload channel, specifying a Upload URL and an optional password.



Upload Channel #4 is	[✓ Enabled] Disabled
Protocol:	Obvius AcquiSuite-XML (alpha-test)
Upload URL:	http://mark.shepard.org/obvius/pw/xmlupload.php5
Password:	[redacted]

During each upload cycle, the AcquiSuite will make one or more HTTP requests to the specified Upload URL. Each HTTP transaction will be a POST request, of Content-Type: text/xml, and the body of the HTTP POST will contain an XML message, the format of which is defined below.

The character encoding is US-ASCII. Characters outside of US-ASCII are encoded as HTML character-entities.

All HTTP requests and replies are in plain, uncompressed text.

Only the HTTP Basic Authentication scheme is supported.

The AcquiSuite sends two types of XML messages, which we call "STATUS" and "LOGFILEUPLOAD".

In each of these messages, the particular AcquiSuite unit making the request identifies itself by including its serial number (which happens to be its Ethernet MAC address) as part of the XML message.

All requests in this protocol are 'stateless' and 'idempotent': The AcquiSuite does not use any mechanism such as Cookies or session-ID's to keep track of session state.

2 Log Cycles vs. Upload Cycles

The AcquiSuite data acquisition server performs two general types of activities – logging data, which means sampling the data points of all connected devices (meters or sensors) at a user-configured, regular interval – and uploading data to the server at a separately configured interval. Uploads may also happen off the schedule: The AcquiSuite may be configured to upload on changes in alarm conditions and it may be configured to perform additional “retry” upload cycles if a scheduled upload cycle fails due communications problems.

During each logging cycle, the AcquiSuite samples every meter or sensor in the system and stores a <record> in its flash memory containing the value of each <point> sampled.

Each <record> is time-stamped showing when the sample was taken. Note that at slow Modbus baud-rates or with many meters on a single Modbus the time required to query each meter results in skewing between the time-stamps of <record’s> sampled during a single logging cycle.

Periodically, the AcquiSuite performs an upload cycle, during which it uploads all accumulated samples from its flash-memory to the server. Any data which could not be uploaded on previous upload-cycles is uploaded first, followed by the most recent set of <record’s>.

The AcquiSuite's user-interface currently imposes the following limits on sampling and upload rates:

- Logging cycle periods can range from as fast as once-per-minute to as slow as once-per-hour. Each log cycle samples and stores to flash memory a single <record> from each device (The format of a <record> is defined below).

- Upload cycle periods can range from as fast as once-per-logging-cycle (i.e., once-per-minute) to as slow as once-per-day.

These limits may be changed in future firmware releases to allow the option of configuring either more or less frequent sampling or uploading.

2.1 Server Provisioning and Maximum XML Message Size

When choosing XML parsing algorithms and provisioning your server's memory, consider these numbers:

- The number of <point>'s per meter can vary from as low as 1 or 2 for some classes of devices, to as high as ~420 for devices such as the Veris 42-channel Branch Current Monitors (each channel has 8 data-points, plus several dozen points not associated with channels).
- The maximum number of meters attached to a single AcquiSuite can be quite large, though the practical number depends on the response time of the devices and of course the amount of data being logged: Modbus devices use an 8-bit address (0-255), and that naturally limits the number of Modbus devices, but the AcquiSuite can also represent devices by <serialnumber>.
- The *maximum logging rate* of the AcquiSuite is 1/minute. The *maximum upload period* is 1/day. These two factors together mean that, worst-case, a single AcquiSuite could be configured to collect 60 * 24 = 1,440 data <record>'s from each meter in a single day.
- Thus, the total amount of data uploaded per day, per AcquiSuite, would be the product of all of the above, times the usual factor for the verbosity of XML ☺

As the AcquiSuite's flash memory starts to fill up, the AcquiSuite will seek to consolidate small files of log-data into fewer, larger files for more efficient storage. A side-effect of this is that if the AcquiSuite has been unable to contact the server for a while, when communication is re-established and the AcquiSuite resumes uploading data, that data will be delivered a fewer yet larger XML messages.

