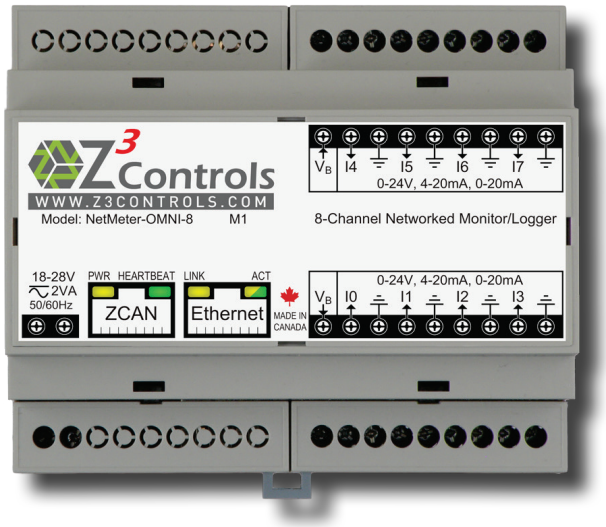
Online Support Portal:
help.z3controls.com



NetMeter-OMNI

Commercial/Industrial Monitor/Logger with Integrated Networking

Data Push API Guide

Data Push Application Programming Interface

Contents

1 Important Notice

During normal use, potentially lethal voltages are connected to the NetMeter hardware. Consequently, the NetMeter hardware module should only be installed and serviced by a qualified electrician.

Please read and follow the Installation Manual for all guidelines and safety procedures associated with the installation and standard operation of this hardware. Any specific application of the NetMeter system should be in accordance with your local standards and practices.

Under no circumstances will Z3 Controls Inc. (Z3 Controls) be responsible or liable for any direct, indirect, or circumstantial damages associated with the usage or application of this equipment. No patent liability will be assumed or associated with Z3 Controls with respect to the usage of information, equipment, circuitry, software or practices described within this manual.

Always check with the Z3 Controls web site for the most up-to-date version of this documentation.

2 Purpose

The purpose of this document is to describe the client-mode Application Programming Interface (the 'API') which allows the NetMeter to "push" data to a remote server using TCP/IP communication.

Potential applications of the client API may include:

- Aggregation of data from multiple NetMeter devices to a centralized server
- Allow NetMeter data to be transferred remotely as an HTTP client and thus overcome firewall issues

The client mode push API is in addition to the server mode API described in a separate document.

3 Introduction

3.1 License

The API is designed to enable interested parties, such as third-party developers, small businesses, technical enthusiasts and students, to incorporate a Z3 Controls product into their custom application as a potential solution. Although the API is the intellectual property of Z3 Controls, Z3 Controls grants customers the right to use the API free of charge, under the terms of the license agreement found on the Z3 Controls website, www.z3controls.com/tla.php.

Please familiarize yourself with the license agreement before implementing the API.

3.2 Overview

Key features of the client mode push API:

- uses the HTTP or HTTPS application layer protocol over TCP/IP
- may be secured using HTTPS (HTTPS is recommended for all public internet communication to prevent interception)
- Data is sent using a simple HTTP POST or GET query
- A server generated API key is used to authenticate credentials
- Data may be synchronized so that the entire contents of the datalog can be replicated on the server. This allows catching-up after long periods of broken communication.

In this document, the term "NetMeter" refers to the NetMeter-OMNI product.

4 NetMeter Push API Configuration

In order to configure the NetMeter to push data to the remote server there are parameters that need to be configured. This is accomplished using the user interface found at: **Setup** → **Data Push Configuration**. An example is depicted below in Figure 1.

