

# Package ‘OnboardClient’

July 21, 2025

**Type** Package

**Title** Bindings for Onboard Data's Building Data API

**Version** 1.0.0

**Description** Provides a wrapper for the Onboard Data building data API <<https://api.onboarddata.io/swagger>>. Along with streamlining access to the API, this package simplifies access to sensor time series data, metadata (sensors, equipment, and buildings), and details about the Onboard data model/ontology.

**License** Apache License (>= 2)

**Imports** data.table (>= 1.14.2), dplyr (>= 1.0.10), httr (>= 1.4.4), jsonlite (>= 1.8.0), lubridate (>= 1.8.0), plyr (>= 1.8.7), rraply (>= 1.2.5), rstudioapi (>= 0.14), tibble (>= 3.1.8), tidyr (>= 1.2.1), tidyselect (>= 1.1.2), stringr (>= 1.4.1)

**Encoding** UTF-8

**RoxygenNote** 7.2.1

**NeedsCompilation** no

**Author** Pranay Shah [aut],  
Christopher Dudas-Thomas [cre, aut],  
Onboard Data [cph, fnd]

**Maintainer** Christopher Dudas-Thomas <[christopher@onboarddata.io](mailto:christopher@onboarddata.io)>

**Repository** CRAN

**Date/Publication** 2022-10-29 09:14:31 UTC

## Contents

api.access . . . . .	2
api.get . . . . .	2
api.post . . . . .	3
api.setup . . . . .	3
api.status . . . . .	4
get_buildings . . . . .	4
get_building_info . . . . .	5
get_deployments . . . . .	5

get_equipment_by_ids . . . . .	6
get_equip_types . . . . .	6
get_metadata . . . . .	7
get_orgs . . . . .	8
get_points_by_ids . . . . .	8
get_point_types . . . . .	9
get_staged_data . . . . .	9
get_timeseries . . . . .	10
get_timeseries_raw . . . . .	10
get_users . . . . .	11
PointSelector . . . . .	11
promote_staged_data . . . . .	12
select_points . . . . .	12
upload_staging . . . . .	13

## Index 14

---

api.access	<i>Access API keys and URL from System Environment</i>
------------	--

---

### Description

Returns the API url and API key.

### Usage

```
api.access()
```

### Value

A named list of API information, containing elements 'url' and 'key'.

---

api.get	<i>API GET call</i>
---------	---------------------

---

### Description

Uses http GET call to return an object from the API.

### Usage

```
api.get(endpoint)
```

### Arguments

endpoint	A character string containing a valid Onboard API endpoint.
----------	---

**Value**

A list or data.frame of the API output.

**Examples**

```
## Not run: whoami <- api.get('whoami')
```

---

api.post	<i>API POST call</i>
----------	----------------------

---

**Description**

Uses http POST call to post objects to the API.

**Usage**

```
api.post(endpoint, json_body, output = "list")
```

**Arguments**

endpoint	A character string containing a valid Onboard API endpoint.
json_body	A JSON payload to give to the POST call.
output	A character string, either "list" (default) or "dataframe", to specify the API output format.

**Value**

A list or data.frame of the API output.

---

api.setup	<i>Set up Onboard API keys and URL in system environment</i>
-----------	--

---

**Description**

Set the Onboard API URL and API keys in the system environment.

**Usage**

```
api.setup(api_type = "prod")
```

**Arguments**

api_type	Provide the API client name.
----------	------------------------------

**Value**

No return value, sets API url and API key in the system environment.

---

api.status	<i>Check the status of your connection with the Onboard API</i>
------------	---

---

**Description**

Provides a status code and message for the API connection.

**Usage**

```
api.status()
```

**Value**

A character string of the API server status and message.

---

get_buildings	<i>Buildings</i>
---------------	------------------

---

**Description**

Retrieve buildings that you have access to.

**Usage**

```
get_buildings(id)
```

**Arguments**

id	(Optional) An integer if you want information on a particular entity. Returns all entities unless this argument is provided.
----	--

**Value**

A data.frame of all building information.

---

get_building_info	<i>Building Info</i>
-------------------	----------------------

---

**Description**

Retrieves building id(s) and name(s). Assigns each to list variables in the parent environment called "id" and "name", and prints each list.