Example #1 – Veris H8036 power meter:

The Veris H8036 power meter provides 26 <points>. If this meter is attached to an AcquiSuite configured to log data once per minute, then each minute, 1 <record> will be created containing 26 <points>. In one hour, 60 <records> containing a total of 60*26 = 1,560 <points> will be generated and stored to the AcquiSuite's flash-memory. Assuming ~80 bytes per <point> when encoded as XML, this single meter will produce ~80*1560 = ~125 kB per hour or ~3 MB per day.

Example #2 – Veris E30 Branch Circuit Monitor meter:

The Veris E30A084 is a 42-channel device. Each channel consists of 8 <points>, plus the meter has 46 other <points> not associated with any channel for a total of 42 * 8 + 46 = 382 <points>. If the Acquisuite is configured to log data once per minute from this meter, in one hour, 60 <records> containing a total of 60*382 = 22,920 <points> will be generated. At ~80 bytes per <point>, this single meter will produce ~80*22920 = ~1,834 kB per hour or 44.0 MB per day! If the logging rate is dropped to once every 15 minutes (4 times per hour), the data rate is reduced by a factor of 15.

3 The Upload Cycle

Periodically, at a configurable interval, each AcquiSuite unit in the field will perform an "upload cycle" to upload all queued meter data in its flash memory to the server.

Each upload cycle will begin with the AcquiSuite sending a "STATUS" XML message, followed by it sending one or more "LOGFILEDATA" XML messages.

3.1 Verifying Message Integrity

The AcquiSuite relies on TCP checksum and the HTTP Content-Length header to detect corrupt or truncated messages.

The most critical task the server must perform when it receives an XML message from the AcquiSuite (in the form of an HTTP POST) is to verify that the message is complete – meaning it was not truncated.

The server should parse the HTTP Content-Length header and verify that the HTTP body matches this length.

If the message somehow was truncated, and the server does not detect this and sends a 'SUCCESS' reply to the AcquiSuite, the AcquiSuite will delete the supposedly-successfully-uploaded data from its flash memory.

3.2 The "STATUS" Message

The "STATUS" XML message provides the server with overall health and status of the AcquiSuite, such as its uptime, the % of free disk space, the firmware version it is running, etc.

```
-------------------- EXAMPLE OF AcquiSuite 'STATUS' XML MESSAGE ----------------
<?xml version="1.0" encoding="US-ASCII" ?>
<DAS>
  <mode>STATUS</mode>
  <name>A8812 Dallas CNT:US CIT:DFW &amp; BLD:Mark1 &lt;mark@shepard.org&gt;</name>
  <serial>0050C230EF52</serial>
  <uptime>145200</uptime>
  <percentblocksinuse>58</percentblocksinuse>
  <percentinodesused>0</percentinodesused>
  <acquisuiteversion>v02.11.0718b</acquisuiteversion>
  <usrversion>v02.11.0714b</usrversion>
  <rootversion>v02.11.0714b</rootversion>
  <kernelversion>2.6.28.10-r1.92b-ge055112</kernelversion>
</DAS>
--------------------------------------------------------------------------------
```

3.2.1 The <serial> field identifies an AcquiSuite unit

The <serial> field provides the serial number of the particular AcquiSuite, assigned by Obvius. This is actually the Ethernet MAC address of the unit. This should be used to uniquely identify each AcquiSuite unit in the field.

In contrast, the <name> field is assigned by the user when the AcquiSuite is installed and may be changed at any time. This can also be used to contain identifying information if you wish to establish your own convention.

