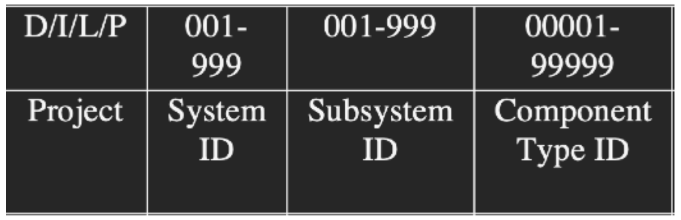
# through the REST API

- Simple examples of how to PUT/PATCH/GET things to/from the HWDB.

- Examples described in this talk are pretty much based on the wiki site:
  https://cdcvs.fnal.gov/redmine/projects/components-db/wiki/Rest_API

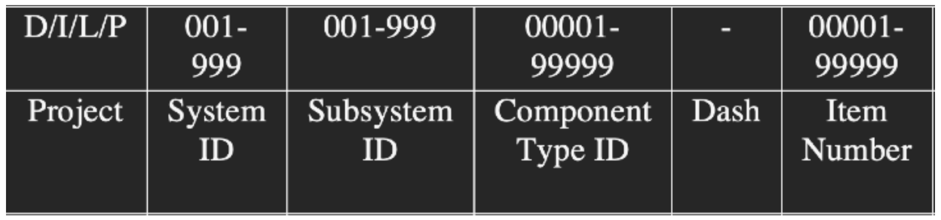
## just a reminding type id, eid, and cid…

- **type id: The first 4 fields of the PID:**



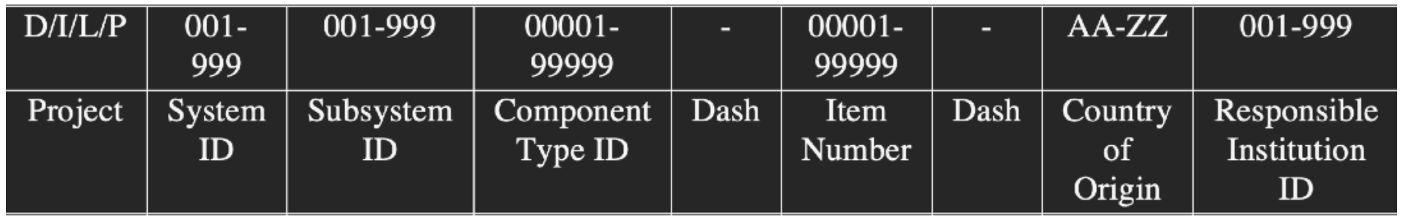
| D/I/L/P | 001-999 | 001-999 | 00001-99999 |
|---|---|---|---|
| Project | System ID | Subsystem ID | Component Type ID |

- **eid : The first 5 fields of the PID:**



| D/I/L/P | 001-999 | 001-999 | 00001-99999 | - | 00001-99999 |
|---|---|---|---|---|---|
| Project | System ID | Subsystem ID | Component Type ID | Dash | Item Number |

- **cid : The first 7 fields of the PID:**



| D/I/L/P | 001-999 | 001-999 | 00001-99999 | - | 00001-99999 | - | AA-ZZ | 001-999 |
|---|---|---|---|---|---|---|---|---|
| Project | System ID | Subsystem ID | Component Type ID | Dash | Item Number | Dash | Country of Origin | Responsible Institution ID |

# Flow of DB entry

1. An architect create Component Types in HWDB based on the PID templates you provide.

did in DAY 1

2. Administrators complete those created Component Types.

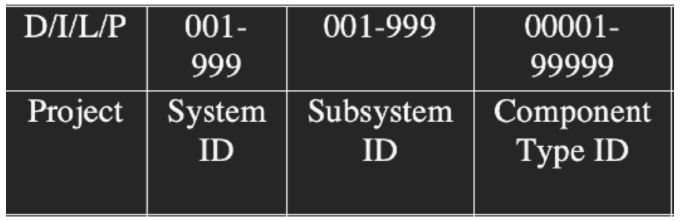
3. Ordinary users can start to enter Items.

Today's task!

## Flow of DB entry

1. An architect create Component Types in HWDB based on the PID templates you provide.

Type id is created:



| D/I/L/P | 001-999 | 001-999 | 00001-99999 |
|---|---|---|---|
| Project | System ID | Subsystem ID | Component Type ID |

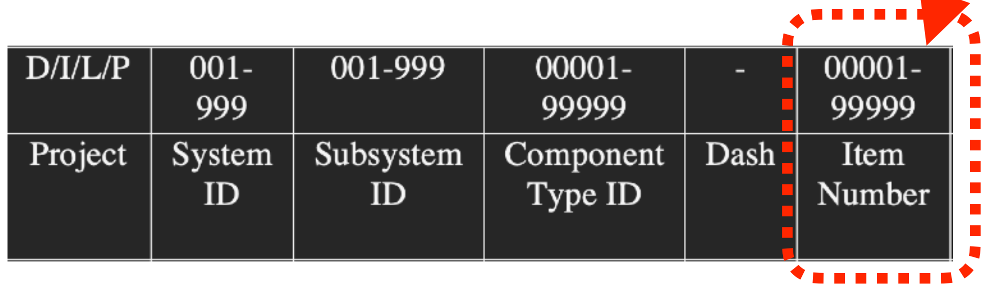
2. Administrators complete those created Component Types.

Type definition is completed:

HWDB generates (increments) it

3. Ordinary users can start to enter Items.

eid and cid are created:



| D/I/L/P | 001-999 | 001-999 | 00001-99999 | - | 00001-99999 |
|---|---|---|---|---|---|
| Project | System ID | Subsystem ID | Component Type ID | Dash | Item Number |

## REST API

- In the following examples, we'll use a curl command.
- curl is a command-line tool to transfer data to or from a server.
  And it supports the protocol we need, https.

- A typical usage is like;
  curl [options] [URL…]

## REST API

- In the rest of this talk, I'll show a bunch of command lines, which usually starts with the following.

  curl --cert-type P12 --cert MyCert.p12:myPSWD 'https://dbwebapi2.fnal.gov:8443/cdbdev/api/…'

  Here,

  ‣ MyCert.p12 is a p12 certificate, obtained from https://cilogon.org

  ‣ myPSWD is the "its password".

- Since they'll show up repeatedly, I will abbreviate them in the following way:

  ‣ curl --cert-type P12 --cert MyCert.p12:myPSWD → CURL

  ‣ https://dbwebapi2.fnal.gov:8443/cdbdev/api → APIPATH

# From DAY 1

- On DAY 1, I took a Component Type “Test_Parts_1” as an example to complete its Type definition by PATCH-ing.
- That Type was defined with this JSON:

```
{
    "part_type_id": "Z00100100048",
    "comments": "Testing...",
    "manufacturers": [7,27],
    "roles": [3,4],
    "properties": {
        "specifications": {
            "ChipSN": "testing Type..."
        }
    },
    "connectors": {
        "MyTest": "Z00100100046"
    }
}
```

- Now let’s insert an Item for this Type.
- The information we need are:
  - Its Type ID : Z00100100048
  - Available manufacturers are 7 (= Hajime Inc) or 27 (= CERN)
  - You must be assigned to one of the Roles : 3 (= type-manager) or 4 (= tester)
  - Its Specifications : In this case, very simple, "ChipSN": String
  - There is only one Component Type that is allowed to be linked: Z00100100046

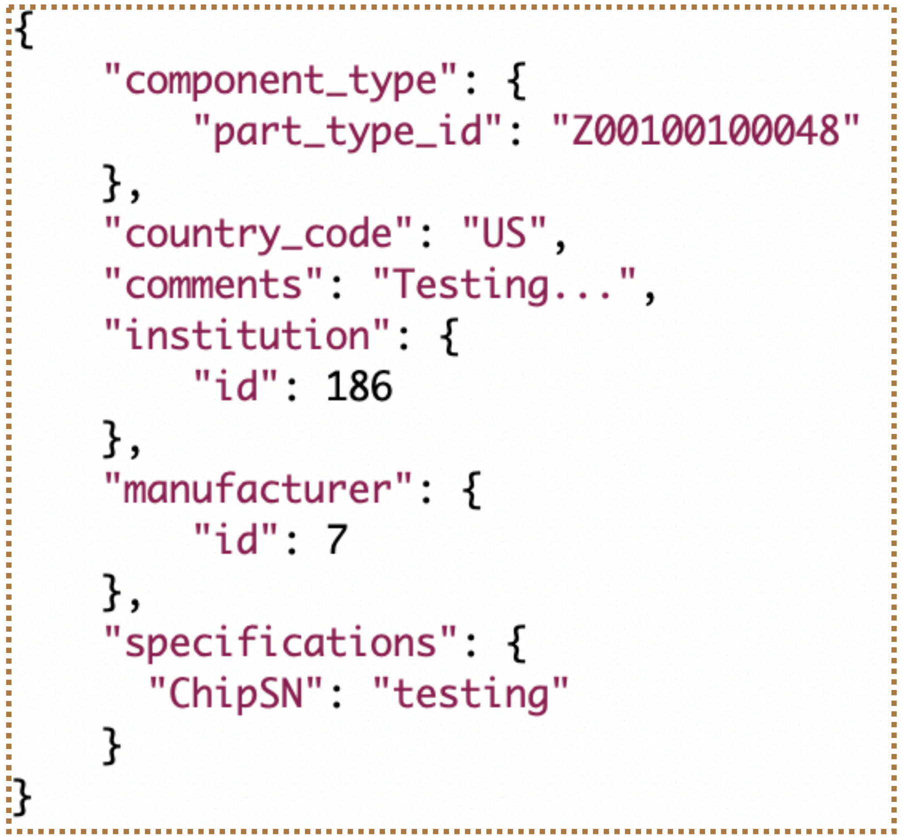
    (we will deal with sub-Components later in this talk)

## Inserting an Item (POST)

- The API endpoint : /api/component-types/<type_id>/components
- An example of actual line:

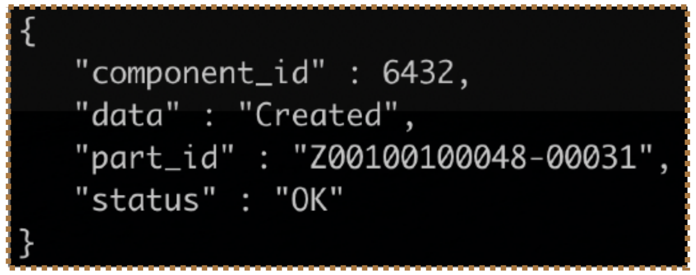
  CURL -H "Content-Type: application/json" -X POST -d @Add_AnItem_Test_Parts_1.json 'APIPATH/component-types/Z00100100048/components'

- This time, we POST an Item. The DB then generates a unique DUNE PID for this Item.
- Again, in this example, the Component Type ID is Z00100100048
- The JSON file looks like this:
  - MUST specify country_code
  - MUST specify institution (e.g., 186 = U of M TC)
  - manufacturer CAN be specified
  - MUST specify specifications.



```
{
    "component_type": {
        "part_type_id": "Z00100100048"
    },
    "country_code": "US",
    "comments": "Testing...",
    "institution": {
        "id": 186
    },
    "manufacturer": {
        "id": 7
    },
    "specifications": {
      "ChipSN": "testing"
    }
}
```

- When executed, you should see a response like this;



```
{
  "component_id" : 6432,
  "data" : "Created",
  "part_id" : "Z00100100048-00031",
  "status" : "OK"
}
```

A PID, Z00100100048-00031, has been created!

