Wattzon Link API 4.0 Documentation

- WattzOn Link API Documentation
  - Definitions
  - Service Introduction
  - Profiles
  - Authentication
  - Use Implications of Authentication
  - Getting Started
    - Creating a Profile
    - Initiating a Data Extraction Job
    - Retrieving Data
    - Testing API Connectivity
- API Resource Specifications
  - General Requirements
  - Base URL
  - About the Examples
  - Basic Services
    - Echo
    - Certificate info
  - ZIP Code Information Services
    - Query ZIP Information
  - Utility and Agent Information Services
    - Find utility provider
    - Query Utility Provider Information
  - Profile Service
    - Query/List Profiles
    - Retrieve Profile
    - Create profile
    - Change Profile
    - Delete Profile
    - Sensitive Fields
    - Get Sensitive Transmission Key
    - List Sensitive Fields
    - Add Sensitive Field
    - Change Sensitive Field
    - Delete Sensitive Field
    - List Tags
    - Add Tag
    - Delete Tag
  - Extraction/Job Service
    - Start Job
    - Job Status
    - Last Job State for Profile
    - Job Status Codes
  - Data Retrieval Service
    - Retrieve Bill-Based Cost and/or Usage Data
    - Retrieve PDF from Bill
    - List Intervals

        - Retrieve Interval
- Notification Support
    - Purpose
    - Configuration
    - Enabling/Disabling Notifications
    - Message Format

WattzOn Link API Documentation

WattzOn Link is a software-as-a-service (SaaS) platform for retrieving billing and usage data from utility providers. This document describes the individual Link API calls and their features.

Definitions

- *Agent*

  The WattzOn software tailored to one particular utility provider that performs data extraction.
- *Client*

  The software interacting with the WattzOn Link API.
- *LSE ID*

  Load Serving Entity (electricity provider) identification number.
- *Job*

  One particular data-extraction run.
- *Profile*

  A data record corresponding to one customer of a utility provider.
- *Sensitive Field*

  The mechanism used to store account credentials.
- *Provider*

  A utility provider.

Service Introduction

Typical API use is a multistep process. For any desired data extraction, the client performs the following steps:

1. Look up the target utility provider.
2. Create a profile for the utility provider's customer.
3. Initiate data extraction job and wait for completion (or for there to be enough data available).
4. Fetch data.

Each one of these steps may involve multiple API calls. Therefore, the Link API is broken into the following service categories that correspond to the above steps:

- ZIP Information Service

Look up ZIP code information based on the ID, see location information, such as latitude, longitude, city, county, etc.

- Utility and Agent Information Service

  Look up utility providers based on ZIP code, see utility provider information, and check agent availability.
- Profile Service

  Create, modify, and delete profiles for customers of utility providers.
- Extraction Service

  Initiates data extraction jobs and checks status of these jobs.
- Data Retrieval Service

  Looks up usage and billing data for a profile or particular job.

Profiles

A profile is a record for one account login on a utility provider. It includes indexing and identifying information as well as account credentials.
Basic profile information includes the following:

- Profile ID: A unique integer (much like a primary key in a database). The API assigns a new Profile ID upon profile creation.
- Utility Provider ID: The utility provider for this profile (acquired with Utility and Agent Information service API calls).
- Label: An arbitrary text field. This is available for profile lookup, so clients can use this field to reference profiles based on their own internal tracking system (such as a Salesforce case ID). The label can be empty.
- Tags: An arbitrary list of text strings for grouping and searching. Examples could include "solar" and "budget billing".
- ZIP Code: ZIP code of the service location. Can be empty, and does not necessarily need to be correct (for example, one utility site login can have more than one service line in multiple ZIP codes; this will not be important).

A profile's account credentials are stored in *sensitive fields*, which are key-value pairs. These fields are write-only in the API: Clients can set and delete keys and values, and read keys, but are unable to read values. The agents for most utility providers require these two sensitive fields:

- username: The username on the utility site.
- password: The password on the utility site.

There are separate API calls for working with sensitive fields in a profile.

Authentication

All client identification and authorization is through a client-side TLS certificate. You do not need to purchase a certificate; WattzOn will act as a certificate authority for its own application. The procedure is as follows:

1. Generate a CSR (certificate signing request). Here's how to do it with openssl:

```
openssl req -new -newkey rsa:2048 -nodes -keyout client.key -out client.csr
```

Don't enter a challenge password. This creates two files: client.csr (the CSR) and client.key (the private key).

2. Send the CSR to your WattzOn technical or sales contact. Do NOT send the private key; keep it secret.
3. WattzOn will sign the CSR and send you a certificate named client.crt. You can now use this in conjunction with your private key (client.key) to connect to the Link API. However, depending on your client software, you may need to perform another step to create a combined certificate-key file.