3.3 The "LOGFILEUPLOAD" Message

The "LOGFILEUPLOAD" XML message delivers the actual meter data to the server.

```
----------------- EXAMPLE OF AcquiSuite 'LOGFILEUPLOAD' XML MESSAGE ------------
<?xml version="1.0" encoding="US-ASCII" ?>
<DAS>
  <mode>LOGFILEUPLOAD</mode>
  <name>A8812 Dallas CNT:US CIT:DFW &amp; EML:Mark1 &lt;mark@shepard.org&gt;</name>
  <serial>0050C230EF52</serial>
  <devices>
    <device>
      <name>DEV:Veris H8238 MTR8</name>
      <address>72</address>
      <type>Veris Multi-Circuit Power Meter 8, H8238, PMM1, MTR8</type>
      <class>7</class>
      <numpoints>28</numpoints>
      <records>
```

```
        <record>
             <time zone="UTC">2011-07-18 00:38:24</time>
             <error text="Ok">0</error>
             <point number="0" name="Energy Consumption" units="kWh" value="0" />
             <point number="1" name="Real Power" units="kW" value="0" />
             <point number="2" name="Reactive Power" units="kVAR" value="0" />
                  ...
             <point number="27" name="Maximum Demand" units="kW" value="0" />
        </record>
        <record>
                  ...
        </record>
        <record>
                  ...
        </record>
      </records>
    </device>
  </devices>
</DAS>
--------------------------------------------------------------------------------
```

3.3.1 The <device> field identifies the device (meter or sensor)

Each "LOGFILEUPLOAD" XML message contains one or more <device> fields, which identify the particular meter, sensor or other data-reporting device for which readings are being uploaded, and for each device, the LOGFILEUPLOAD message will contain one or more <record> fields, each of which will contain the values of all data-points, sampled from the device at a particular instant in time.

A device (meter, sensor, etc.) is identified in two ways:

- by an <address> field (shown below) containing the Modbus Address of the meter or sensor. The <address> value may be from 0 to 255. Note that Modbus Addresses can and do change. These are typically set on DIP switches or in software when the meter is installed.

- by a <serialnumber> field (not shown, but at the same level as the <address>), which contains the unique serial number of the meter or sensor (not to be confused with the *AcquiSuite's* serial number). Serial numbers do not change; they are set at the factory when the meter or sensor is manufactured.

IMPORTANT:

The device serial number can be up to 128 characters and should be treated as case-sensitive.

Some devices have ONLY a Modbus address (no serial number). Others have ONLY a serial number (no Modbus address). Some devices have both.

If a device has BOTH a Modbus address and a serial number, the server should use the Modbus address to identify the device; this is what the AcquiSuite will do. In this case, the serial number can either be ignored (for simplicity) or it can be stored. If the Modbus address remains the same but a serial number changes, this usually means the meter was replaced with a different hardware unit.

3.3.2 The <device><type> field

The <type> field provides a string description of the type of meter or sensor. This is typically the result of a Modbus Identify query from the meter itself.

3.3.3 The <device><class> field

The <class> field is an arbitrary number, usually assigned by Obvius. Class ID numbers may also be assigned by the user to a new class of meters using the AcquiSuite's Modbus Framework feature.

If two devices have identical <class> ID values, this means that the set of <points> and the meaning and <units> of each of those <point>'s is identical. In other words, if you imagine an abstract type (i.e., C "struct" or C++ object) representing a device, devices with identical class IDs can be represented by the same abstract type – or by an identical database table structure.

Usually, two meters with identical <class> IDs will have the same number of points (i.e., <numpoints>). At worst, if one meter has a greater <numpoints>, the points in common will nevertheless have identical meaning and <units>.

Several meters would share a single <class> ID if they provide the same set of data points, yet differ in power rates or CT factors, or if one meter is a "re-branded" version of another.

If the <class> ID of a particular device changes within an upload session or between successive upload sessions, this usually means that the physical meter was replaced with another of an incompatible type (or the Modbus bus addresses were reassigned to achieve the same result). In this case the server should return an HTTP 409 Conflict error since (by definition) the meaning and <units> of the <points> will be arbitrarily different and very likely completely incompatible.

