Ibis Public API V1 Documentation

Revision 1.2 – 2018-09-21 (Beta)

Introduction

The Ibis Public API is a standard RESTful API, with three main endpoints:

- **/config** -- providing data about sockets, data streams, their categorization for reporting purposes, system users, and schedules for socket control.
- **/data** -- providing access to data measured on InteliSockets, including detailed time series data, summary statistics, and baseline forecasts for calculating savings.
- **/control** -- providing the ability to control a socket, which means to turn power on and off for most existing models, but may include other control points in future models.

The API is served via HTTP and accessed over SSL/TLS for transport security.

Authentication during the Beta phase is via an API key passed in the **Authorization** HTTP header. The API server first authenticates the validity of the API key passed, and then uses the organization to which the API key is tied to authorize access to the requested resource. Authorization for resources follows the organization hierarchy, so if you create sub-organizations within Ibis.io to manage different installations, campuses, or companies, you can perform API requests against any resource in any of your sub-organizations using one main API key. You may not access data or resources in organizations that you do not own and will receive an appropriate HTTP error (403) for such requests.

Ibis will be adding OAuth2 as an authentication protocol to the V1 API in the near future, to increase the security of the API and support authentication by applications built by Ibis customers and ISVs.

General Usage Notes

Base Endpoint

All queries to the Ibis API in production begin with the following base URL:

https://data.ibis.io

Resource Identifiers

In the API, resources are identified by ID numbers. In nearly all cases, resource ID numbers will be integers (e.g., 12345), especially if they represent abstract concepts such as a Location or Schedule. The time series of data values collected by an InteliSocket (or "data stream") is also represented by an integer ID field. These ID values can be obtained by relevant **/config** API queries and are passed into many other queries for specific resources.

Physical resources such as InteliSockets are identified instead by ID numbers in hexadecimal (e.g., 2A45B1). These ID numbers are the same as those printed on the socket or other Ibis hardware device. APIs which refer to sockets or other Ibis hardware products always take this hexadecimal ID, whether to query metadata about the socket, or to control its power on/off state.

Controlling a physical socket employs the ID in hexadecimal format today. With dual sockets (IS-302), only one of the two plugs is controllable (the bottom one, with a green box and label around it) and it is this plug that is controlled when sending a request to the Control API.

Sockets and data streams are separate concepts in the Ibis API and Ibis.io because an InteliSocket or other Ibis hardware product may have multiple physical power sockets (e.g., dual sockets) or additional sensors. Each of these data-producing components generates a separate stream of time series data, for which all of the same variables are collected (e.g., power, voltage, power factor, energy).

HTTP Headers

All API requests should include the following header, to ensure that you receive properly formatted responses:

Accept: application/json

Calls to the API are authenticated and authorized by an API key, which is passed in an HTTP header. API keys are issued by Ibis Networks, and are customer-specific and should not be shared or published. The format of the HTTP header is as follows (the example is a mock API key, for illustration purposes):

Authorization: Ibis 9409B249D03C7BF191CB5B17391521834

API keys are available by talking to your reseller or contacting Ibis Networks.

API Response Format

All API responses return valid JSON responses formatted as UTF-8 text.

A valid query returns a response organized into several blocks. The top-level **query** block shows the query associated with the response, the time at which the query was processed on the API server (in Unix timestamp format), and the date/time format used in the **results** block.

The top-level **messages** block records human-readable messages from the server, including explanatory text if there are issues with a query. With a valid query this will usually simply indicate that the status is “ok”.