- The WattzOn Python reference client needs no additional steps; it can use client.crt and client.key directly.
- Some HTTPS clients may require a .pfx/.pf12 file, a DER file, or another format. OpenSSL can convert to these formats.

Use Implications of Authentication

The core Link API is designed to be accessed by systems under the direct control of the customer; it is not meant to be consumed directly by web browsers. Client-side certificates are the practical means to enforce this; they are essential to the security of the system.
As such, the core Link API does not authenticate at the individual profile ("end user") level; this is the responsibility of the customer (and indeed, each customer has their own form of authentication and identification for their own end users). However, WattzOn can assist in developing user interface products that build upon the core API to integrate with a customer's end-user authentication.

Getting Started

Typical use of WattzOn Link involves of the following steps:

1. Create a profile for a utility provider customer.
2. Initiate a data extraction job.
3. Wait for the extraction job to complete, or until the agent has extracted enough utility data.
4. Retrieve the utility data.

For steps 1-3, many WattzOn Link customers opt to use the WattzOn 3in1 product to create profiles and start extraction jobs. This greatly simplifies implementation, because you need only implement step 4; retrieving data after an extraction is a single Link API call.
Let's go through the process of creating a profile with the WattzOn Mock Utility Provider and Mock Agent, a tool for testing your implementation without accessing a true utility provider.
If you need to test your API connection, see Testing API Connectivity.

Creating a Profile

For WattzOn 3in1 customers, choose the "Link Your Data" button in the 3in1 interface to bring up the linking user interface, then:

1. For ZIP code, enter 00007. You'll get a list of utility providers.
2. Choose "Mock Utility Provider (electric+gas)" from the list.
3. Enter anything you like for username and password.
4. Review your information, then link the account.
5. You're done! Skim the few sections for a quick look behind the scenes, and pick up again at Retrieving Data.

Note: The Mock Agent includes special diagnostic features that you can enable by entering specific values for the username; we can provide Mock Agent documentation for details.

To create a profile with the API, you need to enter a ZIP code and a utility provider. You can look up utility providers by ZIP code or name using the Find Utility Provider endpoint. In this case, we'll use the ZIP code 00007 and the utility provider ID 3149, which is the WattzOn Mock Utility Provider.
Create the profile with the following request:
Method: POST; Endpoint: */link/4.0/profiles/*
Data:

```
{
    "provider_id": 3150,
    "zipcode": "00007",
    "label": "your label"
}
```

Result:

```
{
  "id": 516,
  "label": "this is optional",
  "utility_provider": "Mock Utility (electric only)",
  "utility_provider_id": 3150,
  "zipcode": "00007"
}
```

A successful profile creation like this returns a profile_id field; you'll need to know that later. In this example, the ID of the new profile is 516.

Setting Credentials with Sensitive Fields

Most utility agents need credentials (specifically, username and password) for a user's account in order to function. These are known as sensitive fields in the Link API. Using these is one of the more complicated pieces of the API, because you must first get an encryption key through the API (even though the service runs over TLS), and then encrypt the data before sending.
Note: Remember that if you're using WattzOn 3in1, you don't need to use the API for this! You can skip ahead to the data extraction.
For example, to set sensitive field, get a public encryption key with this API call:
Method: GET; Endpoint: */link/4.0/profiles/keys/*
Result:

```
{
  "expires": 1476901022,
  "public_key": "-----BEGIN PUBLIC KEY-----\nMIIBI [bytes omitted]
QAB\n-----END PUBLIC KEY-----"
}
```

Now, using this public encryption key, use PKCS1/OAEP to encrypt the actual value that you're transmitting, and base64-encode the encrypted value. Here's the API call to set the username sensitive field for the profile_id from the previous example (516):
Method: POST; Endpoint: */link/4.0/profiles/516/sensitive/*
Data:

```
{
    "field": "username",
    "value": "BIKJ3IkC1B [bytes omitted] SYxi8boLMX7QQ5K47GI+AQirwg=="
}
```

Result:

```
{
  "message": "Success"
}
```

This endpoint is fairly restrictive; it will not allow encrypted values with expired keys (a 403 error), and will not overwrite an existing key-value pair (a 409 error). If you need to change a field, remove the old one first.
Most profiles require two sensitive fields in order for their agents to function correctly: username and password. Sensitive fields are write-only; you cannot retrieve their values through the API.

