# List of API endpoints covered in this session

- Pages 5/8/9/32 : POST/GET an Item(s) : /component-types/<type_id>/components
- Page 6 : GET an Item : /components/<eid>
- Page 7 : POST Items : /component-types/<type_id>/bulk-add
- Pages 10-16 : Filter Items : /components?
- Page 20 : POST a Test : /components/<eid>/tests
- Page 21 : GET a Test : /components/<eid>/tests/<test_type_name>
- Page 27 : GET a Status : /components/<eid>/Status
- Page 28 : PATCH an Item : /components/<eid>/Status
- Pages 29/30 : PATCH Items : /components/bulk-enable
- Pages 33/34/36 : PATCH/GET a sub-component(s) : /components/<eid>/subcomponents
- Page 35 : GET a parent-component : /components/<eid>/container
- Pages 37/38 : POST/GET a location : /components/<eid>/locations
- Page 39 : GET a bar-code : /get-barcode/<pid>
- Page 39 : GET a QR-code : /get-qrcode/<pid>
- Page 41 : POST/GET Image/info for a Component Type : /component-types/<type_id>/images
- Page 43 : GET an Image : /img/<image_id>
- Pages 44 : POST/GET Image/info for an Item : /components/<eid>/images
- Pages 46-48 : POST/GET Image/info for a Test entry : /component-tests/<oid>/images

# The two Databases

## Development version and Production version

- **Production version : https://dbweb0.fnal.gov/cdb/login/sso**
- **Development version : https://dbweb0.fnal.gov/cdbdev/login/sso**

**You can access to both databases easily through your web browsers**

**Today we'll use the development version,**

**but the usage of the two databases are identical.**

# REST API

- **Many command lines shown in this subsection start with the following.**

  **curl --cert Output.pem --pass YourPhrase 'https://dbwebapi2.fnal.gov:8443/cdbdev/api/version/…'**

  **Here,**

  - **Output.pem and YourPhrase are your certificate and phrase that are obtained/setup in the HWDB Training site, respectively.**
  - **version is a version of the REST API.**

    **As of March 11th, 2024, the latest version is v1.**

- **Since they'll show up repeatedly, we will abbreviate them in the following way:**
  - **curl --cert-type P12 --cert MyCert.p12:myPSWD → CURL**
  - **https://dbwebapi2.fnal.gov:8443/cdbdev/api/v1/ → APIPATH**

    **(https://dbwebapi2.fnal.gov:8443/cdb/api/v1/ for the production version)**

# From the 1st session this morning…

- **We took a Component Type "Front Axle" as an example to complete its Type definition by PATCH-ing.**
- **That Type was defined with this JSON:**


```
{
   "part_type_id": "Z00100400005",
   "comments": "Setting up this Type",
   "manufacturers": [7,27],
   "roles": [3,4],
   "properties": {
      "specifications": {
         "Last name": "Muramatsu",
         "First name": "Hajime"
      }
   },
   "connectors": {
      "My L Wheel": "Z00100400007",
      "My R Wheel": "Z00100400008"
   }
}
```


- **Now let's insert an Item for this Type.**
- **The information we need are:**
  - **Its Type ID : Z00100400005**
  - **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, with the two Keys.**
  - **There are two Component Types that are allowed to be linked:**

    **Z00100400007 and Z00100400008. (we will deal with sub-Components later in this talk)**

# POST an Item

- The API endpoint : /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/Z00100400005/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 Z00100400005
- The JSON file looks like this:
  - MUST specify part_type_id
  - 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": "Z00100400005"
  },
  "country_code": "US",
  "comments": "Testing...",
  "institution": {
    "id": 186
  },
  "manufacturer": {
    "id": 7
  },
  "specifications": {
      "Last name" : "Muramatsu",
      "First name" : "Hajime"
  }
}
```

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

```
{
  "component_id": 153639,
  "data": "Created",
  "part_id": "Z00100400005-00014",
  "status": "OK"
}
```

**A PID, Z00100400005-00001, has been assigned!**

# GET an Item

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

  CURL 'APIPATH/components/Z00100400005-00001'

Again, here is the “data” blob. It’s there..

We see;

- component type id & name
- country Code
- institution id
- manufacturer
- serial number (empty)
- location (not defined, yet)
- specifications
- part_id (= the generated pid)


```
"data": {
  "batch": {
   "id": 3,
   "received": "2020-05-15"
  },
  "comments": "",
  "component_id": 46435,
  "component_type": {
   "name": "Front Axle",
   "part_type_id": "Z00100400005"
  },
  "country_code": "US",
  "created": "2023-09-22T10:50:55.099050-05:00",
  "creator": {
   "id": 12624,
   "name": "Hajime Muramatsu",
   "username": "hajime3"
  },
  "institution": {
   "id": 186,
   "name": "University of Minnesota Twin Cities"
  },
  "location": null,
  "manufacturer": {
   "id": 7,
   "name": "Hajime Inc"
  },
  "part_id": "Z00100400005-00001",
  "serial_number": "",
  "specifications": [
   {
    "First name": "Hajime.",
    "Last name": "Muramatsu"
   }
  ],
  "specs_version": 4,
  "status": {
   "id": 2,
   "name": "temporarily not available"
  }
},
```

# POST multiple Items at once

- The API endpoint : /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/Z00100400005/bulk-add'

- Again, in this example, the Component Type ID is Z00100400005
- The JSON file looks like this:

  Specify how many Items you want to insert by "count".

```
{
  "component_type": {
      "part_type_id": "Z00100400005"
  },
  "country_code": "US",
  "institution": {
      "id": 186
  },
  "manufacturer": {
      "id": 7
  },
  "count": 2
}
```

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

```
{
 "data": [
  {
   "link": {
    "href": "/cdbdev/api/v1/components/Z00100400005-00015",
    "rel": "self"
   },
   "part_id": "Z00100400005-00015"
  },
  {
   "link": {
    "href": "/cdbdev/api/v1/components/Z00100400005-00016",
    "rel": "self"
   },
   "part_id": "Z00100400005-00016"
  }
 ],
 "status": "OK"
}
```

**Two eids, Z00100400005-00002 and Z00100400005-00003, have been assigned!**

# GET a list of Items

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

  CURL 'APIPATH/component-types/Z00100400005/components'

- Shows a list of all entered Items for this Type (Front Axle).
- It shows up to 100 Items at once.
- When more than 100 Items, managed by Pagination.

  I.e., for Type ID = Z00100300030;


```
"pagination": {
  "next": "/cdbdev/api/v1/component-types/Z00100300030/components?page=2&size=100",
  "page": 1,
  "page_size": 100,
  "pages": 2,
  "prev": null,
  "total": 182
 },
```