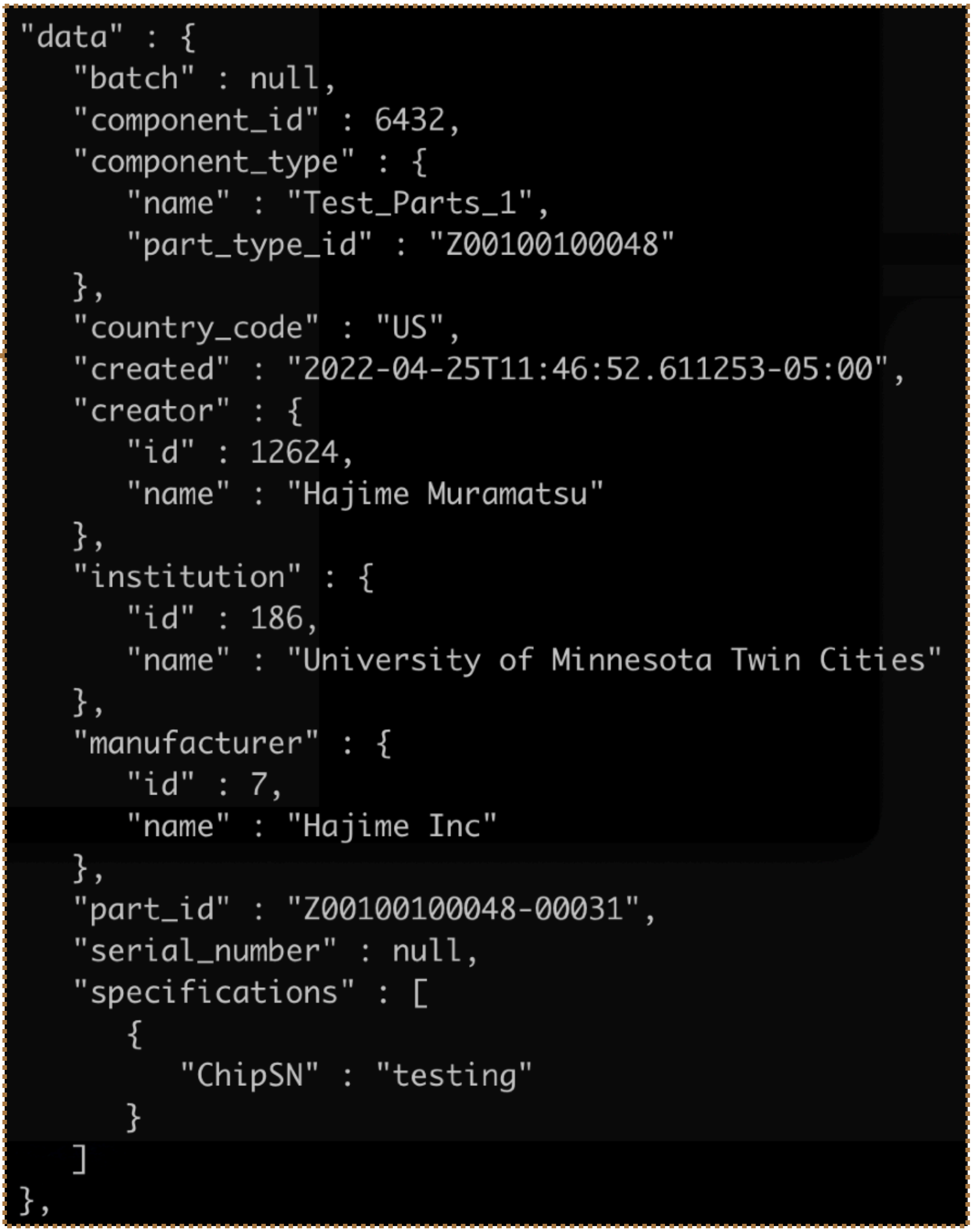
# Again, let's check the Item just created (GET)

- The API endpoint : /api/components/<eid>
- An example of actual line:
  CURL 'APIPATH/components/Z00100100048-00031'

Again, here is the "data" blob. It's there..

We see;

- component type id & name
- country Code
- institution id
- manufacturer
- specifications
- part_id (= the generated pid)



```
"data" : {
   "batch" : null,
   "component_id" : 6432,
   "component_type" : {
      "name" : "Test_Parts_1",
      "part_type_id" : "Z00100100048"
   },
   "country_code" : "US",
   "created" : "2022-04-25T11:46:52.611253-05:00",
   "creator" : {
      "id" : 12624,
      "name" : "Hajime Muramatsu"
   },
   "institution" : {
      "id" : 186,
      "name" : "University of Minnesota Twin Cities"
   },
   "manufacturer" : {
      "id" : 7,
      "name" : "Hajime Inc"
   },
   "part_id" : "Z00100100048-00031",
   "serial_number" : null,
   "specifications" : [
      {
         "ChipSN" : "testing"
      }
   ]
},
```

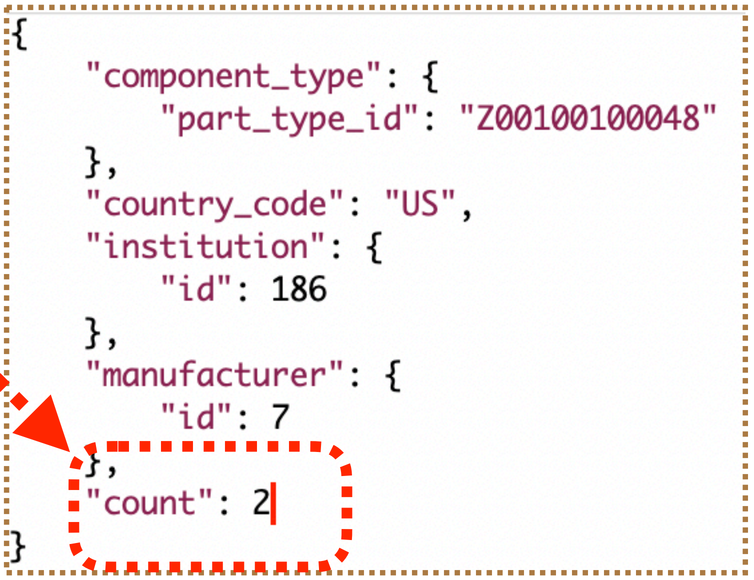
## Inserting a bunch of Items at once (POST)

- The API endpoint : /api/component-types/<type_id>/bulk-add
- An example of actual line:

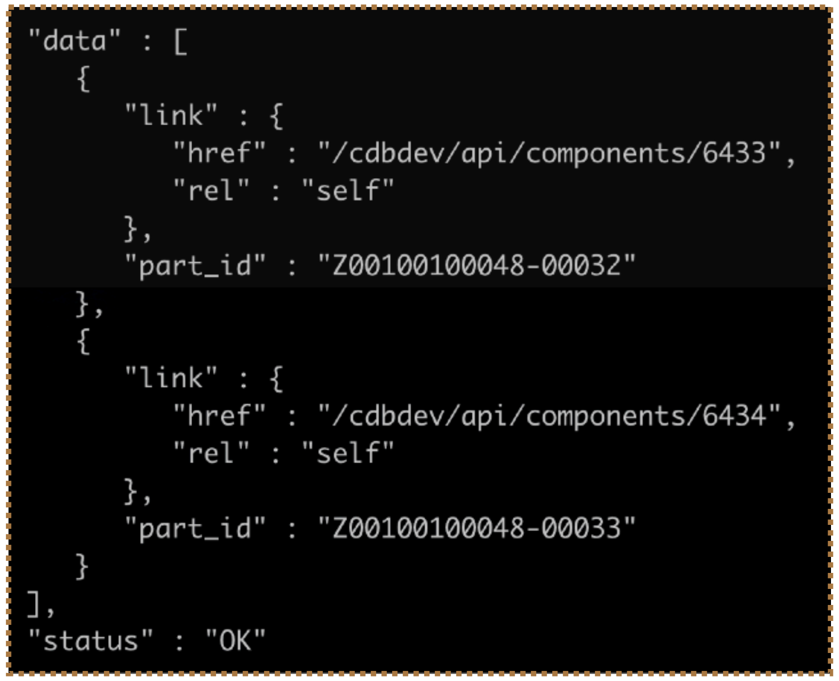
```
CURL -H "Content-Type: application/json" -X POST -d @Add_ItemS_Test_Parts_1.json
'APIPATH/component-types/Z00100100048/bulk-add'
```

- Again, in this example, the Component Type ID is Z00100100048
- The JSON file looks like this:

  Specify how many Items you want to insert by "count".
- When executed, you should see a response like this;



```
{
    "component_type": {
        "part_type_id": "Z00100100048"
    },
    "country_code": "US",
    "institution": {
        "id": 186
    },
    "manufacturer": {
        "id": 7
    },
    "count": 2
}
```



```
"data" : [
   {
      "link" : {
         "href" : "/cdbdev/api/components/6433",
         "rel" : "self"
      },
      "part_id" : "Z00100100048-00032"
   },
   {
      "link" : {
         "href" : "/cdbdev/api/components/6434",
         "rel" : "self"
      },
      "part_id" : "Z00100100048-00033"
   }
],
"status" : "OK"
```

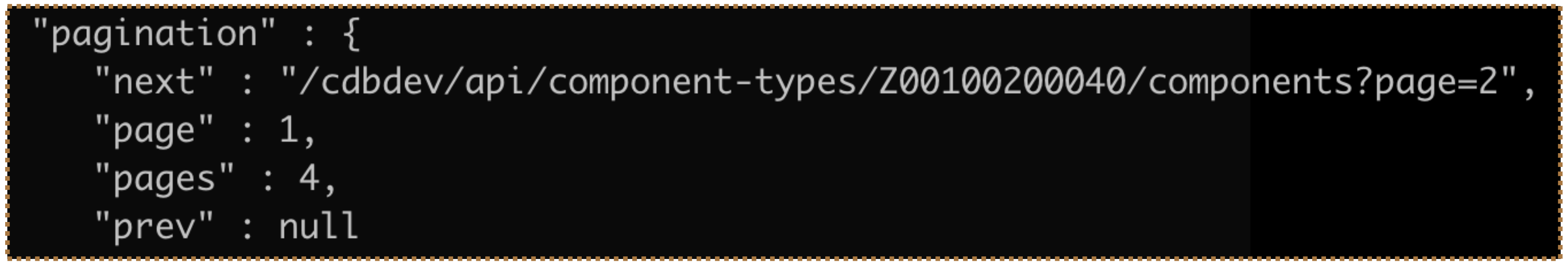
Two eids, Z00100100048-00032 and Z00100100048-00033, have been created!

# Show a list of eid of the all entered Items for a given type id (GET)

- The API endpoint : /api/component-types/<type_id>/components
- An example of actual line:

  CURL 'APIPATH/component-types/Z00100200040/components'

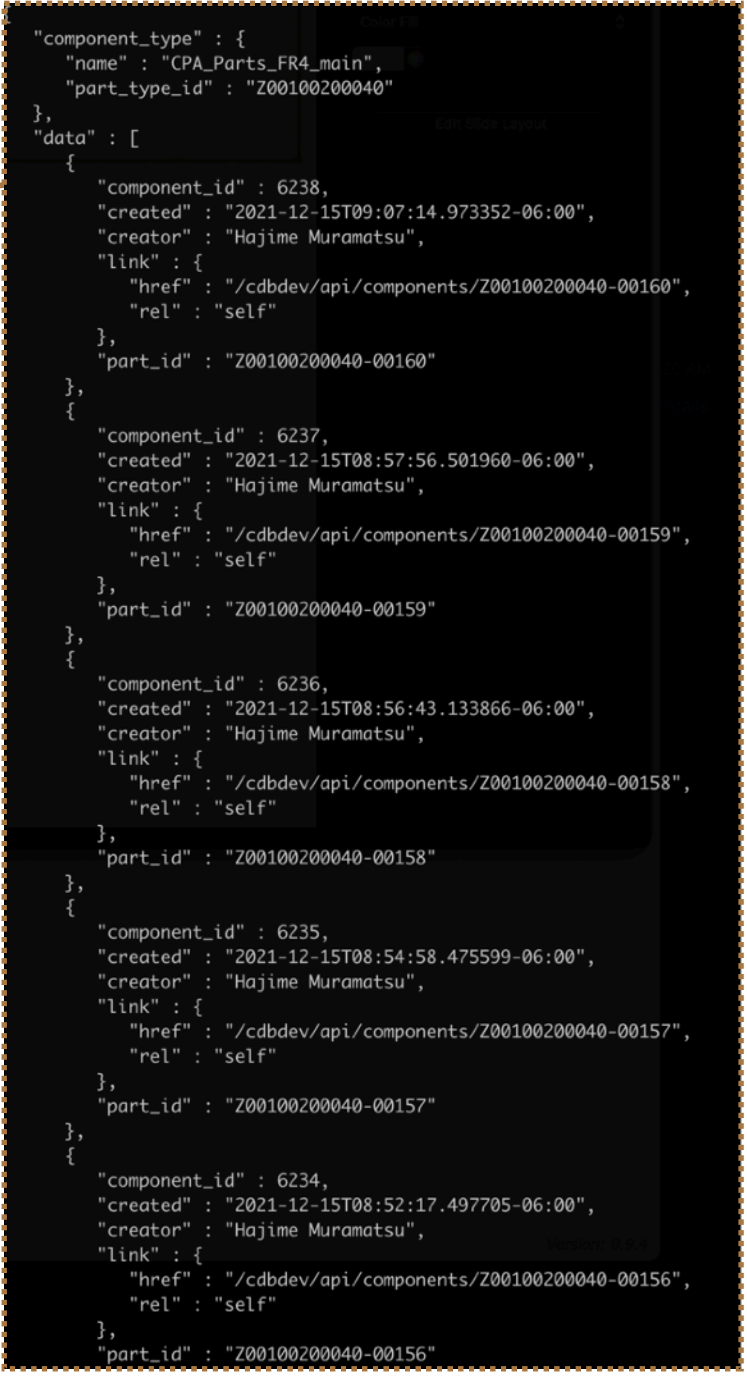
- Shows a list of all entered Items for this Type (CPA_Parts_FR4_main).
- It shows only 50 Items at once.
- When more than 50 Items, managed by Pagination.



```
"pagination" : {
   "next" : "/cdbdev/api/component-types/Z00100200040/components?page=2",
   "page" : 1,
   "pages" : 4,
   "prev" : null
```

- To access to a different page;

  CURL 'APIPATH/component-types/Z00100200040/components?page=4'



```
{
  "component_type" : {
     "name" : "CPA_Parts_FR4_main",
     "part_type_id" : "Z00100200040"
  },
  "data" : [
     {
        "component_id" : 6238,
        "created" : "2021-12-15T09:07:14.973352-06:00",
        "creator" : "Hajime Muramatsu",
        "link" : {
           "href" : "/cdbdev/api/components/Z00100200040-00160",
           "rel" : "self"
        },
        "part_id" : "Z00100200040-00160"
     },
     {
        "component_id" : 6237,
        "created" : "2021-12-15T08:57:56.501960-06:00",
        "creator" : "Hajime Muramatsu",
        "link" : {
           "href" : "/cdbdev/api/components/Z00100200040-00159",
           "rel" : "self"
        },
        "part_id" : "Z00100200040-00159"
     },
     {
        "component_id" : 6236,
        "created" : "2021-12-15T08:56:43.133866-06:00",
        "creator" : "Hajime Muramatsu",
        "link" : {
           "href" : "/cdbdev/api/components/Z00100200040-00158",
           "rel" : "self"
        },
        "part_id" : "Z00100200040-00158"
     },
     {
        "component_id" : 6235,
        "created" : "2021-12-15T08:54:58.475599-06:00",
        "creator" : "Hajime Muramatsu",
        "link" : {
           "href" : "/cdbdev/api/components/Z00100200040-00157",
           "rel" : "self"
        },
        "part_id" : "Z00100200040-00157"
     },
     {
        "component_id" : 6234,
        "created" : "2021-12-15T08:52:17.497705-06:00",
        "creator" : "Hajime Muramatsu",
        "link" : {
           "href" : "/cdbdev/api/components/Z00100200040-00156",
           "rel" : "self"
        },
        "part_id" : "Z00100200040-00156"
```