4.1 Setup

The configuration fields are:

- Push Service Enable: must be checked in order for the push service to push data.
- Host URL: Either an IP address or the DNS name of the remote server and the path within that server. However, do not include the `HTTP://` or `HTTPS://` at the beginning as you would for a browser URL.
- Protocol: whether or not the access is to be encrypted (HTTPS) or non-encrypted (HTTP)
- Port Number: The port number of the remote server: port 80 is typical for HTTP access and port 443 is typical for HTTPS access
- Server Timeout: how long to wait for a server response before giving up
- API Key: enter the server assigned API Key here. The API key may be used by the server to validate and route data from the NetMeter.
- Method: what HTTP method to use: POST or GET[1]
- Data Packet Format: determines if the data is formatted as JSON or CSV
- Data to Transmit: reports on the list of data that will be pushed to the server. In this form, the vales are read-only (cannot be changed). The individual channels may be enabled/disabled in the Sensor Configuration screen.
- Data Object: this is generated by the NetMeter as metadata to describe the configuration of the NetMeter for the benefit of the server. See section 5.3.

[1] Currently, only the POST method is supported.

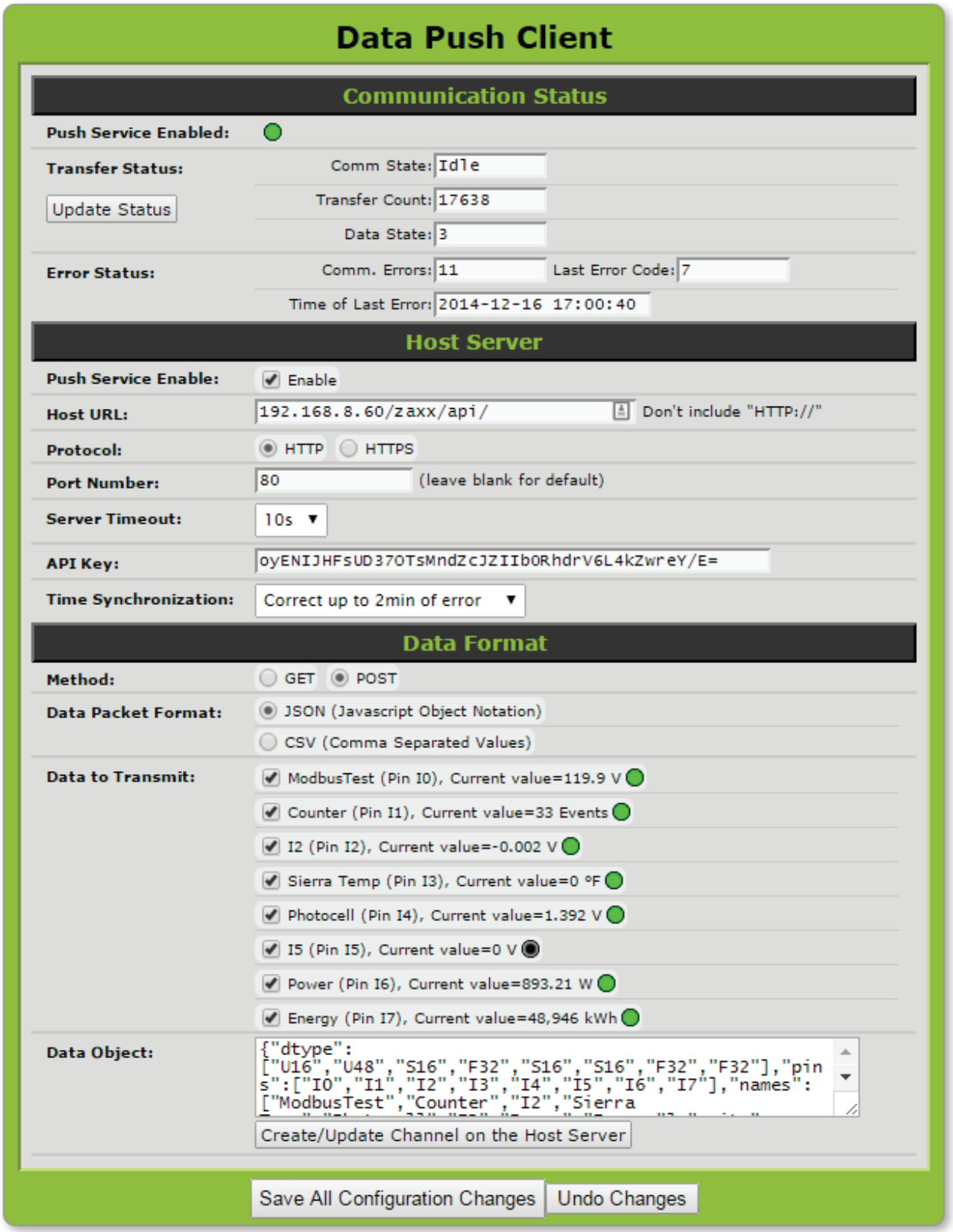


Figure 1: NetMeter-OMNI Data Push Configuration Screen

4.2 Troubleshooting

The **Setup** → **Data Push Configuration** screen is also where the status of the Push API can be monitored and is helpful for troubleshooting. This is depicted in the Communication Status section of Figure 1.

If there is a problem communicating with the server, the "Comm. Errors" field will be incremented and the error type will be indicated in "Last Error Code" in addition to a timestamp of when the last error occurred. The error codes are described in Section 6 below.

Pressing the "Update Status" button will load the latest status information.

"Transfer Count" counts both failed and successful transfers. Under normal operation, the "Transfer Count" value should be much higher than the "Comm. Errors" value. If the "Comm. Errors" keeps pace with the "Transfer Count" value then troubleshooting may be necessary.

If the setup information is incorrect, Error Code 2 is typical: it indicates that there is no server response and could be because the Host URL is incorrect.

Error Codes 3,4 occur when HTTPS (encrypted) communication is enabled and there is a problem securing the communication. Try getting non- encrypted communication to work first.

