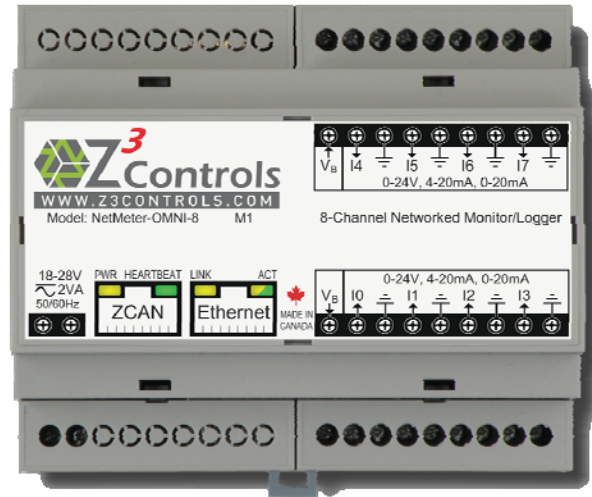




Online Support Portal:
help.z3controls.com



NetMeter-OMNI

**Commercial/Industrial Monitor/Logger with
Integrated Networking**

NETWORK API GUIDE

Preliminary

CONTENTS

| | | |
|---------|---|----|
| 1 | Important Notice | 3 |
| 2 | Purpose | 3 |
| 3 | Introduction..... | 3 |
| 3.1 | License | 3 |
| 3.2 | Overview..... | 4 |
| 4 | Quick Start Tutorial | 4 |
| 4.1 | Connect to the NetMeter-OMNI | 4 |
| 4.2 | Query Real-time NetMeter-OMNI Data | 5 |
| 4.3 | Access NetMeter-OMNI Data from a Program..... | 6 |
| 5 | API Command Format..... | 8 |
| 5.1 | Data Types | 8 |
| 5.1.1 | Writing System Parameters | 9 |
| 5.2 | Sensor Time | 10 |
| 5.3 | Time Values for CSV Data Query Mode | 10 |
| 5.4 | Command Limitations..... | 11 |
| 5.5 | Return Data | 11 |
| 5.6 | Modeless Operation | 12 |
| 6 | API Command Reference..... | 12 |
| 6.1 | Gateway Specific Commands..... | 12 |
| 6.1.1 | Gateway Network Configuration: gnconfig.json | 12 |
| 6.2 | Sensor Specific Commands..... | 15 |
| 6.2.1 | Sensor Information: sinfo.json | 15 |
| 6.2.2 | Sensor Data Query: sdata.json..... | 18 |
| 6.2.2.1 | Sensor Data Query: Realtime Data (mode = rt) | 21 |
| 6.2.2.2 | Sensor Data Query: Realtime Data (mode = rtall) | 24 |
| 6.2.2.3 | Sensor Data Query: Short-term Datalog (mode = ramlog) | 25 |
| 6.2.2.4 | Sensor Data Query: Main Datalog (mode = ml) | 27 |
| 6.2.2.5 | Sensor Data Query: Coarse Main Datalog (mode = mlc) | 30 |
| 7 | Using Excel to Access the API | 31 |
| 7.1 | Alternate Methods..... | 34 |

1 Important Notice

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.

2 Purpose

The purpose of this document is to describe the parameters and operation of the Application Programming Interface (the 'API') which allows client applications to access NetMeter-OMNI data.

Since the Z3 Controls NetMeter-OMNI already provides the user with a fully-integrated, full-featured graphical user interface ('GUI'), most setups will not require the API. However, the API is invaluable in situations where the NetMeter-OMNI is intended to be used with other systems.

Potential applications of the API may include:

- Integration into factory automation or building automation systems
- The aggregation of NetMeter data with data from other systems
- Special applications that harvest NetMeter-OMNI data for various purposes
- Generating custom reports or custom dashboard displays

3 Introduction

This document is specific to the NetMeter-OMNI devices. However, most of the methods described herein are the same or similar to other NetMeter Family devices. For the purposes of generalization, this document sometimes refers to "NetMeter" in a general way. In this context, NetMeter may also be considered the NetMeter-OMNI.

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 at:

<http://z3controls.com/tla.php>

Users are only permitted to use the API commands specified within this documentation. Any additional, or undocumented, API commands may not be incorporated into a user's application(s) without first obtaining the written permission of Z3 Controls.

Please familiarize yourself with the license agreement before using this device.

3.2 Overview

The API is based on HTTP communications where the Z3 Controls device is considered to be an HTTP server. An HTTP client is necessary in order to make use of this device. For example, this may be achieved in the form of JavaScript using a web browser or by using a programming language (such as C, C + + , PHP, Python, etc.) that has socket libraries which enable TCP/IP HTTP client communication.

Note that the API is available using both the HTTP and HTTPS (encrypted) protocols. When using HTTPS it is normal to have a security certificate error.

Additionally, each Z3 Controls NetMeter device can be logically or physically composed of up to 3 parts:

1. **The Gateway:** this is the component of the device containing the HTTP server. The Gateway component has Gateway specific API commands, such as the Gateway Network Setup which can be either Ethernet or WiFi.
2. **The Sensor(s):** each Z3 Controls Gateway will have a sensor, or input/output signal, attached to it –there may be multiple sensors or I/Os on a single Gateway.
3. **Data logging and storage:** Z3 Controls sensors include data storage and logging.

The Gateway and Sensor(s) may be composed of a single unit or they may be physically separate units. For the purposes of the API, they are treated as logically separate units.

The NetMeter-OMNI is a single physical unit that combines gateway, sensor, and storage.

4 Quick Start Tutorial

This section is for those who wish to start programming with the API right away. If anything is unclear in this tutorial, please skip ahead to the subsequent chapters that explain the operation in greater detail.

4.1 Connect to the NetMeter-OMNI

Follow the instructions in the NetMeter-OMNI-8C *Installation and Instruction Manual* to set up network communication to the device.

By default, any NetMeter family product will have a NETBIOS name of NETMETER and will be available at:

<http://netmeter/>

From this point on, it is assumed that your NetMeter-OMNI has the basic configuration setup and is able to display real-time data from the standard web interface using a web browser.

It is also assumed that the default NETBIOS name (“NETMETER”) is being used. If you have changed it, please substitute your new NETBIOS name as required or access it through the direct IP address.

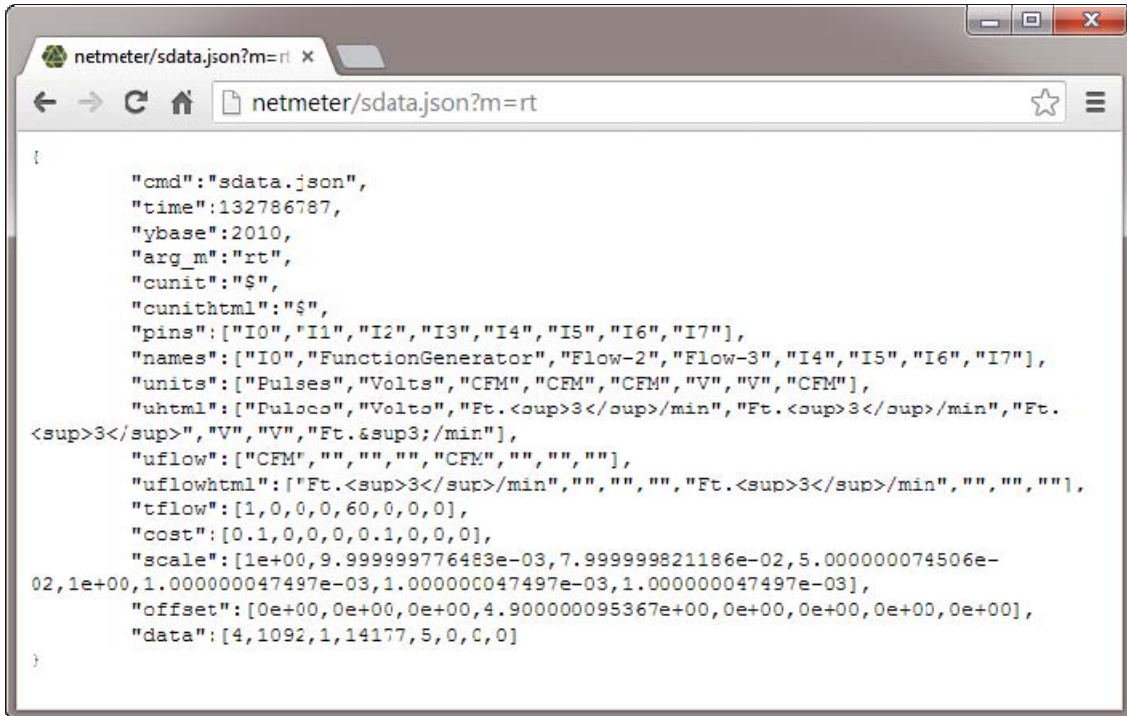
4.2 Query Real-time NetMeter-OMNI Data

The sdata.json query is used for obtaining many types of data. A “query” is a request for data from the NetMeter and it uses the standard URL format used in web applications.

Enter the following query into the address bar of your web browser:

```
http://netmeter/sdata.json?m=rt
```

The following data should be displayed (please note that the results will not be exactly as shown):



```
{
  "cmd": "sdata.json",
  "time": 132786787,
  "ybase": 2010,
  "arg_m": "rt",
  "cunit": "$",
  "cunithtml": "$",
  "pins": ["I0", "I1", "I2", "I3", "I4", "I5", "I6", "I7"],
  "names": ["I0", "FunctionGenerator", "Flow-2", "Flow-3", "I4", "I5", "I6", "I7"],
  "units": ["Pulses", "Volts", "CFM", "CFM", "CFM", "V", "V", "CFM"],
  "uhtml": ["Pulses", "Volts", "Ft. <sup>3</sup>/min", "Ft. <sup>3</sup>/min", "Ft.
<sup>3</sup>", "V", "V", "Ft. <sup>3</sup>/min"],
  "uflow": ["CFM", "", "", "", "CFM", "", "", ""],
  "uflowhtml": ["Ft. <sup>3</sup>/min", "", "", "", "Ft. <sup>3</sup>/min", "", "", ""],
  "tflow": [1, 0, 0, 0, 60, 0, 0, 0],
  "cost": [0.1, 0, 0, 0, 0.1, 0, 0, 0],
  "scale": [1e+00, 9.999999776483e-03, 7.999999921186e-02, 5.000000074506e-
02, 1e+00, 1.000000047497e-03, 1.000000047497e-03, 1.000000047497e-03],
  "offset": [0e+00, 0e+00, 0e+00, 4.900000095367e+00, 0e+00, 0e+00, 0e+00, 0e+00],
  "data": [4, 1092, 1, 14177, 5, 0, 0, 0]
```

Figure 1: Example Result from the sdata.json Query

The resulting data structure in Figure 1 above is in JSON format: a lightweight open standard designed for machine and human-readable data interchange. It is language-independent, with parsers available for most modern programming languages.