## Posting a Test result

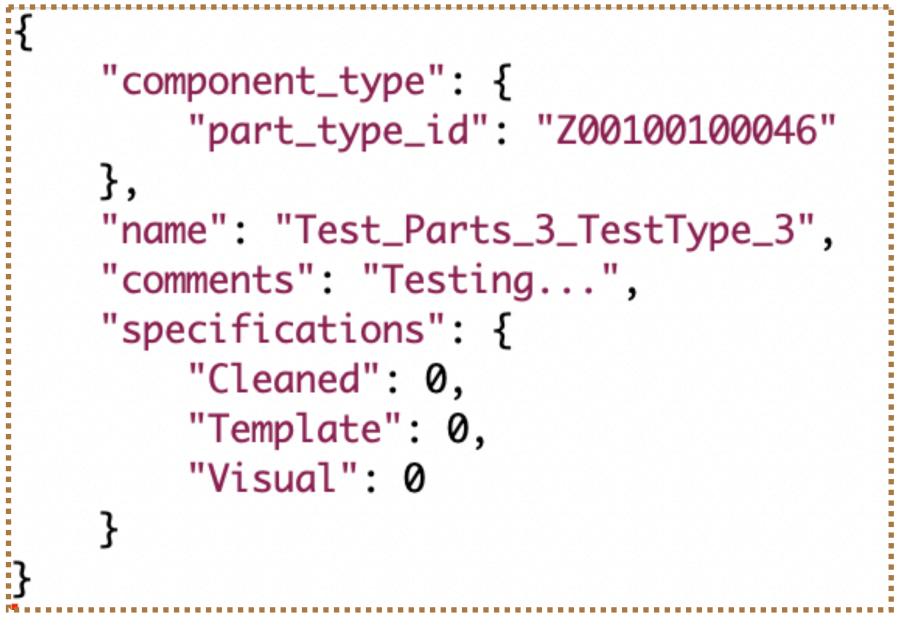
- Now that we can POST/GET an Item,
  we like to POST a test result that is associated with an Item.
- The procedure is similar:
  - Define a Test Type
    (but unlike Component Type, there can be
    multiple Test Types for a given Component Type)
  - Post a Test.

## Creating a Test Type for a given Component Type(POST)

- The API endpoint : /api/component-types/<type_id>/test-types
- An example of actual line:

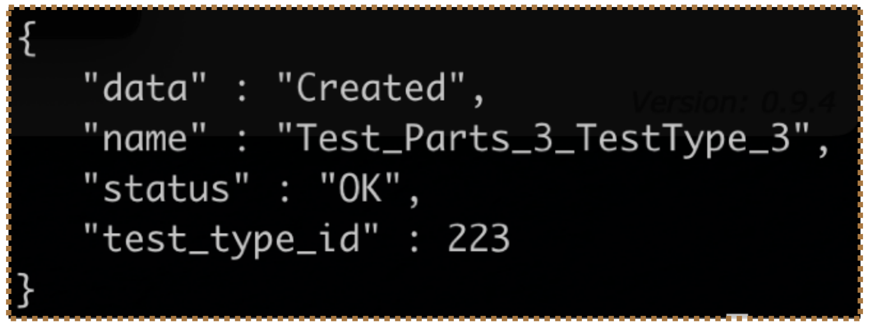
  CURL -H "Content-Type: application/json" -X POST -d @Post_TestType_Test_parts_3.json 'APIPATH/component-types/Z00100100046/test-types'

- In this example, the Component Type ID is Z00100100046
- The JSON file looks like this:
  - Specify a Test Type name
  - Specification is given in JSON



```
{
    "component_type": {
        "part_type_id": "Z00100100046"
    },
    "name": "Test_Parts_3_TestType_3",
    "comments": "Testing...",
    "specifications": {
        "Cleaned": 0,
        "Template": 0,
        "Visual": 0
    }
}
```

- When executed, you should see a response like this;



```
{
   "data" : "Created",
   "name" : "Test_Parts_3_TestType_3",
   "status" : "OK",
   "test_type_id" : 223
}
```

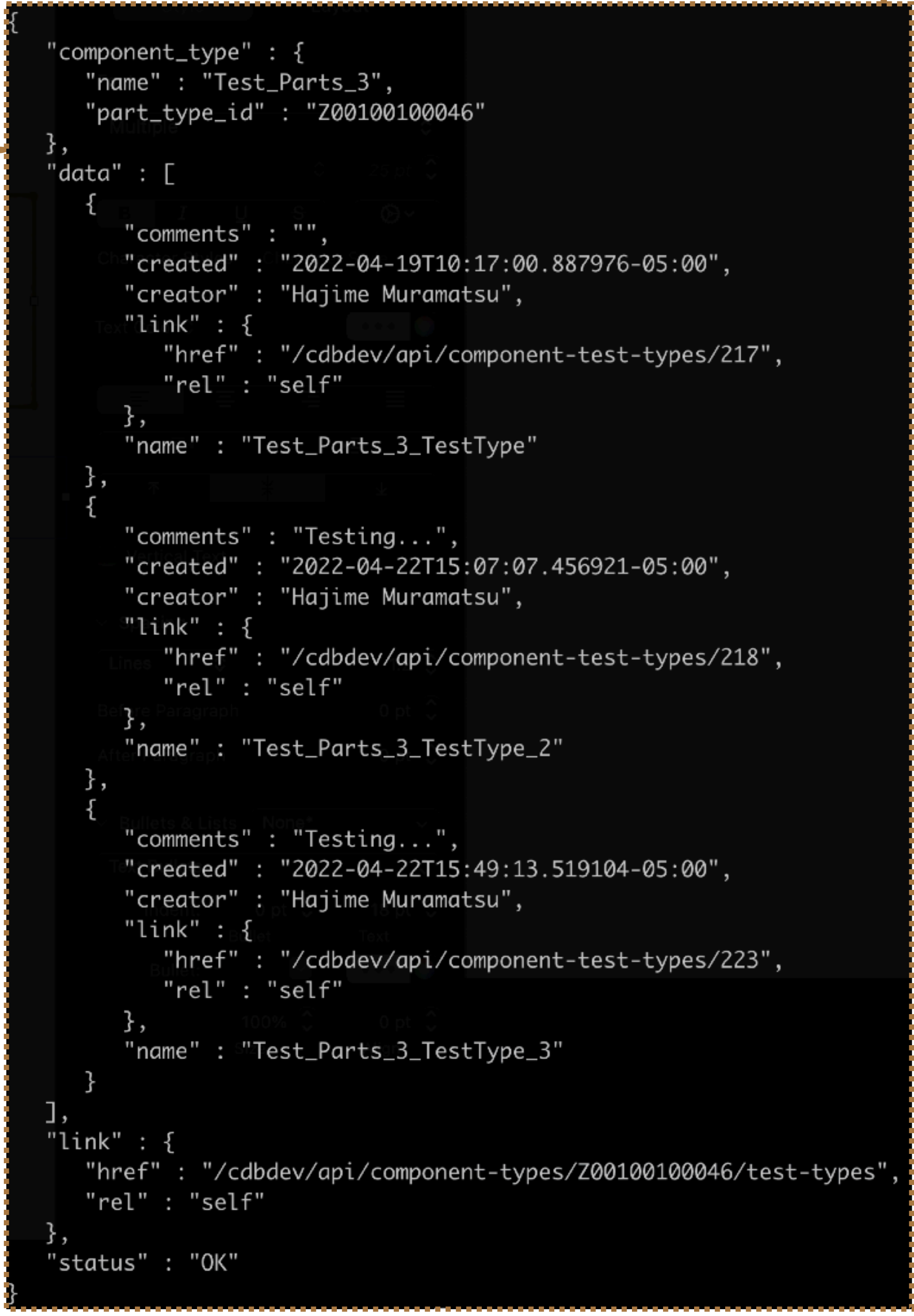
A new Test Type, Test_Parts_3_TestType_3, has been created.

# Checking Test Types (GET)

- The API endpoint : /api/component-types/<type_id>/test-types
- An example of actual line:
  CURL 'APIPATH/component-types/Z00100100046/test-types'

- Shows Test Types that are available for this Component Type.

- In this case, there are 3 Test Types are there, Test_Parts_3_TestType, Test_Parts_3_TestType_2, and Test_Parts_3_TestType_3.



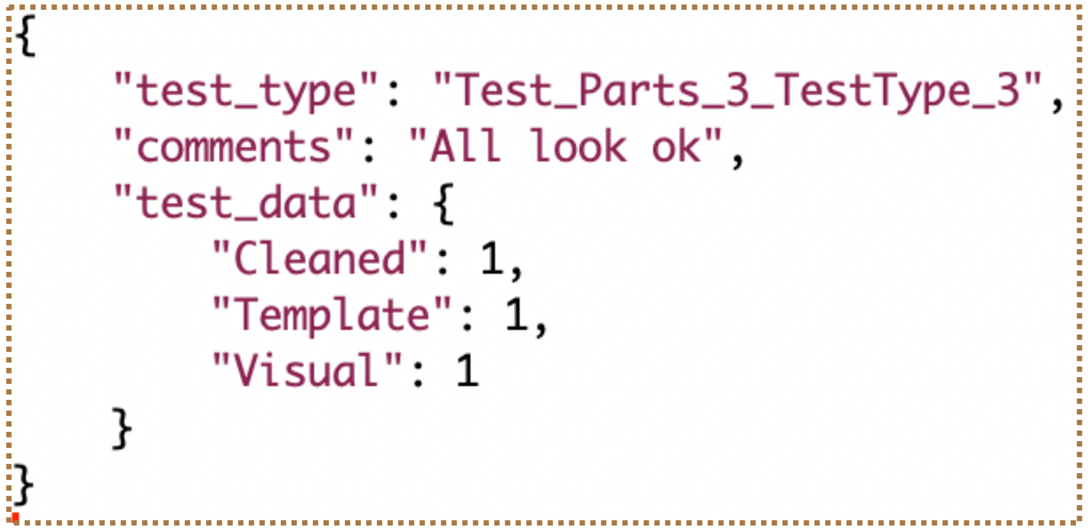
```
{
  "component_type" : {
    "name" : "Test_Parts_3",
    "part_type_id" : "Z00100100046"
  },
  "data" : [
    {
      "comments" : "",
      "created" : "2022-04-19T10:17:00.887976-05:00",
      "creator" : "Hajime Muramatsu",
      "link" : {
        "href" : "/cdbdev/api/component-test-types/217",
        "rel" : "self"
      },
      "name" : "Test_Parts_3_TestType"
    },
    {
      "comments" : "Testing...",
      "created" : "2022-04-22T15:07:07.456921-05:00",
      "creator" : "Hajime Muramatsu",
      "link" : {
        "href" : "/cdbdev/api/component-test-types/218",
        "rel" : "self"
      },
      "name" : "Test_Parts_3_TestType_2"
    },
    {
      "comments" : "Testing...",
      "created" : "2022-04-22T15:49:13.519104-05:00",
      "creator" : "Hajime Muramatsu",
      "link" : {
        "href" : "/cdbdev/api/component-test-types/223",
        "rel" : "self"
      },
      "name" : "Test_Parts_3_TestType_3"
    }
  ],
  "link" : {
    "href" : "/cdbdev/api/component-types/Z00100100046/test-types",
    "rel" : "self"
  },
  "status" : "OK"
}
```

## Posting test results (POST)

- The API endpoint : /api/components/<eid>/tests
- An example of actual line:

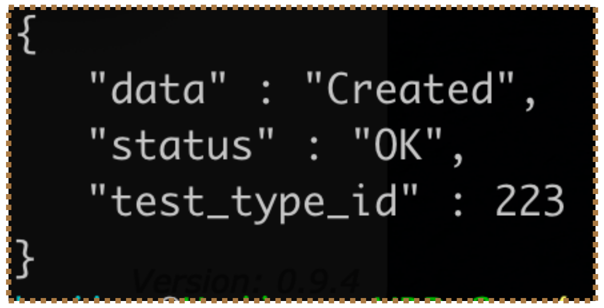
  CURL -H "Content-Type: application/json" -X POST -d @Post_TestResult_Test_parts_3.json 'APIPATH/components/Z00100100046-00008/tests'

- In this example, the external ID is Z00100100046-00008.
- The JSON file looks like this:
  - Need to specify a Test Type Name.
  - Can provide comments.
  - Specification (= test_data) is given in JSON.



```
{
    "test_type": "Test_Parts_3_TestType_3",
    "comments": "All look ok",
    "test_data": {
        "Cleaned": 1,
        "Template": 1,
        "Visual": 1
    }
}
```

- When executed, you should see a response like this;