Initiating a Data Extraction Job

For WattzOn 3in1 customers, the interface starts a data extraction job immediately after the user clicks the Link button after entering their utility provider information. When the job completes, 3in1 sends a notification (see the Retrieving Data section).
In our ongoing example using the profile ID 516, use the following API call to start a data extraction job with the API.
Method: POST; Endpoint: */link/4.0/jobs/*
Data:

```
{
    "profile_id": 516,
    "mode": "bill"
}
```

Result:

```
{
  "job_id": "3b7471eb-7e35-4a45-b081-e9cfa9d71e75",
  "profile_id": 1
}
```

Note the job ID return value. You can check up on the job with the following API call:
Method: GET; Endpoint: */link/4.0/jobs/3b7471eb-7e35-4a45-b081-e9cfa9d71e75/*
Result:

```
{
  "finished": true,
  "job_id": "3b7471eb-7e35-4a45-b081-e9cfa9d71e75",
  "job_status": "AgentState.success"
}
```

You can make this call periodically to see how an agent is progressing. When the agent has completed, the endpoint's finished field will be true. (Note that many agents can make data available before finishing.)

Retrieving Data

You need a profile ID in order to retrieve data. WattzOn 3in1 customers typically get a notification of profile creation through SQS or another system; this notification includes the profile ID.
Here's the API call to fetch the utility data for profile ID 516:
Endpoint: */link/4.0/data/bills/516/*
Result:

```
[
  {
    "account_number": "42626522-7",
    "bill_id": "580167d10640fd48ff1960bf",
    "cost": 21.25,
    "currency": "USD",
    "end_date": "2016-10-31T23:59:59",
```

```
    "meter_number": "67563764",
    "name": "J. Random Person",
    "pdf": "/link/4.0/data/bills/1/580167d10640fd48ff1960bf/pdf",
    "service_address": "123 J. Random Road, Randomtown, RI 00007",
    "service_id": "353267363254",
    "start_date": "2016-10-01T00:00:00",
    "type": "gas",
    "units": "therms",
    "usage": 13.72
  }
]
```


By default, the data retrieval call returns all utility data for a profile, regardless of the job that extracted the data.

Testing API Connectivity

The WattzOn Link API includes an echo service for testing API connectivity. The service simply repeats anything sent to it. You can either send the request using POST parameters or GET query parameters.

Endpoint: */link/4.0/echo?string=value*

Data:


```
{
    "string": "value"
}
```


Result:


```
{
    "string": "value"
}
```


API Resource Specifications

General Requirements

The Link API uses JSON as a serialization format. As such, it requires no specific software to use. Authentication and communication proceed over TLS; clients must present a certificate. One step in creating a profile requires transmission of sensitive information to the API. Clients must encrypt this information using a public-key algorithm before transmission.

Base URL

For most customers, the base URL for the Link is:

`https://api.wattzon.com/`

About the Examples

This document contains a number of curl command line examples to assist in understanding the API parameters. However, these are useful only to a certain degree, as some parts (in particular, the sensitive parameter endpoints) are too lengthy to show with a short command line.

An alternative for experimenting with the API is a scripting language with a command line interface. For instance, WattzOn uses the interactive Python interpreter with the requests library to develop the reference client.

Basic Services

Base path: */link/4.0/*

Echo

path: */link/4.0/echo/*
methods: **POST, GET**

Query Parameters

- string: any string

Body Parameters

- string: any string

Output

- string: the string that was on the original request

Example

```
$ curl -XGET https://api.wattzon.com/link/4.0/echo/?string=Hello --
cert certificate_file.crt --key key_file.key
$ curl -XPOST https://api.wattzon.com/link/4.0/echo/ -d '{"string":
"30"}' --cert certificate_file.crt --key key_file.key
```

The echo API endpoint is a basic sanity check to see if your client-side certificate works.

Certificate info

path: */link/4.0/certificates/info*
method: **GET**

Params

- none

Output

- client_name: the client to which this certificate is associated with
- enabled: check on the certificate validity
- expiration_date: expiration date for the current certificate
- created_date: creation date for the current certificate

Example

```
$ curl -XGET https://api.wattzon.com/link/4.0/certificates/info --cert
certificate_file.crt --key key_file.key
```

This API endpoint check whether your certificate is valid or not against the current API.