Finally, the top-level **results** block contains the substantive response to the API query, in JSON format as well. In the example shown here, the query asked for metadata on the organization whose ID is 8004, which corresponds to the Ibis Networks office in one of our test systems.


```
{
   "query": {
       "url": "/config/v1/501/organizations",
       "execution_time": 1505242627,
       "time_format": "timestamp"
   },
   "messages": {
       "status": "ok"
   },
   "results": [
       {
           "name": "Ibis Networks Office",
           "dp60_days": 45,
           "is_active": true,
           "price_model_id": 13,
           "updated_at": 1504040978,
           "org_type": "customer",
           "mode": "normal",
           "timezone_name": "Pacific/Honolulu",
           "id": 501,
           "core_hours_end_time": {
               "second_of_day": 61200,
               "second": 0,
               "minute": 0,
               "hour": 17
           },
           "core_hours_start_time": {
               "second_of_day": 25200,
               "second": 0,
               "minute": 0,
               "hour": 7
           }
       }
   ]
}
```


You may also see error responses, which in addition to the HTTP response code (indicating the class of error), the response body is JSON-formatted and attempts to provide human readable guidance about the source of the error. In the following

example, an Authorization header was present, but did not have an Ibis-formatted API key. This response was accompanied by a standard HTTP 401 Not Authorized error.

```
{
   "is_test": "False",
   "message": "[auth] Ibis API key required as authentication token",
   "type": "public_api_classifications",
   "isError": true
}
```

Time Format in Parameters and Response Data

API methods may send responses that contain date and time information (not all do). Methods may also take date and time information as input parameters.

Date and time information in the Ibis Public API is formatted in one of three ways:

time_format	Input or Output Format
timestamp	<SSS…>, following Unix format
utc	<YYYY-MM-DD>T<HH:MM:SS>Z
local	<YYYY-MM-DD>T<HH:MM:SS>

"Local" time always refers to the time zone associated with the organization associated with the API key passed in the query. This can get confusing if you have a large organization with many sub-organizations which have different time zones. In such cases, we strongly suggest operating with UTC or Unix timestamps and performing conversions as needed for the display of the data.

Methods that take date/time parameters infer the time format given the above. For example, passing the date and time "2017-01-10T00:00:00" to an API call will be interpreted as a local time, with the time zone treated as the zone listed for the organization in Ibis.io to which the socket and data stream belong. Adding "Z" to the end of an input time parameter tells our API services to interpret the date and time as referring to UTC.

Some methods, especially those that return time stamped information, possess a parameter called "**time_format**". This parameter governs the format in which date time information is returned to the caller. Input time information is still inferred via its format. This allows you to make queries in Unix time stamps, for example, in an automated process, but return UTC formatted date and times for display.

The default format for date and time information is “timestamp,” so if the **time_format** parameter is omitted from a request, all date and time information in API responses will be formatted as Unix timestamps, giving the number of seconds since “epoch”, which is midnight on January 1, 1970 UTC. Unix timestamps are easy to translate into other local time zones using software libraries in nearly every programming language.

There are two important notes about date/time stamps in addition to the general guidelines above.

First, when date and time information are sent as parameters (e.g., to **/data/time_series** to retrieve a set of power measurements), the API is capable of parsing “incomplete” date time strings, assuming they make sense. You can, for example, omit seconds from a UTC timestamp, and the API server will infer that as rounded to the nearest unit as specified in the “**granularity**” parameter. For example, a start time of “2017-08-05”, in hour-level granularity is interpreted as referring to midnight in the organization’s local time zone.

Second, day-level granularity data points **always** refer to an average over the 24 hour period from midnight, in the organization’s local time zone, through midnight of the following day (11:59:59). When you make a query for day-level data in the API, the date format in the results section of the response will always simply refer to the day (e.g., “2017-10-01”, “2017-10-02”, and so on).

Date and time formatting in software systems is a complex topic, particularly when it involves converting between time zones. We cannot support every code library in every language, but if you are developing an application against the Ibis API and have a question about how to format parameters or interpret output, please let us know and we will assist if possible.

Organization Filtering in Queries

For a large organization with a number of sub-organizations of large installations, a query may result in more information than you need. This is because the Ibis API will return information on any resources that the caller is authorized to access, and access is controlled by the organizations associated with your API key credentials. The response to an API call usually includes information for the requested resources from the organization to which the API key is attached, *and all of its sub-organizations*.