```
{
 "component_type": {
  "name": "Front Axle",
  "part_type_id": "Z00100400005"
 },
 "data": [
  {
   "comments": "",
   "component_id": 46435,
   "created": "2023-09-22T10:50:55.099050-05:00",
   "creator": {
    "id": 12624,
    "name": "Hajime Muramatsu",
    "username": "hajime3"
   },
   "link": {
    "href": "/cdbdev/api/v1/components/Z00100400005-00001",
    "rel": "self"
   },
   "location": "",
   "part_id": "Z00100400005-00001",
   "serial_number": "",
   "specifications": [
    {
     "First name": "Hajime.",
     "Last name": "Muramatsu"
    }
   ],
   "status": {
    "id": 2,
    "name": "temporarily not available"
   }
  },
  {
   "comments": null,
   "component_id": 46436,
   "created": "2023-09-22T11:07:46.576790-05:00",
   "creator": {
    "id": 12624,
    "name": "Hajime Muramatsu",
    "username": "hajime3"
```


- To access to a different page;

  CURL 'APIPATH/component-types/Z00100300030/components?page=4'

# Filtering a list of Items for a given Component Type ID

## - /component-types/<Component Type ID>/components? -

- Perhaps, the simple is to pipe the output into something like jq… to dump the PIDs only.

  CURL 'APIPATH/component-types/Z00100400005/components' | jq .data[].part_id

- But we can also do this with the HWDB filtering functionality;

  CURL 'APIPATH/component-types/Z00100400005/components?page=1&size=10&fields=specifications,part_id' | jq

  to dump only 2 fields, Specifications and PID.

- One can search for entries with “serial_number” with a WILD CARD:

  CURL 'APIPATH/component-types/Z00100200040/components?serial_number=SMB%'

  This will dump all entries with “serial_ number = SMB*”.

  Postgres style wildcards characters '%' and '_' can be used.

  E.g., if you want to dump only “serial_number=SMB***98”:

  CURL 'APIPATH/component-types/Z00100200040/components?serial_number=SMB___98%'

# Filtering a list of Items - 1

## - /components? -

- **Filter for a certain Manufacturer (case-insensitive);**

  CURL 'APIPATH/components?manufacturer=hajime%20inc' | jq .data[].part_id

- **Filter for a certain Number in Item Specifications;**

  CURL 'APIPATH/components?specs=SiPM_Strip_ID==4548' | jq .data[].part_id

**The right hand side here happens to be numbers.**
**But it also works for strings (case-sensitive).**
**In fact… see the next page.**


```
"specifications": [
  {
    "Documentation": "https://something.com/whatever",
    "SiPM_Strip_ID": 4548,
    "Test_Box_ID": "M1b3",
    "Tray_Number": 20,
    "Vendor_Box_Number": 14,
    "Vendor_Delivery_ID": "HPK_Ciemat_03",
    "_meta": {
      "_column_order": [
        "Vendor_Delivery_ID",
        "Vendor_Box_Number",
        "Tray_Number",
        "SiPM_Strip_ID",
        "Test_Box_ID",
        "Documentation"
      ]
    }
  }
],
```

# Filtering a list of Items - 2

## - /components? -

- For numbers:
  - ==
  - !=
  - <
  - <=
  - >
  - >=
- For strings/substrings:
  - ==
  - !=
  - ~ (regex search : case sensitive) : See later pages
  - ~* (regex search : case insensitive) : See later pages

# Filtering a list of Items - 3

## - /components? - regex -

- Suppose I have the following in one of the Items (D00400300001-00340):


```
"specifications": [
  {
    "Documentation": "https://something.com/whatever",
    "SiPM_Strip_ID": 4548,
    "Test_Box_ID": "Mib3",
    "Tray_Number": 20,
    "Vendor_Box_Number": 14,
    "Vendor_Delivery_ID": "HPK_Ciemat_03",
    "_meta": {
      "_column_order": [
        "Vendor_Delivery_ID",
        "Vendor_Box_Number",
        "Tray_Number",
        "SiPM_Strip_ID",
        "Test_Box_ID",
        "Documentation"
      ]
    }
  }
],
```


- case sensitive : Vendor_Delivery_ID~HPK_Ciemat_
- case sensitive : Vendor_Delivery_ID~HPK_CiemaT_ : Nothing returned.
- case insensitive : Vendor_Delivery_ID~*HPK_CiemaT_

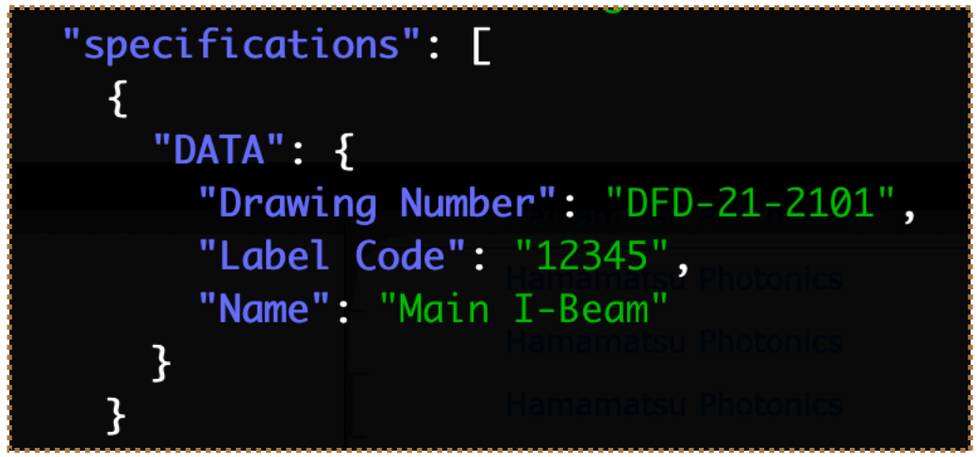
- E.g.,

  CURL 'APIPATH/components?specs=Vendor_Delivery_ID~HPK_Ciemat_' | jq .data[].part_id

# Filtering a list of Items - 4

## - /components? - nested structure -

**- Suppose I have the following in one of the Items (D00599800001-00379):**




**You can still filter it simply via;**

**Name==Main I-Beam**

**or**

**Label Code==12345**

**- E.g.,**

**CURL 'APIPATH/components?specs=Name==Main%20I-Beam' | jq .data[].part_id**

# Filtering a list of Items - 5

## - /components? - test data -

**We can apply a filter to Test Data in the same way!**

**- Suppose I have the following in one of the Items (D00400300001-00380):**