ZIP Code Information Services

ZIP service base path: */link/4.0/zips*

Query ZIP Information

path: */link/4.0/zips/{zipcode}*
method: **GET**

URI Parameters

- zipcode: ZIP ID

Output

- zipcode: ZIP code
- city: city where zipcode is located
- county: county where zipcode is located
- state: state where zipcode is located
- latitude: latitude of location
- longitude: longitude of location
- error: error message (if failure)

Errors

- 400 (Bad Request): Error loading the zipcode
- 404 (Not Found): ZIP (code) not found
- 403 (Forbidden): Permission denied

Example

```
$ curl -XGET https://api.wattzon.com/link/4.0/zips/00007 --cert
certificate_file.crt --key key_file.key
```

Utility and Agent Information Services

Utility service base path: */link/4.0/utilities*

Find utility provider

path: */link/4.0/utilities*
method: **GET**

Query Parameters

- zipcode: ZIP code (optional if name parameter present)
- name: name to match (optional if zipcode parameter present)
- type: utility type (e.g. electricity, gas, water) (optional)

Output

- provider_ids: list of utility provider IDs (WattzOn IDs; not LSE IDs)
- error: error message (if failure)

Errors

- 400 (Bad Request): Name or Zipcode not found in URI
- 404 (Not Found): Utility Provider not found
- 403 (Forbidden): Permission denied

Example

```
$ curl -XGET https://api.wattzon.com/link/4.0/utilities/?zipcode=00007
--cert certificate_file.crt --key key_file.key
```

Query Utility Provider Information

path: */link/4.0/utilities/{provider_id}*
method: **GET**

URI Parameters

- provider_id: utility provider ID (e.g. from a find call)

Output

- name: utility name
- type: utility types supported by the utility: comma-separated list of types (e.g. electricity, gas, water)
- lseid: LSE ID (if available)
- homepage_url: link to the provider's website
- error: error message (if failure)

Errors

- 400 (Bad Request): Error loading the Provider ID
- 404 (Not Found): Utility Provider not found
- 403 (Forbidden): Permission denied

Example

```
$ curl -XGET https://api.wattzon.com/link/4.0/utilities/3150 --cert
certificate_file.crt --key key_file.key
```

Profile Service

service base path: */link/4.0/profiles*

Query/List Profiles

path: */link/4.0/profiles*
method: **GET**

Query Parameters

- id: (optional) single profile ID or list of profile IDs to list
- provider (optional): utility provider ID to query
- start_date (optional): minimum creation date to query
- end_date (optional): latest creation date to query
- tags (optional): comma-separated list of tags to match against profiles
- label (optional): a string identifier for the profile to query

Output

- profiles: profile list; will consist of IDs
- error: error message (if failure)

Errors

- 403 (Forbidden): Permission denied

Should not return errors (other than a 403 if you're not allowed to use the service at all), but may return an empty list.

Example

```
$ curl -XGET
https://api.wattzon.com/link/4.0/profiles/?label=sample%20profile --
cert certificate_file.crt --key key_file.key
```

Retrieve Profile

path: */link/4.0/profiles/{profile_id}*
method: **GET**

URI Parameters

- profile_id: profile ID to change

Output

- utility_provider: utility provider (from Utility Provider Service, if the profile is associated with one)
- utility_provider_id: utility provider ID (from Utility Provider Service, if the profile is

associated with one)
- label: label associated to the profile (if it was added)
- tags: tags associated to the profile (if they were added)
- created: creation date for the profile
- modified: last modification date for the profile
- error: error message (if failure)

Errors

- 400 (Bad Request): Error loading the JSON
- 404 (Not Found): Profile not found
- 403 (Forbidden): Permission denied

Example

```
$ curl -XGET https://api.wattzon.com/link/4.0/profiles/1 --cert
certificate_file.crt --key key_file.key
```

Create profile

path: */link/4.0/profiles*
method: **POST**

Body Parameters

- provider_id: utility provider (from Utility Provider Service)
- zipcode: account ZIP code (may be required for some providers)
- label: text field to identify the profile (optional)

Output

- profile_id: newly-created profile ID (if success)
- zipcode: zipcode associated to the new profile (if success)
- label: label associated to the new profile (if it was added)
- error: error message (if failure)

Errors

- 400 (Bad Request): Error loading the JSON
- 404 (Not Found): ZIP not found
- 404 (Not Found): Utility provider not found
- 403 (Forbidden): Permission denied

Example

```
$ curl -XPOST https://api.wattzon.com/link/4.0/profiles -d
'{"provider_id": 3150, "zipcode": "00007", "label": "new testing
profile"}' --cert certificate_file.crt --key key_file.key
```

Change Profile

path: */link/4.0/profiles/{profile_id}*
method: **PUT**

URI Parameters

- profile_id (in URL): profile to change

Body Parameters

- provider_id: new provider (optional)
- zipcode: account ZIP code (optional)
- label: text field to identify the profile (optional)

Output

- profile fields (reflecting changed values)
- error: error message (if failure)

Errors

- 400 (Bad Request): Error loading the JSON
- 400 (Bad Request): Error loading the Profile ID
- 404 (Not Found): Profile not found
- 404 (Not Found): Utility provider not found
- 404 (Not Found): ZIP not found
- 403 (Forbidden): Permission denied