3.3.4 The <device><numpoints> field

<numpoints> gives the number of <points>.

Note that <numpoints> may increase for a particular device within an upload session or between successive uploads. This should be rare. Assuming the <class> ID is unchanged, this indicates that a <point> or points were added to the device. This usually corresponds with a firmware upgrade – i.e., the data records logged before the AcquiSuite firmware will have one <numpoints> value, while the data records logged after the AcquiSuite firmware upgrade will have a higher <numpoints> value.

From the server's point of view, when the AcquiSuite uploads all its data records, <numpoints> will appear to increase within the data stream from the device.

3.3.5 The <device><name> field

The <name> field is assigned by the customer or installer to a particular meter or sensor. This is an arbitrary string.

3.3.6 The <device><record> field

Each <record> represents an event which occurred at an instant in time and which has been recorded by the AcquiSuite.

Usually, the "event" in question is the meter's data points being sampled, but other events are also recorded in the data stream of <records> from a particular device, such as attempts to sample which resulted in Modbus errors (e.g., because the device was offline or did not respond), or AcquiSuite system events such as starting or stopping of logging for a particular device (e.g., because the AcquiSuite is rebooting). The <error> field tells you which event occurred.

Each <record> field contains:

- a <time> field, which is the absolute time (in UTC) when the sample was collected by the AcquiSuite. The resolution of these time-stamps is 1 second;
- a <error> field, which is the Modbus Error Code returned by the device (or 0 for success). The <error text="…"> attribute gives the error text corresponding to the error code;
- one or more <point ...> fields which provide the value of each data point, along with its name and units.

3.3.7 The <record><error> field

The <error> field contains the numeric error, with a "text" attribute giving the text corresponding to the error code. As explained above, it tells you which type of "event" occurred at the <time> of this <record>. The most common type of event by far is that a sample of the meters data points was collected. In this case, <error> will be 0 indicating data was successfully sampled from the meter by the AcquiSuite.

If <error> is not zero, its value will be either a POSIX/Linux "errno" code, a Modbus error code or an AcquiSuite-defined error code. These 3 types of error codes do not overlap.

If an error occurs due to an internal problem with the AcquiSuite or with TCP/IP networking, <error> will contain a Linux errno value as follows. If the error is network-related, it typically means the meter is being accessed by Modbus/TCP. Other errors are possible – only the most common are shown.

5	Input/output error (EIO)
100	Network is down (ENETDOWN)
101	Network is unreachable (ENETUNREACH)
102	Network dropped connection because of reset
103	Software caused connection abort (ECONNABORTED)
104	Connection reset by peer (ECONNRESET)
105	No buffer space available (ENOBUFS)
106	Transport endpoint is already connected (EISCONN)
107	Transport endpoint is not connected (ENOTCONN)
108	Cannot send after transport endpoint shutdown (ESHUTDOWN)
110	Connection timed out (ETIMEDOUT)
111	Connection refused (ECONNREFUSED)
113	No route to host (EHOSTUNREACH)

If an error occurs trying to sample the device, <error> will contain the particular Modbus Error Code:

128	Error out of range
129	Illegal function (function not allowed by slave device)
130	Illegal data address
131	Illegal data value
132	Slave device failed to handle the query
133	Acknowledge
134	Slave device busy
135	Negative Acknowledge
136	Memory parity error

137	Error undefined
138	Gateway path unavailable
139	Device failed to respond
140	Received invalid data checksum (check for Modbus address conflicts)
141	Received response from unexpected device
142	Received unsolicited query, assume another device is present
143	Modbus device probe function got some good and some error message replies

Obvius also defines some additional <error> values outside the normal range of POSIX and Modbus errors to indicate events related to the AcquiSuite system itself:

160	Start Logging
161	Stop Logging
162	AcquiSuite system time change
163	AcquiSuite system auto-restart
164	Log entry corrupt
165	Device restart
192	Device does not match configuration

<error> 192 indicates the AcquiSuite itself has detected a conflict of <class> ID values between its configuration file for a device and what the AcquiSuite currently detects as the device's class when probing the device.