Higher Error Codes are documented in Table 1.

4.3 Internal Setup Parameter Storage

Internally, the setup information is stored in 2 system parameters under the gnconfig.json query[2]: "pushs0" and "pushd0". The user interface will write to these parameters so there is no need to manipulate them manually. For reference, the interpretation of these parameters is described as follows:

"pushs0" is a comma separated list laid out as:

Enable, Port number, Protocol, Server Timeout, Host name, URL

Where the elements are described as:

- [0] Enable: 1 = enable, 0 = disable. Set to 1 to enable the push service.
- [1] Port number: this is the TCP/IP port number of the server. Should be an integer between 0 and 65535
- [2] Protocol: 0 = HTTP, 1 = HTTPS
- [3] Server Timeout: The period of time to wait for a server response. Should be an integer value in seconds.
- [4] Host name: the IP address or host name of the server where the NetMeter will push the data. Do not include "HTTP://". This should be a string of 48 characters maximum.

[2] The gnconfig.json query is described in the NetMeter-OMNI "Network API Guide" document available at www.z3controls.com.

- [5] URL: this is the path, relative to the host, for the receiving API. This should be a string of 48 characters maximum.

"pushd0" is also a comma separated list but laid out as:

Update rate, Timeout action, Method, Packet format, Timestamp format, Data to transmit, Reserved, TSkew, API Key

Where the elements are described as:

- [0] Update rate (seconds)
- [1] Timeout action: reserved for future use, set to 0
- [2] Method: 0 = GET, 1 = POST. Reserved for future use, set to 1.
- [3] Packet format: 0 = JSON, 1 = CSV
- [4] Timestamp format: reserved for future use, set to 0
- [5] Data to transmit: 32 bit unsigned integer with 1 bit per channel for enable
- [6] Reserved (set to 0)
- [7] TSkew: the maximum number of seconds that the NetMeter will try to compensate for skew between the server clock and the NetMeters internal clock.
- [8] API Key: this is a string that is generated by the server to verify the identity and routing of data.

5 POST Method API

The POST API method provides a simple way to push data to a remote server.

The format of the request is depicted as follows:

```
POST /zaxx/api/ HTTP/1.1\r\n
Host: 192.168.8.60\r\n
Accept: */*\r\n
Content-Type: text/plain\r\n
User-Agent: NetMeter\r\n
ApiKey: g0QctaEd8u9qA/c5Y/vSIoKLhdiVYBor1eHcspDdbFI=\r\n
ApiTime: 1400435085\r\n
Content-Length: 28\r\n
\r\n
[[1400435085,4,222,0,14386]]
```

The highlighted fields in the above packet will vary according to the setup.

