buildingOS_

510.907.0400
9AM-5PM Pacific Time
support@luciddg.com

Data Push Connector API

Overview

BuildingOS integrates with over 150 different building system control services, hardware manufacturers, and software vendors. In order to enable a new integration as fast as possible, this Push Connector provides four standard data formats that are designed for quick onboarding of new customer data. Provinding developers a scalable way to send data to one centralized cloud database, it enables new customers and partners to quickly deploy new and existing technology to the BuildingOS ecosystem of over 6,000 commercial buildings in 380+ organizations.

Data Transmission Standards

In the building intelligence field, two standards have emerged as ideal methods of automating machine to machine data communications from sensors, meters, gateways, databases, PCs, servers, and other mobile and industrial equipment to BuildingOS.

HTTP-POST (preferred)

This is our preferred method for transmitting data from modern applications that support a REST-based framework. Please note the following:

- Files pushed from your application to BuildingOS must be an HTTP POST request
- All requests must pass https://rest.buildingos.com/ as the URL (note the trailing "/" is significant)
- Only a POST is accepted. If a GET is sent the response will include a JSON error message and a status code of 409.

510.907.0400
Pacific Time
support@luciddg.com

- Only one file can be sent at a time. If more than one file is sent, the request will be denied and an error message will be returned with a status code of 409.
- Files can be of type .csv, .xml, or .json.
- JSON, XML, and CSV files must be passed as ‘Form-data’, with the appropriate Content-Type (*application/json* for json, *text/plain* for xml & csv)
- Sample Visual Basic example for transmitting files from a Windows directory available on request

FTP (legacy)

This method supports older applications that natively support file delivery via **sFTP** only. Please note the following:

- Standard FTP is **not** supported due to security concerns, you must use sFTP
- FTP over SSL (FTPS) is **not** supported at this time
- A URL, username, and password, and/or private key will be provided that is unique to each device or PC sending files.
- Files must be placed at the **root** of the FTP account - Do not create any subfolders or your files will not be transmitted to our cloud servers
- The FTP server is effectively an FTP -> HTTP-POST converter - all successfully transferred files are immediately sent as HTTP-POST request to BuildingOS and removed from the FTP root directory
- Whenever possible, we recommend sending files directly as an HTTP-POST request to remove this extra hop in the data transmission path for enhanced stability and on-going reliability

Application authentication & URI schema

The BuildingOS Push Connector does not require authentication. Devices and systems identify themselves through a unique URI specified in the *datasource* element of the HTTP request.

The *datasource* value should be a URI composed of the following elements:

<application>://<integration>/<gateway>

- **application –** always **bos**, which is shorthand for BuildingOS
- **integration** - a string of numbers, letters, or dashes (-) that serves as a unique identifier of a class of devices or gateways (i.e. Obvius-AcquiSuite). This is linked to the Tile displayed under the Integrations page in BuildingOS.
- **gateway –** a string of numbers, letters or underscores (_) that serves as a unique identifier of the device, building or other object in the source system. We recommend using a serial

number, MAC Address, UUID, or other series of alphanumeric characters easily found on a device or invoice for future support.

The following are examples of valid data source URIs

- bos://buildingos-csv/carnegie_library
- bos://buildingos-json/001EC60003DD
- bos://foobar-industries/123456_abcd
- bos://data-exporter-9000/site3125A_001

When developing your application, Lucid Integration or Solutions Engineers can assist with vetting URIs and ensuring compatibility with our system. All URIs must be unique within our system to ensure proper data mapping.

Meter Catalog

A meter catalog enumerates the metering points which will be reported to BuildingOS, and supplying a meter catalog enables automated discovery and quick meter commissioning in BuildingOS. Though optional, it is highly recommended that you implement the catalog as it significantly streamlines the setup process.

Once a catalog has been created following the first push of data, a user must log in to BuildingOS and connect the points in the catalog to BuildingOS so the system knows how to interpret the data that is sent, how to categorize it and what facility it is associated with.