```
"test_data": {
  "Test Results": [
    {
      "Date": "01-26-2024-06:13 UTC",
      "Location": "Milano Bicocca",
      "Operator": "Andrea Falcone; Claudia Brizzolari; Francesco Terranova; Luca Meazza; Maritza Delgado; Maurizio Perego",
      "Polarization": "Rev",
      "SiPM": [
        {
          "Comment": "",
          "I": [
            1.2604860486048603e-06,
            7.759315931593157e-07,
            1.2117011701170138e-07,
            8.730783078307831e-07,
            1.4496849684968533e-07,
            4.010081008100806e-07,
            5.816921692169214e-07,
            2.153726372637264e-06,
            1.612241224122412e-06,
            -6.832583258325834e-07
```


**The followings also work:**

- **I[0]==1.2604860486048603e-06 : the 1st index**
- **I[0]>1.260486e-06 : the 1st index**
- **I[1]==7.759315931593157e-07 : the 2nd index**
- **I[*]==1.2117011701170138e-07 : Any of them**

**Here, the right-hand sides happen to be numbers. But this works for strings as well.**

# Filtering a list of Items - 6

## - /components? - test data -

- E.g.,

  CURL 'APIPATH/components?testdata=I[0]==1.2604860486048603e-06' | jq .data[].part_id

  If your terminal (shell) complains, try;

  CURL 'APIPATH/components?testdata=I\[0\]==1.2604860486048603e-06' | jq .data[].part_id

# Summary of
# Filtering Items through the REST API

- /api/v1/component-types/<Component Type ID>/components?<XXX>
- /api/v1/components?<XXX>

- <XXX> can be:
  - page
  - size
  - fields (comma-separated list)
  - part_id
  - serial_number
  - part_type_id
  - manufacturer
  - creator
  - comments
  - JSON attribute (e.g., Name==Main I-Beam)
  - country_of_origin
  - resp_institution

- and multiple <XXX> can be combined with “&”.
- Again, Postgres wildcard characters % and _ can be used as well.

# POST 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 : DONE in the previous session
    (but unlike Component Type, there can be
    multiple Test Types for a given Component Type)
  - Post a Test.

from previous session

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

- The API endpoint : /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_10",
  "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_10",
  "status" : "OK",
  "test_type_id" : 599
}
```

A new Test Type, Test_Parts_3_TestType_10, has been created.

from previous session

# Checking Test Types (GET)

- The API endpoint : /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.

- It's a long list…

  3 Test Types can be seen here,

  Test_Parts_3_TestType_10,

  My QC check 2", and

  CPAFC Cryo Pos QC check.


```
"data" : [
 {
   "comments" : "Testing...",
   "created" : "2024-03-11T05:24:51.211047-05:00",
   "creator" : {
     "id" : 12624,
     "name" : "Hajime Muramatsu",
     "username" : "hajime3"
   },
   "id" : 599,
   "link" : {
     "href" : "/cdbdev/api/v1/component-test-types/599",
     "rel" : "self"
   },
   "name" : "Test_Parts_3_TestType_10"
 },
 {
   "comments" : "",
   "created" : "2023-07-30T20:13:55.920030-05:00",
   "creator" : {
     "id" : 12624,
     "name" : "Hajime Muramatsu",
     "username" : "hajime3"
   },
   "id" : 477,
   "link" : {
     "href" : "/cdbdev/api/v1/component-test-types/477",
     "rel" : "self"
   },
   "name" : "My QC check 2"
 },
 {
   "comments" : "Setting up Test Type",
   "created" : "2023-05-22T12:39:44.212142-05:00",
   "creator" : {
     "id" : 12624,
     "name" : "Hajime Muramatsu",
     "username" : "hajime3"
   },
   "id" : 464,
   "link" : {
     "href" : "/cdbdev/api/v1/component-test-types/464",
     "rel" : "self"
   },
   "name" : "CPAFC Cryo Pos QC check"
 },
```

# POST test results

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

  CURL -H "Content-Type: application/json" -X POST -d @Post_TestResult_Test_parts_10.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_10",
  "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_id": 17813,
 "test_type_id": 599
}
```

# Check test results (GET)

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

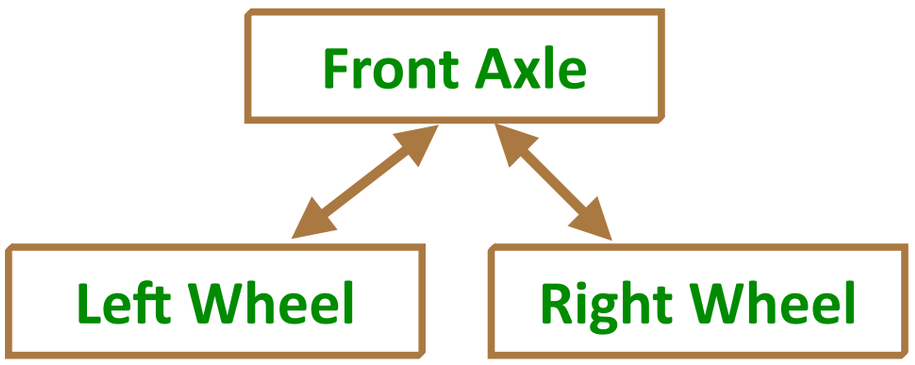
  CURL 'APIPATH/components/Z00100100046-00008/tests/Test_Parts_3_TestType_10'

- They are in the data blob.


```
"data": [
  {
   "comments": "All look ok",
   "created": "2024-05-22T09:43:37.049580-05:00",
   "creator": {
    "id": 12624,
    "name": "Hajime Muramatsu",
    "username": "hajime3"
   },
   "id": 17813,
   "images": [],
   "link": {
    "href": "/cdbdev/api/v1/component-tests/17813",
    "rel": "details"
   },
   "methods": [
    {
     "href": "/cdbdev/api/v1/component-tests/17813/images",
     "rel": "Images"
    }
   ],
   "test_data": {
    "Cleaned": 1,
    "Template": 1,
    "Visual": 1
   },
   "test_spec_version": -1,
   "test_type": {
    "id": 599,
    "name": "Test_Parts_3_TestType_10"
   }
  }
],
```

# Connecting sub-Components

- In the previous session (see the next few slides),
  you have already defined to have sub-Component Types,
  Z00100400007 (Left Wheel) and Z00100400008 (Right Wheel)
  in your Component Type definition, Z00100400005 (Front Axle).



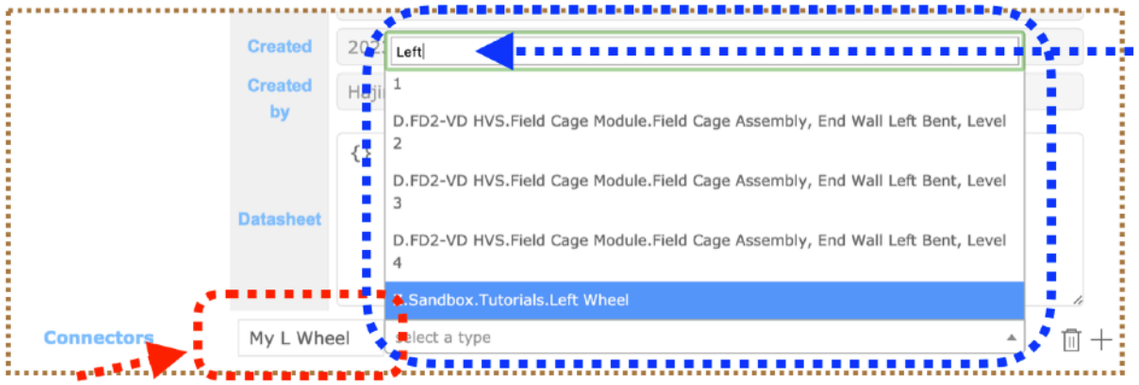

- The JSON file for its Type definition looks like this:

```
{
  "part_type_id": "Z00100400005",
  "comments": "Setting up this Type",
  "manufacturers": [7,27],
  "roles": [3,4],
  "properties": {
    "specifications": {
       "Last name": "Muramatsu",
       "First name": "Hajime"
    }
  },
  "connectors": {
     "My L Wheel": "Z00100400007",
     "My R Wheel": "Z00100400008"
  }
}
```

from previous session

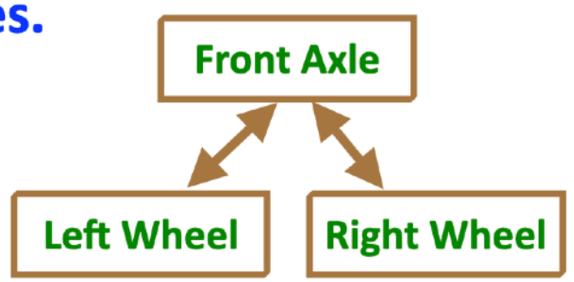
# Defining sub-Component Types

**WANT Left Wheel and Right Wheel to be sub-components of Front Axle**



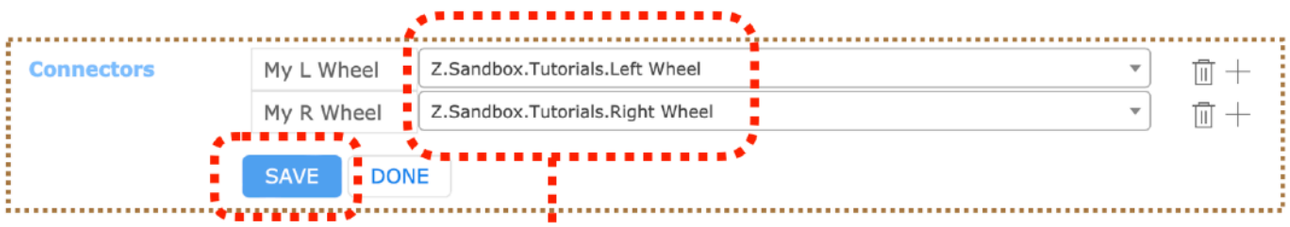

**Provide the corresponding sub-Component Type Name.**

**Entering a few characters will give you a list of matched candidates.**

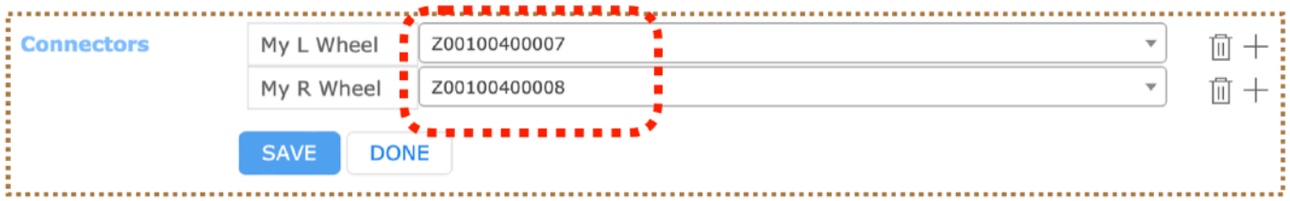



**Give your preferred name for its functional position**

**(can be any, but needs to be unique within this Component Type)**




**Now the two Type Names are entered.**



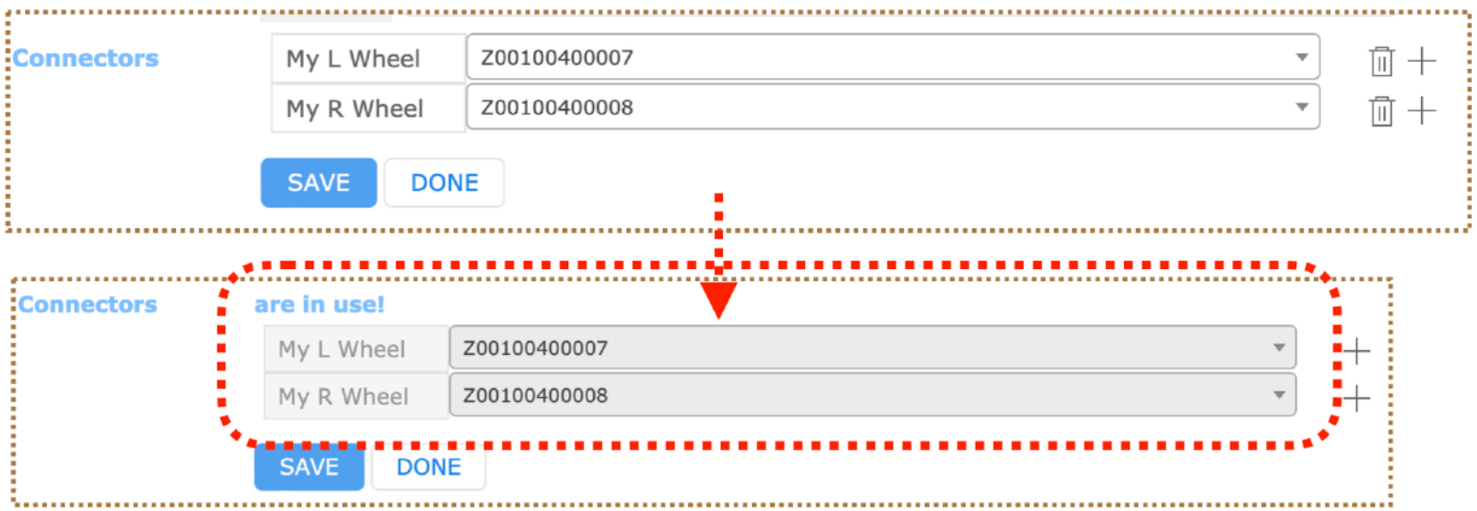

**Saving triggers the conversion from Type Names to Type IDs.**

**That is it! You are done!**

from previous session

# Cautions in defining Connectors - 1

- Once you define these sub-component Types and start to actually use them (i.e., linking existing Items as subcomponents), you will not be allowed to delete those sub-comp. Types in use. Else, there would be inconsistencies between the Type definition and Items that are linked.
- Because of this, the Type definition will be FROZEN once the corresponding Item is linked (we'll link Items on the 2nd day).



from previous session

# Cautions in defining Connectors - 2

- Even after they are frozen, however, you are allowed to add new sub-Component Type.
  (so even if you make a mistake in defining Types, you could just ignore it and introduce a new (correct) one).


- Or you could undo the all linked sub-components (Day 2 material).
  When this is done, you will be able to edit the existing sub-component Type definition (even to delete).

- We are about to make links to the existing Items.
  Make sure your subcomponent Type definitions are correct!

- Before making links to sub-components, make sure that those sub-components are in Status = available.
- If not, how do we make them available?
  and vice versa. How do we make then unavailable?

# Find out its Status

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

  CURL 'APIPATH/components/Z00100400005-00001'

GET its Item as usual, in which you can find its Status info.