The command has “m=rt” (real-time data) which outputs the instantaneous values for the main function of each channel.

If the parameter “s=1” is added (see Figure 2), the resulting data will be displayed in scaled units (volts, milliamps, Deg C, and so on). Using the “s” parameter is the simplest method to get quick results. When the “s” (scale) parameter is not specified, then the values in the “scale” and “offset” arrays must be applied.

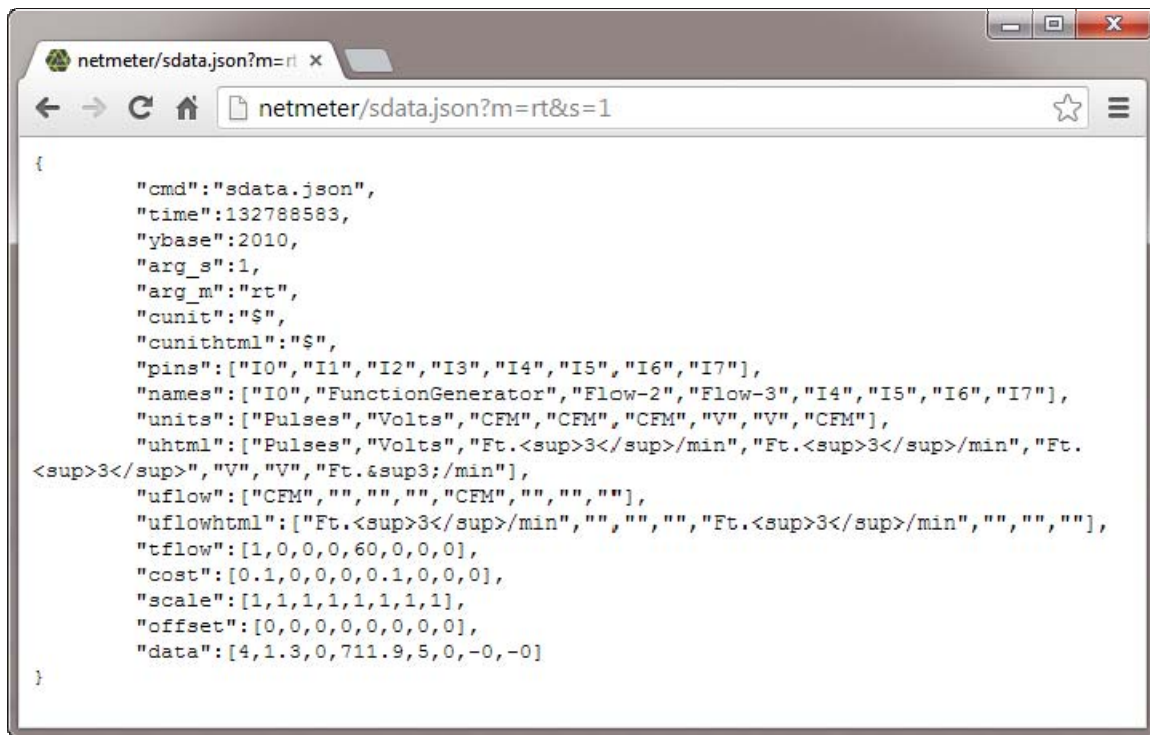


Figure 2: Example Result from the sdata.json Query with Scaled Data

4.3 Access NetMeter-OMNI Data from a Program

Instead of a web browser, your preferred programming language may also be used to execute the above query and then access the returned data structure. Nearly all available computer languages provide for TCP/IP socket support, either natively or through readily available libraries.

Consider the popular server side scripting language PHP: the cURL library can be used for just this purpose. The library is readily available, although you may need to enable this feature in your PHP setup.

Listing 1 is a simple PHP program that performs the query using the cURL library.

Listing 1: PHP Code to Perform Simple NetMeter-OMNI Query

```

<?php
$url = 'http://netmeter/sdata.json?m=rt&s=1';
echo curl_download($url);

function curl_download($url)
{
    // is cURL installed yet?
    if (!function_exists ('curl_init')){
        die ('Sorry cURL is not installed!');
    }

    // OK cool - then let's create a new cURL resource handle
    $ch = curl_init();

    // Now set some options (most are optional)

    // Set URL to download

```

```
curl_setopt($ch, CURLOPT_URL, $url);

// Include header in result? (0 = yes, 1 = no)
curl_setopt($ch, CURLOPT_HEADER, 0);

// Should cURL return or print out the data? (true = return, false = print)
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

// Timeout in seconds
curl_setopt($ch, CURLOPT_TIMEOUT, 10);

// Download the given URL, and return output
$output = curl_exec($ch);

// Close the cURL resource, and free system resources
curl_close($ch);

return $output;
}
?>
```

In Listing 1, the main program is only 2 lines of code (Line 3 and 4). The function `curl_download($url)` is a generic function that can be used to perform a query and return the results as a text string.

Rather than use the data from the query in string format, it can be made available as a data object such that the PHP program can use it more easily. This is illustrated in Listing 2 below. Note that the `curl_download($url)` function has been pushed out to an external include file for convenience.

The PHP function `json_decode()` on line 6 is used to set the variable `$sdata` as a PHP data object.

Listing 2: PHP Code to Perform NetMeter-OMNI Query and Parse the JSON String

```
<?php
include_once "curl_download.php";

$url = 'http://netmeter/sdata.json?m=rt&s=1';
$sdata=json_decode(curl_download($url));

for($i=0; $i<count($sdata->data); $i++)
{
    echo "\n". $sdata->names[$i]. "=". $sdata->data[$i]. " ". $sdata->units[$i];
}
?>
```

Running the above PHP script will return results similar to:

```
I0=4 Pulses
FunctionGenerator=1.1 Volts
Flow-2=0 CFM
Flow-3=711.5 CFM
I4=5 CFM
I5=0 V
I6=0 V
I7=0 CFM
```

The examples above illustrate how easy it is to incorporate data from a NetMeter into another application with just a few lines of code. The PHP example can easily translate

into other programming languages since cURL and JSON libraries exist for most of the popular languages in current use.

5 API Command Format

There are two basic types of API command:

1. **Read Commands:** Commands that query the state of the Gateway or Sensor(s)
2. **Write Commands:** Commands that configure the Gateway or Sensor(s)

Commands to the Gateway are in the form of HTTP: 'GET' requests over the Ethernet interface. 'GET' requests take this general form:

```
http://netmeterpath/command?var1=aaa&var2=bbb&var3=ccc
```

Where:

“netmeterpath” is the full path to the NetMeter device or its IP address. The default NETBIOS name for the NetMeter is “netmeter” when using the factory default network setup.

“command” is the API defined command name such as “sinfo.json” for sensor setup information or “sdata.json” for sensor data

“var1”, “var2”, “var3” are parameter names that the command processes

“aaa”, “bbb”, “ccc” are values for the parameters

Note that the order in which parameters are present in the query has no effect on the behaviour of the command.

5.1 Data Types

Parameters are case sensitive. Generally, they are lower case unless otherwise noted.

Parameters may be one of those described in Table 1.

Table 1: Mnemonics for Parameter Data Types

| Mnemonic | Description |
|----------|--|
| U8 | Unsigned value between 0 and 255 (8 bits). Hex values can be used with a “0x” prefix |
| U16 | Unsigned value between 0 and 65,535 (16 bits) |
| U32 | Unsigned value between 0 and 4,294,967,295 (32 bits) |
| INT8 | Signed 8 bit value between -128 to 127 |
| INT16 | Signed 16 bit value between -32,768 to 32,767 |
| INT32 | Signed 32 bit value between -2147483648 to 2147483647 |
| INT48 | Signed 48 bit value between -140737488355328 to 140737488355327 |
| F32 | 32 bit floating point number |
| F64 | 64 bit floating point number |
| BOOL | Boolean: 0 or 1 |

| Mnemonic | Description |
|----------|--|
| STR | Text string: a maximum of 46 characters unless otherwise noted. |
| MAC | A string in the format of a MAC (Media Access Controller) address such as "00:04:A3:AB:12:34" |
| IP | A string formatted as an IP address or mask such as "192.168.1.1" |
| (W) | The (W) annotation indicates that a parameter may also be written. See Section 5.1.1 for additional information. |

5.1.1 Writing System Parameters

System parameters are parameters that change the state of the NetMeter and are stored in non-volatile memory. Non-system parameters are only relevant to a specific query.

Most applications designed to access NetMeters directly will not need to write NetMeter system parameters. If required, however, system parameters are written in a similar fashion to that of non-system parameters, using the HTTP URL encoded 'GET' query as demonstrated below:

```
http://netmeter/command?parameter=value
```

Where:

“parameter” is the name of the parameter to be written.

“value” is the numeric or string data to be written to the parameter. “value” must be a valid URL/URI encoded value where special characters are escaped as defined by the W3C (World Wide Web Consortium) RFC 3986.

URLs can only be sent over the Internet using the ASCII character-set.

Since URLs often contain characters outside the ASCII set, the URL must be converted into a valid ASCII format.

URL encoding replaces the 'invalid' ASCII characters with a "%" followed by two hexadecimal digits.

URLs cannot contain spaces. URL encoding normally replaces a space with a + sign.