Real-time Data Formats

JSON Schema

The JSON schema is composed of two parts, which may be passed in unison or at different instances, depending on required implementation and/or device interaction:

A *meter catalog* is composed of two required elements and up to three optional elements.

- *meterId* [required] – A unique string specifying the data point for which a value is being reported (Example: Electricity_Meter_1). The uniqueness of this string is only relative to the *gateway* - the same ID can be used again for each gateway.
- *meterName* [required] – A human readable string that the BuildingOS user will understand to aid in the commission of this point in BuildingOS
- *meterUnits* [optional] – Industry standard codes that represent the meter/points unit of measure (Example: kWh) - this could be very helpful during commissioning and is recommended.
- *meterLocation* [optional] – A string. On occasion the meterId is not sufficient to uniquely identify a point and in these cases a meterLocation can be used to provide an additional attribute for identification. If the meterLocation is used with the catalog, it must be used with the readings as well.
- *meterDescription* [optional] – A human readable string that can be used to add additional information for the user to aid in commissioning in BuildingOS.
- *meterTags* [optional] – A comma separated list of strings that enable grouping of meters/points (repeat

An *energy reading* is comprised of three required values plus two optional values:

- *timestamp* [required] - The time of the reading, expressed as an ISO8601-format timestamp (Example: `2013-01-17T09:23:00Z`).
    - Please note: Timestamps are assumed to represent the **beginning** of an interval. For example, a timestamp of 1:30 in quarter-hour data represents the interval from 1:30 to 1:44.
- *meterId* [required] - A unique string specifying the data point for which a value is being reported. (Example: `Electricity_Meter_1`). The uniqueness of this string is only relative to the *gateway* - the same ID can be used again for each gateway.
- *value* [required] - The floating-point value associated with the reading (Example: 21445.543)
- *meterLocation* [optional] – A string. On occasion the meterId is not sufficient to uniquely identify a point, thus the meterLocation can be used to provide an additional attribute for identification.
- *buildingLocalTime* [optional] - A boolean true/false value. true indicates the timestamp reflects your local time zone; false indicates that the timestamp reflects UTC. This value defaults to false.

CSV Schema

BuildingOS can generate Meter Catalogs from two types of CSV schemas natively, to reflect the two most common formats found with existing Building Automation exports.

- Header-based
    - This format is such that the first set of values in each .csv row identify the timestamp, which each subsequent value representing data for the corresponding timestamp.
    - The first row contains the unique identifier (VendorMeterID) that serves to identify each subsequent row of values
    - Timestamps must be ordered from oldest to newest
    - Example:

Timestamp	{meterId_1}	{meterId_2}	{meterId_3}	{meterId_4}
{oldest_timestamp}	{value}	{value}	{value}	{value}
{newest_timestamp}	{value}	{value}	{value}	{value}

- Tuple-based (no headers)
    - This format is such that each row contains one or two identifiers, with no headers
    - Timestamps must be ordered from oldest to newest
    - Example:

{meterId_1}	{oldest_timestamp}	{value}	{optional_meterLocation_1}
{meterId_2}	{oldest_timestamp}	{value}	{optional_meterLocation_1}
{meterId_3}	{oldest_timestamp}	{value}	{optional_meterLocation_1}
{meterId_1}	{newest_timestamp}	{value}	{optional_meterLocation_1}
{meterId_2}	{newest_timestamp}	{value}	{optional_meterLocation_1}
{meterId_3}	{newest_timestamp}	{value}	{optional_meterLocation_1}

510.907.0400
Pacific Time
support@luciddg.com

Bill Data Formats

JSON Schema

The JSON file top level elements are constructed as follows:

```
{
  "datasource":"bos://buildingos-bills/<org-url-element>",
  "constructCatalog":true | false,
  "constructReadings":true | false,
  "bills":[],
  "meterCatalog":[],
  "readings":[]
}
```