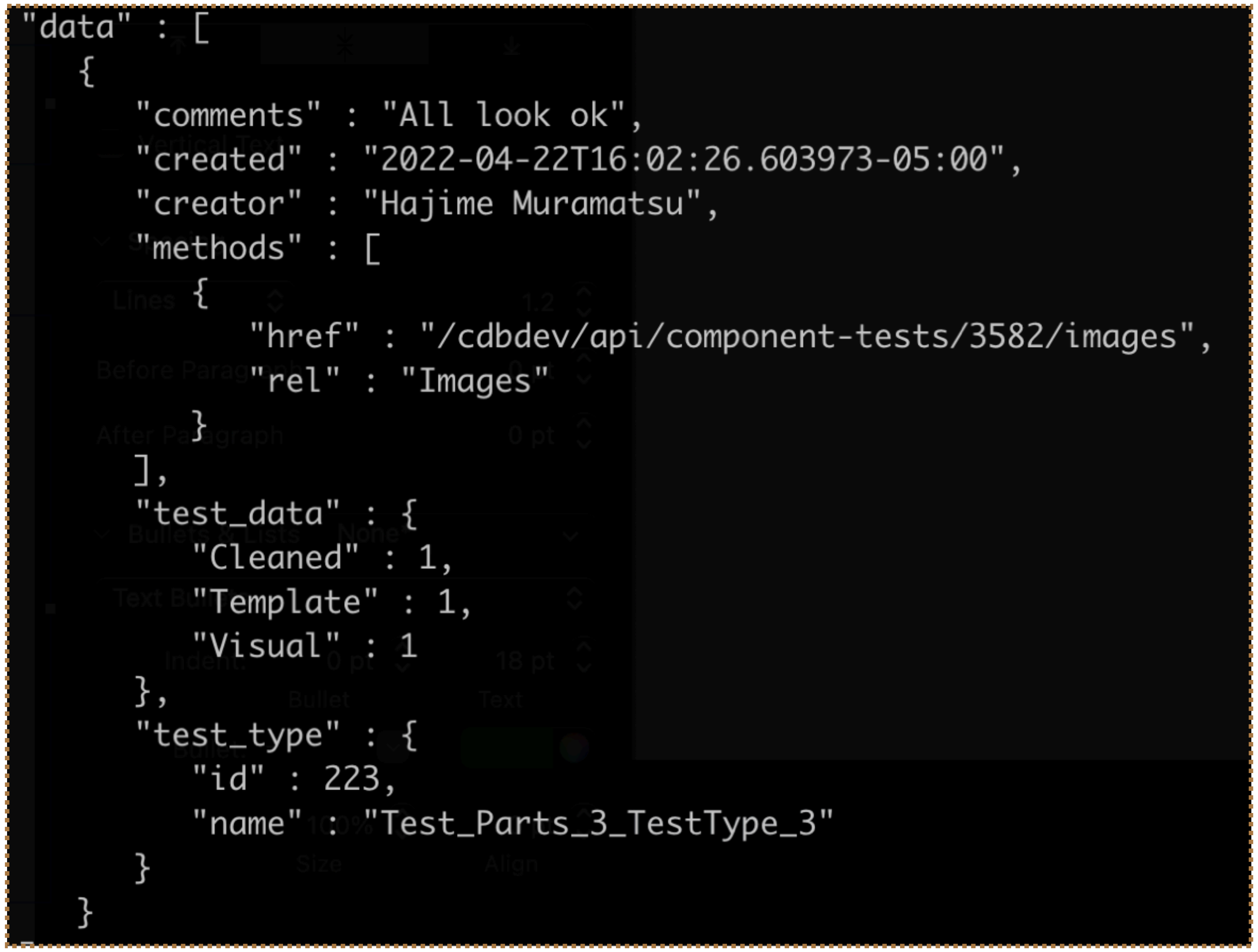
```
{
   "data" : "Created",
   "status" : "OK",
   "test_type_id" : 223
}
```

## Check test results (GET)

- The API endpoint : /api/components/<eid>/tests/<test_type_name>
- An example of actual line:

  CURL 'APIPATH/components/Z00100100046-00008/tests/Test_Parts_3_TestType_3'

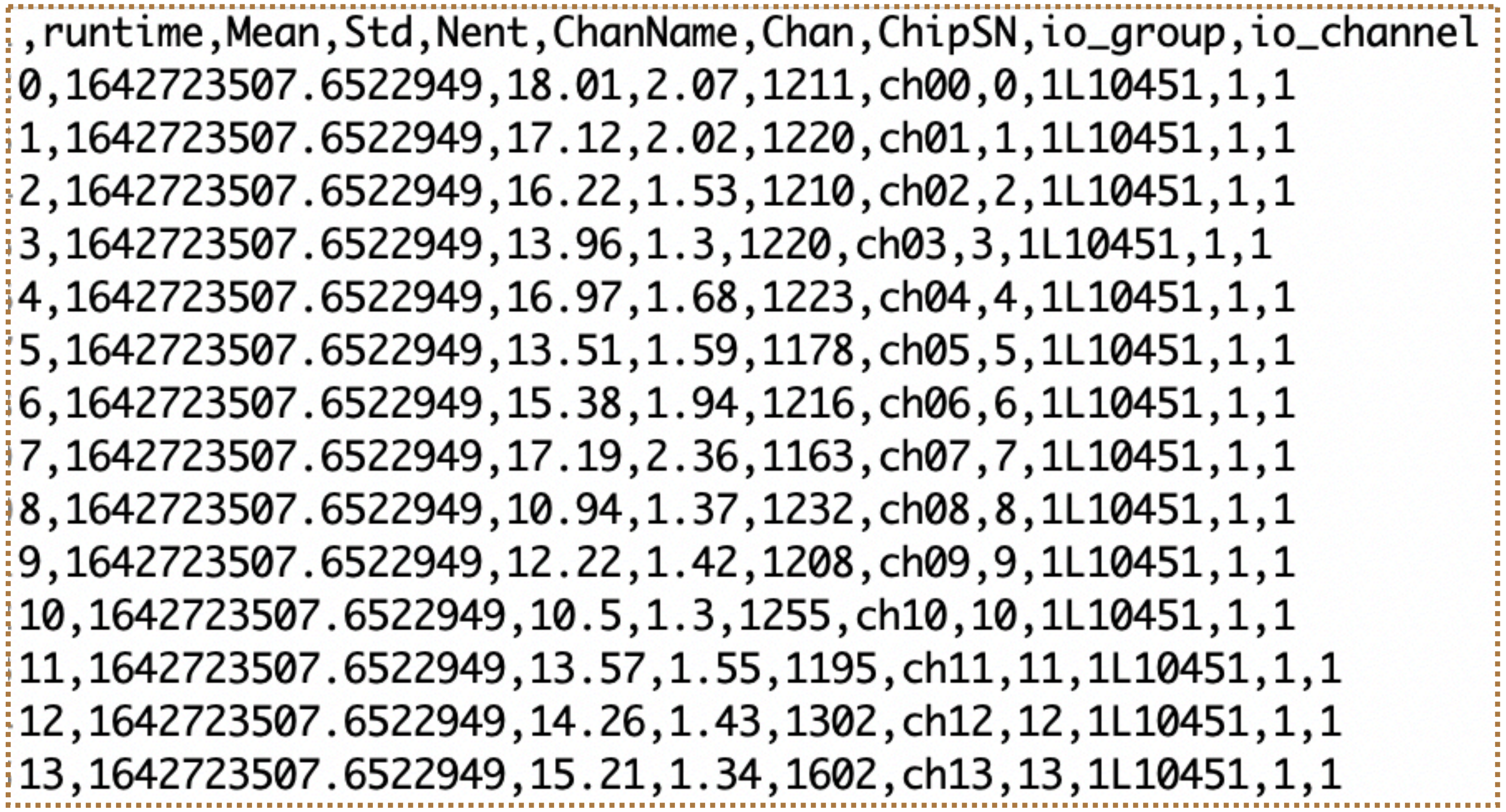
- They are in the data blob.



```
"data" : [
   {
      "comments" : "All look ok",
      "created" : "2022-04-22T16:02:26.603973-05:00",
      "creator" : "Hajime Muramatsu",
      "methods" : [
         {
            "href" : "/cdbdev/api/component-tests/3582/images",
            "rel" : "Images"
         }
      ],
      "test_data" : {
         "Cleaned" : 1,
         "Template" : 1,
         "Visual" : 1
      },
      "test_type" : {
         "id" : 223,
         "name" : "Test_Parts_3_TestType_3"
      }
   }
```

## An example of posting a long list of test results

- Suppose… I want to upload test (measurement) results on 333 different ASICs. And I want to store info on each 64 channels of those ASICs.
- Starting from a csv file that looks like this:

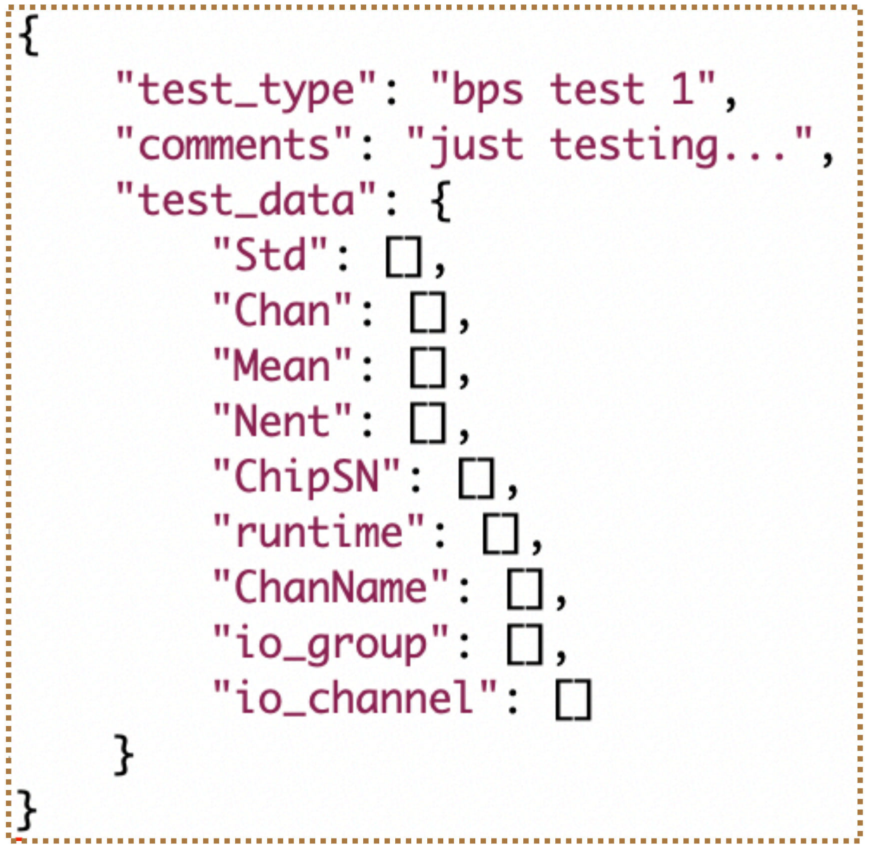


```
,runtime,Mean,Std,Nent,ChanName,Chan,ChipSN,io_group,io_channel
0,1642723507.6522949,18.01,2.07,1211,ch00,0,1L10451,1,1
1,1642723507.6522949,17.12,2.02,1220,ch01,1,1L10451,1,1
2,1642723507.6522949,16.22,1.53,1210,ch02,2,1L10451,1,1
3,1642723507.6522949,13.96,1.3,1220,ch03,3,1L10451,1,1
4,1642723507.6522949,16.97,1.68,1223,ch04,4,1L10451,1,1
5,1642723507.6522949,13.51,1.59,1178,ch05,5,1L10451,1,1
6,1642723507.6522949,15.38,1.94,1216,ch06,6,1L10451,1,1
7,1642723507.6522949,17.19,2.36,1163,ch07,7,1L10451,1,1
8,1642723507.6522949,10.94,1.37,1232,ch08,8,1L10451,1,1
9,1642723507.6522949,12.22,1.42,1208,ch09,9,1L10451,1,1
10,1642723507.6522949,10.5,1.3,1255,ch10,10,1L10451,1,1
11,1642723507.6522949,13.57,1.55,1195,ch11,11,1L10451,1,1
12,1642723507.6522949,14.26,1.43,1302,ch12,12,1L10451,1,1
13,1642723507.6522949,15.21,1.34,1602,ch13,13,1L10451,1,1
```

There are 333×64 = 21312 lines there…

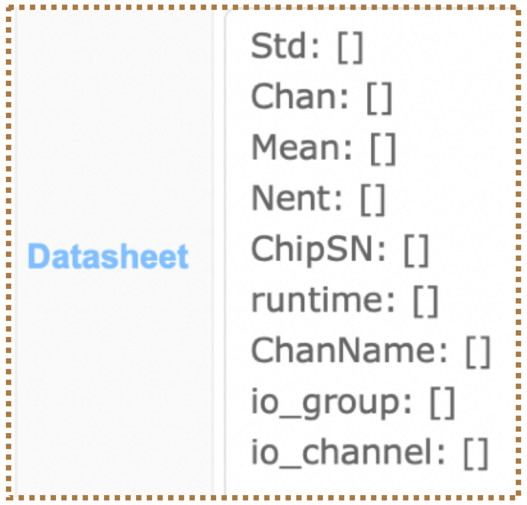
- There are a couple of ways to do this…
  - One could define an ASIC as an Item and store the info of the corresponding 64 channels to its Test Log.