**Usage**

```
get_building_info(buildings, verbose = TRUE)
```

**Arguments**

buildings	Integer, character, or vectors of those types, providing building id(s) or name(s). You can provide multiple buildings at once.
verbose	Logical. If TRUE (default), print status messages.

**Value**

A data.frame of building info with two columns, 'id' and 'name'.

---

get_deployments	<i>Deployments</i>
-----------------	--------------------

---

**Description**

Get all deployments in your organization.

**Usage**

```
get_deployments(org_id)
```

**Arguments**

org_id	organization id
--------	-----------------

**Value**

A data.frame of all deployments.

get\_equipment\_by\_ids *Equipment by ID*

---

**Description**

Queries equipment by their ids.

**Usage**

```
get_equipment_by_ids(id)
```

**Arguments**

id                    Integer or integer vector, containing one or many equipment ids.

**Value**

A data.frame of the requested equipment, or an empty list if no equipment matches those ids.

**Examples**

```
## Not run:
equipment <- get_equipment_by_ids(c(1000,1001))

# If you are using the point selector function:
query <- PointSelector()

query$buildings <- 101
query$equipment_types <- 'ahu'

selection <- select_points(query)

equipment <- get_equipment_by_ids(selection$equipment)

## End(Not run)
```

---

get equip\_types            *Equipment Types*

---

**Description**

Query all equipment types from Onboard's Data Model.

**Usage**

```
get equip_types()
```

**Value**

A data.frame containing all equipment types.

---

get_metadata	<i>Metadata</i>
--------------	-----------------

---

**Description**

Retrieves points and equipment for a given building or selection and outputs a clean metadata data.frame.

**Usage**

```
get_metadata(buildings, selection, verbose = TRUE)
```

**Arguments**

buildings	Integer, character, or vectors of those types, providing building id(s) or name(s). You can provide multiple buildings at once.
selection	Selection list from point selector.
verbose	Logical. If TRUE (default), print status messages.

**Value**

A data.frame of clean metadata for the requested points.

**Examples**

```
## Not run:
metadata <- get_metadata(buildings=c(427,"Laboratory"))

OR

query <- PointSelector()

query$buildings <- 427
query$equipment_types <- 'ahu'
query$point_types <- c('Supply Air Temperature','Supply Air Static Pressure')

selection <- select_points(query)

metadata <- get_metadata(selection)

## End(Not run)
```

---

`get_orgs`*Organizations*

---

**Description**

Retrieve Organizations that you have access to.

**Usage**

```
get_orgs(id)
```

**Arguments**

`id` (Optional) An integer if you want information on a particular entity. Returns all entities unless this argument is provided.

**Value**

A data.frame of organization information.

---

`get_points_by_ids`*Points by ID*

---

**Description**

Queries data points by their ids.

**Usage**

```
get_points_by_ids(id)
```

**Arguments**

`id` Integer or list of integers. One or many point ids.

**Value**

A data.frame of the requested points, or an empty list if there are no points with those ids.

**Examples**

```
## Not run:
points <- get_points_by_ids(c(10000,10001))

# If you are using the point selector function:
query <- PointSelector()

query$buildings <- 101
query$equipment_types <- 'ahu'
query$point_types <- c('Supply Air Temperature', 'Supply Air Static Pressure')

selection <- select_points(query)

points <- get_points_by_ids(selection$points)

## End(Not run)
```

---

get_point_types	<i>Point Types</i>
-----------------	--------------------

---

**Description**

Queries all point types, measurements and their units from Onboard's Data Model and returns a clean output.

**Usage**

```
get_point_types()
```

**Value**

A data.frame containing all point types.

---

get_staged_data	<i>Get Staged Data</i>
-----------------	------------------------

---

**Description**

Gets metadata from the staging area.

**Usage**

```
get_staged_data(building, verbose = TRUE)
```

**Arguments**

building	Character vector or integer corresponding to the building name or id. If you enter multiple building ids or names, only the first entry is considered.
verbose	Logical. If TRUE (default), prints status and progress messages.

**Value**

A data.frame of metadata from the staging area.