A URL/URI encoding function is provided in libraries for many programming languages. In JavaScript, a valid query may be constructed as:

```
var query="http://netmeter/info.json?" + pname + "=" + encodeURIComponent(pvalue);
```

Where:

“pname” is a variable containing the parameter name

“pvalue” is a variable containing the parameter value

“encodeURIComponent()” is a function built into JavaScript that will safely escape “pvalue”

In PHP the same query is constructed as:

```
$query='http://netmeter/info.json?' . $pname . '=' . rawurlencode($pvalue);
```

Parameters may be erased to their default value by setting them to the backspace character (encoded as "%08"). For example:

```
http://netmeter/command?parameter=%08
```

Typically, parameters may only be written when the client is authenticated. Authentication may be included in the URL by using the URL encoded method:

```
http://username:password@netmeter/...
```

For security purposes, this method should only be used by scripts over a secure local/VPN network.

5.2 Sensor Time

Some queries input time as a parameter or return time values as data. Time stamps are 32 bit unsigned numbers that work as follows:

- The NetMeter maintains a temperature compensated internal battery backed real time clock that is used to timestamp data.
- Time values used by the NetMeter are the number of seconds since zero hour of January 1 of the “base year” and referenced to the UTC time zone. The base year is defined by the “ybase” value available through the “sinfo.json” (sensor info) query. In the case where ybase = “2010”, a time value of zero would be 00:00:00 (midnight) of January 1, 2010 of the UTC time zone. A time value of 1 is exactly 1 second past that and so on.
- Internally, the NetMeter sensor always operates in UTC time (Zulu Time Zone). For the NetMeter built-in web application, time as shown in the user interface is localized by the application software running in the web browser using whichever localization rules the client device happens to be using. Consequently, a user accessing a NetMeter from a laptop in the EST time zone will see that the current sensor time, as reported by the GUI, is the same as their laptop. Another user with a laptop configured to a different time zone will see that the current time matches their alternate local time. However, the numerical value transmitted by the NetMeter will be exactly the same in both time zones.

Conversion of NetMeter time to the time format of an application language is fairly straightforward. For example, JavaScript time is measured by the number of milliseconds occurred since midnight, January 1, 1970. Consequently, JavaScript time is calculated by multiplying the NetMeter time by 1000 and adding the number of milliseconds from midnight January 1, 1970 until midnight January 1 of ybase. The following JavaScript functions may be used for this purpose:

```
// Convert the Z3 Controls NetMeter time format into a JavaScript style serial number
function fromZ3Time(z3time, ybase)
{
    var UTCOffset = parseInt(Date.UTC(ybase,0,1)); // number of ms to add to time stamps
    return z3time*1000+UTCOffset;
}

// Convert a JavaScript style serial number to the the Z3 Controls NetMeter time format
function toZ3Time(time, ybase)
{
    var UTCOffset=parseInt(Date.UTC(ybase,0,1)); // number of ms to subtract from JavaScript time
    return (time-UTCOffset)/1000;
}
```

5.3 Time Values for CSV Data Query Mode

When the “sdata.json” or “sdata.csv” query specifies that the CSV format be applied, then data time stamps will be reported using the Excel floating point format. This applies to the

data logger query modes (“ramlog”, “ml”, and “mlc”) as detailed in 6.2.2.3, 6.2.2.4, and 6.2.2.5.

Excel stores all dates as integers and all times as decimal fractions. With this system, Excel can add, subtract, or compare dates and times just like any other numbers, and all dates are manipulated by using this system.

In this system, the serial number 1 represents 1/1/1900 12:00:00 a.m. Times are stored as decimal numbers between .0 and .99999, where .0 is 00:00:00 and .99999 is 23:59:59. The date integers and time decimal fractions can be combined to create numbers that have a decimal and an integer portion. For example, the number 32331.06 represents the date and time 7/7/1988 1:26:24 a.m.

The NetMeter-OMNI will output time in the Excel floating point format for the “ramlog”, “ml”, and “mlc” data queries when the “csv” mode is specified.

Unfortunately, when cells are date formatted by excel, there is no concept of time zones. Consequently, times will be shown in the UTC (ZULU) reference time zone by default as this is the NetMeter-OMNI format. This may be shifted to the desired local time using the “utc” query parameter.

5.4 Command Limitations

Due to the limited buffering capacity inside the Gateway processor, HTTP 'GET' requests are limited to a maximum of 120 characters. This should not be a concern regarding query requests because they are designed to be brief.

5.5 Return Data

When the HTTP 'GET' request is processed by the Gateway, it will typically respond with a JSON formatted string unless otherwise noted.

JSON (JavaScript Object Notation) is an open source lightweight data-interchange format. It is easy for programmers to read and write and also easy for machines to parse and generate. JSON is a text format that is entirely language independent and is supported by most common programming/scripting languages such as the C-family of languages (C, C + +, C#), Java, JavaScript, Perl, Python, PHP, Tcl, Matlab, and many others.

More information about JSON is available online at <http://www.json.org/>

An example of a JSON response from the NetMeter-OMNI:

```
{
  "cmd": "sdata.json",
  "time": 132789281,
  "ybase": 2010,
  "arg_hdr": 0,
  "arg_m": "rt",
  "data": [4, 688, 0, 14068, 5, -1, -1, -1]
}
```

This is a response containing real-time information from the NetMeter-OMNI. It is a data object that contains a series of data elements such as “cmd”, “time” and so forth and may be set in any order. The values for these elements can be strings, numbers, arrays, or even hierarchical data objects.

The JSON responses that are described in this document may contain additional data elements to those listed above, depending upon the firmware revision. These can usually be ignored.

5.6 Modeless Operation

API commands are designed to be as modeless as possible. That is, the API assumes that multiple clients wish to access data simultaneously. Consequently, each command is self-contained and does not rely on the state or “mode” of the gateway/sensor from a previous command.

6 API Command Reference

There are two main classes of commands defined by the Gateway:

1. **Gateway specific commands:** these commands are processed by the Gateway itself. No communication with the sensor(s) is required.
2. **Sensor specific commands:** these commands are sent directly to the sensor(s) for processing.

6.1 Gateway Specific Commands

Gateway specific commands are commands that are processed by the Gateway itself. Gateway commands lead with the letter ‘g’.

6.1.1 Gateway Network Configuration: gnconfig.json

Returns the gateway Ethernet/WiFi network configuration.

| | | |
|------------|---|--|
| Command | gnconfig.json | |
| Parameters | ledfast | This optional parameter causes the heartbeat LED to blink faster for 2 seconds |
| | When no parameters are used, a report of the gateway is generated (see example below) | |

Example for using the gnconfig.json query:

Example Query:

```
http://netmeter/gnconfig.json
```

Example Response:

```
{
  "model": "NetMeter-OMNI-8C",
  "hwver": "1.0",
  "fwver": "1.0.0",
  "fwdate": "2014-03-17",
  "fwbuild": "0282",
  "eflash": "32",
  "mac": "00:04:A3:F5:44:6B",
  "mach": "0x023D47B2",
  "ip": "192.168.8.78",
  "mask": "255.255.255.0",
  "gateway": "192.168.8.1",
  "dns1": "192.168.8.1",
  "dns2": "0.0.0.0",
  "defip": "192.168.2.200",
  "defmask": "255.255.255.0",
  "hostname": "OMNI 2",
  "dhcp": "1",
  "wtype": "0",
  "fail": "0",
  "remip": "192.168.8.60",
  "remmac": "20:CF:30:AE:AD:1E",
  "auth": "0x80",
  "debug": "0",
  "dfit":
  {
    "ip": "192.168.2.200",
    "mask": "255.255.255.0",
    "gateway": "192.168.2.75",
    "dns1": "192.168.2.75",
    "dns2": "0.0.0.0",
    "hostname": "NETMETER",
    "dhcp": "1"
  },
  "nvmem":
  {
    "ip": "",
    "mask": "",
    "gateway": "",
    "dns1": "",
    "dns2": "",
    "hostname": "OMNI 2",
    "dhcp": "",
    "homepg": "ilog.html?list=[2],[4]&tspan=16h&nav=1&auto=1"
  }
}
```

This data structure is described in Table 2.

Table 2: Description of the Data Structure Returned by the "gnconfig.json" Query

| Mnemonic | Type | Description |
|----------|------|--|
| model | STR | Hardware model number of the Z3 Controls Gateway |
| hwver | STR | Gateway hardware version |
| fwver | STR | Gateway firmware version |
| fwdate | STR | Gateway firmware build date |
| fwbuild | STR | Gateway firmware build number |
| eflash | STR | Amount of flash memory storage in megabytes |
| mac | MAC | Gateway mac address |

| Mnemonic | Type | Description |
|----------------|---------|---|
| mach | STR | A special hash of the MAC address |
| ip | IP(W) | Current IP address |
| mask | IP(W) | Current IP mask |
| gateway | IP(W) | Current IP address of the internet gateway (this is separate from the Z3 "Gateway") |
| dns1 | IP(W) | Current IP address for the first domain name server used by the NetMeter-OMNI |
| dns2 | IP(W) | Current IP address for the second domain name server used by the NetMeter-OMNI |
| hostname | STR(W) | The NETBIOS name used to identify the NetMeter-OMNI on the network |
| dhcp | BOOL(W) | DHCP (Dynamic Host Configuration Protocol) enabled (= 1) or disabled (= 0) |
| wtype | U8 | Wireless Network Type: "0" indicates wired Ethernet |
| remip | IP | IP address of the remote client |
| remmac | MAC | MAC address of the remote client |
| auth | U8 | Authentication level of the client: "0x80" indicates admin level authentication. |
| dflt.ip | IP | Factory default IP address |
| dflt.mask | IP | Factory default IP mask |
| dflt.gateway | IP | Factory default IP address of the internet gateway |
| dflt.dns1 | IP | Factory default IP address for the first domain name server used by the NetMeter-OMNI |
| dflt.dns2 | IP | Factory default IP address for the second domain name server used by the NetMeter-OMNI |
| dflt.hostname | STR | The factory default NETBIOS name used to identify the NetMeter-OMNI on the network |
| dflt.dhcp | BOOL | Factory default DHCP setting |
| nvmem.ip | IP | Power-on default IP address |
| nvmem.mask | IP | Power-on default IP mask |
| nvmem.gateway | IP | Power-on default IP address of the internet gateway |
| nvmem.dns1 | IP | Power-on default IP address for the first domain name server used by the NetMeter-OMNI |
| nvmem.dns2 | IP | Power-on default IP address for the second domain name server used by the NetMeter-OMNI |
| nvmem.hostname | STR | The power-on default NETBIOS name used to identify the NetMeter-OMNI on the network |
| nvmem.dhcp | BOOL | Power-on default DHCP setting |
| nvmem.homepg | STR | Home page link |

6.2 Sensor Specific Commands

This section documents commands that are specific to energy Sensor class devices.