3.3.8 The <record><point> field

Each meter or sensor has one or more data points. For Modbus devices, a data point is typically read by reading a particular Modbus register such as 40001, where the exact register number depends on the meter's class. Since all Modbus registers are 16-bits wide, some data points may require reading more than a single Modbus register to get the point's value (e.g., two Modbus registers must be read if the data point is a 32-bit float value).

Meters provide data in a wide variety of formats: big- or little-endian, signed or unsigned integers, fixed- or floating point, IEEE floats or even custom floating-point formats. The AcquiSuite takes care of converting all of these disparate formats into engineering units and numeric values encoded as decimal or hexadecimal strings.

Each time the AcquiSuite performs a logging cycle it will sample the current readings from each meter or sensor and create a corresponding <record>. This <record> will contain a <point> for each data point the meter or sensor provides.

A single <point> has the following attributes:

3.3.8.1 The <point number="…"> attribute

The AcquiSuite assigns a sequential index number to each point ranging from 0 to <numpoints>-1 and the number= attribute provides this number, in decimal. Note that <points> may appear in any order within the XML message!

3.3.8.2 The <point name="…"> attribute

The <point's> name= attribute provides a string which is a human-readable description of the data point. Examples: Energy Consumption, Phase-A Current, Frequency, Channel 1 Pulse Count

Some meters allow the AcquiSuite's installer to customize the names of data points – this is typical on multi-channel meters. Other meter's data point names are fixed strings and cannot be changed by the user.

3.3.8.3 The <point units="…"> attribute

The <point's> units= attribute provides the abbreviation for the engineering units of the data point.

Note that the units of some data points may be partially configurable by the user. For instance, the units of pulse count inputs may be configured to be gallons per hour (gph) or gallons per minute (gpm).

3.3.8.4 The <point value="…"> attribute

Finally, the value= attribute gives the points value as a decimal number, rendered as a string. If the value= contains a "." (decimal point) it should be parsed as a float using the sscanf() "%f" format, otherwise integers may be parsed with the sscanf() "%i" format. Hexadecimal integers will be prefixed with "0x".

In some cases, a <point> may not *have* a value; the AcquiSuite indicates this by sending **value="NULL"**.

Typical reasons a point would not have a value:

- The AcquiSuite got an error trying to query the meter, or
- the meter is in "Consumption Only" mode and the point is not an Energy point, or

- the meter reported a float "NaN" value (NaN = Not-a-number) or other invalid numeric value, or
- the meter reported an over- or under-range value (off-scale high or off-scale low).

If the <record><error> field shows a non-zero (error) code, all the <point's> will usually have value="NULL".

Example:

```
   <point number="1" name="Real Power" units="kW" value="NULL" />
```

4 The Server's Reply

The AcquiSuite expects to receive an XML message in reply to each of its POST's. This XML reply has the same format for both the STATUS and LOGFILEUPLOAD messages.

```
--------------------------- REPLY INDICATING SUCCESS ---------------------------
<?xml version="1.0"?>
<DAS>
  <result>SUCCESS</result>
</DAS>
--------------------------- REPLY INDICATING FAILURE ---------------------------
<?xml version="1.0"?>
<DAS>
  <result>FAILURE: one-line error message without HTML tags</result>
</DAS>
--------------------------------------------------------------------------------
```

'SUCCESS' tells the AcquiSuite that its message was received and processed correctly. In the case of a LOGFILEUPLOAD message, this tells the AcquiSuite that the server has stored the meter's data and the AcquiSuite can now delete the data from its flash-memory.

'FAILURE' is typically returned by the server if the server is unable to process the request for some reason such: out of disk space, database down for maintenance, AcquiSuite's serial number is not recognized, password is invalid, etc.

A one-line error message may optionally be included with the 'FAILURE' keyword. This is displayed by the AcquiSuite in its Connection Test and also in its system log files of upload sessions.