Example

```
$ curl -XPUT https://api.wattzon.com/link/4.0/profiles/268 -d
'{"provider_id": 3149, "zipcode": "00007", "label": "new testing
profile with changes"}' --cert certificate_file.crt --key key_file.key
```

Delete Profile

path: */link/4.0/profiles/{profile_id}*
method: **DELETE**

URI Parameters

- profile_id: profile ID to delete

Output

- 200 (Deleted)
- error: error message (if failure)

Errors

- 400 (Bad Request): Error loading the Profile ID

- 404 (Not Found): Profile not found
- 403 (Forbidden): Permission denied

Example

```
$ curl -XDELETE https://api.wattzon.com/link/4.0/profiles/2 --cert
certificate_file.crt --key key_file.key
```

Sensitive Fields

Most utility data extraction agents require a profile to include two sensitive fields, username and password, for the utility account credentials.
When a client sets a sensitive field, it must encrypt the value before sending to the WattzOn API. The steps to do so are as follows:

1. Acquire a transmission key. This is a public key with an expiration time.
2. Encrypt the value using the PKCS1/OAEP algorithms, and base64-encode the encrypted value.
3. Pass the encoded and encrypted value as the value parameter in the add or modify field endpoint.

Get Sensitive Transmission Key

path: */link/4.0/profiles/keys*
method: **GET**

Params

- none

Output

- public_key: public key for transmitting sensitive data
- expires: key expiration (Unix timestamp)
- error: error message (if failure)

Errors

- 403 (Forbidden): Permission denied

Example

```
$ curl -XGET https://api.wattzon.com/link/4.0/profiles/keys --cert
certificate_file.crt --key key_file.key
```

List Sensitive Fields

path: */link/4.0/profiles/{profile_id}/sensitive*
method: **GET**

URI Parameters

- profile_id: profile ID to change

Output

- fields: list of fields (keys only, no values)
- error: error message (if failure)

Errors

- 400 (Bad Request): Error loading the Profile ID
- 404 (Not Found): Profile not found
- 403 (Forbidden): Permission denied

Example

```
$ curl -XGET https://api.wattzon.com/link/4.0/profiles/268/sensitive --cert certificate_file.crt --key key_file.key
```

Add Sensitive Field

path: */link/4.0/profiles/{profile_id}/sensitive*
method: **POST**

URI Parameters

- profile_id: profile ID to change

Body Parameters

- field: field ID (e.g. username, password)
- value: encrypted field value (using a valid key obtained with the transmission key endpoint); must be PKCS1/OAEP-encoded, then base64-encoded

Output

- 201 (Success/Created)
- error: error message (if failure)

NOTE: A POST for a profile ID/field ID combination that already exists is an error (see PUT below).

Errors

- 400 (Bad Request): Error loading the JSON
- 400 (Bad Request): Error loading the Profile ID
- 404 (Not Found): Profile not found
- 403 (Forbidden): Key expired
- 403 (Forbidden): Decryption failed
- 403 (Forbidden): Permission denied
- 409 (Conflict): Field already exists

Example

```
$ curl -XPOST https://api.wattzon.com/link/4.0/profiles/268/sensitive
-d '{"field": "username", "value": "new encrypted user name"}' --cert
certificate_file.crt --key key_file.key
```
NOTE: The new encrypted user name in this example could be quite long.

Change Sensitive Field

path: */link/4.0/profiles/{profile_id}/sensitive*
method: **PUT**

URI Parameters

- profile_id: profile ID to change

Body Parameters

- field: field ID (e.g. username, password)
- value: encrypted field value (using a valid key obtained with the transmission key endpoint); must be PKCS1/OAEP-encoded, then base64-encoded

Output:

- message: success
- error: error message (if failure)

Errors

- 400 (Bad Request): Error loading the JSON
- 400 (Bad Request): Error loading the Profile ID
- 404 (Not Found): Profile not found
- 404 (Not Found): Sensitive field not found
- 403 (Forbidden): Key expired
- 403 (Forbidden): Decryption failed
- 403 (Forbidden): Permission denied

Example

```
$ curl -XPUT https://api.wattzon.com/link/4.0/profiles/268/sensitive -
d '{"field": "username", "value": "new encrypted user name"}' --cert
certificate_file.crt --key key_file.key
```
NOTE: The new encrypted user name in this example could be quite long.