Queries can be filtered by organization to narrow their scope. This is done by inserting an organization ID (obtained by querying the **/config/v1/organizations** method) in the query path, after the API version number and before any methods. For example, here in the Ibis test lab, if we issue the query:

/config/v1/data_streams/intelisockets

the response contains information on data streams (e.g., which socket they associated with, their start and end times) for several dozen organizations, amounting to tens of thousands of entries in the response JSON. If we do not need all of that data, and for example are simply interested in devices within Ibis Network's main office, we could issue the following query instead:

/config/v1/501/data_streams/intelisockets

This query returns only those sockets listed under the sub-organization for the Ibis Networks office, which is a considerably smaller list than the previous query. Filtering your queries by sub-organization is an important technique in large installations, because returning large result sets takes much longer, and can be costly for heavy API use.

Note that sub-organization filtering is optional in all queries, and is independent of the target ID passed to a query, which may filter a query down to a single resource. For example, the following queries return the same (single) resource:

/config/v1/hardware/intelisockets/a7de7d

/config/v1/8004/hardware/intelisockets/a7de7d

Thus, sub-organization filtering is mainly useful when you are querying a collection of resources.

Classifications: Devices and Locations

The main use case for the Ibis API V1 is to allow customers to retrieve usage data and other information for subsets of InteliSockets and associated data streams. The data may then be used to build your own reports, perform your own analyses, or create custom monitoring for conditions that do not have generic alerts within our platform. In any sizable installation, you will frequently want to build those reports or analyses on subsets of data, broken out by the type of hardware you have plugged into the sockets, by location within your installation, or both.

Ibis.io facilitates this by classifying data streams by Location and Device (class), which you assign at installation time, and change anytime in the user interface. Many of our default reports in Ibis.io then show you usage summaries broken down by location or device class.

Within the API you can also query the list of Location and Device classes, find their ID numbers, and then use these IDs to filter requests for data stream IDs. This allows you to then retrieve time series data from categories of devices (e.g., computers, LCD projectors) or locations (e.g., Building A), or both.

Configuration API

The Configuration API, with its endpoint at /config/v1, responds to GET requests for data about resources within the Ibis InteliNetwork system. This is the same information you can see and configure within Ibis.io, along with numerical IDs to facilitate further API queries, and additional metadata in some cases.

Organizations

The **/config/v1/organizations** method is used to retrieve information about a specific organization, or sub-organizations under a top-level admin organization. This information includes metadata such as the organization's time zone, or core hours start and end, which can be useful in processing time series data retrieved from the API and performing a variety of analyses.

`GET` /config/v1/<org_id>/organizations

parameters	contents	required?
<org_id>/	organization id number, for filtering response scope	no
active=	true | false	no

Example:

/config/v1/organizations?active=true

Retrieve information about the organization to which the API key belongs, and all of its sub-organizations, filtering out any inactive organizations.

Data Streams

The **/config/v1/data_streams** method is used to retrieve metadata on the data streams coming from hardware plugged into InteliSockets. This metadata includes data stream ID numbers, their Device and Location classes, and the timestamp giving the earliest data stored in Ibis.io. All of this is useful in crafting further queries for time series data. You can retrieve this information for classes, or for individual sockets or data streams. An individual data stream is queried by giving its ID as the final path parameter; other types of queries are fashioned using combinations of the parameters in the query string.

`GET` /config/v1/<org_id>/data_streams/intelisockets/<ds_id>

parameters	contents	required?

<org_id>/	organization id number, for filtering response scope	no
<ds_id>/	data stream id number	no
sockets=	comma separated list of hexadecimal id numbers	no
active=	true | false	no
locations=	Location id number(s), filters data stream list returned	no
devices=	Device id number(s), filters data stream list returned	no

Examples:

/config/v1/8004/data_streams/intelisockets/30452
Retrieves metadata on a single data stream in the Ibis Networks Office test lab.

/config/v1/data_streams/intelisockets?devices=1&active=true
Metadata from all active computers (device class = 1) from authorized organizations

Hardware

The **/config/v1/hardware** method retrieves the current control state of an InteliSocket. At present, this state is either “on” or “off,” recording whether the socket is shut off, or providing power to the associated plug.