```
  "location": null,
  "manufacturer": {
   "id": 7,
   "name": "Hajime Inc"
  },
  "part_id": "D00599800001-00379",
  "serial_number": "testing Serial Number 4",
  "specifications": [
   {
    "DATA": {
     "Drawing Number": "DFD-21-2101",
     "Label Code": "12345",
     "Name": "Main I-Beam"
    }
   }
  ],
  "specs_version": 11,
  "status": {
   "id": 3,
   "name": "permanently not available"
  }
 },
```


Or you can GET only Status info as well.

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

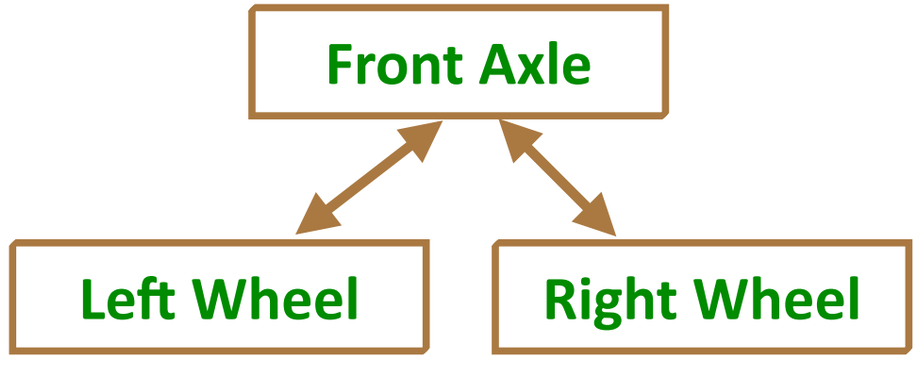
  CURL 'APIPATH/components/Z00100400005-00001/status'


```
{
 "component": {
  "id": 152958,
  "part_id": "D00599800001-00379"
 },
 "data": {
  "status": {
   "id": 3,
   "name": "permanently not available"
  }
 },
```


Status is also described by integers:

- 1 = available
- 2 = temporarily not available
- 3 = permanently not available

# Make it (un)available!




- And let's say the following two Items already exist in the HWDB:

  Z00100400007-00001 (Left Wheel)

  Z00100400008-00001 (Right Wheel)

- You want to make them available first.

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

  CURL -H "Content-Type: application/json" -X PATCH -d @Patch_Item_Status.json

  'APIPATH/components/Z00100400007-00001/status'

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

```
{
 "component_id": 46438,
 "data": "Created",
 "new_status": "available",
 "part_id": "Z00100400007-00001",
 "status": "OK"
}
```

Patch_Item_Status.json

```
{
  "component": {
    "part_id": "Z00100400007-00001"
  },
  "status": {
       "id": 1
  }
}
```

# Make multiple EIDs available at once!

- One could do so with ‘bulk-enable’ command.
- Let’s say you want to enable 3 EIDs,
  from Z00100400007-00002 and Z00100400007-00003.

- The API endpoint : /components/bulk-enable
- An example of actual line:

```
CURL -H "Content-Type: application/json" -X PATCH -d @enableBulkSub.json
     'APIPATH/components/bulk-enable'
```

enableBulkSub.json

```
{
  "data": [
    {
      "part_id": "Z00100400007-00002"
    },
    {
      "part_id": "Z00100400007-00003"
    }
  ]
}
```

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


```
{
 "data": [
  {
   "link": {
    "href": "/cdbdev/api/v1/components/46439",
    "rel": "self"
   },
   "message": "enabled",
   "part_id": "Z00100400007-00002"
  },
  {
   "link": {
    "href": "/cdbdev/api/v1/components/46440",
    "rel": "self"
   },
   "message": "enabled",
   "part_id": "Z00100400007-00003"
  }
 ],
 "status": "OK"
}
```


- Here, “enabled” means “available”.

# Or make multiple EIDs unavailable at once!

- We just need to disable them
- Let's say you want to

  enable Z00100400007-00001, but disable Z00100400007-00002.

- The API endpoint : /components/bulk-enable
- An example of actual line:

  CURL -H "Content-Type: application/json" -X PATCH -d @enableBulkSub.json

  'APIPATH/components/bulk-enable'

- When executed,

  you should see a response like this;


```
{
 "data": [
  {
   "link": {
    "href": "/cdbdev/api/v1/components/46438",
    "rel": "self"
   },
   "message": "enabled",
   "part_id": "Z00100400007-00001"
  },
  {
   "link": {
    "href": "/cdbdev/api/v1/components/46439",
    "rel": "self"
   },
   "message": "disabled",
   "part_id": "Z00100400007-00002"
  }
 ],
 "status": "OK"
}
```


enableBulkSub.json