6.2.1 Sensor Information: `sinfo.json`

The sensor information command (`sinfo.json`) is designed to report on the configuration and operation of the sensor acquisition subsystem of the NetMeter-OMNI. It contains information that typically doesn't change over time as the NetMeter-OMNI collects data. For example, the type of CT in use, or the name and description of the sensor.

| Command | <code>sinfo.json</code> | |
|------------|-------------------------|--|
| Parameters | <code>id</code> | <p>This is an optional string that will be sent back with the response. It can be used by a requester to tag the response with an identifier.</p> <p>Example: <code>sinfo.json?id=abc123</code> will return a data structure with the data member <code>arg_id</code> set to <code>abc123</code>.</p> |
| | <code>logenable</code> | <p>This optional parameter is used to enable/disable the data logger.</p> <p><code>sinfo.json?logenable=on</code> will enable the data logger</p> <p><code>sinfo.json?logenable=off</code> will disable the data logger</p> |
| | <code>logclr</code> | <p>This optional parameter is used to clear all data in the data logger.</p> <p><code>sinfo.json?logclr=AMSURE</code> will clear the data</p> |

Example of the `sinfo.json` query:

Example Query:

`http://netmeter/sinfo.json`

Example Response:

```
{
  "model": "NetMeter-OMNI -8C",
  "hwver": "1.0",
  "fwver": "1.0.0",
  "fwdate": "2014-03-17",
  "fwbuilid": "0282",
  "logmem": 4096,
  "channels": 8,
  "ybase": 2010,
  "sensen": 1,
  "time": 132793823,
  "vpsumv": 20200,
  "batok": "1",
  "afeerr": 0,
  "ml": {"t0": 126720300, "size": 12582912, "used": 4191822},
  "tboot": 132771934,
  "tdown": 132771636,
  "feature": 0,
  "afefwver": 15,
  "logenable": 1,
  "sdata": {"ramlog": 1, "ml": 5, "mlc": 300},
  "alertstat": ["0x00000000", "0x00000000"],
  "inputs": [{"name": "I0", "phy": "Vi", "func": "adp", "min": [0, 0], "max": [24, 20]},
    {"name": "I1", "phy": "Vi", "func": "adp", "min": [0, 0], "max": [24, 20]},
    {"name": "I2", "phy": "Vi", "func": "adp", "min": [0, 0], "max": [24, 20]},
    {"name": "I3", "phy": "Vi", "func": "adp", "min": [0, 0], "max": [24, 20]},
    {"name": "I4", "phy": "Vi", "func": "adp", "min": [0, 0], "max": [24, 20]},
    {"name": "I5", "phy": "Vi", "func": "adp", "min": [0, 0], "max": [24, 20]},
    {"name": "I6", "phy": "Vi", "func": "adp", "min": [0, 0], "max": [24, 20]},
    {"name": "I7", "phy": "Vi", "func": "adp", "min": [0, 0], "max": [24, 20]}],
  "label": "OMNI -Factory-1",
  "desc": "OMNI Factory 1",
  "maiperiod": "5",
  "cunit": "$, $, 2",
  "iname0": "I0, , CFM, Ft. <sup>3</sup>\\mi n",
  "icfg0": "0, 2, 1, 0, . 4, 1. 5, 1, 0, . 1, 1",
  "iname1": "FunctionGenerator, Vol ts, , ,",
  "icfg1": "0, 0, 10, 0, . 5, 1. 5, 1, 0, 0, 0",
  "iname2": "Flow-2, CFM, Ft. <sup>3</sup>\\mi n, ,",
  "icfg2": "0, 0, 80, 0, . 5, 1. 5, 1, 0, 0, 0",
  "iname3": "Flow-3, CFM, Ft. <sup>3</sup>\\mi n, ,",
  "icfg3": "1, 0, 50, 4. 9, . 5, 1. 5, 1, 0, 0, 0",
  "iname4": "I4, CFM, Ft. <sup>3</sup>, CFM, Ft. <sup>3</sup>\\mi n",
  "icfg4": "0, 2, 1, 0, . 5, 1. 5, 1, 0, 0. 1, 60",
  "iname5": "I5, , , ,",
  "icfg5": "0, 0, 1, 0, . 5, 1. 5, 1, 0, 0, 0",
  "iname6": "I6, , , ,",
  "icfg6": "0, 0, 1, 0, . 5, 1. 5, 1, 0, 0, 0",
  "iname7": "I7, CFM, Ft. &sup3; \\mi n, ,",
  "icfg7": "0, 0, 1, 0, . 5, 1. 5, 1, 0, 0, 0",
}
```

The returned data structure is described in Table 3

Table 3: Description of the Data Structure Returned by the "sinfo.json" Query

| Mnemonic | Type | Description |
|----------|------|---|
| model | STR | Hardware model number of the Z3 Controls sensor |
| hwver | STR | Sensor hardware version |
| fwver | STR | Sensor firmware version |
| fwdate | STR | Sensor firmware build date |
| fwbuild | STR | Sensor firmware build number |

| Mnemonic | Type | Description |
|------------|----------|---|
| ybase | U16 | Year base: January 1 at midnight (hour 0) of ybase is the zero point in time for all time values. See Section 5.2 for a description of how time stamp values are formulated. |
| sensen | BOOL | Sensor data storage enable: 0 when disabled, 1 when the sensor is enabled to store data history |
| time | U32 | The current sensor time based on the time the query was transmitted. See Section 5.2 for a description of how time stamp values are formulated. |
| vpsumv | U16 | NetMeter-OMNI power supply voltage (in millivolts). This has roughly +/- 10% accuracy. |
| batok | STR | Real time clock backup battery status: 0 when not present or needs replacement, 1 indicates that it is OK |
| afeerr | U32 | Indicates communication errors with the data acquisition subsystem. Should read as 0. |
| tboot | U32 | The time of the last device bootup. |
| ttdown | U32 | The time of the last device reboot or loss of primary power. |
| feature | U16 | Sensor feature set: a value of "0" denotes the standard NetMeter-OMNI |
| label | STR (W) | The user defined label for the NetMeter-OMNI sensor |
| desc | STR (W) | The user defined description of the NetMeter-OMNI sensor |
| senab | BOOL (W) | Power-on sensor enable: determines the state of the sensor enable bit after a power-on reset. |
| afefwver | U16 | Firmware version of the data acquisition subsystem. |
| logenable | BOOL (W) | Indicates that the datalog is enabled for data capture. Should be 1 for normal operation |
| sdata | Object | A descriptor for the query modes available for the datalogger sdata commands. The number indicates the number of seconds between samples for each of the query modes. |
| inputs | Object | Descriptor for each of the inputs: "name": the default name of the input "phy": the modes supported by the input. "V" indicates support for voltage mode input with voltage reported in mV, "i" indicates support for current mode input with current reported in μ A. "func": the functions supported for each input. "a" for analog, "d" for digital, and "p" for pulse count. "min": the minimum value of each of the phy modes (in volts for "V" and in mA for "i"). "max": the maximum value each of the phy modes. |
| mainperiod | U16 (W) | The period (in seconds) of the main datalog. |
| cunit | STR (W) | Currency units: this is a comma separated list with 3 elements: the plain text currency units, the HTML currency units, and the number of decimal places for currency display. |

| Mnemonic | Type | Description |
|-----------------|---------|--|
| iname0 - iname7 | STR (W) | Input channel parameters: this is a comma separated list with 5 elements: <ul style="list-style-type: none"> ▪ User assigned input name ▪ User assigned primary units (plain text) ▪ User assigned primary units (html) ▪ User assigned secondary units used for flow calculations (plain text) ▪ User assigned secondary units (html) |
| icfg0 - icfg7 | STR (W) | Input channel configuration: this is a comma separated list with 9 elements: <ul style="list-style-type: none"> ▪ Mode: the input mode. Follows the sequence of the "inputs.phy" object. For the NetMeter-OMNI-8C that is: 0 = voltage, 1 = current ▪ Function: the input function. Follows the sequence of the "inputs.func" object. For the NetMeter-OMNI-8C that is: 0 = analog, 1 = digital, and 2 = pulse count. ▪ Scale: the user assigned scale factor (F32 format) ▪ Offset: the user assigned offset (F32 format) ▪ Threshold Low: voltage (in V) or current (in mA) below which the input is considered LOW as a binary or pulse input (F32 format) ▪ Threshold High: voltage (in V) or current (in mA) above which the input is considered HIGH as a binary or pulse input (F32 format) ▪ Mainlog enable: 1 = data for this channel is logged, 0 = data for this channel is not logged. ▪ Advanced log enable ▪ Cost: The cost-per-unit for the channel (F32 format) |

6.2.2 Sensor Data Query: sdata.json

The query "sdata.json" is used to obtain sensor data, either real time or historical. It is a multimode command with many parameters. For clarity, each mode will be documented separately below.

There are 2 categories of data available from the NetMeter-OMNI:

1. Data captured directly from the sensor subsystem in real-time
2. Data that has been stored in on-board memory.

There are a number of data storage arrays in the NetMeter-OMNI:

- Ramlog: a volatile array of short term data. This contains the most recent 5 min of data for each channel at a sample rate of 1 second. The data is the current (for current mode inputs in units of μA) or voltage (for voltage mode inputs in units of mV). The data recorded here ignores the “func” (function) setting for the channel and records the data as if it were set to “a” analog mode; Consequently, pulse counters do not get recorded in the ramlog.
- Mainlog: A non-volatile storage where either the current, voltage, binary, or pulse count value is stored for each channel (depending on the function select).

Data contained in data queries can be sent as integer numbers that must be scaled in order to convert them to standard units. This is referred to as raw data. This approach has two primary advantages:

- Integer data is typically smaller to transmit as text compared to floating point numbers
- The integer data is native to the data acquisition and digital signal processing subsystem. Consequently, the highest precision is maintained during data transmission.