`GET` /config/v1/<org_id>/hardware/intelisockets/<hw_id>

parameters	**contents**	**required?**
<org_id>/	organization id number, for filtering response scope	no
<hw_id>/	hardware id number in hexadecimal	no

Example:

/config/v1/8004/hardware/intelisockets
Retrieve the hardware control state for all InteliSockets installed for the specified organization.

/config/v1/8004/hardware/intelisockets/a7de7d
Retrieve the hardware control state for a specific InteliSocket, identified by its hexadecimal hardware ID (which is also printed on the socket itself).

Devices

The **/config/v1/devices** method retrieves the current set of Device class categories set up by an organization within Ibis.io. The ID numbers returned are then useful for filtering in queries for data streams (see **/config/v1/data_streams**).

`GET` /config/v1/<org_id>/devices/<dev_id>

parameters	contents	required?
<org_id>/	organization id number, for filtering response scope	no
<dev_id>/	device id number	no

Example:

/config/v1/8004/devices

Retrieves the list of global and organization-specific device categories

/config/v1/8004/devices/1

Retrieves the list of global and org-specific categories under “Computer”

Locations

The **/config/v1/locations** method retrieves the current set of Location class categories set up by an organization within Ibis.io. The ID numbers returned are then useful for filtering in queries for data streams (see **/config/v1/data_streams**).

`GET` /config/v1/<org_id>/locations/<loc_id>

parameters	contents	required?
<org_id>/	organization id number, for filtering response scope	no
<loc_id>/	location id number	no

Example:

/config/v1/8004/locations

Retrieves the list of organization-specific location categories

/config/v1/8004/locations/12

Retrieves the list of org-specific categories under a root location (e.g., “Building 1”)

Schedules

The **/config/v1/schedules** method retrieves the available schedules for an organization (and sub-organizations), and the details for individual schedules. Each schedule refers,

potentially, to sets of Device and Locations, as well as individual sockets if desired, and records times that Ibis.io ought to turn sockets off and then back on to achieve energy savings.

There are two views of a schedule. The first returns the abstract schedule, which applies on/off controls on weekdays, versus weekends or holidays, to specified groups of sockets. The second view of a schedule "flattens" it out to a specific time range, giving the specific times that control events will occur within a range defined by **start_time** and **end_time**. This second query form means that API users do not have to manually calculate the effect of a schedule, and can simply retrieve a list of control events.

That list of specific control events can either reflect control events that already occurred in the past, or if you specify start_time and end_times that are in the future, the **/config/v1/schedules/flattened** method will project the schedule for that future time frame and tell you when the schedule will turn those sockets on and off.

`GET` /config/v1/<org_id>/schedules/<sch_id>

parameters	**contents**	**required?**
<org_id>/	organization id number, for filtering response scope	no
<sch_id>/	schedule id number	no

`GET` /config/v1/<org_id>/schedules/<sch_id>/flattened?<query_parameters>

parameters	**contents**	**required?**
<org_id>/	organization id number, for filtering response scope	no
<sch_id>/	schedule id number	no
start_time=	<time_input>	yes
end_time=	<time_input>	yes
time_format=	local | utc | timestamp	no

Examples:

/config/V1/8004/schedules

Retrieves the list of schedules that have been configured in the Ibis offices

/config/v1/8004/schedules/15
Retrieves the details of a specific schedule

/config/v1/8004/schedules/76/flattened?start_time=2017-12-01T00:00Z&end_time=2017-12-31T23:59Z
Projects the effect of a specific schedule on its configured sockets, for the month of December 2017

Data API

The Data API, with its endpoint at **/data/v1**, responds to GET requests for data collected for data streams through InteliSockets.

Time Series

Data collected from InteliSockets are in the form of time series with values averaged over a specified time interval (granularity), and are numerical values paired with a date and time value. The **/data/v1/time_series** method allows retrieval of this type of data from Ibis.io.