The POST, Host, and Content-Length headers are standard HTTP headers. The non-standard headers are:

- ApiKey: The API Key generated by the host and transmitted by the NetMeter. It is used to identify how the request is to be routed so that it can be stored properly in the servers database.
- ApiTime: This is a timestamp of the current NetMeter time clock. It is the time that the header is transmitted, not necessarily the time that the NetMeter data was collected. The format of the timestamp is documented in the next section.

5.1 Timestamp Format

Timestamps sent by the NetMeter push API are in Unix epoch time. That is, the number of seconds since midnight January 1, 1970[3] in the UTC/GMT time zone.

5.2 Data Format

The data sent by the NetMeter is a JSON[4] formatted string or CSV (Comma Separated Vales). The JSON format is an array of arrays (2D array). The general format is:

```
[
    [timestamp(0),data(0,0),data(0,1),…,data(0,n-1)],
    [timestamp(1),data(1,0),data(1,1),…,data(1,n-1)],
    …
    [timestamp(m-1),data(m-1,0),data(m-1,1),…,data(m-1,n-1)]
]
```

Where:

- timestamp(0)…timestamp(m-1) is the timestamp for each of the data sets
- data(0,0),data(0,1),…,data(0,n-1) is the data for timestamp(0). The layout of data is setup dependent. Each data element is for a channel: data(t,0) for channel 0 … data(t,1) for channel 1 and so on.
- There may be 1 or more data sets. The maximum data payload size is 4kB.

Note that whitespace has been added to the above for readability. The whitespace will not exist in the actual data.

Normally, only a single time stamped data set is transmitted. However, if there has been a period of lost communication, and data has become backlogged, then a larger data packet containing multiple time stamped data sets will be transmitted.

The equivalent CSV data payload is:

```
timestamp(0),data(0,0),data(0,1),…,data(0,n-1)
timestamp(1),data(1,0),data(1,1),…,data(1,n-1)
…
timestamp(m-1),data(m-1,0),data(m-1,1),…,data(m-1,n-1)
```

For the CSV format, each timestamped dataset is terminated with a “newline” character (“\n”).

5.3 Metadata

Metadata is the information about how data from the NetMeter is to be interpreted. The NetMeter user interface (**Setup** → **Data Push Configuration**) generates the metadata and provides the option of automatically pushing it up to the server. This only needs to be done once during setup.

The metadata is a JSON object that describes the attributes of each channel such as:

---

[3] The NetMeter internally stores time as the number of seconds since January 1, 2010 of the UTC/GMT. However, it is converted to Unix epoch time before it is transmitted by the push API.

[4] JSON is a standard (ECMA-404) defined by the ECMA (European Computer Manufacturers Association). For more information, see json.org and www.ecma-international.org/publications/standards/Ecma-404.htm

- "dtype": the data type for the channel. The NetMeter-OMNI has 2 possible data types: S16 (signed 16 bit integer) and U48 (Unsigned 48 bit integer)
- "names": the user assigned name or label for each channel.
- "units": the user assigned units of measure for each channel.
- "uhtml": the HTML version of user assigned units of measure for each channel. These may contain HTML tags for subscript, superscript or special characters that are not available in standard plain text.


```
{
    "dtype":["S16","S16","U48","S16","U48","S16","S16","U48"],
    "pins":["I0","I1","I2","I3","I4","I5","I6","I7"],
    "names":["Light Sensor(Int)","I1","Steam Mass","Steam Flow Rate","Motion Sensor","Fan
Voltage","Light Sensor","Fan Tachometer"],
    "units":["V","V","lb","lb/hr","Events","V","V","Revolutions"],
    "uhtml":["V","V","lb","lb/hr","Events","V","V","Revolutions"],
    "uflow":["","","lb/min","","Events/min","","","RPM"],
    "uflowhtml":["","","lb/min","","Events/min","","","RPM"],
    "tflow":[0,0,60,0,60,0,0,60],
    "scale":[0.004,0.001,1,0.2228125035763,1,0.001,0.001,0.5],
    "offset":[0,0,0,-891.25,0,0,0,0],
    "model":"NetMeter-OMNI-8C",
    "hwver":"1.0",
    "fwver":"1.0.0",
    "fwdate":"2014-06-07",
    "fwbuild":"0352",
    "mac":"00:04:A3:CE:C1:37",
    "label":"OmniDemo",
    "desc":"NetMeter-OMNI Show Demo Unit"
}
```