My JSON for this Test Type would look like this:



```
{
    "test_type": "bps test 1",
    "comments": "just testing...",
    "test_data": {
        "Std": [],
        "Chan": [],
        "Mean": [],
        "Nent": [],
        "ChipSN": [],
        "runtime": [],
        "ChanName": [],
        "io_group": [],
        "io_channel": []
    }
}
```

or in the WEB UI form:



| | |
|---|---|
| Datasheet | Std: []<br>Chan: []<br>Mean: []<br>Nent: []<br>ChipSN: []<br>runtime: []<br>ChanName: []<br>io_group: []<br>io_channel: [] |

The idea here is that:

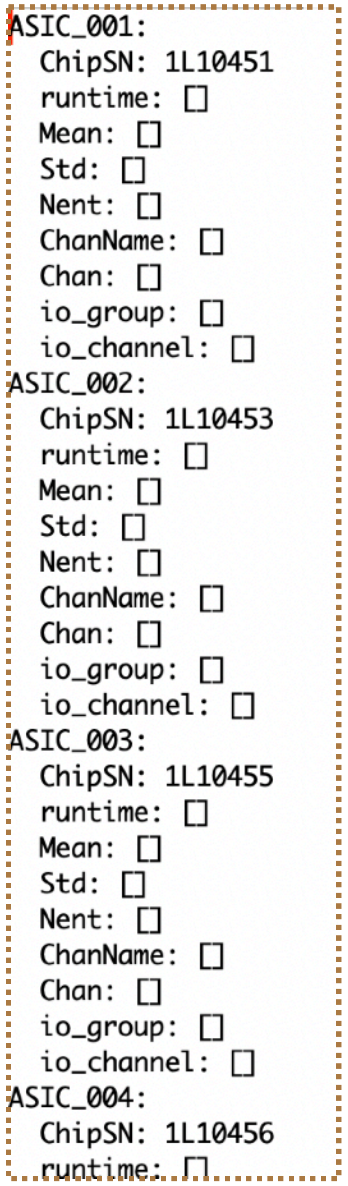
➡ Store each channel info in arrays

➡ I'll have 333 Items to upload.

- **Another approach could be put everything into a single Item.**
  **Not a good idea (probably doesn't make much sense…)**
  **But let me do this way just for testing purpose.**

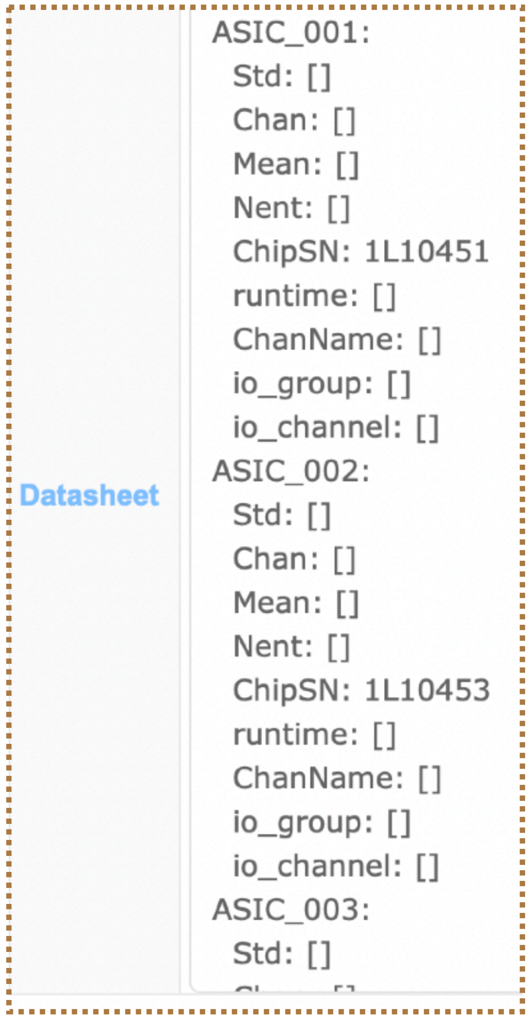
- **We want to put the info of the all 333 ASIC's into a single Test Log.**

**My JSON for this Test Type would look like this:**



```
ASIC_001:
  ChipSN: 1L10451
  runtime: []
  Mean: []
  Std: []
  Nent: []
  ChanName: []
  Chan: []
  io_group: []
  io_channel: []
ASIC_002:
  ChipSN: 1L10453
  runtime: []
  Mean: []
  Std: []
  Nent: []
  ChanName: []
  Chan: []
  io_group: []
  io_channel: []
ASIC_003:
  ChipSN: 1L10455
  runtime: []
  Mean: []
  Std: []
  Nent: []
  ChanName: []
  Chan: []
  io_group: []
  io_channel: []
ASIC_004:
  ChipSN: 1L10456
  runtime: []
```

**or in the WEB UI form:**



```
ASIC_001:
  Std: []
  Chan: []
  Mean: []
  Nent: []
  ChipSN: 1L10451
  runtime: []
  ChanName: []
  io_group: []
  io_channel: []
ASIC_002:
  Std: []
  Chan: []
  Mean: []
  Nent: []
  ChipSN: 1L10453
  runtime: []
  ChanName: []
  io_group: []
  io_channel: []
ASIC_003:
  Std: []
```

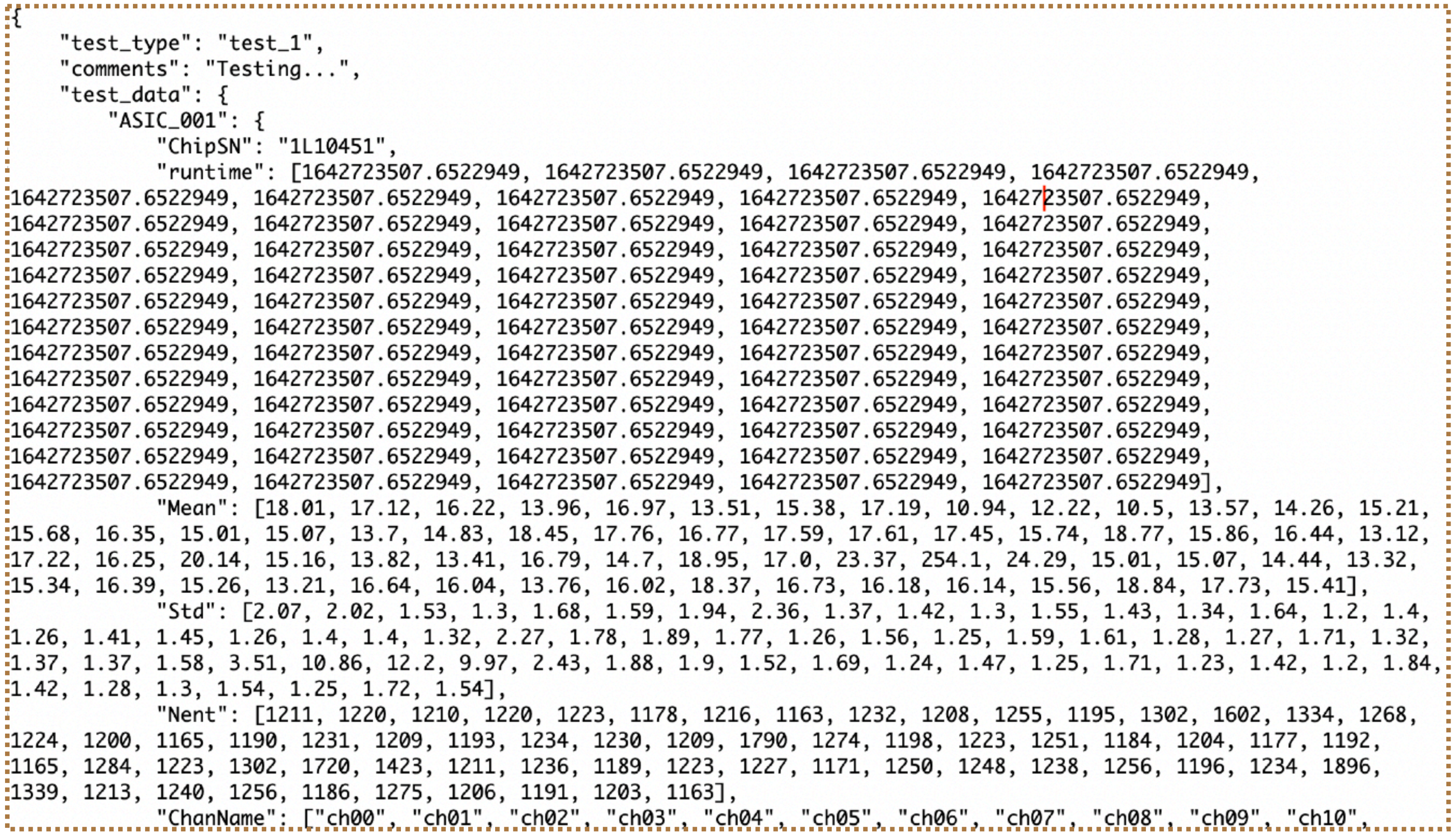
- **Nested Keys**
- **Key names represent ASIC numbers.**
- **Certainly don't want to do this manually!**
  **Use scripts/apps to generate them!**

## To upload this, the API endpoint is the same as before:

- The API endpoint : /api/components/<eid>/tests
- An example of actual line:

  CURL -H "Content-Type: application/json" -X POST -d @Generated_test_1.json 'APIPATH/components/Z00100100048-00033/tests'

But I generated the JSON file by a script…



```
{
    "test_type": "test_1",
    "comments": "Testing...",
    "test_data": {
        "ASIC_001": {
            "ChipSN": "1L10451",
            "runtime": [1642723507.6522949, 1642723507.6522949, 1642723507.6522949, 1642723507.6522949,
1642723507.6522949, 1642723507.6522949, 1642723507.6522949, 1642723507.6522949, 1642723507.6522949,
1642723507.6522949, 1642723507.6522949, 1642723507.6522949, 1642723507.6522949, 1642723507.6522949,
1642723507.6522949, 1642723507.6522949, 1642723507.6522949, 1642723507.6522949, 1642723507.6522949,
1642723507.6522949, 1642723507.6522949, 1642723507.6522949, 1642723507.6522949, 1642723507.6522949,
1642723507.6522949, 1642723507.6522949, 1642723507.6522949, 1642723507.6522949, 1642723507.6522949,
1642723507.6522949, 1642723507.6522949, 1642723507.6522949, 1642723507.6522949, 1642723507.6522949,
1642723507.6522949, 1642723507.6522949, 1642723507.6522949, 1642723507.6522949, 1642723507.6522949,
1642723507.6522949, 1642723507.6522949, 1642723507.6522949, 1642723507.6522949, 1642723507.6522949,
1642723507.6522949, 1642723507.6522949, 1642723507.6522949, 1642723507.6522949, 1642723507.6522949,
1642723507.6522949, 1642723507.6522949, 1642723507.6522949, 1642723507.6522949, 1642723507.6522949,
1642723507.6522949, 1642723507.6522949, 1642723507.6522949, 1642723507.6522949, 1642723507.6522949,
1642723507.6522949, 1642723507.6522949, 1642723507.6522949, 1642723507.6522949, 1642723507.6522949],
            "Mean": [18.01, 17.12, 16.22, 13.96, 16.97, 13.51, 15.38, 17.19, 10.94, 12.22, 10.5, 13.57, 14.26, 15.21,
15.68, 16.35, 15.01, 15.07, 13.7, 14.83, 18.45, 17.76, 16.77, 17.59, 17.61, 17.45, 15.74, 18.77, 15.86, 16.44, 13.12,
17.22, 16.25, 20.14, 15.16, 13.82, 13.41, 16.79, 14.7, 18.95, 17.0, 23.37, 254.1, 24.29, 15.01, 15.07, 14.44, 13.32,
15.34, 16.39, 15.26, 13.21, 16.64, 16.04, 13.76, 16.02, 18.37, 16.73, 16.18, 16.14, 15.56, 18.84, 17.73, 15.41],
            "Std": [2.07, 2.02, 1.53, 1.3, 1.68, 1.59, 1.94, 2.36, 1.37, 1.42, 1.3, 1.55, 1.43, 1.34, 1.64, 1.2, 1.4,
1.26, 1.41, 1.45, 1.26, 1.4, 1.4, 1.32, 2.27, 1.78, 1.89, 1.77, 1.26, 1.56, 1.25, 1.59, 1.61, 1.28, 1.27, 1.71, 1.32,
1.37, 1.37, 1.58, 3.51, 10.86, 12.2, 9.97, 2.43, 1.88, 1.9, 1.52, 1.69, 1.24, 1.47, 1.25, 1.71, 1.23, 1.42, 1.2, 1.84,
1.42, 1.28, 1.3, 1.54, 1.25, 1.72, 1.54],
            "Nent": [1211, 1220, 1210, 1220, 1223, 1178, 1216, 1163, 1232, 1208, 1255, 1195, 1302, 1602, 1334, 1268,
1224, 1200, 1165, 1190, 1231, 1209, 1193, 1234, 1230, 1209, 1790, 1274, 1198, 1223, 1251, 1184, 1204, 1177, 1192,
1165, 1284, 1223, 1302, 1720, 1423, 1211, 1236, 1189, 1223, 1227, 1171, 1250, 1248, 1238, 1256, 1196, 1234, 1896,
1339, 1213, 1240, 1256, 1186, 1275, 1206, 1191, 1203, 1163],
            "ChanName": ["ch00", "ch01", "ch02", "ch03", "ch04", "ch05", "ch06", "ch07", "ch08", "ch09", "ch10",
```

