

DevHub SDK Documentation

version 2.0



April 12, 2010

Contents

DevHub Module SDK Documentation	1
Using the SDK	1
Creating/Editing a Module	1
Running Your Module	2
Style Checking Your Module	2
Testing Your Module	2
Packaging Your Module	2
API Reference	3
The Module Base Class	3
Module Data Sources	3
Site-specific Data	5
HTTP Request Data	6
GET Data	6
POST Data	6
HTTP Headers	6
Geolocation Data	6
User Location	7
Site/User Location	7
Index	9

DevHub Module SDK Documentation

This document is meant to describe the various features of the DevHub Module SDK. There are two main sections:

- [Using the SDK](#)
- [API Reference](#)

Using the SDK

The DevHub platform uses Python 2.5 as its programming language. If you are not familiar with it, the book "[Dive Into Python](#)" is a good introduction. It is available in multiple formats, including a physical book and a (free) HTML version.

Note

There is also a book by the same author called "Dive into Python 3". While it is also a good book, it is not relevant to the DevHub platform, as the platform uses Python 2.5, not Python 3.x.

Creating/Editing a Module

When you create a module, you need to supply some information about the module:

Name

The internal name of the module. Note that this may change, if there is already a module with that name in the system. To avoid this, your best bet is to prefix the name with the name of your company. The naming convention is the same as a Python module/package. From Python's style guide:

Modules should have short, all-lowercase names. Underscores can be used in the module name if it improves readability. Python packages should also have short, all-lowercase names, although the use of underscores is discouraged. [\[PEP8\]](#)

[PEP8](#) <http://www.python.org/dev/peps/pep-0008/>

Display Name

The name of the module that is shown to the user in the editor.

Author Name

Your name, or the name of the technical contact.

Author Email

Your email address, or the address of the technical contact. This is not shown to the users. It is used internally, in case there is a problem with the module.

Description

The description of what the module does. This is shown to the user in the editor.

When the metadata is saved for the first time, a skeleton module is created for you, the location of which is listed on the front page. There are three files in particular that are important (the paths are relative to the `modules` directory):

`$module/__init__.py`

This is where the **BaseModule** subclass is located. Please see BaseModule's documentation for details on how to implement it.

`$module/tests.py`

This is where the [unit tests](#) for your module are located. One test has been provided, but you are encouraged to add more as the functionality for your module grows. The SDK uses the [unittest/unittest2](#) modules for implementing the tests. Please see [Testing Your Module](#) for more details.

templates/modules/\$module/template.html

This is the skeleton template for a module. The notable thing about this file is that the folder that it is located in can have more than one template file, so that all of the templates for a module are selectable when you [run a module](#). The templating system used by the DevHub platform is the [Jinja2](#) library.

Running Your Module

For each module on the front page, there is a form that allows you to run the module that you created with example site and HTTP request data. The form options are as follows:

Template

The template that you wish to use to render the results generated by your module.

Properties

Modules can have one or more properties. Since property values can be different types, the simplest way to set them is by passing [JSON](#) to the form. Below is an example of a module which uses a property. If you don't pass any JSON, the module will output `{'username': 'test'}`. If you pass `{'user': 'foo'}` into the form, the module will output `{'username': 'foo'}`.

```
from dhplatform.modules import BaseModule

class Module(BaseModule):
    '''A module that shows properties.'''

    user = 'test'

    def run(self):
        return {
            'username': user,
        }
```

Profile

This enables support for [profiling](#) your module. For an example of the profiling output, which follows the rendered template, please see the [Python Module of the Week](#) entry on profiling.

Style Checking Your Module

Due to the moderation process for modules, it is necessary for the code to conform to a readable code style. The DevHub platform uses Python's style guide ([PEP 8](#)) as the coding standard. When you click on the "Check for style conformity" link for a module, the SDK checks it for PEP 8 compliance, plus things such as syntax errors and unused imports.

Testing Your Module

Running your module's unit tests is as simple as clicking a link. When clicked, all of the unit tests for the given module are run. If they all passed, it prints "PASSED!" Otherwise, a list of the errors and failures are printed.

Packaging Your Module

When your module is in a workable state, you may wish to submit it to the DevHub platform moderation queue. To do that, it needs to be packaged in a certain format. This format can be generated by the "Package the module to send to DevHub" link. When clicked, it will run the [style conformity check](#) and the [unit test suite](#). If both of these checks pass, the module will be packaged and the path to the module will be printed to the screen.

API Reference

The API reference covers three major areas:

- The [module API](#)
- The [data source API](#)
- [Data](#) made available via the module API (i.e., the **data** attribute)

The Module Base Class

class `dhplatform.modules.BaseModule` (`request`, `site`, `**kwargs`)

Base class for all DevHub platform modules.