```
{
  "data": [
    {
       "part_id": "Z00100400007-00001"
    },
      {
      "part_id": "Z00100400007-00002",
        "enabled": false
    }
  ]
}
```

# Let’s link subcomponents now!

- There are two ways to do this.

  One is to directly POST an Item, while specifying sub comp EIDs.

  The other is to PATCH existing an Item, while specifying sub comp EIDs only.

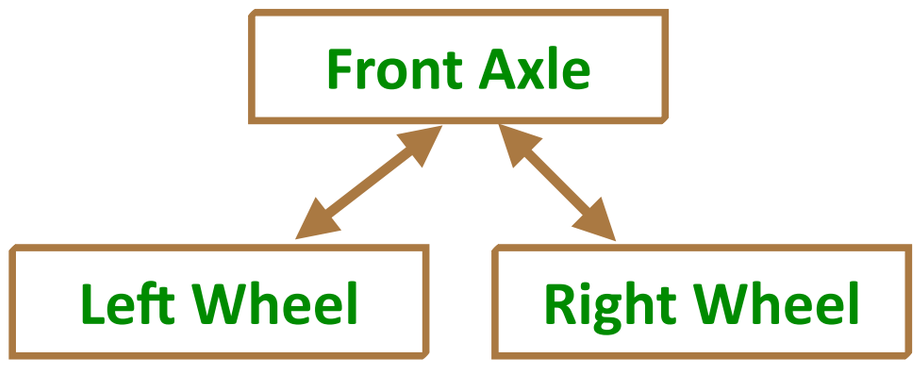
- In the following pages,

  we assume you have already made your sub comp EIDs available.

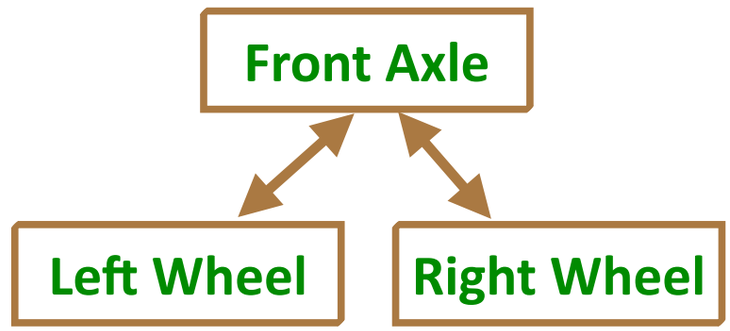
  In our example, we have sub-component EIDs to be linked:

  Z00100400007-00001 (Left Wheel)

  Z00100400008-00001 (Right Wheel)



# Linking Items (POST)




- Posting an Item, while specifying subcomponent EIDs.

  In this case, the procedure is identical to what page 5 shows.

- The API endpoint : /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/Z00100400005/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.
- When executed, you should see a response like this;

```
{
  "component_id": 153657,
  "data": "Created",
  "part_id": "Z00100400005-00017",
  "status": "OK"
}
```

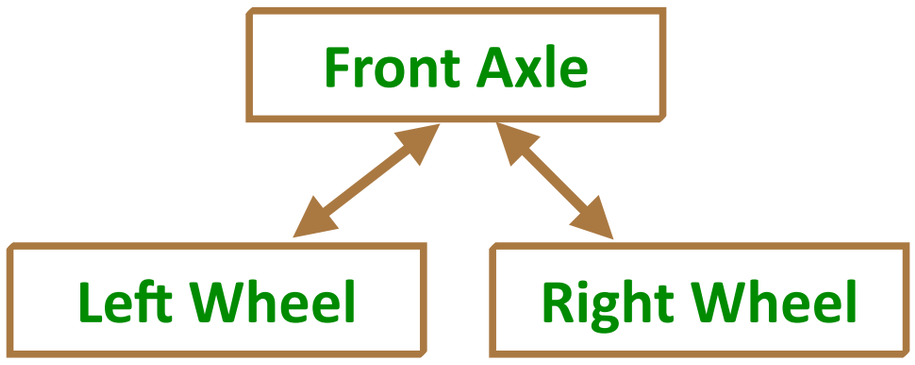
```
{
  "component_type": {
    "part_type_id": "Z00100400005"
  },
  "country_code": "US",
  "comments": "Testing...",
  "institution": {
    "id": 186
  },
  "manufacturer": {
    "id": 7
  },
  "specifications": {
        "Last name": "Muramatsu",
        "First name": "Hajime"
  },
  "subcomponents": {
    "My L Wheel": "Z00100400007-00001",
    "My R Wheel": "Z00100400008-00001"
  }
}
```

# Linking Items (PATCH)

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

- The API endpoint : /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/Z00100400005-00001/subcomponents'
```




```
{
  "component": {
       "part_id": "Z00100400005-00001"
  },
  "subcomponents": {
       "My L Wheel": "Z00100400007-00002",
       "My R Wheel": "Z00100400008-00002"
  }
}
```

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

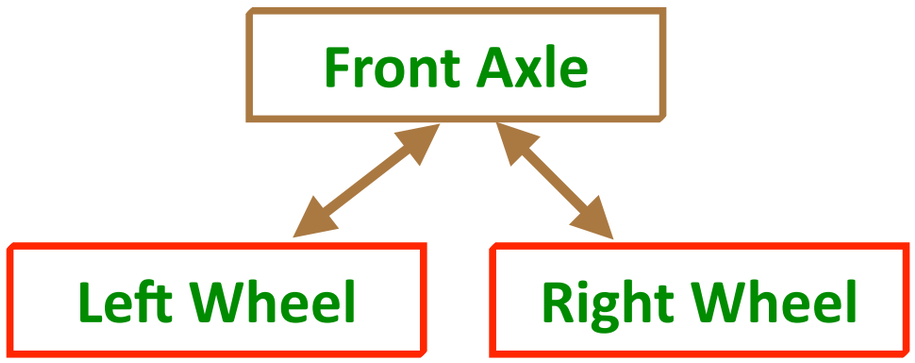
```
{
 "component_id": 46435,
 "data": "Updated",
 "part_id": "Z00100400005-00001",
 "status": "OK"
}
```

# GET sub-Components

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

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

  CURL 'APIPATH/components/Z00100400005-00001/subcomponents'




- Remember:

  Z00100400005-00001 (Front Axle)

  Z00100400007-00002 (Left Wheel)

  Z00100400008-00002 (Right Wheel)

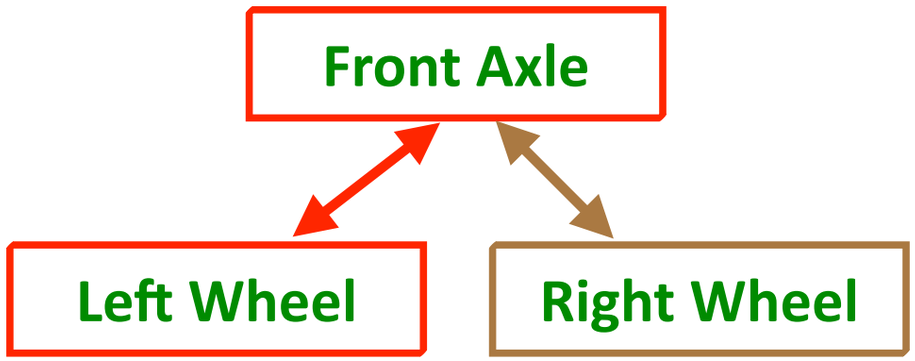

```
"data": [
 {
  "comments": null,
  "created": "2024-05-22T10:45:08.718759-05:00",
  "creator": {
   "id": 12624,
   "name": "Hajime Muramatsu",
   "username": "hajime3"
  },
  "functional_position": "My L Wheel",
  "link": {
   "href": "/cdbdev/api/v1/components/Z00100400007-00002",
   "rel": "self"
  },
  "operation": "mount",
  "part_id": "Z00100400007-00002",
  "type_name": "Left Wheel"
 },
 {
  "comments": null,
  "created": "2024-05-22T10:45:08.727897-05:00",
  "creator": {
   "id": 12624,
   "name": "Hajime Muramatsu",
   "username": "hajime3"
  },
  "functional_position": "My R Wheel",
  "link": {
   "href": "/cdbdev/api/v1/components/Z00100400008-00002",
   "rel": "self"
  },
  "operation": "mount",
  "part_id": "Z00100400008-00002",
  "type_name": "Right Wheel"
 }
],
```