Alternately, many commands have an option for the data to be scaled inside the NetMeter. This is referred to as scaled data. With scaled data, some small loss of precision is seen, or an expansion of the data size can result when high floating point precision is specified.

When the raw data mode is used, the process of obtaining data from the NetMeter proceeds as follows:

1. Execute the normal sdata.json query which will include the scale/offset information. This is required only once for a series of data queries. It provides the scale/offset values that are used to convert the raw sensor data into standard values such as M^3 , CFM, $^{\circ}\text{C}$, etc.
2. Execute multiple sdata.json queries with the header disabled (`hdr=0`) and scale it according to the results of the sdata.json query.

Raw data is converted to the specified units using the following formula:

$$\text{ScaledData}[i] = (\text{RawData}[i] * \text{scale}[i]) + \text{offset}[i]$$

The common parameters for the sdata.json command are documented in Table 4.

Table 4: "sdata.json" Query Parameters

| Command | "sdata.json" or "sdata.csv" | |
|------------|-----------------------------|--|
| Parameters | m (STR) | <p>This is the mode for the data.json command. The modes are:</p> <ul style="list-style-type: none"> ▪ "rt": realtime data ▪ "ml": main data log at full resolution ▪ "mlc": main data log at coarse resolution ▪ "ramlog": the past 5 minutes at 1s resolution |
| | id (STR) | <p>This is an optional string that will be sent back with the response. It can be used by a requester to tag the response with an identifier.</p> |
| | s (U8) | <p>This is an optional integer parameter to specify that data should be scaled according to the user specified scale factor. The number specified sets the number of decimal places of precision.</p> <p>When the "s" parameter is not specified, all data is sent as raw data that must be scaled according to the "scale" and "offset" arrays.</p> |
| | hdr (BOOL) | <p>Optional parameter to enable (1,2) or disable (0) the provision of the header information. The critical header header information is transmitted by default (hdr = 1). An extended header is transmitted (hdr = 2) includes additional documentation about the sensor. This includes the "label", "desc", "model", and "mac" information.</p> |
| | csv (BOOL) | <p>Specifies the use of CSV format (csv = 1) versus the default JSON format (csv = 0 or not specified). When csv = 1 is specified, the output format is a comma separated values list instead of JSON format.</p> |

Note that the sdata query can be initiated using "sdata.json" or the alias "sdata.csv".

6.2.2.1 Sensor Data Query: Realtime Data (mode=rt)

The “mode=rt” data query returns real-time data for the primary function of each channel (voltage, current, pulse count, or binary)

| | | |
|------------|-----------------|---|
| Command | sdata.json?m=rt | |
| Parameters | id (STR) | This is an optional string that will be sent back with the response. It can be used by a requester to tag the response with an identifier. |
| | s (U8) | This is an optional integer parameter to specify that data should be scaled according to the user specified scale factor. The number specified sets the number of decimal places of precision. When the “s” parameter is not specified, all data is sent as raw data that must be scaled according to the “scale” and “offset” arrays. |
| | hdr (U8) | Optional parameter to enable (1,2) or disable (0) the provision of the header information. The critical header information is transmitted by default (hdr = 1). An extended header is transmitted (hdr = 2) includes additional documentation about the sensor. This includes the "label", "desc", "model", and "mac" information. |

mode=rt Example (raw data):

Example Query:

<http://netmeter/sdata.json?m=rt>

Example Response:

```
{
  "cmd": "sdata.json",
  "time": 131914931,
  "ybase": 2010,
  "arg_hdr": 1,
  "arg_m": "rt",
  "cunit": "$",
  "cunithtml": "$",
  "pins": ["I0", "I1", "I2", "I3", "I4", "I5", "I6", "I7"],
  "names": ["I0", "FunctionGenerator", "Flow-2", "Flow-3", "I4", "I5", "I6", "I7"],
  "units": ["Pulses", "Volts", "CFM", "CFM", "CFM", "V", "V", "CFM"],
  "uhtml": ["Pulses", "Volts", "Ft. <sup>3</sup>/mi n", "Ft. <sup>3</sup>/mi n", "Ft. <sup>3</sup>", "V", "V", "Ft. &sup3;/mi n"],
  "uflow": ["CFM", "", "", "CFM", "", "", ""],
  "uflowhtml": ["Ft. <sup>3</sup>/mi n", "", "", "Ft. <sup>3</sup>/mi n", "", "", ""],
  "tflow": [1, 0, 0, 0, 60, 0, 0, 0],
  "cost": [0.1, 0, 0, 0, 0.1, 0, 0, 0],
  "scale": [1e+00, 9.99999776483e-03, 7.99999821186e-02, 5.00000074506e-02, 1e+00, 1.000000047497e-03, 1.000000047497e-03, 1.000000047497e-03],
  "offset": [0e+00, 0e+00, 0e+00, 4.900000095367e+00, 0e+00, 0e+00, 0e+00, 0e+00],
  "data": [4, 387, -1, 2510, 5, -1, 0, 0]
}
```

Example Query:

```
http://netmeter/sdata.json?m=rt&hdr=2
```

Example Response:

```
{
  "cmd": "sdata.json",
  "time": 131917017,
  "ybase": 2010,
  "arg_hdr": 2,
  "arg_m": "rt",
  "cunit": "$",
  "cunithtml": "$",
  "label": "OMNI -Factory-1",
  "desc": "OMNI Factory 1",
  "model": "NetMeter-OMNI-8C",
  "mac": "00:04:A3:F5:44:6B",
  "pins": ["I0", "I1", "I2", "I3", "I4", "I5", "I6", "I7"],
  "names": ["I0", "FunctionGenerator", "Flow-2", "Flow-3", "I4", "I5", "I6", "I7"],
  "units": ["Pulses", "Volts", "CFM", "CFM", "CFM", "V", "V", "CFM"],
  "uhtml": ["Pulses", "Volts", "Ft. <sup>3</sup>/mi n", "Ft. <sup>3</sup>/mi n", "Ft. <sup>3</sup>", "V", "V", "Ft. &sup3;/mi n"],
  "uflow": ["CFM", "", "", "CFM", "", "", ""],
  "uflowhtml": ["Ft. <sup>3</sup>/mi n", "", "", "Ft. <sup>3</sup>/mi n", "", "", ""],
  "tflow": [1, 0, 0, 0, 60, 0, 0, 0],
  "cost": [0.1, 0, 0, 0, 0.1, 0, 0, 0],
  "scale": [1e+00, 9.99999776483e-03, 7.99999821186e-02, 5.00000074506e-02, 1e+00, 1.000000047497e-03, 1.000000047497e-03, 1.000000047497e-03],
  "offset": [0e+00, 0e+00, 0e+00, 4.900000095367e+00, 0e+00, 0e+00, 0e+00, 0e+00],
  "data": [4, 341, 0, 2178, 5, -1, -2, -2]
}
```

The returned data contains the following elements:

| Mnemonic | Type | Description |
|----------|------|---|
| cmd | STR | The command for the query: "sdata.json" |

| Mnemonic | Type | Description |
|-----------|------|---|
| time | U32 | The current sensor time based on the time the query was transmitted. See Section 5.2 for a description of how time stamp values are formulated. |
| ybase | U16 | Year base: January 1 at midnight (hour 0) of ybase is the zero point in time for all time values. See Section 5.2 for a description of how time stamp values are formulated. |
| cunit | STR | Text format of the Currency Units as defined in the sensor configuration |
| cunithtml | STR | HTML format of the Currency Units as defined in the sensor configuration |
| pins | STR | A list of the physical pin names associated with each channel of data |
| names | STR | A list of the logical signal names associated with each channel of data as defined in the sensor configuration |
| units | STR | A list of the text mode units of measure for each channel of data as defined in the sensor configuration. These values may be used for display when plain text is desired and HTML rendering is <u>not</u> possible. |
| uhtml | STR | A list of the HTML version of the units of measure for each channel of data as defined in the sensor configuration. These values may be used for display when HTML rendering is possible (allows special characters to be displayed). |
| uflow | STR | A list of the text mode units of measure for each channel of data when that channel is to be converted to a flow rate. For example, when pulse mode is measuring cubic feet (CF) and the data will be displayed as CF per minute (CFM). |
| uflowhtml | STR | This is the HTML version of uflow. |
| tflow | U32 | The reference time period for flow calculations (in seconds). For example: if 1 pulse represents 1CF, then a tflow value of 60 would require the application to use a Δt value of 60s. |
| cost | F32 | The cost per unit for each channel as defined in the sensor configuration. |
| scale | F32 | The scale factor to convert the raw sensor data vales into the standard units of measure as defined in the sensor configuration. When data is being displayed in floating point format, the scale will be 1 as the defined scale will be applied by the NetMeter-OMNI. |
| offset | F32 | The offset to be added to the scaled sensor data vales as defined in the sensor configuration. The offset should be added <u>after</u> the scale value has been applied. When data is being displayed in floating point format, the offset will be zero as any defined offset will be applied by the NetMeter-OMNI. |

| Mnemonic | Type | Description |
|----------|------------------|---|
| data | S16, U48, or F32 | The realtime data for each of the channels. This will be either integer format ("s" not defined) or floating point (s is defined). For integer mode, analog values are in S16 format and are in units of mV or μ A. |

In the next example, the data is scaled by the NetMeter-OMNI instead of having to post process it:

Example Query:

```
http://netmeter/sdata.json?m=rt&s=1
```

Example Response:

```
{
  "cmd": "sdata.json",
  "time": 132870026,
  "ybase": 2010,
  "arg_s": 1,
  "arg_m": "rt",
  "unit": "$",
  "unithtml": "$",
  "pins": ["I0", "I1", "I2", "I3", "I4", "I5", "I6", "I7"],
  "names": ["I0", "FunctionGenerator", "Flow-2", "Flow-3", "I4", "I5", "I6", "I7"],
  "units": ["Pulses", "Volts", "CFM", "CFM", "CFM", "V", "V", "CFM"],
  "uhtml": ["Pulses", "Volts", "Ft. <sup>3</sup>/mi n", "Ft. <sup>3</sup>/mi n", "Ft. <sup>3</sup>/mi n", "V", "V", "Ft. &sup3;/mi n"],
  "uflow": ["CFM", "", "", "", "CFM", "", "", ""],
  "uflowhtml": ["Ft. <sup>3</sup>/mi n", "", "", "", "Ft. <sup>3</sup>/mi n", "", "", ""],
  "tflow": [1, 0, 0, 0, 60, 0, 0, 0],
  "cost": [0.1, 0, 0, 0, 0.1, 0, 0, 0],
  "scale": [1, 1, 1, 1, 1, 1, 1, 1],
  "offset": [0, 0, 0, 0, 0, 0, 0, 0],
  "data": [4, 4.1, 0.1, 717.5, 5, 0, 0, 0]
}
```

6.2.2.2 Sensor Data Query: Realtime Data (mode=rtall)

The "mode=rtall" data query returns real-time data for both the analog state and the counter value for each channel.