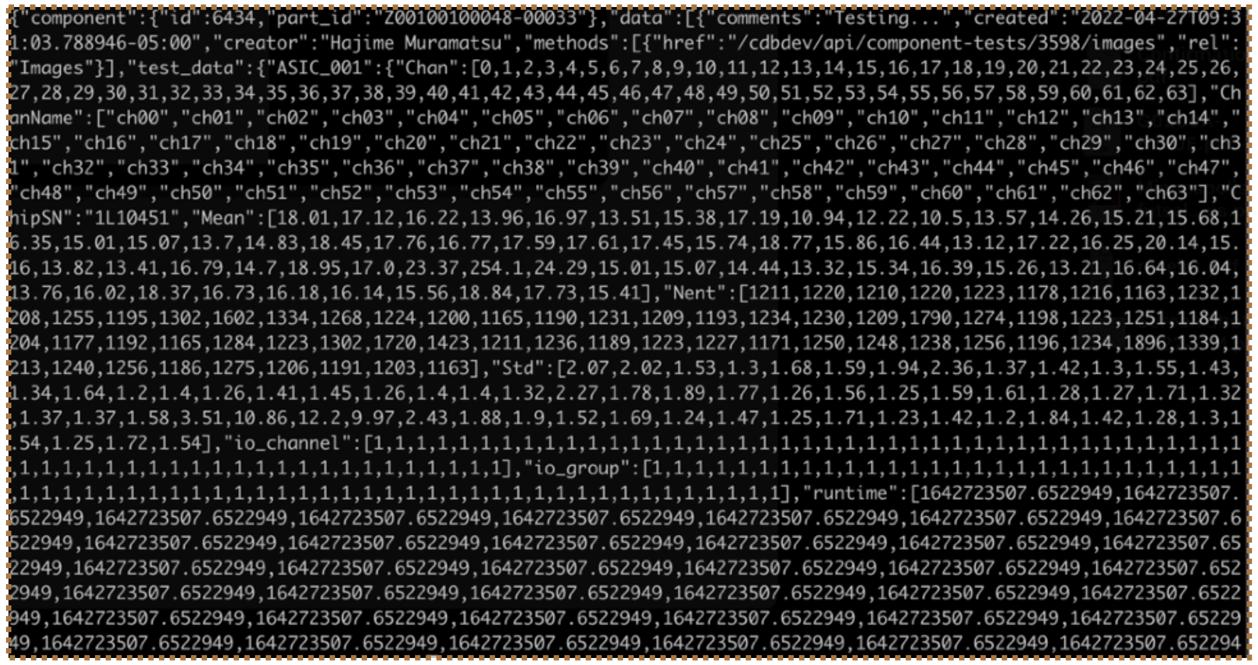
- It took about 5-20 sec, depending on my neighbors(?), to upload this from my home.

## And we can check if the stuff is there in the same way.

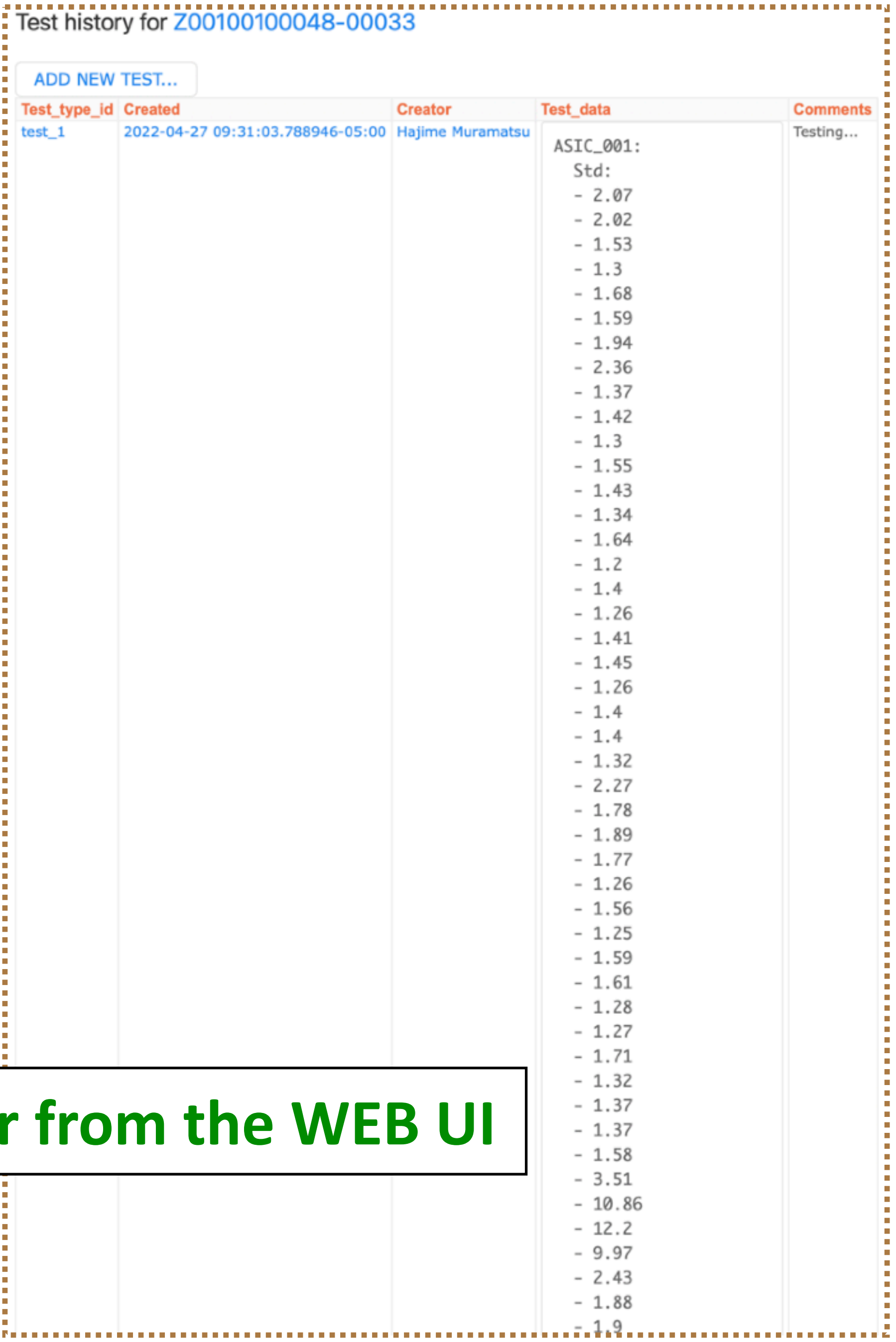
- The API endpoint : /api/components/<eid>/tests/test_type_name
- An example of actual line:

  CURL 'APIPATH/components/Z00100100048-00033/tests/test_1'

… it's there..



```
{"component":{"id":6434,"part_id":"Z00100100048-00033"},"data":[{"comments":"Testing...","created":"2022-04-27T09:3
1:03.788946-05:00","creator":"Hajime Muramatsu","methods":[{"href":"/cdbdev/api/component-tests/3598/images","rel":
"Images"}],"test_data":{"ASIC_001":{"Chan":[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,
27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63],"Ch
anName":["ch00","ch01","ch02","ch03","ch04","ch05","ch06","ch07","ch08","ch09","ch10","ch11","ch12","ch13","ch14","
ch15","ch16","ch17","ch18","ch19","ch20","ch21","ch22","ch23","ch24","ch25","ch26","ch27","ch28","ch29","ch30","ch3
1","ch32","ch33","ch34","ch35","ch36","ch37","ch38","ch39","ch40","ch41","ch42","ch43","ch44","ch45","ch46","ch47",
"ch48","ch49","ch50","ch51","ch52","ch53","ch54","ch55","ch56","ch57","ch58","ch59","ch60","ch61","ch62","ch63"],"C
hipSN":"1L10451","Mean":[18.01,17.12,16.22,13.96,16.97,13.51,15.38,17.19,10.94,12.22,10.5,13.57,14.26,15.21,15.68,1
6.35,15.01,15.07,13.7,14.83,18.45,17.76,16.77,17.59,17.61,17.45,15.74,18.77,15.86,16.44,13.12,17.22,16.25,20.14,15.
16,13.82,13.41,16.79,14.7,18.95,17.0,23.37,254.1,24.29,15.01,15.07,14.44,13.32,15.34,16.39,15.26,13.21,16.64,16.04,
13.76,16.02,18.37,16.73,16.18,16.14,15.56,18.84,17.73,15.41],"Nent":[1211,1220,1210,1220,1223,1178,1216,1163,1232,1
208,1255,1195,1302,1602,1334,1268,1224,1200,1165,1190,1231,1209,1193,1234,1230,1209,1790,1274,1198,1223,1251,1184,1
204,1177,1192,1165,1284,1223,1302,1720,1423,1211,1236,1189,1223,1227,1171,1250,1248,1238,1256,1196,1234,1896,1339,1
213,1240,1256,1186,1275,1206,1191,1203,1163],"Std":[2.07,2.02,1.53,1.3,1.68,1.59,1.94,2.36,1.37,1.42,1.3,1.55,1.43,
1.34,1.64,1.2,1.4,1.26,1.41,1.45,1.26,1.4,1.4,1.32,2.27,1.78,1.89,1.77,1.26,1.56,1.25,1.59,1.61,1.28,1.27,1.71,1.32
,1.37,1.37,1.58,3.51,10.86,12.2,9.97,2.43,1.88,1.9,1.52,1.69,1.24,1.47,1.25,1.71,1.23,1.42,1.2,1.84,1.42,1.28,1.3,1
.54,1.25,1.72,1.54],"io_channel":[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],"io_group":[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],"runtime":[1642723507.6522949,1642723507.
6522949,1642723507.6522949,1642723507.6522949,1642723507.6522949,1642723507.6522949,1642723507.6522949,1642723507.6
522949,1642723507.6522949,1642723507.6522949,1642723507.6522949,1642723507.6522949,1642723507.6522949,1642723507.65
22949,1642723507.6522949,1642723507.6522949,1642723507.6522949,1642723507.6522949,1642723507.6522949,1642723507.652
2949,1642723507.6522949,1642723507.6522949,1642723507.6522949,1642723507.6522949,1642723507.6522949,1642723507.6522
949,1642723507.6522949,1642723507.6522949,1642723507.6522949,1642723507.6522949,1642723507.6522949,1642723507.65229
49,1642723507.6522949,1642723507.6522949,1642723507.6522949,1642723507.6522949,1642723507.6522949,1642723507.652294
```



Test history for Z00100100048-00033

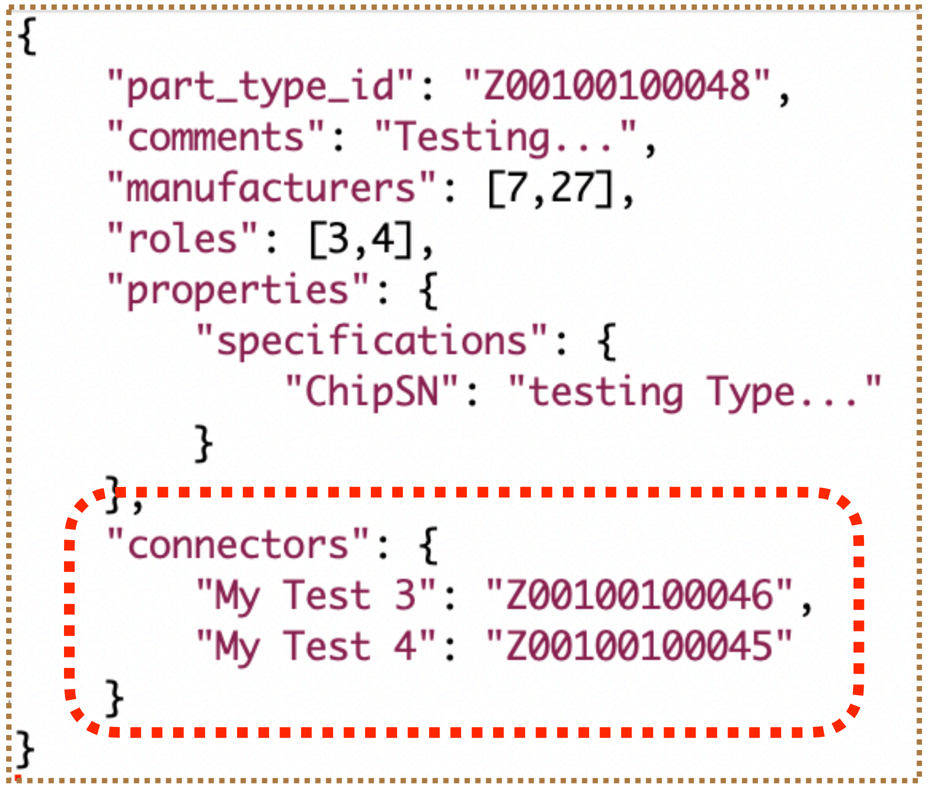
| Test_type_id | Created | Creator | Test_data | Comments |
|---|---|---|---|---|
| test_1 | 2022-04-27 09:31:03.788946-05:00 | Hajime Muramatsu | ASIC_001: Std: - 2.07 - 2.02 - 1.53 - 1.3 - 1.68 - 1.59 - 1.94 - 2.36 - 1.37 - 1.42 - 1.3 - 1.55 - 1.43 - 1.34 - 1.64 - 1.2 - 1.4 - 1.26 - 1.41 - 1.45 - 1.26 - 1.4 - 1.4 - 1.32 - 2.27 - 1.78 - 1.89 - 1.77 - 1.26 - 1.56 - 1.25 - 1.59 - 1.61 - 1.28 - 1.27 - 1.71 - 1.32 - 1.37 - 1.37 - 1.58 - 3.51 - 10.86 - 12.2 - 9.97 - 2.43 - 1.88 - 1.9 | Testing... |

or from the WEB UI

## Connecting sub-Components

- Suppose (from DAY1), you have already defined to have sub-Component Types, Z00100100046 (Test_Parts_3) and Z00100100045 (Test_Parts_4) in your Component Type definition, Z00100100048 (Test_Parts_1).