# GET a parent-Component

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

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

  CURL 'APIPATH/components/Z00100400007-00002/container'




- Remember:

  Z00100400005-00001 (Front Axle)

  Z00100400007-00002 (Left Wheel)

  Z00100400008-00002 (Right Wheel)


```
"data": [
 {
  "comments": null,
  "container": {
   "component_type": {
    "name": "Front Axle",
    "part_type_id": "Z00100400005"
   },
   "link": {
    "href": "/cdbdev/api/v1/components/Z00100400005-00001",
    "rel": "self"
   },
   "part_id": "Z00100400005-00001"
  },
  "created": "2024-05-22T10:45:08.718759-05:00",
  "creator": {
   "id": 12624,
   "name": "Hajime Muramatsu",
   "username": "hajime3"
  },
  "functional_position": "My L Wheel",
  "link": {
   "href": "/cdbdev/api/v1/components/Z00100400005-00001",
   "rel": "self"
  },
  "operation": "mount",
  "part_id": "Z00100400007-00002",
  "type_name": "Left Wheel"
 }
],
```

# Clearing sub-Components (PATCH)

- And we can UNDO the links.

- The API endpoint : /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/Z00100400005-00001/subcomponents'

- Here is a JSON file to remove sub-Components and its response when executed:

```
{
  "component": {
    "part_id": "Z00100400005-00001"
  },
  "subcomponents": {
    "My L Wheel": null,
    "My R Wheel": null
  }
}
```

```
{
  "component_id": 46435,
  "data": "Updated",
  "part_id": "Z00100400005-00001",
  "status": "OK"
}
```

# GET location of an Item

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

  CURL 'APIPATH/components/Z00100300030-00103/locations'

- Returns in Data[].
- The latest shows up at the top.


```
"data": [
  {
   "arrived": "2024-04-10T14:50:12-05:00",
   "comments": "now testing through the REST API",
   "created": "2024-04-10T14:09:44.316278-05:00",
   "creator": "Hajime Muramatsu",
   "id": 92,
   "link": {
    "href": "/cdbdev/api/v1/locations/92",
    "rel": "details"
   },
   "location": {
    "id": 187,
    "name": "University of Mississippi"
   }
  },
  {
   "arrived": "2024-04-10T09:51:44.735194-05:00",
   "comments": "testing...",
   "created": "2024-04-10T09:51:44.735194-05:00",
   "creator": "Hajime Muramatsu",
   "id": 91,
   "link": {
    "href": "/cdbdev/api/v1/locations/91",
    "rel": "details"
   },
   "location": {
    "id": 186,
    "name": "University of Minnesota Twin Cities"
   }
  },
  {
   "arrived": "2024-04-10T14:50:12-05:00",
   "comments": "now testing through the REST API",
   "created": "2024-04-04T08:16:44.333658-05:00",
   "creator": "Hajime Muramatsu",
   "id": 15,
   "link": {
    "href": "/cdbdev/api/v1/locations/15",
    "rel": "details"
   },
   "location": {
    "id": 9,
    "name": "Universidade Federal de Alfenas"
   }
  },
```

# POST a new location

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

  CURL -H "Content-Type: application/json" -X POST -d @Post_ALocation.json 'APIPATH/components/Z00100300030-00103/locations'

- Specify the location by its Institution ID.
- You can omit its time-zone in the Arrived Date.

  The CST will be assumed when time-zone is omitted.

Post_ALocation.json

```
{
  "arrived": "2024-05-10 13:27:18",
  "comments": "now testing through the REST API",
  "location": {
    "id": 187
  }
}
```

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

```
{
  "data": "Created",
  "id": 95,
  "status": "OK"
}
```

# Downloading Bar/QR codes (GET)

- The API endpoint : /get-barcode/<pid>
  /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)


# POST image files

- We can POST/GET images.
- Most of the image formats are supported:
  jpeg, tiff, pdf, bmp, png

- There are 3 ways to POST/GET images.
  - POST/GET images for a given Component Type
  - POST/GET images for a given Item
  - POST/GET images for a given Test entry

In the followings, we'll go through each of these 3 cases.

# POST an image file

## - for a given Component Type -

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

  CURL -H "comments=testing from curl" -F "image=@myimage.pdf" 'APIPATH/component-types/Z00100100048/images'

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

```
{
  "data": "Created",
  "image_id": "fa636f5c-186d-11ef-ba26-2f96aeb61232",
  "status": "OK"
}
```

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

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

  CURL 'APIPATH/component-types/Z00100100048/images'

See the respond on the next page…

```
{
  "component_type": {
    "name": "Test_Parts_1",
    "part_type_id": "Z00100100048"
  },
  "data": [
    {
      "comments": null,
      "created": "2024-05-22T14:03:26.007944-05:00",
      "creator": {
        "id": 12624,
        "name": "Hajime Muramatsu",
        "username": "hajime3"
      },
      "image_id": "fa636f5c-186d-11ef-ba26-2f96aeb61232",
      "image_name": "myimage.pdf",
      "library": "type",
      "link": {
        "href": "/cdbdev/api/v1/img/fa636f5c-186d-11ef-ba26-2f96aeb61232",
        "rel": "self"
      }
    },
    {
      "comments": null,
      "created": "2024-05-22T14:09:14.068344-05:00",
      "creator": {
        "id": 12624,
        "name": "Hajime Muramatsu",
        "username": "hajime3"
      },
      "image_id": "c9dfdd88-186e-11ef-ba26-c76e454071ee",
      "image_name": "myimage.pdf",
      "library": "type",
      "link": {
        "href": "/cdbdev/api/v1/img/c9dfdd88-186e-11ef-ba26-c76e454071ee",
        "rel": "self"
      }
    }
  ],
  "link": {
    "href": "/cdbdev/api/v1/component-types/Z00100100048/images",
    "rel": "self"
  },
  "status": "OK"
}
```


- **When multiple images exist, it shows all of them.**

- **Record the image_id which you will need to retrieve the image.**
  **And now let's download one of them!**
  **Next page…**