Example Query:

```
http://netmeter/sdata.json?m=rtall&s=1
```


Example Response:

```
{
  "cmd": "sdata.json",
  "time": 132871062,
  "ybase": 2010,
  "arg_s": 1,
  "arg_m": "rtall",
  "cunit": "$",
  "cunithtml": "$",
  "pins": ["I0", "I1", "I2", "I3", "I4", "I5", "I6", "I7"],
  "names": ["I0", "FunctionGenerator", "Flow-2", "Flow-3", "I4", "I5", "I6", "I7"],
  "units": ["Pulses", "Volts", "CFM", "CFM", "CFM", "V", "V", "CFM"],
  "uhtml": ["Pulses", "Volts", "Ft. <sup>3</sup>/mi n", "Ft. <sup>3</sup>/mi n", "Ft. <sup>3</sup>", "V", "V", "Ft. &sup3;/mi n"],
  "uflow": ["CFM", "", "", "CFM", "", "", ""],
  "uflowhtml": ["Ft. <sup>3</sup>/mi n", "", "", "Ft. <sup>3</sup>/mi n", "", "", ""],
  "tflow": [1, 0, 0, 0, 60, 0, 0, 0],
  "cost": [0.1, 0, 0, 0, 0.1, 0, 0, 0],
  "scale": [1, 1, 1, 1, 1, 1, 1, 1],
  "offset": [0, 0, 0, 0, 0, 0, 0, 0],
  "din": 10,
  "cnt": [4, 1469, 106, 568, 5, 0, 0, 0],
  "ain": [0, 11.5, 0.1, 717.3, 0, -0, -0, -0]
}
```

The returned data is similar to the “rt” mode query with the following differences:

| Mnemonic | Type | Description |
|----------|------------|---|
| din | U16 | Digital input state: The interpretation of the input voltage/current as digital data according to the threshold settings. The value should be parsed as a decimal integer number and the resulting value used as binary data where channel 0 is the least significant bit (bit 0) and channel 7 is the most significant bit (bit 7) |
| cnt | U48 or F32 | The pulse counter value. |
| ain | U16 or F32 | Analog input value. |

6.2.2.3 Sensor Data Query: Short-term Datalog (mode=ramlog)

The “mode = ramlog” data query returns the data from the short-term datalog. The short-term datalog contains the last 5 minutes of analog data (current or voltage) at a resolution of 1 second. This log is stored in volatile memory and is cleared each time the NetMeter-OMNI is powered up.

The ramlog is intended to provide fast data for viewing the state of each analog input channel. This may be useful to debug setup problems or to view analog inputs in fast real-time.

| | | |
|------------|---------------------|--|
| Command | sdata.json?m=ramlog | |
| Parameters | id (STR) | This is an optional string that will be sent back with the response. It can be used by a requester to tag the response with an identifier. |

| | |
|------------|--|
| s (U8) | <p>This is an optional integer parameter to specify that data should be scaled according to the user specified scale factor. The number specified sets the number of decimal places of precision.</p> <p>When the “s” parameter is not specified, all data is sent as raw data that must be scaled according to the “scale” and “offset” arrays.</p> |
| hdr (U8) | <p>Optional parameter to enable (1,2) or disable (0) the provision of the header information. The critical header header information is transmitted by default (hdr = 1). An extended header is transmitted (hdr = 2) includes additional documentation about the sensor. This includes the "label", "desc", "model", and "mac" information.</p> |
| span (U32) | <p>Time Span for the data of the query. Defaults to all available data.</p> |
| t0 (U32) | <p>Start Time for the data of the query. Defaults to 0 (all data in the ramlog is transmitted). All data after, but not including t0, will be transmitted. Consequently, incremental updates of this information may be obtained by setting t0 to the time stamp of the most recently loaded value.</p> |
| utc (S32) | <p>Time offset for csv mode: an offset used for the csv query mode to convert the UTC time of the NetMeter-OMNI to the desired time zone. The “utc” value is added to the NetMeter-OMNI time when the floating point time value is calculated. Consequently, utc is an integer number in seconds and may be positive or negative.</p> <p>See section 5.3 for more details.</p> |

Example Query:

Query:

```
http://netmeter/sdata.json?m=ramlog
```

Example Response:

```
{
  "cmd": "sdata.json",
  "time": 122656021,
  "ybase": 2010,
  "arg_m": "ramlog",
  "cunit": "$",
  "cunithtml": "$",
  "pins": ["I0", "I1", "I2", "I3", "I4", "I5", "I6", "I7"],
  "names": ["Steam-0", "Flow-0", "Steam-1", "Flow-1", "I4", "I5", "I6", "I7"],
  "units": ["mA", "CFM", "mA", "CFM", "V", "V", "V", "V"],
  "uhtml": ["mA", "Ft<sup>3</sup>/min", "mA", "Ft<sup>3</sup>/min", "V", "V", "V", "V"],
  "scale": [1.0e-03, 1.0e-03, 1.0e-03, 1.0e-03, 1.0e-03, 1.0e-03, 1.0e-03, 1.0e-03],
  "offset": [0e+00, 0e+00, 0e+00, 0e+00, 0e+00, 0e+00, 0e+00, 0e+00],
  "data": [[122656021, -12, 4447, -12, 4430, 696, 244, 179, 179],
    [122656020, -13, 4442, -11, 4427, 696, 242, 177, 177],
    [122656019, -12, 4439, -8, 4424, 694, 244, 179, 179],
    [122656018, -13, 4428, -9, 4425, 694, 244, 178, 178],
    [122656017, -12, 4424, -8, 4425, 696, 245, 179, 179],
    ...
    [122655722, -11, 2356, -10, 4468, 699, 244, 178, 178],
    [122655721, -11, 2351, -9, 4467, 701, 244, 178, 178]]
}
```

6.2.2.4 Sensor Data Query: Main Datalog (mode=ml)

The “mode = ml” data query returns the data from the main datalog. The main datalog is stored in non-volatile memory and the sample period may be configured as 5s, 15s, 30s, 1m, 2m, 5m.

The main datalog is configured to store either the analog value of the input or the pulse counter value for each channel.

| Command | sdata.json?m=ml | |
|------------|-----------------|---|
| Parameters | id (STR) | This is an optional string that will be sent back with the response. It can be used by a requester to tag the response with an identifier. |
| | s (U8) | This is an optional integer parameter to specify that data should be scaled according to the user specified scale factor. The number specified sets the number of decimal places of precision. When the “s” parameter is not specified, all data is sent as raw data that must be scaled according to the “scale” and “offset” arrays. |
| | hdr (U8) | Optional parameter to enable (1,2) or disable (0) the provision of the header information. The critical header information is transmitted by default (hdr = 1). An extended header is transmitted (hdr = 2) includes additional documentation about the sensor. This includes the “label”, “desc”, “model”, and “mac” information. |
| | t0 (U32) | Start Time for the data of the query. Defaults to 0. |

t1 (U32) End Time for the data of the query. Defaults to the current time.

span (U32) Time Span for the data of the query. Defaults to all available time.

Note: specify only one or two of the parameters t0, t1, span. The valid combinations are:

- t0, t1: data between t0 and t1
 - t0,span: data between t0 and t0 + span
 - t1,span: data between t1-span and t1
 - t0: data from t0 until now
 - t1: earliest data up until t1
-

interval (U8) The interval of the data to return in units of the sample interval.

The default value of 1 results in all data being returned (the interval is 1 main-log period).

A value of 2 returns every other data point (the interval is 2 main-log periods). A value of 3,4,5... returns every 3rd, 4th, 5th... data point.

The alignment of the data is based on the modulo boundary. For example: if the main-log period is 1-minute and an interval of 5 is selected (interval=5) then the data will be aligned to an even 5 minute value (00,05,10,15,20,25,30,35,40,45,50,55).

utc (S32) Time offset for csv mode: an offset used for the csv query mode to convert the UTC time of the NetMeter-OMNI to the desired time zone. The "utc" value is added to the NetMeter-OMNI time when the floating point time value is calculated. Consequently, utc is an integer number in seconds and may be positive or negative.

See section 5.3 for more details.