Delete Sensitive Field

path: */link/4.0/profiles/{profile_id}/sensitive/{field}*
method: **DELETE**

URI Parameters

- profile_id: profile ID to change
- field: target field

Output

- message: deleted
- error: error message (if failure)

Errors

- 400 (Bad Request): Error loading the JSON
- 404 (Not Found): Profile not found
- 404 (Not Found): (Sensitive) Field not found
- 403 (Forbidden): Decryption failed
- 403 (Forbidden): Permission denied

Example

```
$ curl -XDELETE
https://api.wattzon.com/link/4.0/profiles/268/sensitive/username --
cert certificate_file.crt --key key_file.key
```

List Tags

path: */link/4.0/profiles/{profile_id}/tags*
method: **GET**

URI Parameters

- profile_id: profile ID to change

Output

- tags: list of tags
- error: error message (if failure)

Errors

- 400 (Bad Request): Error loading the Profile ID
- 404 (Not Found): Profile not found
- 403 (Forbidden): Permission denied

Example

```
$ curl -XGET https://api.wattzon.com/link/4.0/profiles/268/tags --cert
certificate_file.crt --key key_file.key
```

Add Tag

path: */link/4.0/profiles/{profile_id}/tags*

method: **POST**

URI Parameters

- profile_id: profile ID to change

Body Parameters

- name: name of tag to add

Output

- 200 (Success; if tag already exists and newly attached to profile)
- 201 (Created; if newly created)
- error: Error message (if failure)

Errors

- 400 (Bad Request): Error loading the Profile ID
- 400 (Bad Request): Error loading the JSON
- 404 (Not Found): Profile not found
- 403 (Forbidden): Permission denied

Example

```
$ curl -XPOST https://api.wattzon.com/link/4.0/profiles/268/tags -d
'{"name": "new tag"}' --cert certificate_file.crt --key key_file.key
```

Delete Tag

path: */link/4.0/profiles/{profile_id}/tags*
method: **DELETE**

URI Parameters

- profile_id: profile ID to change

Body Parameters

- name: name of tag to delete

Output

- message: deleted
- error: error message (if failure)

Errors

- 400 (Bad Request): Error loading the Profile ID
- 400 (Bad Request): Error loading the JSON

- 404 (Not Found): Profile not found
- 404 (Not Found): Tag not found
- 403 (Forbidden): Permission denied

Example

```
$ curl -XDELETE https://api.wattzon.com/link/4.0/profiles/268/tags -d
'{"name": "new tag"}' --cert certificate_file.crt --key key_file.key
```

Extraction/Job Service

service base path: */link/4.0/jobs*

Start Job

path: */link/4.0/jobs*
method: **POST**

Body Parameters

- profile_id: profile to start data extraction on
- mode: bill or interval (default: bill)
- send_notification: true or false (default: true)

NOTE: send_notification field must be a boolean value. For more information on notifications, see Notifications.

Output

- job_id: job ID
- profile_id: profile ID of running job
- error: error message (if failure)

Errors

- 400 (Bad Request): Error loading the JSON
- 404 (Not Found): Profile not found
- 404 (Not Found): No agent for provider
- 404 (Not Found): Tag not found
- 403 (Forbidden): Permission denied

Example

```
$ curl -XPOST https://api.wattzon.com/link/4.0/jobs -d '{"profile_id":
268}' --cert certificate_file.crt --key key_file.key
```

Job Status

path: */link/4.0/jobs/{job_id}*
method: **GET**

URI Parameters

- job_id: job ID (from start job call)

Output

- job_id: job ID (if successful)
- job_status: job status *See status list*
- finished: true, if the job has finished; false, if the job is still running or queued
- detail: status details (if available)
- error: error message (if failure)

Errors

- 400 (Bad Request): Error loading the Job ID
- 404 (Not Found): Job not found
- 403 (Forbidden): Permission denied

Example

```
$ curl -XGET https://api.wattzon.com/link/4.0/jobs/7ede4372-2d9e-4294-
a83f-e034040edd99 --cert certificate_file.crt --key key_file.key
```

Last Job State for Profile

path: */link/4.0/jobs/profiles/{profile_id}/*
method: **GET**

URI Parameters

- profile_id: profile ID to query

Output

- job_id: job ID (if successful)
- job_status: job status *See status list*
- finished: true, if the job has finished; false, if the job is still running or queued
- detail: status details (if available)
- error: error message (if failure)

Errors

- 400 (Bad Request): Error loading the Job ID
- 404 (Not Found): Job not found
- 403 (Forbidden): Permission denied

Example

```
$ curl -XGET https://api.wattzon.com/link/4.0/jobs/profiles/268 --cert
certificate_file.crt --key key_file.key
```

Job Status Codes

The following are status codes that a job status call can return. First, the terminal states (that is, the extraction job has completed), in approximate order of frequency:

- success - extraction successeful
- partial_success - extraction successeful, but some utility site errors occurred, preventing download of certain records
- credential_error - incorrect login credentials (such as utility site username and password)
- action_needed - account problem on utility site (need to accept terms and conditions, need to reset password, etc.)
- unexpected_presentation - utility agent encountered a
- incomplete_parameters - agent missing credentials (usually a username
- timeout_site - utility site timed out and/or password sensitive field)
- site_error - utility site error detected
- terminated - agent aborted from Link side
- agent_failure - agent failed in a non-presentation related manner
- timeout_agent - agent timed out

The following are in-progress status codes, in rough order of the sequence that they occur. Of particular note here is that after an agent has successfully passed the login state without a credential_error or action_needed status message, the utility site credentials are valid and the extraction will typically be successful.