The metadata can be pushed to the server using the "Create/Update Channel on the Host Server" button. When this button is clicked, the JSON metadata is sent to the specified server using POST. The receiving API on the host can differentiate the transaction from regular data transactions by recognizing the presence of the URI encoded parameter "wrobj" (write object). In this mode, the API Key is presented as a URL variable "ApiKey". For example:

```
hostpath?wrobj=1&ApiKey=123ABC
```

Where

- `hostpath` is the host URL established in the Host URL configuration field of the setup page
- `wrobj=1` indicates that the POSTed payload is the JSON data object
- `ApiKey` is the URI safe value of the API key

5.3.1 Considerations for Host Metadata Implementation

There are several implementation details that host servers need to be provisioned for:

1. The host needs to handle changes in NetMeter configuration
2. The host needs to allow "Cross-Origin Resource Sharing" (CORS)

5.3.1.1 Changes in NetMeter Configuration

When the host receives a "wrobj" command after a previous "wrobj" command it is possible that the new data layout is incompatible with the original. This may require the previous data to have to be cleared. One way to deal with this is for the host to capture

the new data object without making it active unless there are no conflicting data types or quantity of data items. If there is a conflict, then the host could require the user to login to allow the change to be accepted with the consequence that previous data will be lost.

5.3.1.2 Cross-Origin Resource Sharing

The host server needs to enable Cross-Origin Resource Sharing by setting a header field in the HTTP/HTTPS response. The response should include the following in the header:

```
Access-Control-Allow-Origin: *
```

Control over header information is dependant on the server-side application serving the API. Information for setting up CORS for various server platforms can be found at:

http://enable-cors.org/server.html

For example, to add the CORS authorization to the header using Apache, simply add the following line inside either the `<Directory>`, `<Location>`, `<Files>` or `<VirtualHost>` sections of your server config (usually located in a *.conf file, such as httpd.conf or apache.conf), or within a `.htaccess` file:

```
Header set Access-Control-Allow-Origin "*"
```

5.4 Server Response

The POST API expects only simple responses similar to[5]:

```
HTTP/1.1 200 OK\r\n
Access-Control-Allow-Origin: *\r\n
Content-Length: 10\r\n
\r\n
ServerTime,DataTime,Optional…
```

Additional header lines may be added by the server and will be ignored by the NetMeter.

The response header must include the first line with the Status Code. The valid HTTP Status Codes are:

- 200 to indicate that the server received and understood the POST query
- 404 to indicate that the URL is not found
- 403 to indicate that the command is forbidden (lacks the proper credentials)

Any response in the range of 200-299 is considered “OK”. Anything outside of this will be counted as an error.

For debug purposes, the NetMeter counts the number of error responses (responses that are not the value 200-299) and also captures the value of the last error response along with a timestamp to indicate when that response was received.

The only other required header information is the “`Content-Length`” that will be used to size the data payload.

The server response data payload is in the form of a comma separated list of numbers in this order:

[5] The header field “Access-Control-Allow-Origin” is not required for normal data transfers but must be present for the “wrobj” response.

- *ServerTime*: is a server timestamp: the time according to the server. Optionally, the NetMeter may use this as a time synchronization reference to de-skew NetMeter time to align it with server time.
- *DataTime*: is the timestamp of the most up-to-date data received by the server. The server may set this to 0 (zero) and the NetMeter will ignore it. See ☐ below for more details.
- *Optional*: additional numeric values may be transmitted back to the NetMeter. This data may be used by the NetMeter to controls outputs such as relays on accessory devices. However, it is otherwise ignored.