Example Query:

Query:

```
http://netmeter/sdata.json?m=ml &span=120&s=3
```

Example Response:

```
{
  "cmd": "sdata.json",
  "time": 132885418,
  "ybase": 2010,
  "arg_s": 3,
  "arg_m": "ml",
  "arg_span": 120,
  "cunit": "$",
  "cunithtml": "$",
  "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],
  "cost": [0, 0, 0, 0, 0, 0, 0, 0],
  "scale": [1, 1, 1, 1, 1, 1, 1, 1],
  "offset": [0, 0, 0, 0, 0, 0, 0, 0],
  "data": [[132885285, 0.252, 0.009, 40, -889.022, 896, 8.375, 3.71, 348935913.5],
[132885300, 0.244, 0.008, 40, -889.245, 896, 8.377, 3.706, 348937498.5],
[132885315, 0.24, 0.008, 40, -889.245, 896, 8.376, 3.696, 348939083.5],
[132885330, 0.224, 0.008, 40, -889.245, 896, 8.372, 3.689, 348940668],
[132885345, 0.244, 0.008, 40, -889.245, 896, 8.377, 3.687, 348942253],
[132885360, 0.248, 0.008, 40, -889.245, 896, 8.378, 3.683, 348943838],
[132885375, 0.248, 0.009, 40, -889.245, 896, 8.375, 3.677, 348945423],
[132885390, 0.244, 0.008, 40, -889.245, 896, 8.379, 3.671, 348947008],
[132885405, 0.244, 0.008, 40, -889.467, 896, 8.376, 3.669, 348948593]]
}
```

Here is the same query in CSV mode:

Query:

<http://netmeter/sdata.json?m=ml&span=120&s=3&csv=1>

Example Response:

```
time,132885554
ybase,2010
arg_s,3
arg_m,ml
arg_span,120
cunit,$
cunithtml,$
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
cost,0,0,0,0,0,0,0,0
scale,1,1,1,1,1,1,1,1
offset,0,0,0,0,0,0,0,0
data
41717.0256944,0.244,0.008,40,-889.245,896,8.376,3.665,348950178
41717.0258681,0.24,0.007,40,-889.467,896,8.374,3.656,348951763
41717.0260417,0.236,0.008,40,-889.467,896,8.376,3.65,348953348
41717.0262153,0.24,0.007,40,-889.467,896,8.373,3.646,348954933
41717.0263889,0.24,0.007,40,-889.467,896,8.374,3.644,348956518
41717.0265625,0.244,0.008,40,-889.467,896,8.376,3.64,348958102.5
41717.0267361,0.24,0.007,40,-889.467,896,8.375,3.637,348959687.5
41717.0269097,0.244,0.007,40,-889.467,896,8.375,3.635,348961273
41717.0270833,0.244,0.008,40,-889.467,896,8.375,3.629,348962858
```

Note the time stamp is in the Excel floating point format

6.2.2.5 Sensor Data Query: Coarse Main Datalog (mode=mlc)

The “mode = mlc” data query returns the coarse interval data from the main datalog. This is the same data as the “mode = ml” data query except that the data is more coarse in time interval as defined in Table 5.

Table 5: Relationship Between the Mainlog Period Setting and the Data Interval of the “mlc” Query

| Mainlog Period | Mainlog Coarse Interval |
|----------------|-------------------------|
| 5 s | 5 min |
| 15 s | 15 min |
| 30 s | 15 min |
| 1 min | 1 hour *default |
| 2 min | 1 hour |
| 5 min | 1 hour |

This query is useful for obtaining data for a long time span where the full data resolution is not required. The “interval” option with the “mode = ml” command can be used to obtain similar results. However, “mode = mlc” is faster to process.

Example:

Example Query for the “mlc” mode:

```
http://netmeter/sdata.json?m=mlc&span=7200&s=2
```

Example Response:

```
{
  "cmd": "sdata.json",
  "time": 132885927,
  "ybase": 2010,
  "arg_s": 2,
  "arg_m": "mlc",
  "arg_span": 7200,
  "cunit": "$",
  "cunithtml": "$",
  "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],
  "cost": [0, 0, 0, 0, 0, 0, 0, 0],
  "scale": [1, 1, 1, 1, 1, 1, 1, 1],
  "offset": [0, 0, 0, 0, 0, 0, 0, 0],
  "data": [[132879600, 0.23, 0.01, 40, -889.24, 896, 8.37, 0.19, 348335193.5],
    [132880500, 0.23, 0.01, 40, -889.24, 896, 8.38, 0.09, 348430273],
    [132881400, 0.26, 0.01, 40, -889.24, 896, 8.38, 4.44, 348525372],
    [132882300, 0.22, 0.01, 40, -889.47, 896, 8.37, 0.07, 348620476],
    [132883200, 0.21, 0.01, 40, -889.47, 896, 8.37, 0.07, 348715585],
    [132884100, 0.2, 0.01, 40, -889.47, 896, 8.38, 0.01, 348810692.5],
    [132885000, 0.23, 0.01, 40, -889.24, 896, 8.37, 3.87, 348905797],
    [132885900, 0.25, 0.01, 40, -889.24, 896, 8.38, 3.57, 349000898]
  ]
}
```

7 Using Excel to Access the API

Data from the NetMeter-OMNI may be automatically imported into Excel using the CSV mode of the API. This can be done in a number of ways. One of the simplest is to use the “Get External Data” → “From Text” option as pictured in Figure 3 (Excel 2010).

Step 1: Start a new workbook in Excel and press “Get External Data” → “From Text”

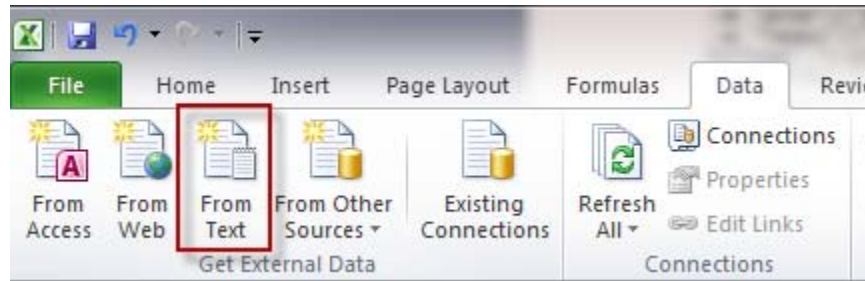


Figure 3: Get External Data → From Text

For the “Import Text File” dialog, enter the link for the sdata.json, something such as:

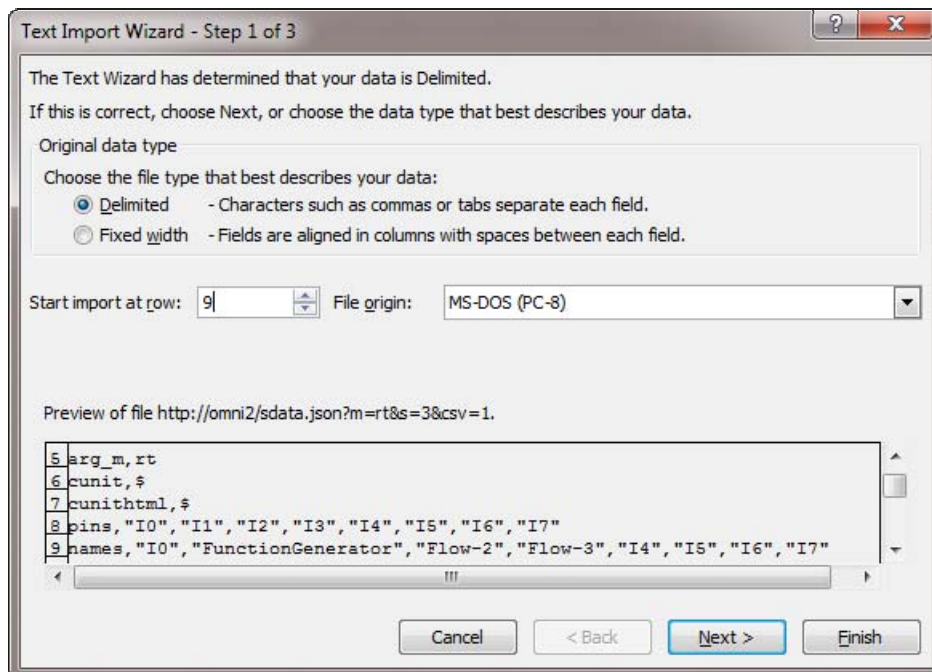
```
http://netmeter/sdata.json?m=rt&s=3&csv=1
```

This will capture the realtime data in the CSV format (change “netmeter” as required).

Press “Open” to continue.

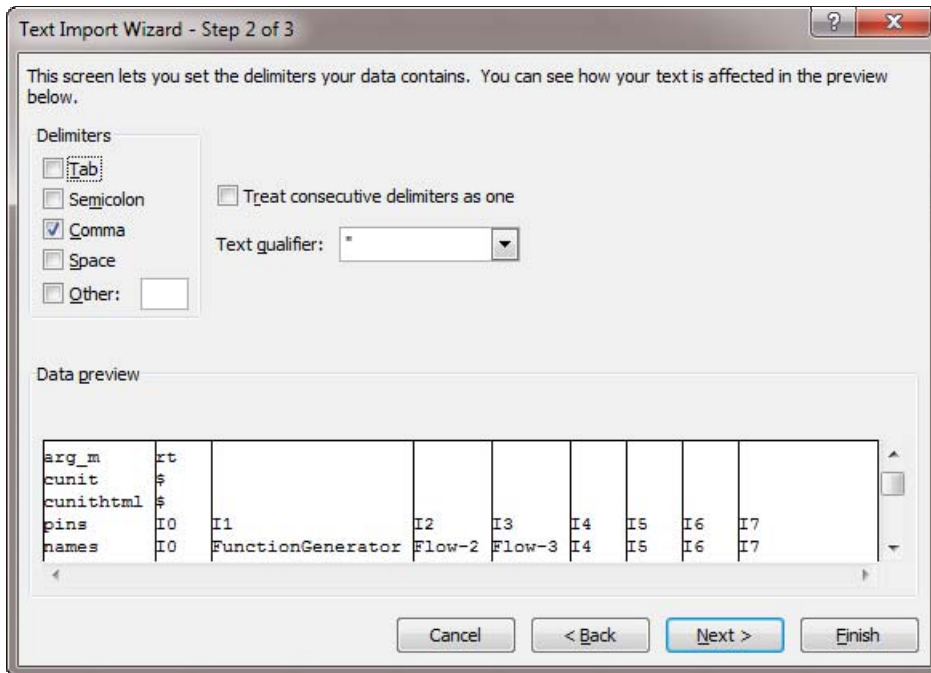
Step 2: Text Import Wizard

If the link was entered properly, the “Text Import Wizard” will show a preview of the data



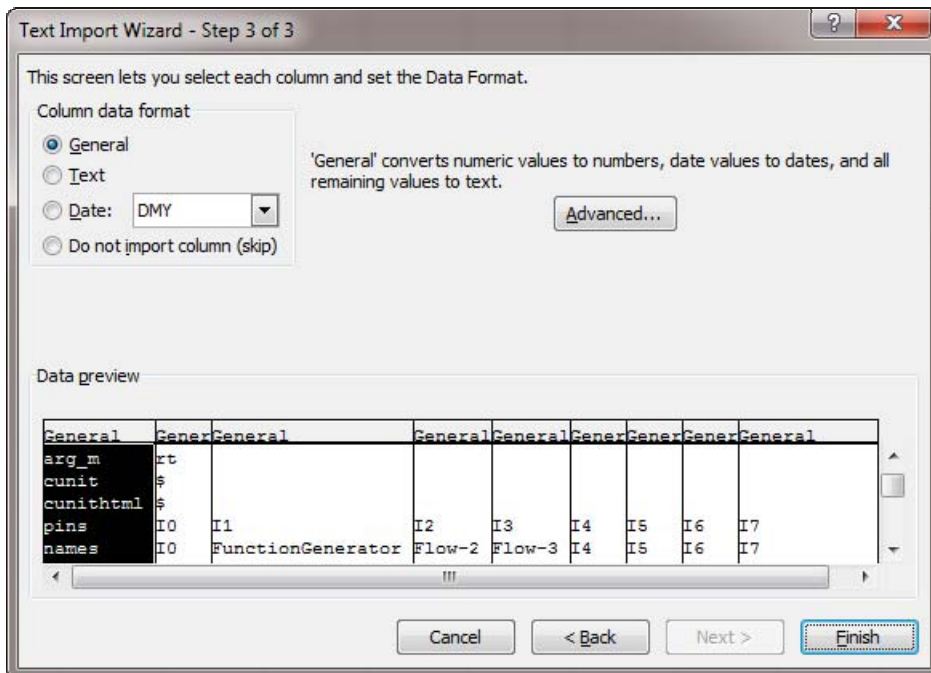
Set the “Start import at row” as required. Row 9 allows the signal names to be captured.