- The JSON file for its Type definition looks like this:



```
{
    "part_type_id": "Z00100100048",
    "comments": "Testing...",
    "manufacturers": [7,27],
    "roles": [3,4],
    "properties": {
        "specifications": {
            "ChipSN": "testing Type..."
        }
    },
    "connectors": {
        "My Test 3": "Z00100100046",
        "My Test 4": "Z00100100045"
    }
}
```

# Linking Items (POST)

- There are two ways to do this.
- One way is to make the links as you post a new Item.
  In this case, the procedure is identical to what page 8 shows.

- The API endpoint : /api/component-types/<type_id>/components
- An example of actual line:

```
CURL -H "Content-Type: application/json" -X POST -d @Add_AnItem_Test_Parts_1-2.json
'APIPATH/component-types/Z00100100048/components'
```

- The JSON file looks like this:
  - You just need to add the "subcomponents" blob.

- Be careful that you can NOT link Items that have been already linked to other Items.

```
{
    "component_type": {
        "part_type_id": "Z00100100048"
    },
    "country_code": "US",
    "comments": "Testing...",
    "institution": {
        "id": 186
    },
    "manufacturer": {
        "id": 7
    },
    "specifications": {
      "ChipSN": "testing"
    },
    "subcomponents": {
        "My Test 3": "Z00100100046-00001",
        "My Test 4": "Z00100100045-00004"
    }
}
```

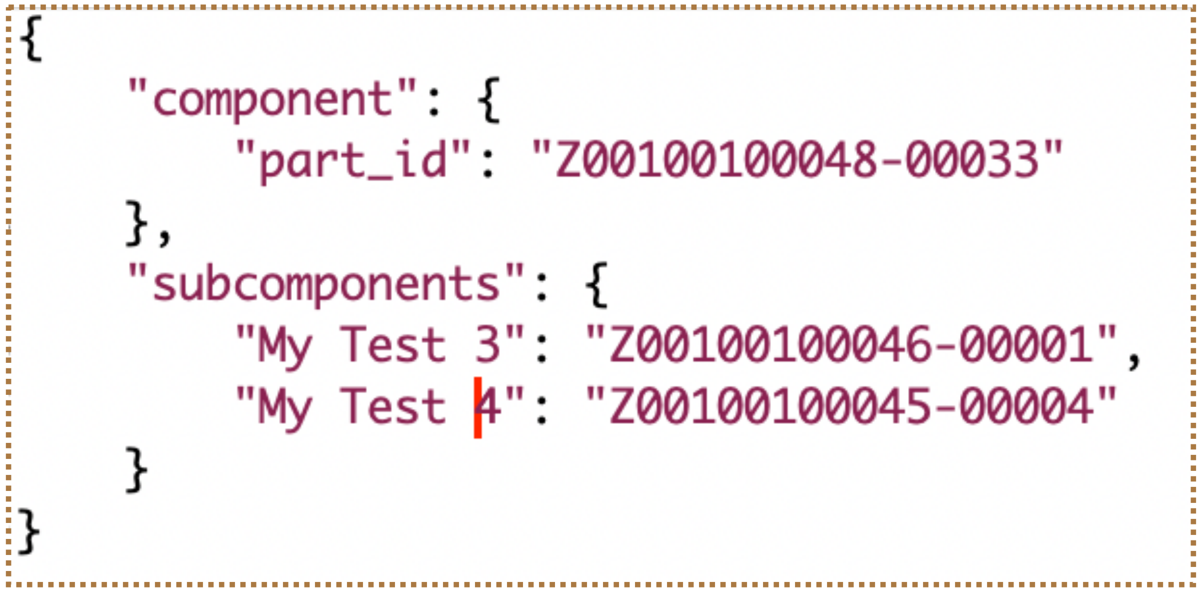
## Linking Items (PATCH)

- The other way is to PATCH an existing Item entry and add links.

- The API endpoint : /api/components/<eid>/subcomponents
- An example of actual line:

```
CURL -H "Content-Type: application/json" -X PATCH -d @Patch_AnItem_Test_Parts_1.json
'APIPATH/components/Z00100100048-00033/subcomponents'
```

- To PATCH an Item, Z00100100048-00033 and add links to Z00100100046-00001 and Z00100100045-00004, my JSON file would look like this:



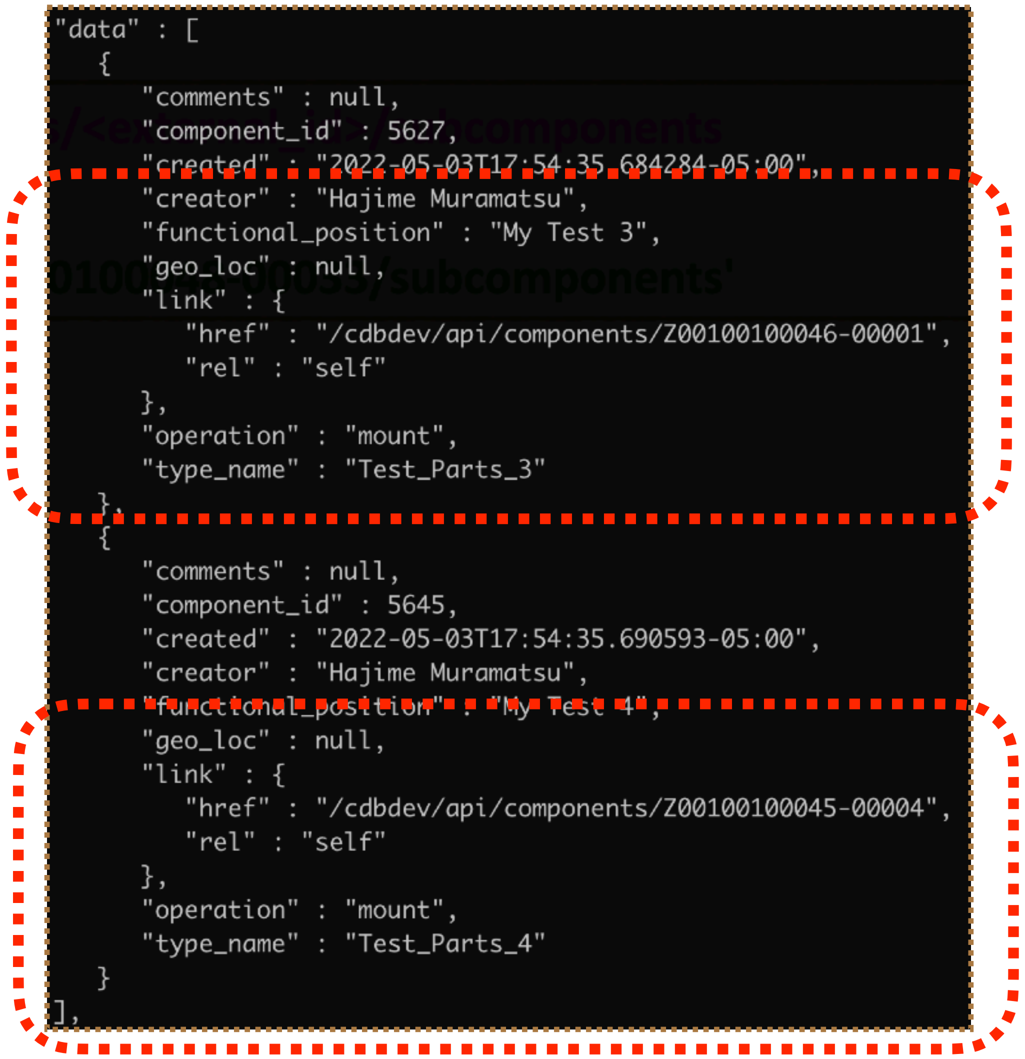
```
{
    "component": {
        "part_id": "Z00100100048-00033"
    },
    "subcomponents": {
        "My Test 3": "Z00100100046-00001",
        "My Test 4": "Z00100100045-00004"
    }
}
```

# Checking sub-Components (GET)

- And we can get the sub-Component information as well.

- The API endpoint : /api/components/<eid>/subcomponents
- An example of actual line:

  CURL 'APIPATH/components/Z00100100048-00033/subcomponents'



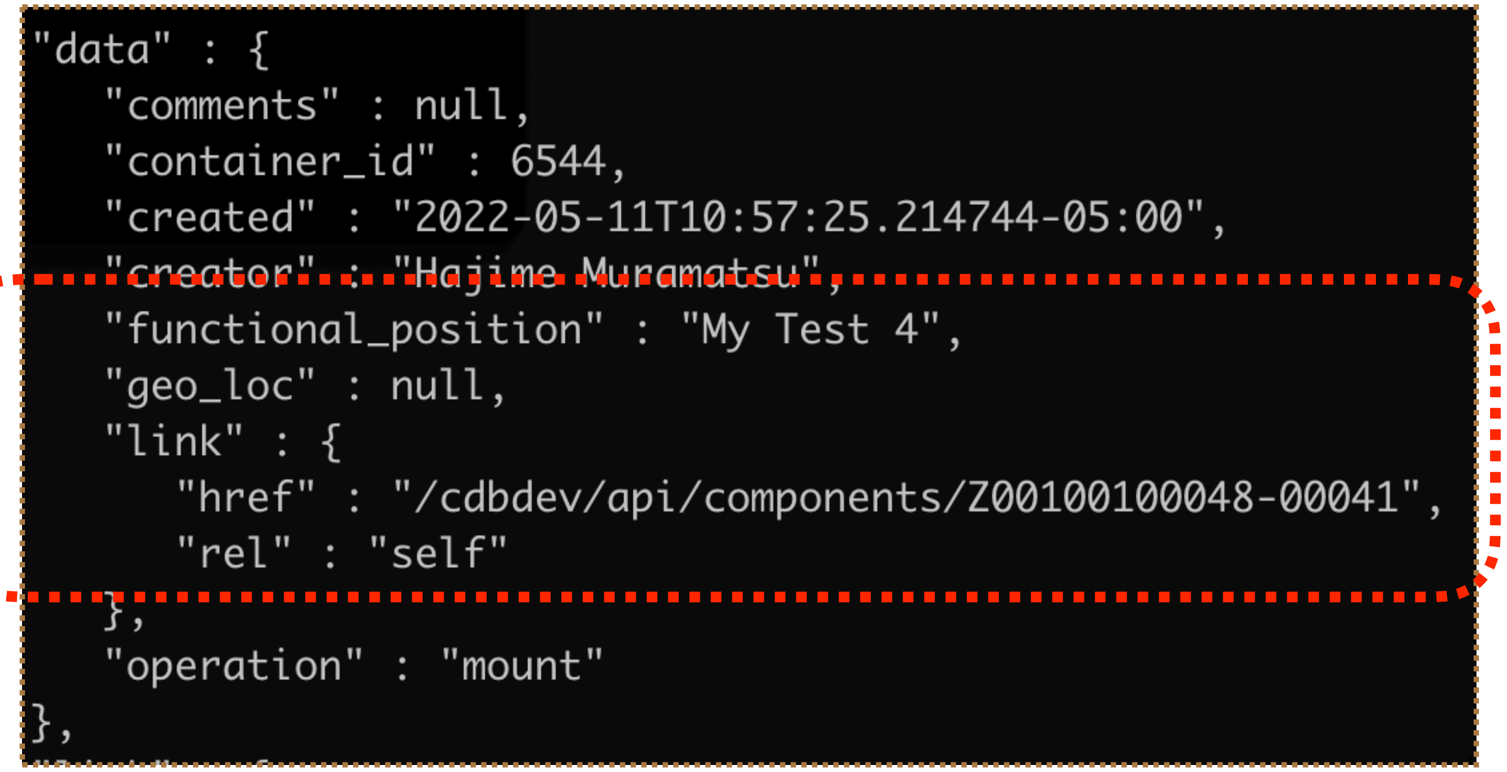
```
"data" : [
   {
      "comments" : null,
      "component_id" : 5627,
      "created" : "2022-05-03T17:54:35.684284-05:00",
      "creator" : "Hajime Muramatsu",
      "functional_position" : "My Test 3",
      "geo_loc" : null,
      "link" : {
         "href" : "/cdbdev/api/components/Z00100100046-00001",
         "rel" : "self"
      },
      "operation" : "mount",
      "type_name" : "Test_Parts_3"
   },
   {
      "comments" : null,
      "component_id" : 5645,
      "created" : "2022-05-03T17:54:35.690593-05:00",
      "creator" : "Hajime Muramatsu",
      "functional_position" : "My Test 4",
      "geo_loc" : null,
      "link" : {
         "href" : "/cdbdev/api/components/Z00100100045-00004",
         "rel" : "self"
      },
      "operation" : "mount",
      "type_name" : "Test_Parts_4"
   }
],
```

## Checking sub-Components (GET)

- can also look at one of the daughter Items and see which Item it is contained.