The XML reply (whether a 'SUCCESS' or 'FAILURE') may also contain an optional set of <notes>, like so:

```
------------------------ EXAMPLE SHOWING OPTIONAL <notes> ----------------------
<DAS>
  <result>...</result>
  <notes>
    <note> connecting to database server: foobar </note>
    <note> creating table xyz for serial-number 123456789ABCDEF </note>
    <note> storing data... </note>
    <note> data stored successfully </note>
    ...
  </notes>
</DAS>
--------------------------------------------------------------------------------
```

These <notes> are not parsed by the AcquiSuite -- they are included in the AcquiSuite's upload log (but only if the AcquiSuite's logging level is set to 'Full Debug with Trace'. The intent of the <notes> field is to provide a free-form area where the server can output debugging or diagnostic information to help the AcquiSuite's installer or user debug problems.

4.1 How the AcquiSuite Responds to Errors

When parsing the server's reply, the AcquiSuite also interprets the HTTP status code.

The AcquiSuite tries to use a "common-sense" interpretation of the HTTP status code to make it easy to integrate into existing web frameworks where it may not always be possible to control the specific HTTP error in all cases.

In cases of an upload failure, the server can influence exactly how the AcquiSuite responds to the failure by the particular HTTP status code the server returns.

For instance, normally, the server would reply to the AcquiSuite with an XML message (as defined above) and with HTTP status code "200 OK", regardless of whether the reply was a 'SUCCESS' or a 'FAILURE'. By default, the AcquiSuite treats all 'FAILURE's as transient communications problems, and so will retry any 'FAILURE's several times during a single upload session (the number of retry attempts can be configured on the AcquiSuite).

Note that in all cases where the server fails to accept data, the data is left in the AcquiSuite's flash memory. When the AcquiSuite's flash memory fills up, data records are deleted in oldest-first order to make room for new data. This means that if the server does choose to reject data with a 409 or 406 error and the problem cannot be resolved in a timely fashion, the AcquiSuite will eventually delete the data and continue normal operation without any human intervention required.

4.1.1 "Authentication" Errors – permanent, affects upload session

Some failures -- such as a bad password failure or invalid URL – are considered "permanent" instead of "transient", in that they usually cannot be fixed by retrying immediately, and retrying simply wastes server CPU time and network bandwidth. If the server replies with any of the HTTP status codes below, the AcquiSuite will abort the entire upload session:

HTTP 401 Unauthorized

HTTP 402 Payment Required

HTTP 403 Forbidden

HTTP 404 Not Found

HTTP 407 Proxy Auth Required

The server can also return one of the above HTTP errors if the server is down for maintenance and wishes to minimize the bandwidth while it is offline.

4.1.2 Device <class> Conflict Errors – permanent, affects data stream from a device

The server can tell the AcquiSuite to skip the remaining data for the current device (meter or sensor) -- and continue with the next device -- by returning:

HTTP 409 Conflict

This lets the server continue uploading data from other devices, while skipping over one particular device.

The HTTP 409 Conflict code is typically used when the server cannot resolve format differences between its database and the latest data from a device. This typically happens if a meter is replaced with another meter of an entirely different class (and so, by definition, the meter's set of data-points is incompatible with the previous meter's). Rather than upload the data and creating havoc by trying to "round pegs in a square database", the server can reject the device's data, leaving it on the AcquiSuite until the problem can be resolved (this typically requires human intervention.)

4.1.3 Data Not Acceptable Errors – permanent, affects a block of <record>s

Finally, the server can tell the AcquiSuite that a particular XML message was received correctly but due to a problem with the message's contents cannot be accepted by returning:

HTTP 406 Not Acceptable

This causes the AcquiSuite to skip over the particular block of <record>'s of data contained in the XML message but to continue with the next block of <record>'s (if any) for the current device. This would typically be used if the data values appeared corrupt or at least cannot be interpreted with certainty by the server. Like 409 Conflict problems, this also typically requires human intervention.