Click “Next” to continue:

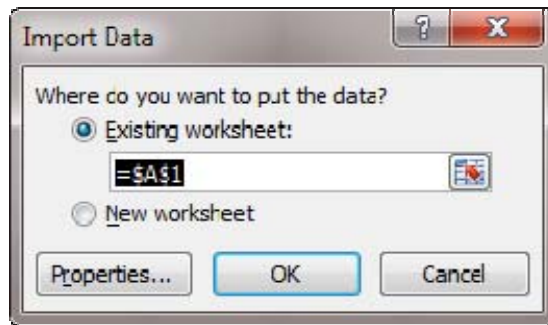


Select "Comma" as the delimiter instead of Tab.

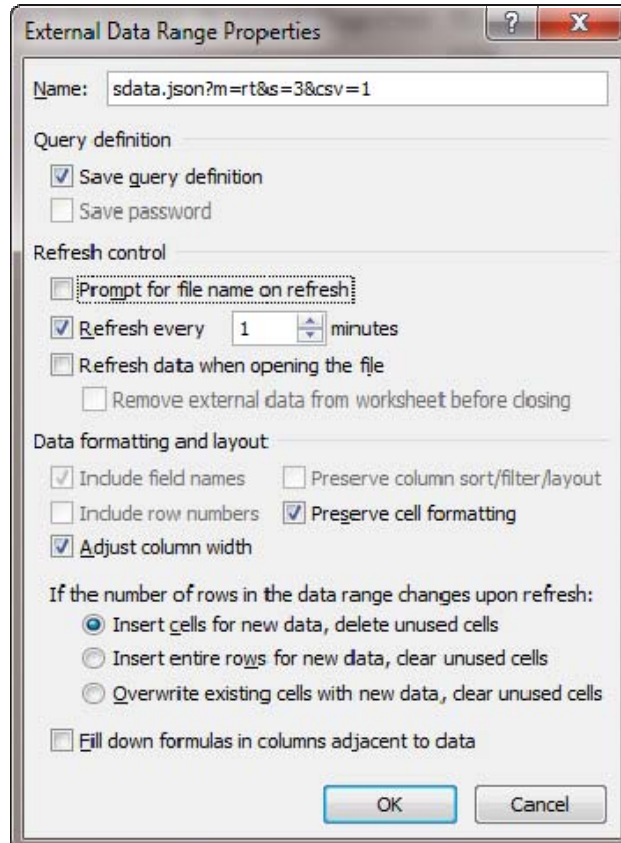
Click "Next" to continue:



Click "Finish" to continue:



Select the cell to place the data and press OK:



Step 3: Set the properties.

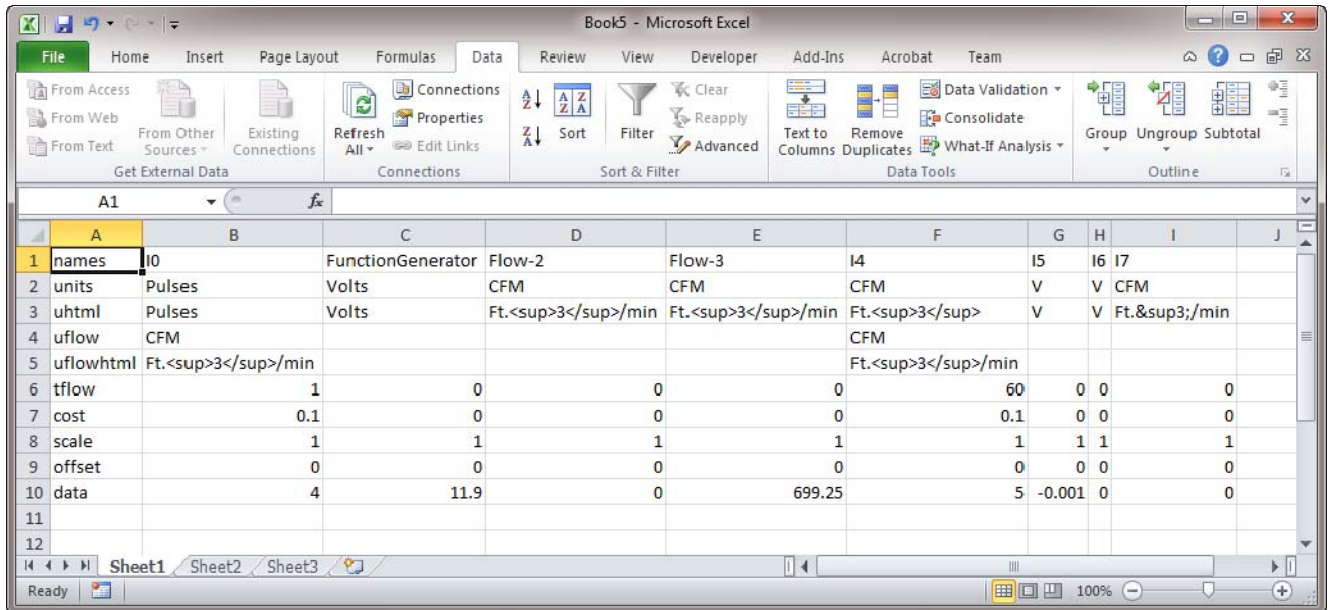
Uncheck "Prompt for file name on refresh".

Select "Refresh every 1 minutes".

Un-select "Adjust column width"

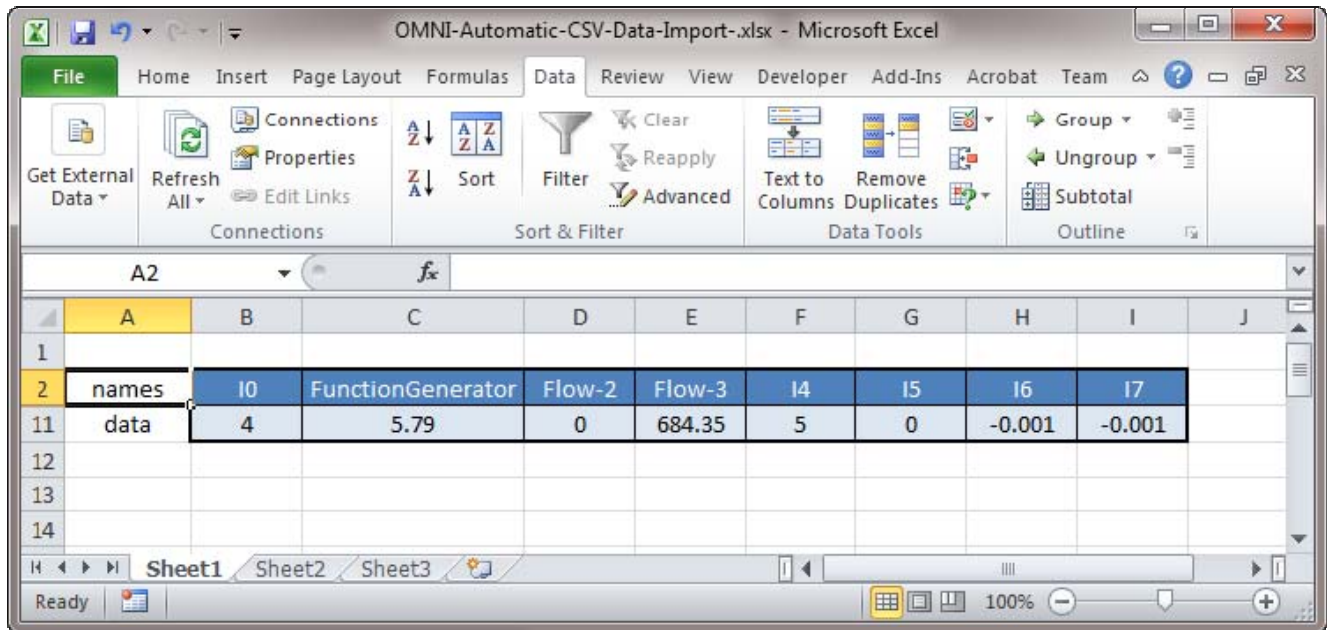
Press OK.

The query will be performed and the data populated into the spreadsheet:



Step 4: Format the Data

Format the cells and hide rows you don't need for the desired result:



The data in the spreadsheet will automatically update every minute as selected above.

When the insertion cell is selected, use the "Properties" command to change the data feed properties as desired.

7.1 Alternate Methods

Alternately, the JSON mode data may be imported using plug-ins such as:

- cDataSet.xlsm (<http://ramblings.mcpher.com/Home/excelquirks/json>)
- VB-JSON (<http://www.ediy.co.nz/vbjson-json-parser-library-in-vb6-xidc55680.html>)



www.z3controls.com

support@z3controls.com

Phone: 1-877-454-4436

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

Copyright © 2013 Z3 Controls Inc.