- *datasource* [required] – the BuildingOS URI, described above in "How does my application authenticate?". The vendor name is agreed up between Lucid and the partner. The organization is known by Lucid (the urlElement of the organization) and is relayed to the vendor when the account is set up. This element is **required** if not located within the JSON, an error message will be returned with a status code of 409.
- *constructCatalog* [optional] – If true this tells BuildingOS to construct a meter catalog to enable BuildingOS ConnectNow functionality. If false, the partner **must** send the separate meterCatalog element (see below) for the BuildingOS user to be able to leverage Connect Now functionality. If passing false, the attribute **must** be available whenever sending JSON to ensure consistency. This defaults to true
- *constructReadings* [optional] – If true this tells BuildingOS to disaggregate and construct meter readings using BuildingOS calendarization of dates and time from the meter readings in the bill. If false, the partner must send the separate readings element (see below) for the BuildingOS user to be able utilize the data within BuildingOS. If passing false, the attribute **must** be available whenever sending JSON to ensure consistency. This defaults to true. Since the bills element is not required no error is returned if the bills element is not present.

Bills element

The bills element is a list of associative arrays (dictionaries/maps) corresponding to monthly billing items. This element **is not** required and can be sent separately. The attributes startDate, endDate, meterId, consumption and cost are required. If these attributes are not present the system will return an error message with a status code of 409. The bills element is structured as follows:

```
"bills":[
  {
     "startDate":"12/09/2013",
     "endDate":"12/16/2013",
     "accountNumber":"123456-789",
     "meterId":"dummy_lnp_234567",
     "billId":"dummy_lnp_123",
     "consumptionUnit":"kWh",
     "consumption":149607,
     "demandUnit":"kW",
     "demand":392,
     "costUnit":"USD",
     "cost":17844.13,
     "consumptionCost":944.39,
     "demandCost": 844.04,
     "serviceCharges": 74.89,
     "taxes": 56.13,
     "otherCharges": 348.11

   }
]
```

- *startDate* [required] – the en_US formatted date the billing period began.
- *endDate* [required] – the en_US formatted date the billing period ended.
- *accountNumber* [optional]– the account number is a string value.
- *meterId* [required] – The unique vendor identifier for this meter. This string value only needs to be unique to the vendor.
- *billId* [optional] – The unique bill identifier for this meter. This field can be used to push edits or changes to a specific bill.
- *consumptionUnit* [optional] – the consumption unit is a string that defines the standard unit for the source systems unit of measure for consumption. The default is kWh.
- *consumption* [required] – an integer or float representation of the consumption for this period.
- *demandUnit* [optional] – the demand unit is a string that defines the standard unit for the source systems unit of measure for demand. The default is kW.
- *demand* [optional] – an integer or float representation of the demand for this period.
- *costUnit* [optional] – The three letter ISO 4217 currency code. This defaults to USD.
- *cost* [required] – The float value (without locale formatting) of the cost.

- *consumptionCost* [optional] – The float value (without locale formatting) of the consumption cost.
- *demandCost* [optional] – The float value (without locale formatting) of the demand cost.
- *serviceCharges* [optional] – The float value (without locale formatting) of the cost of service charges.
- *taxes* [optional] – The float value (without locale formatting) of the cost of taxes.
- *otherCharges* [optional] – The float value (without locale formatting) of the cost adjustments, credits, and any other charges.

Meter Catalog element

The meterCatalog element is a list of associative arrays (dictionaries, maps) that are used by BuildingOS to define meters. This element can be sent to BuildingOS separately but will not be processed unless constructCatalog is false in the main elements of the JSON (see above). This is used when greater detail is desired when defining the meters for the readings related to bills. This element **is not** required and can be sent separately. The following is an example of the meterCatalog element:

```
"meterCatalog": [
     {
        "meterId": "dummy_lnp_123456",
        "meterName": "CC4 Sub Main T",
        "meterLocation": "westend",
        "meterUnits": "kWh",
        "meterDescription": "West Mechanical Room",
     }
  ]
```

A meter catalog is composed of two required elements and up to three optional elements.