5.5 Server Response: Initialization Special Case

When the NetMeter comes out of a reboot/power-on, or if the data push client is first enabled, the NetMeter needs to determine how up-to-date the server is versus the NetMeter's internal memory. This is accomplished by the NetMeter transmitting an empty array. When the server sees this, it should respond as usual even though no data is written to the server. What is most critical, is that the server respond with the "*DataTime*" value as previously defined. This is how "*DataTime*" will be acted upon:

- For *DataTime*=0 the NetMeter will transmit only newly logged data that occurs after the reboot/power-on.
- For any positive integer value of "*DataTime*", the NetMeter will transmit all available data that has a timestamp newer than "*DataTime*".

In the case where the server has never received data from the NetMeter (the NetMeter has been freshly added) it is possible to upload all available historical data by having the server respond with *DataTime*=1 for the initial response.

A typical use case for server response is:

- If the request from the NetMeter is a zero length array, respond with "*DataTime*" set to the highest timestamp value of all previously recorded data.
- If the request from the NetMeter is a zero length array, and the server database has never recorded any data from the NetMeter, respond with "*DataTime*" set to 1.
- Otherwise, return 0.

This procedure will ensure that the NetMeter's internal database is synchronized with the server.

5.6 Clock Skew Compensation

As part of the client-host transaction, the host transmits it's current time back to the NetMeter in the response data. The NetMeter may be configured to use this value to try to synchronize it's internal clock with the server clock. This feature is enabled by setting the "Time Synchronization" value in the Data Push Client setup page.

When Time Synchronization is enabled the server clock is used as a reference. The NetMeter will add or drop 1 second per minute until the difference between NetMeter and server clock is ±1 second. This avoids large and sudden time shifts. It also helps avoid clock jitter and constant re-adjustment.

In order to avoid any problems with the server transmitting an incorrect time value, or any possible setup problems, the NetMeter will only make small time adjustments according to the value set by the Time Synchronization setting (up to ±15 minutes). The NetMeter's time clock must be reasonably close to the server clock value to begin with.

The server may have the NetMeter ignore Time Synchronization by returning a value of 0 for server time.

5.7 Timeout Handling

A timeout occurs when the server is unable to respond before the timeout period. This may be for a number of reasons:

- Network communication has been lost
- The server is busy
- There is a setup problem

When a transmission is initiated by the NetMeter, it will send out the data packet and wait for the response. If no response is received by the NetMeter before the timeout period then the NetMeter will terminate the current communication attempt. If a server response is in flight at that time, it will be lost.

At the next regular update time, the previous data will be transmitted in addition to the latest data timestamp. A timeout may happen again thus causing additional data sets to be queued for later transmission. Once communication has been restored, and data can move again, the accumulated backlog of data will be transmitted. If communication has been unavailable for a lengthy time, multiple requests may occur until all of the backlogged data has been transmitted.

6 Error Codes

When errors occur during API communication, an error code is generated. The latest error code is displayed in the Push API setup page. The error codes are:

Table 1: Push API Error Codes

Code	Description
2	Timeout waiting for a server response
3, 4	Problem with securing the socket using SSL when in HTTPS mode
5	Timeout or disconnect when the NetMeter was trying to send the header to the server
6	Timeout or disconnect when the NetMeter was trying to send the payload to the server
7	Timeout or disconnect while trying to receive the "Content-Length" header of the server response
8	Timeout or disconnect while trying to parse the Content-Length header value of the server response

Code	Description
9	Timeout or disconnect while trying to process the response payload

Appendix A Change Log

Revision	Change Description
1.0.2	First release
1.0.3	• Documented the need for Access-Control-Allow-Origin (CORS) in order to enable the NetMeter to push the metadata object to the server. • Documented the CSV format • Figure 1 updated to reflect the most recent updates • Troubleshooting section added (section 4.2)

www.z3controls.com

support@z3controls.com

Phone: 1-877-454-4436

4261-A14 Highway #7 East, Unit 290
Markham, ON L3R 9W6
Canada