5 Sample Code to Receive Uploads

5.1 *Minimal PHP Sample to Receive ‘AcquiSuite-XML’ Uploads*

```php
<?php
// xmlupload_minimal.php -- absolutely minimal PHP script to
//              receive and valid XML and reply to AcquiSuite.
//       by Mark N. Shepard
//       $Id: xmlupload_minimal.php,v 1.2 2011/08/02 17:50:31 mns Exp $

// Uncomment to disable uploads if server is down for maintanence.
// header("Status: 503 Server Busy");
// exit(0);

$contentLength = intval($_SERVER['CONTENT_LENGTH']);

$body = file_get_contents('php://input');

// Critical: Verify we received the ENTIRE post of data from the AcquiSuite
// by comparing the actual length of $body against the Content-Length:
// header. If the message from the AcquiSuite was truncated, fail.

if ($contentLength <= 0 || strlen($body) < $contentLength)
{       header("Status: 400 Bad Request");
        exit(0);
}

// Parse the XML to verify it hasn't been truncated; assumes that PHP will
// throw an exception and bomb out if the XML is syntactically invalid.
//
// For a good intro to XML parsing in PHP, see:
//   "XML for PHP developers, Part 2: Advanced XML parsing techniques"
//   IBM developerWorks
//   http://www.ibm.com/developerworks/xml/library/x-xmlphp2/index.html

$dasMsg = new SimpleXMLElement($body);

// TODO:  insert code to do stuff w/ $dasMsg

header("Status: 200 OK");
header("Pragma: no-cache");
header("Content-Type: text/xml");

// Send 'SUCCESS' response to AcquiSuite; this will allow AcquiSuite to delete data
// on its end. Only do this if you are ABSOLUTELY SURE XML you received is valid
// and you have properly stored or processed it!

printf("<?xml version=\"1.0\"?>\n");              // XML response to AcquiSuite.
printf("<DAS>\n");
printf("<result>SUCCESS</result>\n");
printf("</DAS>\n");
?>
```

6 References for XML Parsing

For a good intro to XML parsing in PHP, please see:

XML for PHP developers, Part 2: Advanced XML parsing techniques
IBM developerWorks
http://www.ibm.com/developerworks/xml/library/x-xmlphp2/index.html

7 Frequently Asked Questions

Q1: Why isn't the XML compressed with one of the standard HTTP compression methods?

A1: Obvius' BuildingManagerOnline protocol already provides gzip-compression and is much more efficiently encoded than XML. The AcquiSuite-XML protocol was intended to be very simple to use with existing web frameworks, and, unfortunately, while HTTP can negotiate a compression-method for the server's response, HTTP has no standard mechanism to negotiate compression on the client's [the AcquiSuite's] initial request. In this respect, HTTP is optimized for downloading rather than uploading content from the server.

Q2: How can we get the "current status" or "health" of a meter, for display in a user-interface or dashboard?

A2: The "status" of a meter or sensor is only knowable when the AcquiSuite tries to sample it, therefore the "current status" is simply the <error> code and text from the most recently dated <record> uploaded from the meter. If the <error> code is 0, the meter functioned properly when sampled.

The "staleness" or "freshness" of this information is determined by how often the AcquiSuite is configured to log data from the meter and how often it uploads data to your server.

Q3: When the user "focuses on" or selects a meter in our dashboard, how can we obtain the very latest data from the meter? How can we boost the sampling rate to show real-time data?

A3: The best way is to query the AcquiSuite directly, using HTTP from your server. The AcquiSuite allows the current status and current readings of any meter or sensor to be queried via an HTTP request to the AcquiSuite. Such a query also causes the AcquiSuite to immediately refresh its data from the meter; repeated queries will provide new data every 3 seconds. This real-time data is provided in the AcquiSuite-XML format, so it should be easily compatible with the historical data uploaded by the AcquiSuite.