Modules should have access to the [HTTP Request Data](#), [Site-specific Data](#), and [Geolocation Data](#) if they are available.

Variable status: The HTTP status code that the page containing the module should send (defaults to 200). The valid values are 200 and 404. You should only use this in important circumstances.

data

The attributes from the [request](#), [site](#), and [geolocation](#) input that have been made available to modules.

Return type: `dict`

run ()

The "main" method of the module. This is the method that is called by the platform to execute the module's code. This must be implemented by the subclass, otherwise an **exception** will be thrown.

Returns: The data generated by the module for use in its template(s).

Return type: `dict`

Module Data Sources

The base classes for module data sources.

class `dhplatform.datasourcesource.base.DataSource` (`method`, `uri`, `params={}`, `credentials=None`, `max_age=None`, `proxy_info=None`)

The base class for HTTP-based data sources.

Note

the DevHub platform automatically caches all HTTP requests (utilizing HTTP headers such as ETag and Last-Modified), so adding HTTP caching logic to the module code is redundant, unless the API in question does not have support for said headers.

Example usage:

```
uri = 'http://api.example.com/messages'
# These are the POST "form" variables to be sent to the URI.
params = {
    'api_key': '12345',
    'user': '{username}',
    'message': 'Hello, world!',
}
data = {
    'username': 'devhub',
```

```

}
data_source = DataSource('POST', uri, params)
# Usually, the data param here is BaseModule.data (as an object attr).
result = data_source.retrieve(data)
# result now contains some string of data from the HTTP response body.

```

Parameters:

- **method (str)** -- The HTTP method.
- **uri (str)** -- The base URI of the data source.
- **params** (depends on the method: if the method is GET, **dict**; if the method is POST, **dict** (for classic form-based requests) or **str** (usually for sending files/data via API requests).) -- The parameters for the request. Format/usage depends on the method.
- **credentials (None or dict)** -- The authentication credentials needed for the request. The keys for the credentials, if supplied, depend on the authentication method used. By default, username and password are supported for all of the authentication methods that [httplib2](#) has built-in.
- **max_age (None or int)** -- The maximum age of the cached response (in seconds).
- **proxy_info (tuple (host [str], port [int]))** -- Info needed to connect to an intermediary SOCKS proxy.

handle_credentials ()

Applies credentials to the HTTP request. This should not be called directly; it is called if you are writing your own **DataSource** subclass, override this method for different credentials handling (for example, **OAuth handling**).

query

The exact query parameters/data that was passed to the data source, that were originally from the constructor. This property is generated when **retrieve()** is run.

Return type: **dict** or **str**, depending on the HTTP method and Content-Type header.

request_uri

The full request URI (e.g., if the request method was GET, then the query string would be attached), generated after **retrieve()** is run.

Return type: **str**

retrieve (data, use_uri_templates=True)

Executes the HTTP request.

Parameter: **data (dict)** -- The data used for URI template substitution. This is usually **BaseModule.data**.

Returns: The HTTP response body.

Return type: **None** or **str**

Raises: **DataSourceError**

set_custom_headers (headers)

Sets custom HTTP headers for a data source. If you have previously set custom headers, and **headers** contains the same header names but different values, the values will be overwritten. For example:

```

d = DataSource('GET', 'http://www.example.com/')
d.set_custom_headers({'a': 'b', 'b': 'c'})
d.set_custom_headers({'b': 'a', 'c': 'd'})

```

will result in the following headers being sent: {'a': 'b', 'b': 'a', 'c': 'd'}.

Parameter: headers (**dict**) -- The HTTP headers to send to the URI.

exception `dhplatform.datasource.base.DataSourceError` (method, uri, params, code, msg)
A **DataSource**-related error.

Parameters:

- method (**str**) -- The HTTP method used by the data source.
- uri (**str**) -- The base URI used by the data source.
- params (**dict** or **str**, depending on the HTTP method and Content-Type header.) -- The query parameter(s)/data used by the data source.
- code (**int**) -- The HTTP status code sent by the data source.
- msg (**str**) -- Either the HTTP status message sent by the data source, or an error message sent by the data source in the body of the response.

Variable method: See the `method` parameter of the constructor.

Variable uri: See the `uri` parameter of the constructor.

Variable params: See the `params` parameter of the constructor.

Variable code: See the `code` parameter of the constructor.