Queries can target a single socket or data stream, or ask for data from a list of sockets or data streams. Each query must target one "field key," which identifies a variable or measurable quantity collected from each socket (e.g., voltage, power). A valid query also specifies a time range, within the valid data range of the data stream.

The valid range for which a data stream has stored data is identifiable by a prior query to **/config/v1/data_streams/intelisockets**, using the "**start_time**" field. Callers of this method can specify a detailed date and time parameter for the "**end_time**" field, or use the literal "**now**" to get data up through the most recent stored data points.

Finally, a query must specify the granularity of data required for data in the requested range. Minute-level data is available for 45 days, with hour and day-level granularity available for the past 365 days.

`GET /data/v1/<org_id>/time_series/intelisockets/<field_key>?<query_parameters>`

parameters	contents	required?
<org_id>/	organization id number, for filtering response scope	no
<field_key>/	power | energy | power_factor | voltage | current	yes

data_streams=	comma separated list of id numbers	yes*
sockets=	comma separated list of hexadecimal id numbers	yes*
start_time=	<time_input>	yes
end_time=	<time_input> | now	yes
granularity=	day | hour | minute	yes
time_format=	utc | local | timestamp	no

*Time Series requires exclusive use of either data_streams or sockets as a parameter.

Examples:

/data/v1/8004/time_series/intelisockets/power?data_streams=30318,30445,30452&start_time=2017-08-08T20:00:00Z&end_time=2017-08-09T19:59:00Z&granularity=minute

Retrieves power usage data for three data streams, for a 24 hour period, at minute level

/data/v1/8004/time_series/intelisockets/voltage?data_streams=30452&start_time=2017-09-01T00:00:00Z&end_time=now&granularity=hour&time_format=utc

Retrieves voltage data averaged on an hourly basis, from the beginning of Sept 2017 until the present, with date/time information returned in ISO 8601 format in the UTC timezone.

/data/v1/8004/time_series/intelisockets/voltage?data_streams=30452&start_time=2017-09-01&end_time=now&granularity=day

Retrieves voltage data averaged over 24 hour cycles, corresponding to the organization's local time zone. Dates are returned in the response.

Baselines

The **/data/v1/baselines** method allows retrieval of the average power and energy usage for a data stream, over weekdays, individual days of the week, weekends, and holidays. This information forms the "baseline" against which we measure savings when you turn sockets off and use schedules to manage after-hours, weekend, and holiday energy usage.

Baselines are available in hour and day granularity.

GET
/data/v1/<org_id>/baselines/intelisockets/<data_stream_id>?<query_parameters>

parameters	contents	required?
<org_id>/	organization id number, for filtering response scope	no
<data_stream_id>/	data stream id number	yes
time_format=	utc | local | timestamp	no
granularity=	day | hour | minute	yes

Example:

/data/v1/8004/baselines/intelisockets/30452?granularity=day

Retrieves the current day-level baseline used to calculate savings for a single data stream

Control API

The Control API responds to HTTP PATCH requests to change the state of hardware options on InteliSockets. Current models allow the control of the power state of the socket plug (or plugs, for the IS-302 dual socket). Future models may include sensor control points integrated with the InteliSocket hardware, and thus the control API may be extended.

On-Off Control

Turning a socket plug on or off is done by sending a PATCH request with the following URI format, using the hexadecimal socket ID rather than an integer data stream ID. The body of the request is blank, with the parameters all occurring in the URL.

PATCH /control/v1/<org_id>/intelisockets/<hw_id>?<query_parameters>

parameters	contents	required?
<org_id>/	organization id number, for filtering response scope	no
<hw_id>/	hardware id number in hexadecimal	yes
?new_state	on | off	yes

Example:

/control/v1/8004/intelisockets/a9e5ce?new_state=off

Turns off the power to a specific socket in the Ibis Networks test lab organization

Revision History

2017-09-20: Initial release of full public beta
2017-12-13: Updates to time formats and additional time series example
2018-09-21: Update to remove unneeded header on Control API