- The API endpoint : /api/components/<eid>/container
- An example of actual line:

  CURL 'APIPATH/components/Z00100100045-00004/container'



```
"data" : {
   "comments" : null,
   "container_id" : 6544,
   "created" : "2022-05-11T10:57:25.214744-05:00",
   "creator" : "Hajime Muramatsu",
   "functional_position" : "My Test 4",
   "geo_loc" : null,
   "link" : {
      "href" : "/cdbdev/api/components/Z00100100048-00041",
      "rel" : "self"
   },
   "operation" : "mount"
},
```

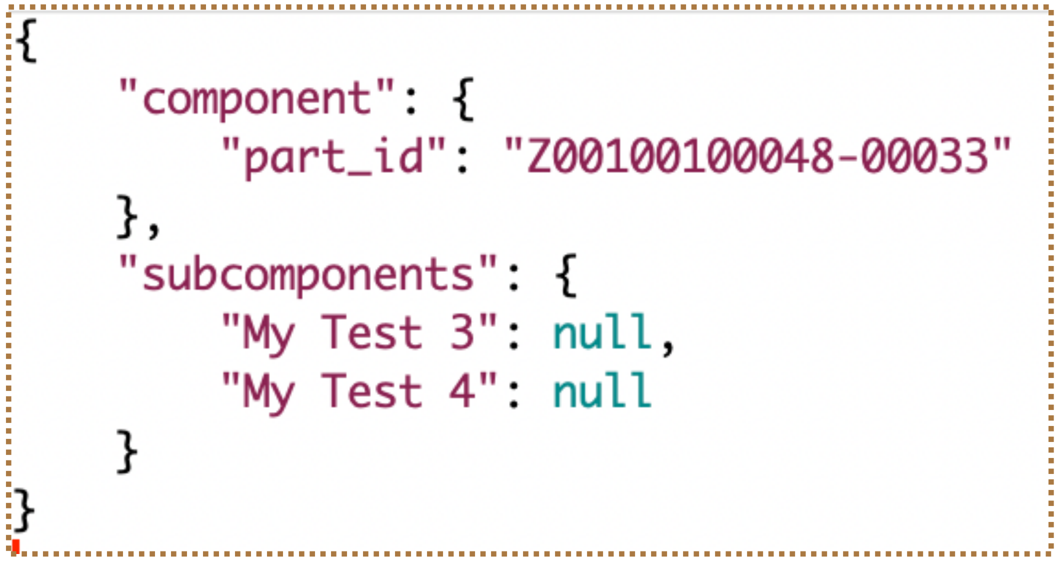
## Clearing sub-Components (PATCH)

- **And we can UNDO the links.**

- **The API endpoint : /api/components/<eid>/subcomponents**
- **An example of actual line:**

**CURL -H "Content-Type: application/json" -X PATCH -d @Patch_AnItem_Test_Parts_1_clean.json 'APIPATH/components/Z00100100048-00033/subcomponents'**

- **Here is a JSON file to remove sub-Components:**



```
{
    "component": {
        "part_id": "Z00100100048-00033"
    },
    "subcomponents": {
        "My Test 3": null,
        "My Test 4": null
    }
}
```

# Downloading Bar/QR codes (GET)

- The API endpoint : /api/get-barcode/<pid>
  /api/get-qrcode/<pid>
- An example of actual line:

```
CURL 'APIPATH/get-barcode/Z00100100048-00033-US186' --output test.png
```

- Not much to say here…
  except that the QR code provides a hyperlink to the corresponding Item page.
  (might become very handy to scan them with your smart phones)

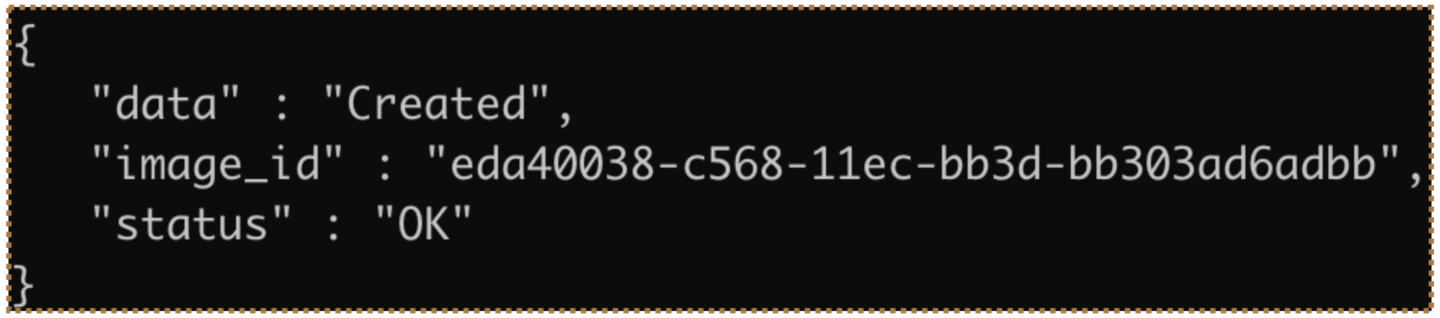
Z00100100048-00033-US186

# POSTing an image file

- The API endpoint : /api/components/<eid>/images
- An example of actual line:

  CURL -H "comments=testing from curl" -F "image=@10-0_10-1-6k_small.pdf" 'APIPATH/components/Z00100100048-00033/images'

- When executed, you should see a response like this;



```
{
   "data" : "Created",
   "image_id" : "eda40038-c568-11ec-bb3d-bb303ad6adbb",
   "status" : "OK"
}
```

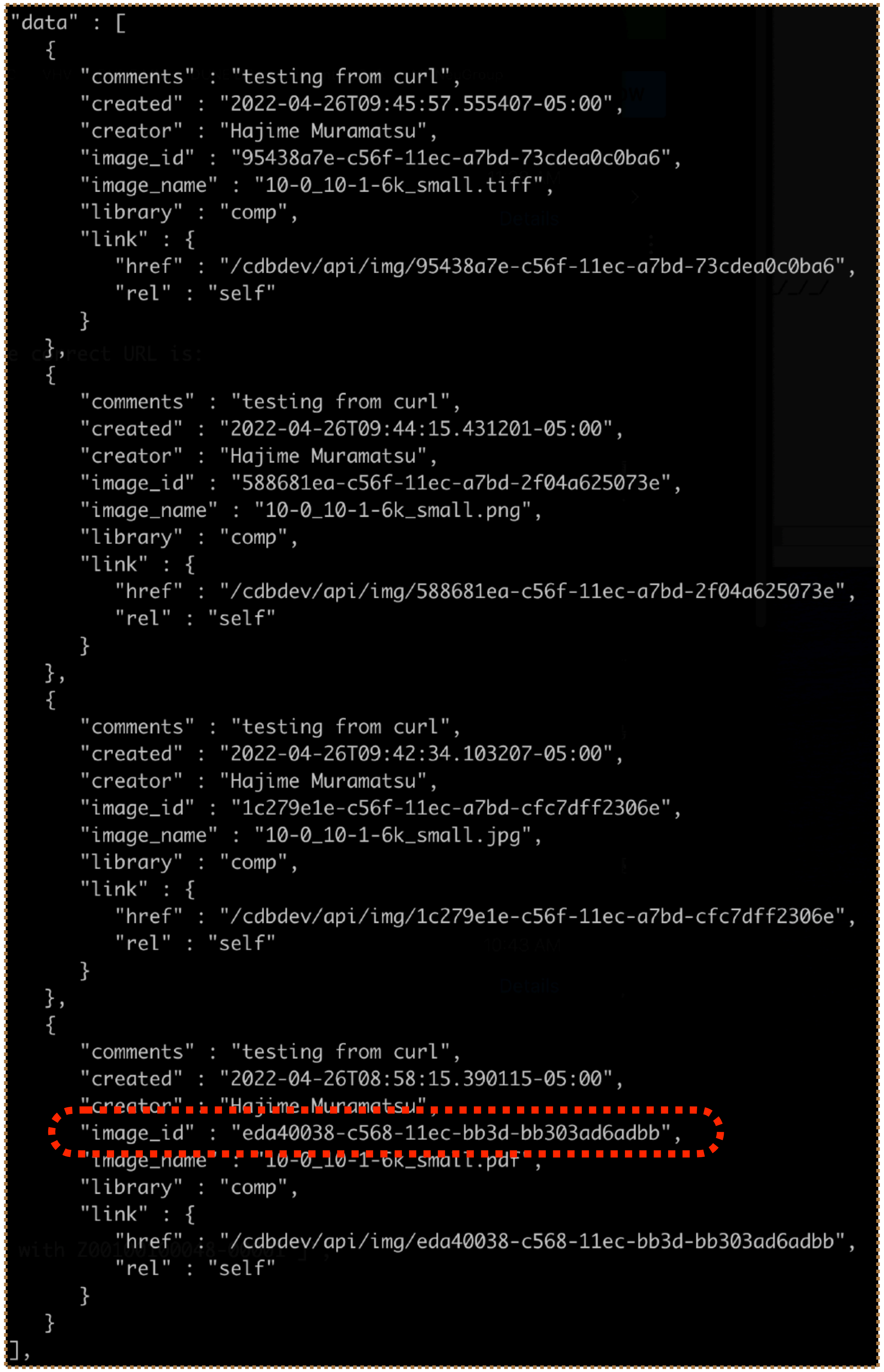
- Let's check that to see if it is there:

  The API endpoint : /api/components/<eid>/images

- An example of actual line:

  CURL 'APIPATH/components/Z00100100048-00033/images'

  See the respond on the next page…



```
"data" : [
  {
    "comments" : "testing from curl",
    "created" : "2022-04-26T09:45:57.555407-05:00",
    "creator" : "Hajime Muramatsu",
    "image_id" : "95438a7e-c56f-11ec-a7bd-73cdea0c0ba6",
    "image_name" : "10-0_10-1-6k_small.tiff",
    "library" : "comp",
    "link" : {
      "href" : "/cdbdev/api/img/95438a7e-c56f-11ec-a7bd-73cdea0c0ba6",
      "rel" : "self"
    }
  },
  {
    "comments" : "testing from curl",
    "created" : "2022-04-26T09:44:15.431201-05:00",
    "creator" : "Hajime Muramatsu",
    "image_id" : "588681ea-c56f-11ec-a7bd-2f04a625073e",
    "image_name" : "10-0_10-1-6k_small.png",
    "library" : "comp",
    "link" : {
      "href" : "/cdbdev/api/img/588681ea-c56f-11ec-a7bd-2f04a625073e",
      "rel" : "self"
    }
  },
  {
    "comments" : "testing from curl",
    "created" : "2022-04-26T09:42:34.103207-05:00",
    "creator" : "Hajime Muramatsu",
    "image_id" : "1c279e1e-c56f-11ec-a7bd-cfc7dff2306e",
    "image_name" : "10-0_10-1-6k_small.jpg",
    "library" : "comp",
    "link" : {
      "href" : "/cdbdev/api/img/1c279e1e-c56f-11ec-a7bd-cfc7dff2306e",
      "rel" : "self"
    }
  },
  {
    "comments" : "testing from curl",
    "created" : "2022-04-26T08:58:15.390115-05:00",
    "creator" : "Hajime Muramatsu",
    "image_id" : "eda40038-c568-11ec-bb3d-bb303ad6adbb",
    "image_name" : "10-0_10-1-6k_small.pdf",
    "library" : "comp",
    "link" : {
      "href" : "/cdbdev/api/img/eda40038-c568-11ec-bb3d-bb303ad6adbb",
      "rel" : "self"
    }
  }
],
```

- **When multiple images exist, it shows all of them, with the latest being on the top.**

- **Accepted formats:**
  - **jpeg**
  - **tiff**
  - **pdf**
  - **bmp**
  - **png**
  - **…**

- **Let's download one of them. Next page…**

## Downloading an image file

- The API endpoint : /api/img/<image_id>
- An example of actual line:

```
CURL 'APIPATH/img/eda40038-c568-11ec-bb3d-bb303ad6adbb' -o myimage.pdf
```

# List of API endpoints covered in tutorials

- PATCH/GET a Component Type : /api/component-types/<type_id>
- POST/GET an Item(s) : /api/component-types/<type_id>/components
- GET an Item : /api/components/<eid>
- POST Items : /api/component-types/<type_id>/bulk-add
- PATCH/GET a sub-component(s) : /api/components/<eid>/subcomponents
- GET a parent-component : /api/components/<eid>/container
- POST/GET a Test Type(s) : /api/component-types/<type_id>/test-types
- POST a Test : /api/components/<eid>/tests
- GET a Test : /api/components/<eid>/tests/<test_type_name>
- GET a bar-code : /api/get-barcode/<oid>
- GET a QR-code : /api/get-qrcode/<oid>
- POST/GET an Image/info : /api/components/<eid>/images
- GET an Image : /api/img/<image_id>

## Now, can you do these?

- POST an Item with and without sub-Components specified.
  (you need to define your Component Type accordingly first)
- GET the posted Item.
- POST multiple Items at once.
- GET a list of the all entered Items for a certain Component Type.
- POST a Test Type for a certain Component Type.
- POST a Test result using the Test Type you just POSTed.
- GET the Test result you just POSTed.
- PATCH an Item to add sub-Components.
- GET info of the sub-Components you just added.
- GET info of its parent Component.
- PATCH an Item to remove the sub-Components you just added.
- GET bar- & QR- codes of an Item.
- POST an image.
- GET the image you just POSTed.