- queued - job queued
- init - agent loading initial data
- start - agent starting
- login - agent logging in to utility site
- account_navigation - agent navigating utility site account
- bill_extract - agent extracting bill (usually downloading a PDF)
- bill_processing - agent processing a bill
- interval_extract - agent extracting interval data
- interval_processing - agent processing an interval file
- logout - agent logging out of utility site
- worker_wait - agent cleanup

NOTE: WattzOn may add new status codes as necessary, so this list should not be presumed to be complete. Always use the finished field of the job status endpoint to determine if the job is complete.

Data Retrieval Service

service base path: */link/4.0/data*

Retrieve Bill-Based Cost and/or Usage Data

path: */link/4.0/data/bills/{profile_id}*
method: **GET**

URI Parameters

- profile_id: target profile

Query Parameters

- job_id: specific job (optional; if unused, it will retrieve all bills for the given profile id)

Output

- data: list of data associated with bill periods (see below)
- error: error message (if failure)

See the Retrieving Data section of Getting Started for sample JSON output.
Many types of data are available and extracted from utility bills. Different fields are organized by type for convenience and clarity, like gas, electricity, water or billing. More types and fields may be added in the future.
Utility types like electricity, gas and water are self-explanatory, however the billing type is a special kind of bill.
The billing type arose from the need from billing and payment information. To accomplish this, we added a new billing type to Link. This record reflects bill amounts, due dates, and other payment related information. This is separate and in addition to the existing electricity, gas and water type records, as payment information does not naturally fit into those categories (i.e. you could make a payment against both your electric & gas bills in the case of a combined utility like Pacific Gas & Electric).
Record types and fields available vary between utility providers and user. Below is an indicative list of the fields that may, but are not guaranteed, to appear in results of the Retrieve Bills endpoint.
Each record, regardless of type, may contain the following fields:

- name: name of the account holder.
- service_address: street address where service is registered and performed.
- account_number: identifier associated with the utility account.
- bill_date: issue date of the bill.
- type: record type (e.g. gas, electricity, billing, water).
- pdf: if a PDF is available, a URI to retrieve the file; otherwise empty.
- bill_id: ID for the individual bill (specific to utility provider).
- cached: true or false; indicates if this record is a cached value from a previous extraction.
- cached_job_id: job ID that extracted the record.
- currency: currency of extracted data (currently USD).

Specific to the electricity or gas type record, it may contain the following:

- service_id: identifier for the utility account.
- meter_number: identifier for the meter device for the given utility.
- start_date: billing period start date (as ISO8601 string; in UTC)
- end_date: billing period end date (as ISO8601 string; in UTC)
- date: some older agents do not include the start and end date; in this case, the "date" field is an interpolated value. Older agents can be converted to give more precise date fields upon request.
- cost: cost of consumed energy.
- usage: amount of energy used.
- units: unit of measurement (e.g. KWh, therms).
- retail_energy_provider: name of retail energy provider (if available)
- rate_plan: payment or rate plan for energy charges
- exported_energy: energy generated and not consumed.

Specific to the billing type record, it may contain the following:

- previous_balance: amount owed in last billing period, if any.
- payment_received: amount of payment made, if any.
- past_due_balance: amount of any past-due balance, if any.
- due_date: due date of any outstanding balance.

- current_charges: total amount of any additional charges for this period, include fees and credits.
- total_amount_due: total amount due, including any fees or credits by the bill holder.
- payment_date: date payment was made, if any.

Errors

- 400 (Bad Request): Error loading the Profile ID
- 404 (Not Found): Profile not found
- 404 (Not Found): Job not found
- 404 (Not Found): Bills not found
- 403 (Forbidden): Permission denied

Example

```
$ curl -XGET https://api.wattzon.com/link/4.0/data/bills/268 --cert
certificate_file.crt --key key_file.key
```

Retrieve PDF from Bill

path: */link/4.0/data/bills/{profile_id}/{bill_id}/pdf*
method: **GET**

URI Parameters

- profile_id: profile ID to retrieve
- bill_id: bill ID to retrieve

Output

- PDF file (200 status code; raw data)
- error: error message (if failure)

Errors

- 400 (Bad Request): Error loading the Profile ID
- 404 (Not Found): Error loading the Bill ID
- 404 (Not Found): Profile not found
- 404 (Not Found): PDF not found
- 404 (Not Found): Bill not found
- 403 (Forbidden): Permission denied

Example

```
$ curl -XGET
https://api.wattzon.com/link/4.0/data/bills/268/584763470640fd4a6a1c51
88/pdf --cert certificate_file.crt --key key_file.key
```

NOTE: bill_id field can be found in the usage data from the /link/4.0/data/bills/{profile_id} endpoint. It also includes the complete URL, which can be copied and used, instead of building the URL each time.