---

<code>get_timeseries</code>	<i>Time-Series Data</i>
-----------------------------	-------------------------

---

**Description**

Provides clean time-series

**Usage**

```
get_timeseries(start_time, end_time, point_ids)
```

**Arguments**

start_time	Start Time in UTC.
end_time	End Time in UTC.
point_ids	Point IDs for which timeseries data needs to be queried.

**Value**

A wide data.frame of time-series data, with timestamp and all requested point IDs as columns.

---

<code>get_timeseries_raw</code>	<i>Raw Time-Series Data</i>
---------------------------------	-----------------------------

---

**Description**

Retrieves timeseries data in raw format.

**Usage**

```
get_timeseries_raw(start_time, end_time, point_ids)
```

**Arguments**

start_time	Start Time in UTC.
end_time	End Time in UTC.
point_ids	Point IDs for which timeseries data needs to be queried.

**Value**

A long data.frame of time series data, with point id, timestamp, and raw point values as columns.

---

get_users	<i>Users</i>
-----------	--------------

---

**Description**

Retrieve all user info in your organization.

**Usage**

```
get_users(id)
```

**Arguments**

`id` (Optional) An integer if you want information on a particular entity. Returns all entities unless this argument is provided.

**Value**

A data.frame of all user information.

---

PointSelector	<i>PointSelector</i>
---------------	----------------------

---

**Description**

A list of parameters to query metadata.

**Usage**

```
PointSelector()
```

**Value**

An empty named list of possible point selection criteria.

**Examples**

```
## Not run:
query <- PointSelector()

query$buildings <- 101
query$equipment_types <- 'ahu'
query$point_types <- c('Supply Air Temperature', 'Supply Air Static Pressure')

## End(Not run)
```

---

promote\_staged\_data     *Promote data on Staging Area*

---

### Description

Promote valid data on the staging area to the live building.

### Usage

```
promote_staged_data(building, data_to_promote, verbose = TRUE)
```

### Arguments

building	Character vector or integer corresponding to the building name or id. If you enter multiple building ids or names, only the first entry is considered.
data_to_promote	(Optional) If missing, all valid topics are promoted. A data.frame containing columns 'e.equip_id' & 'p.topic'.
verbose	Logical. If TRUE (default), prints status and progress messages.

### Value

A named list containing any errors that may have occurred during data promotion.

---

select\_points     *Select Points*

---

### Description

Returns a list of ids based on the input query from PointSelector. Uses http POST call to query data.

### Usage

```
select_points(query)
```

### Arguments

query	query supplied from PointSelector.
-------	------------------------------------

### Value

A named list of all the points requested by the query.

**Examples**

```
## Not run:
query <- PointSelector()

query$buildings <- 427
query$equipment_types <- 'ahu'
query$point_types <- c('Supply Air Temperature', 'Supply Air Static Pressure')

selection <- select_points(query)

## End(Not run)
```

---

upload_staging	<i>Upload to Staging Area</i>
----------------	-------------------------------

---

**Description**

Uploads data to the staging area.

**Usage**

```
upload_staging(building, data_to_upload, skip_topics = FALSE, verbose = TRUE)
```

**Arguments**

building	Character vector or integer corresponding to the building name or id. If you enter multiple building ids or names, only the first entry is considered.
data_to_upload	A data.frame to upload to the staging area. Must contain e.equip_id and p.topic columns.
skip_topics	Logical. If True, the uploaded topics will be assigned ‘__SKIP__’ equip_id.
verbose	Logical. If TRUE (default), prints status and progress messages.

**Value**

A named list containing any errors that may have occurred during data upload.

# Index

api.access, 2  
api.get, 2  
api.post, 3  
api.setup, 3  
api.status, 4

get\_building\_info, 5  
get\_buildings, 4  
get\_deployments, 5  
get\_equip\_types, 6  
get\_equipment\_by\_ids, 6  
get\_metadata, 7  
get\_orgs, 8  
get\_point\_types, 9  
get\_points\_by\_ids, 8  
get\_staged\_data, 9  
get\_timeseries, 10  
get\_timeseries\_raw, 10  
get\_users, 11

PointSelector, 11  
promote\_staged\_data, 12

select\_points, 12

upload\_staging, 13