CWC Portal
Api Documentation

Version 2.0

cwc-potal.com

May 27, 2021

Contents

List of Code Listings

1 Integration Consideration

1.1 Base Url

`https://cwc-portal.com/portal/arduino/1.0/apiv2.php`

To access make sure the URL is set to https because server has forced to always use secure connection and it cannot be accessed from non-https connection.

1.2 Required Parameters

- **user_key** - User key provided to you by the admin
- **api_key** - API Key provided to you by the admin

Two important parameters are required **user_key** and **api_key** as a **QUERY** parameter. For example the api_key provided to you is **ABCDEFGHIJK** and user_key is **THISISAUSERKEY**. Then the API url will look like: `https://cwc-portal.com/portal/arduino/1.0/apiv2.php?user_key=THISISAUSERKEY&api_key=ABCDEFGHIJK`

> Make sure you never forget your keys and most important with the api_key, forgetting **api_key** will need to be manually reset by the server admin where as **user_key** can be recovered upon request.

1.3 Request

The URL can be request with any OPTIONS of methods, but primarily we work with GET and POST methods. If the request is basic that requires fetching then it is better to implement GET whereas pushing data to server or having extra bits of data for certain interfaces might be enabled and will require POST method. Any further request requires all the parameters defined in required parameters section. A simple request code is provided in the examples section.

To access data further, need another request parameter using either GET or POST which is the **action**. And to perform further action inside individual along with the **action**, need to pass **interface** with the respective accessible unit ID returned from the response.

1.4 Response

The request responds with JSON formatted data that can be processed in any language. The response primarily contains three parameters `success`, `message` and `now`.

- `success` - This is the response if request was good
- `message` - This is the reason if failed or response on what happened with the data
- `now` - This is the server time Europe/London in **DATE_ISO8601** format

```
{
   "success":true,
   "message":"Welcome to CWC API, you should be able to access you proper methods",
   "now":"2021-05-25T09:13:24+0100"
}
```

Listing 1: Example of basic response

1.5 Fetching Accessible Units

Adding `action=units` to either **GET** or **POST** method starts fetching the accessible units ID along with their name set to their portal.

An extra parameter along with default responses are sent which is `units` and is the list of Units that are available for the API and are enabled at portal. Individual units has following responses:

- **name** - A simple name set at the portal
- **id** - An identifier that will be used for the further requests
- **last_update** - This is the last updated time stored at portal at portal's timezone.(check **now** if for current server time)

2 Salford Requests

Fetching data from individual interface has certain action but for all Salford units, the action that is defined is **data** only. So to access data from respective accessible unit requires two query parameters to be sent from either `GET` or `POST`. Parameters that are required are:

- **interface** - This is the interface **id** we received from the accessible units.
- **action** - This will be **data**

For example interface id on the list of accessible units is 530, then the query data will be as `interface=530&action=data`

After the request, the response will be similar with basic responses along with the interface detail and the data in it.

- `details` - This one consist of the basic detail along with the timezone the unit is in, update cycle which is how long we define the unit becomes offline and when the unit was updated.
- `blocks` - This one contains the displayable blocks at the portal with analogs with setpoints, alarms with flow and output status inside the blocks. There could be multiple blocks but for cases inside Salford, all blocks are of single type. All **analogs** have name, id, value and the units. **Alarms** has alarm state and total flow and depend on type. If **type** is 2 then we use the flow status because it is of pulse type and on other cases we treat it as regular alarm with its status. And at last the **outputs** has status of different outputs inside the block where **high** as 1 and **low** as 0.

3 Examples

*Note: Any line breaks in example code are to format within the page, please check corresponding language if line break supports the code.

```php
<?php
$base_url = "https://cwc-portal.com/portal/arduino/1.0/apiv2.php";
$user_key = "THISISAUSERKEY";
$api_key  = "ABCDEFGHIJK";
$response = json_decode(file_get_contents(
    "$base_url?user_key=$user_key&api_key=$api_key"));
// here we get proper response that can be handled
?>
```

Listing 2: Example to fetch basic request on PHP

Above example can be use to test if the API key is working well.

```python
import urllib.request


base_url = "https://cwc-portal.com/portal/arduino/1.0/apiv2.php";
user_key = "THISISAUSERKEY";
api_key  = "ABCDEFGHIJK";

url = "{0}?user_key={1}&api_key={2}&action=units" .format(base_url,user_key,api_key)
f = urllib.request.urlopen(url)
print(f.read().decode('utf-8'))
# can be further processed using json.loads() using json library
# make sure clearing any opened request after things are done
```

Listing 3: Example to fetch units on Python3

Above example fetches the units that are available for the specific key.

```php
<?php
$base_url = "https://cwc-portal.com/portal/arduino/1.0/apiv2.php";
$user_key = "THISISAUSERKEY";
$api_key  = "ABCDEFGHIJK";
$interface = 530;
$response = json_decode(file_get_contents(
    "$base_url?user_key=$user_key&api_key=$api_key&action=data&interface=$interface"));
// here we get proper response that can be handled
?>
```

Listing 4: Example to fetch data from a specific unit

Above example fetches data from specific unit and we receive response as below.

There are blocks but only one blocks are available for units inside Salford. Each block has a name along with analogs and alarms inside it. Individual analog consists of values, units and their setpoints. The **aid** defined is unique for their individual unit/blocks so can be referenced on 3rd party end. Each blocks might have alarms and if alarm **type** is 2 then we use the **pulse** type and use respective values and for **type** is 0 then we use regular alarm with alarm status.

```
{
  "success":true,
  "message":"Reading Good",
  "now":"2021-05-27T15:36:22+0100",
  "details":{
    "name":"8C:5C:E4:E0:79:6D",
    "last_update":"2021-05-27 15:35:45",
    "tz":"Europe\/London",
    "update_cycle":"60"
  },
  "blocks":[
    {
      "name":"Roof top irrigation 1",
      "last_update":"2021-05-27 15:35:45",
      "mode":"3",
      "analogs":[
        {
          "aid":"0",
          "name":"Climbers",
          "units":"%",
          "value":"7",
          "recharge":"240",
          "cycle_pulses":"0.15",
          "start":"25",
          "stop":"30",
          "dp":"0"
        }
      ],
      "alarms":[
        {
          "aid":"0",
          "status":"1",
          "type":"2",
          "name":"Climbers ",
          "last_change":"2021-04-28 11:32:37",
          "healthy_name":"Healthy",
          "faulty_name":"Alarm",
          "pulse_total":"0.10",
          "pulse_units":"M3"
        }
      ],
      "outputs":[
        {
          "oid":"1",
          "status":"0",
          "name":"Wonderwall soilnoid ",
          "mode":"0",
          "last_update":"2021-05-27 15:35:42",
          "high_state":"On",
          "low_state":"Off"
        }
      ]
    }
  ]
}
```

Listing 5: Response data from an interface