List Intervals

path: */link/4.0/data/intervals/{profile_id}*
method: **GET**

URI Parameters

- profile_id: target profile

Output

- intervals: list of available interval periods (see below)
- error: error message (if failure)

A single record of the interval period list consists of the following fields:

- meter_id: meter, account, or individual usage point identifier
- month_id: the ID of the interval's month (format: YYYY-DD)
- first_interval: start of first interval (as Unix timestamp)
- last_interval: start of last interval (as Unix timestamp)
- complete: indicates when the interval period includes all possible future data (true/false); used to show when future interval extractions will yield more data
- extraction_time: extraction date/time (as Unix timestamp)
- source_file_size: size of source interval data
- job_id_extracted: job ID that extracted this period

NOTE: Interval period boundaries may not occur precisely on UTC month boundaries; it may occur adjusted to a provider's timezone instead.

Errors

- 400 (Bad Request): Profile ID Required
- 404 (Not Found): Profile not found
- 404 (Not Found): No intervals found
- 403 (Forbidden): Permission denied

Example

```
$ curl -XGET https://api.wattzon.com/link/4.0/data/intervals/268/ --
cert certificate_file.crt --key key_file.key
```

Retrieve Interval

path: */link/4.0/data/intervals/{profile_id}/{meter_id}/monthly/{ym}*
method: **GET**

URI Parameters

- profile_id: target profile
- meter_id: target meter
- ym: month to load (format: YYYY-MM)

Query Parameters

- format: source, csv, json (default: source)

Output

- interval data file (200 status code; raw data or normalized data (csv and json formats))
- error: error message (if failure)

Errors

- 400 (Bad Request): Profile ID Required
- 400 (Bad Request): Meter ID Required
- 400 (Bad Request): Year-month Required
- 400 (Bad Request): Format not supported
- 404 (Not Found): Profile not found
- 404 (Not Found): Normalized data not available
- 404 (Not Found): Interval not found
- 403 (Forbidden): Permission denied

Example

```
$ curl -XGET
https://api.wattzon.com/link/4.0/data/intervals/268/__mock_meter__268/
monthly/2015-01/ --cert certificate_file.crt --key key_file.key
```

NOTE: meter_id and ym fields can be found in the interval data from the /link/4.0/data/intervals/{profile_id}/ endpoint.

Notification Support

Purpose

After starting a utility provider data extraction, there are two options for checking on the extraction job status. The first is Link's Job Status API call, which is ideal for interactive applications; it's immediate and can provide insight into running jobs as well as the status of completed job.
The second option is a notification via Amazon's SQS (Simple Queue Service) mechanism. This is especially useful when the data consumer's API access is decoupled from the end user interface (WattzOn 3in1 users, for example), and needs a way to track newly-created profiles. It can also be used to inform server software of new data from additional extractions on existing profiles (as would be the case on regularly-scheduled extraction jobs).

Configuration

You'll need to provide WattzOn with an Amazon SQS endpoint and associated access ID. Your technical contact can help you with this.

Enabling/Disabling Notifications

When configured, SQS notifications are sent by default. You can disable notifications for a

particular job by setting the send_notification field to false in the Extraction Start Job call.

Message Format

An SQS notification is in JSON format. Here's an example for a successful extraction for profile ID 121:

```
{
    "app": "link",
    "profile_id": 121,
    "job_id": "62f1c29e-aca8-40b8-aa9f-eb436539e3fe",
    "tags": ["example", "tags"],
    "created_at": "2017-01-16 04:47:17.155559+00:00",
    "final_state": "success",
    "mode": "bill",
    "existing_records": 12,
    "new_records": 1
}
```

These fields are: * app: The application (in our case, always link). * profile_id: Profile that the data extraction was run for. * job_id: Extraction Job ID string. * tags: List of tags associated with the profile. * created_at: Timestamp (in UTC) of notification message. * final_state: End-of-job status code. See Job Status Codes for a list. * mode: Extraction mode (bill or interval). * new_records: Number of records that this job extracted. * existing_records: Number of records that already existed for the profile before this extraction job started.