Variable msg: See the `msg` parameter of the constructor.

class `dhplatform.datasource.base.OAuthDataSourceMixin`
Mixin object to deal with [OAuth](#)-based authentication. ¹

¹ NOTE: No module currently uses this functionality. If you need it, please contact DevHub!

class `dhplatform.datasource.feed.FeedDataSource` (method, uri, params={}, credentials=None, max_age=None, proxy_info=None)
Bases: `dhplatform.datasource.base.DataSource`
A data source where the HTTP response body is an Atom/RSS feed, and the output is a [feedparser](#) dictionary.

class `dhplatform.datasource.json.JSONDataSource` (method, uri, params={}, credentials=None, max_age=None, proxy_info=None)
Bases: `dhplatform.datasource.base.DataSource`
A data source where the HTTP response body is [JSON](#), and the output is the deserialized data (via [simplejson](#)).

class `dhplatform.datasource.xml.XMLDataSource` (method, uri, params={}, credentials=None, max_age=None, proxy_info=None)
Bases: `dhplatform.datasource.base.DataSource`
A data source where the HTTP response body is XML, and the output is an [ElementTree](#) object.

Site-specific Data

The following data about a site is made available to modules via the `data` property, prefixed with `site_`:

- domain (**unicode**): The second-level and top-level parts of the site's domain name, for example, `example.com`.
- subdomain (**unicode**): The third-level part of the site's domain name, for example, `www`.

HTTP Request Data

- `formatted_domain` (**unicode**): The full, canonical domain name that the site uses to identify itself in links. Depending on the preference of the site owner, this may be, for example, `www.example.com` or `example.com`. When creating permalinks, this field should be used.
- `keywords` (**unicode**): The keywords associated with the site.
- `about` (**unicode**): A description of the site.
- `width` (**int**): The width of the site, usually used for the layout.

HTTP Request Data

The following data is made available to modules from the HTTP request via the **data** property:

- [GET variables](#) (if any)
- [POST data](#) (if any)
- certain [HTTP headers](#)

GET Data

GET variables are prefixed with `get_`, so for example, if you had a URI which looked like `http://www.example.com/?foo=bar`, you would retrieve the value of `foo` by referencing `self.data['get_foo']`.

POST Data

POST variables are prefixed with `post_`, so for example, if a user submitted a form where `<form method="post">` and you wanted to access the `name` field, you would reference `self.data['post_name']`.

HTTP Headers

The following HTTP headers have been made available to modules (the `self.data` keys are in parentheses):

- `Host` (`HTTP_HOST`)
- `User-Agent` (`HTTP_USER_AGENT`)

While this is not an HTTP header, the IP address of the visitor is also available, with the key `REMOTE_ADDR`.

Geolocation Data

Geolocation data is made available to modules via the **data** property, in one of two forms: user location, and site/user location. Both forms contain the following keys:

- `latitude` (**float**): North is positive, south is negative.
- `longitude` (**float**): East is positive, west is negative.
- `city` (**unicode**): Usually localized to the language of the region.
- `region` (**unicode**): The abbreviation for a region, for example, `WA`.
- `region_name` (**unicode**): The full name of a region.
- `zip_code` (**unicode**): The ZIP/postal code, if applicable.
- `country_code` (**unicode**): The abbreviation for a country, for example, `US`.
- `country_name` (**unicode**): The full name of a country.
- `weather_id` (**unicode**): The weather ID, as used at weather.com.

User Location

This data is created by calculating the site visitor's location using their IP address. The user location keys are prefixed with `geo_`, so for example, when you access the user's city, you would reference `self.data['geo_city']`.

Site/User Location

This form also retrieves the geolocation of the site visitor, but if the site itself has a location associated with it, that data is used instead. The site/user location keys are prefixed with `location_`, so for example, when you access the user's city, you would reference `self.data['location_city']`.

Index

B

BaseModule (class in dhplatform.modules)

D

data (dhplatform.modules.BaseModule attribute)

DataSource (class in dhplatform.datasource.base)

DataSourceError

dhplatform.datasource.base (module)

dhplatform.datasource.feed (module)

dhplatform.datasource.json (module)

dhplatform.datasource.xml (module)

dhplatform.modules (module)

F

FeedDataSource (class in dhplatform.datasource.feed)

H

handle_credentials()
(dhplatform.datasource.base.DataSource method)

J

JSONDataSource (class in dhplatform.datasource.json)

O

OAuthDataSourceMixin (class in dhplatform.datasource.base)

Q

query (dhplatform.datasource.base.DataSource attribute)

R

request_uri
(dhplatform.datasource.base.DataSource attribute)

retrieve()
(dhplatform.datasource.base.DataSource method)

run() (dhplatform.modules.BaseModule method)

S

set_custom_headers()
(dhplatform.datasource.base.DataSource method)

X

XMLDataSource (class in dhplatform.datasource.xml)