- *meterId* [required] – A unique string specifying the data point for which a value is being reported (Example: `Electricity_Meter_1`). The uniqueness of this string is only relative to the *gateway* - the same ID can be used again for each gateway.
- *meterName* [required] – A human readable string that the BuildingOS user will understand to aid in the commission of this point in BuildingOS.
- *meterLocation* [optional] – A string. On occasion the meterId is not sufficient to uniquely identify a point and in these cases a meterLocation can be used to provide an additional attribute for identification. If the meterLocation is used with the catalog, it must be used with the readings as well.

- *meterUnits* [optional] – Industry standard codes that represent the meter/points unit of measure (Example: kWh) - this could be very helpful during commissioning and is recommended. This defaults to kWh.
- *meterDescription* [optional] – A human readable string that can be used to add additional information for the user to aid in commissioning in BuildingOS.

Readings element [optional]

The readings element is only sent when a vendor has a different calendarization algorithm for the disaggregation of the bills than is available in BuildingOS. The readings element are associated with the meters and will not be processed unless constructReadings is false. This element **is not** required and can be sent separately. The following is an example of the readings element:

```
"readings": [
     {
        "meterId": “dummy_lnp_123456",
        "billId": “dummy_lnp_123",
        "meterLocation": "westside",
        "buidlingLocalTime": true,
        "timestamp": "2013-12-11T23:12:00",
        "value": 123456.78
     }
  ]
```

The meterId, timestamp and value are required in all readings. Readings will not be processed if all three are not available. If meterId is not available an error message will be returned with a status code 409.

- *meterId* [required] - The meterId is a string that the source system maintains to uniquely identify a meter referenced in the bills. This must the same value as the **meterId** in the bill element (see above).
- *billId* [optional] - The billId is a string that the source system maintains to uniquely identify a bill referenced to a meter. This must the same value as the **billId** in the bill element (see above). This field can be used to push edits or changes to a specific bill.
- *meterLocation* [optional] – The meterLocation is used when processing data within BuildingOS. This should only be used when a catalog was created with the meterLocation for a vendor that **will not** construct readings from the bills element, but will process readings using what is defined here. This is an additional attribute that is available for vendors to ensure the uniqueness of a given meter. Its type is string.

- *buildingLocalTime* [optional] – If true the timestamp is assumed to be in the time zone of the associated building and will be converted to UTC for processing. The default is false. All timestamps are assumed to be in UTC.
- *timestamp* [required] - The ISO 8601 formatted date/time as a string. There is no need to designate timezone with the time stamp. The timezone is relative to the building that the meter is associated with.
- *value* [required] – The integer or float value for this reading.

CSV Schema

The CSV schema mirrors our manual bill upload format. Note the following two rows are a descriptive example, the first provides meta-data on the type of values expected, and the second is a sample reading for Account '12345678' and Service/MeterID '7654321'

Start Date	End Date	Account Number	ServiceId	Consumption Unit	Consumption	Demand Unit	Peak Demand	Total Cost ($)	Consumption Cost ($)	Demand Cost ($)	Taxes ($)	Service Charges ($)	Supply Charges ($)
MM/DD/YY	MM/DD/YY	Alphanumeric	Alphanumeric	(Energy)	(Value)	(Power)	(Value)	Omit $	Omit $	Omit $	Omit $	Omit $	Omit $
1/1/15	2/16/15	12345678	7654321	kWh	1,234	kW	8.1	1,234.56	765.43	432.10	37.03	0.00	0.00

Server Response

BuildingOS will respond to an HTTP-POST transmission with one of the following codes:

- In response to a valid HTTP POST, BuildingOS will return 200 OK (in addition, JSON files will return a response indicating how many catalog entries and/or readings created)
- A 400 error will be returned if:
  - the 'file' and 'datasource' keys are missing in the Request (or JSON file)
  - the Content-Type attributes are specified incorrectly
  - the system is unable to determine the proper gateway type
  - no or invalid data is passed for the 'file' key.
- A 500 error will be returned if:
  - the wrong URL is passed (i.e. api.buildingos.com instead of rest.buildingos.com)
  - the system is down (please contact Lucid support)
- Please contact Lucid support if any other status codes are returned