# Downloading an image file

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

  CURL 'APIPATH/img/fa636f5c-186d-11ef-ba26-2f96aeb61232' -o myimage.pdf

# POST an image file

## - for a given Item -

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

  CURL -H "comments=testing from curl" -F "image=@myimage.pdf" 'APIPATH/components/Z00100100048-00033/images'

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


```
{
  "data": "Created",
  "image_id": "0c5cb65e-189c-11ef-bcbf-af36911a955c",
  "status": "OK"
}
```


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

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

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

See the respond on the next page…

```
{
  "comments": "testing from curl",
  "created": "2022-04-26T08:58:15.390115-05:00",
  "creator": {
    "id": 12624,
    "name": "Hajime Muramatsu",
    "username": "hajime3"
  },
  "image_id": "eda40038-c568-11ec-bb3d-bb303ad6adbb",
  "image_name": "10-0_10-1-6k_small.pdf",
  "library": "comp",
  "link": {
    "href": "/cdbdev/api/v1/img/eda40038-c568-11ec-bb3d-bb303ad6adbb",
    "rel": "self"
  }
 },
 {
  "comments": "testing from curl",
  "created": "2022-04-26T09:42:34.103207-05:00",
  "creator": {
    "id": 12624,
    "name": "Hajime Muramatsu",
    "username": "hajime3"
  },
  "image_id": "1c279e1e-c56f-11ec-a7bd-cfc7dff2306e",
  "image_name": "10-0_10-1-6k_small.jpg",
  "library": "comp",
  "link": {
    "href": "/cdbdev/api/v1/img/1c279e1e-c56f-11ec-a7bd-cfc7dff2306e",
    "rel": "self"
  }
 },
 {
  "comments": "testing from curl",
  "created": "2022-04-26T09:44:15.431201-05:00",
  "creator": {
    "id": 12624,
    "name": "Hajime Muramatsu",
    "username": "hajime3"
  },
  "image_id": "588681ea-c56f-11ec-a7bd-2f04a625073e",
  "image_name": "10-0_10-1-6k_small.png",
  "library": "comp",
  "link": {
    "href": "/cdbdev/api/v1/img/588681ea-c56f-11ec-a7bd-2f04a625073e",
    "rel": "self"
  }
 },
```


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

- **And you already know how to download them with the corresponding image_id!**

# POST an image file - 1

## - for a given Test entry -

- We can also post an image(s) to each of the Test entries.
- To do this,
  we need to know a unique ID that is assigned to each of the entries, “oid”.
- We can obtain oid(s) by GET a particular Test result(s).
- As an example, we will use:
  PID = Z00100200040-00001
  Test Type name = CPA_Parts_FR4_main QC check

(Or see page 21)

- The API endpoint : /components/<eid>/tests/<test_type_name>
- An example of actual line:
  CURL ‘APIPATH/components/Z00100200040-00001/tests/CPA_Parts_FR4_main QC check?history=true'

- See the next page for its response…

```
"data": [
 {
  "comments": "here is the 1st test of the test result...",
  "created": "2020-11-16T10:13:48.704421-06:00",
  "creator": {
   "id": 12624,
   "name": "Hajime Muramatsu",
   "username": "hajime3"
  },
  "id": 110,
  "images": [
   {
    "id": "12529424-86a6-11eb-a31b-936fd91bb429",
    "name": "DFD-20-A017.pdf"
   }
  ],
  "link": {
   "href": "/cdbdev/api/v1/component-tests/110",
   "rel": "details"
  },
  "methods": [
   {
    "href": "/cdbdev/api/v1/component-tests/110/images",
    "rel": "Images"
   }
  ],
  "test_data": {
   "Cleaned": 0,
   "Comments": null,
   "Template": 1,
   "Visual": 40
  },
  "test_spec_version": -1,
  "test_type": {
   "id": 15,
   "name": "CPA_Parts_FR4_main QC check"
  }
 },
 {
  "comments": "here is the 1st test of the test result...",
  "created": "2020-11-16T10:13:18.708351-06:00",
  "creator": {
   "id": 12624,
   "name": "Hajime Muramatsu",
   "username": "hajime3"
  },
  "id": 109,
  "images": [],
  "link": {
   "href": "/cdbdev/api/v1/component-tests/109",
   "rel": "details"
  },
  "methods": [
   {
    "href": "/cdbdev/api/v1/component-tests/109/images",
    "rel": "Images"
   }
  ],
  "test_data": {
   "Cleaned": 0,
   "Comments": "All looked/passed good...",
   "Template": 1,
   "Visual": 40
  },
  "test_spec_version": -1,
  "test_type": {
   "id": 15,
   "name": "CPA_Parts_FR4_main QC check"
  }
 },
```


- Two test results are shown.
- The 1st one has an image, while the 2nd one does not.

- Notice that, independent of whether there is an image(s) associated with its Test or not, there is always an OID assigned to each of the entries.

- Let’s POST an image to OID = 109!

# POST an image file - 2

## - for a given Test entry -

- The API endpoint : /component-tests/<oid>/images
- An example of actual line:

  CURL -H "comments=testing from curl" -F "image=@myimage.pdf" 'APIPATH/component-tests/109/images'

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

```
{
  "data": "Created",
  "image_id": "ccdd1168-18a0-11ef-ad0c-0bc7358d82dc",
  "status": "OK"
}
```

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

- The API endpoint : /component-tests/<oid>/images
- An example of actual line:

  CURL 'APIPATH/component-tests/109/images'

See the respond on the next page…

```
{
"data": [
 {
  "comments": null,
  "created": "2024-05-22T20:07:14.164978-05:00",
  "creator": {
   "id": 12624,
   "name": "Hajime Muramatsu",
   "username": "hajime3"
  },
  "image_id": "ccdd1168-18a0-11ef-ad0c-0bc7358d82dc",
  "image_name": "myimage.pdf",
  "library": "test",
  "link": {
   "href": "/cdbdev/api/v1/img/ccdd1168-18a0-11ef-ad0c-0bc7358d82dc",
   "rel": "self"
  }
 }
],
"link": {
 "href": "/cdbdev/api/v1/component-tests/109/images",
 "rel": "self"
},
"status": "OK"
}
```


- **It's there, along with its corresponding image_id as well.**

# We have only covered basic usage of the available REST API commands.

# Please explore other useful commands!

- Such as;

  CURL 'APIPATH/projects'

  CURL 'APIPATH/systems/<Project>'

  CURL 'APIPATH/subsystems/<Project>/<System ID>'

  CURL 'APIPATH/component-types/<Project>/<System ID>/<Subsystem ID>'

  CURL 'APIPATH/countries'

  CURL 'APIPATH/institution'

  CURL 'APIPATH/manufacturers'

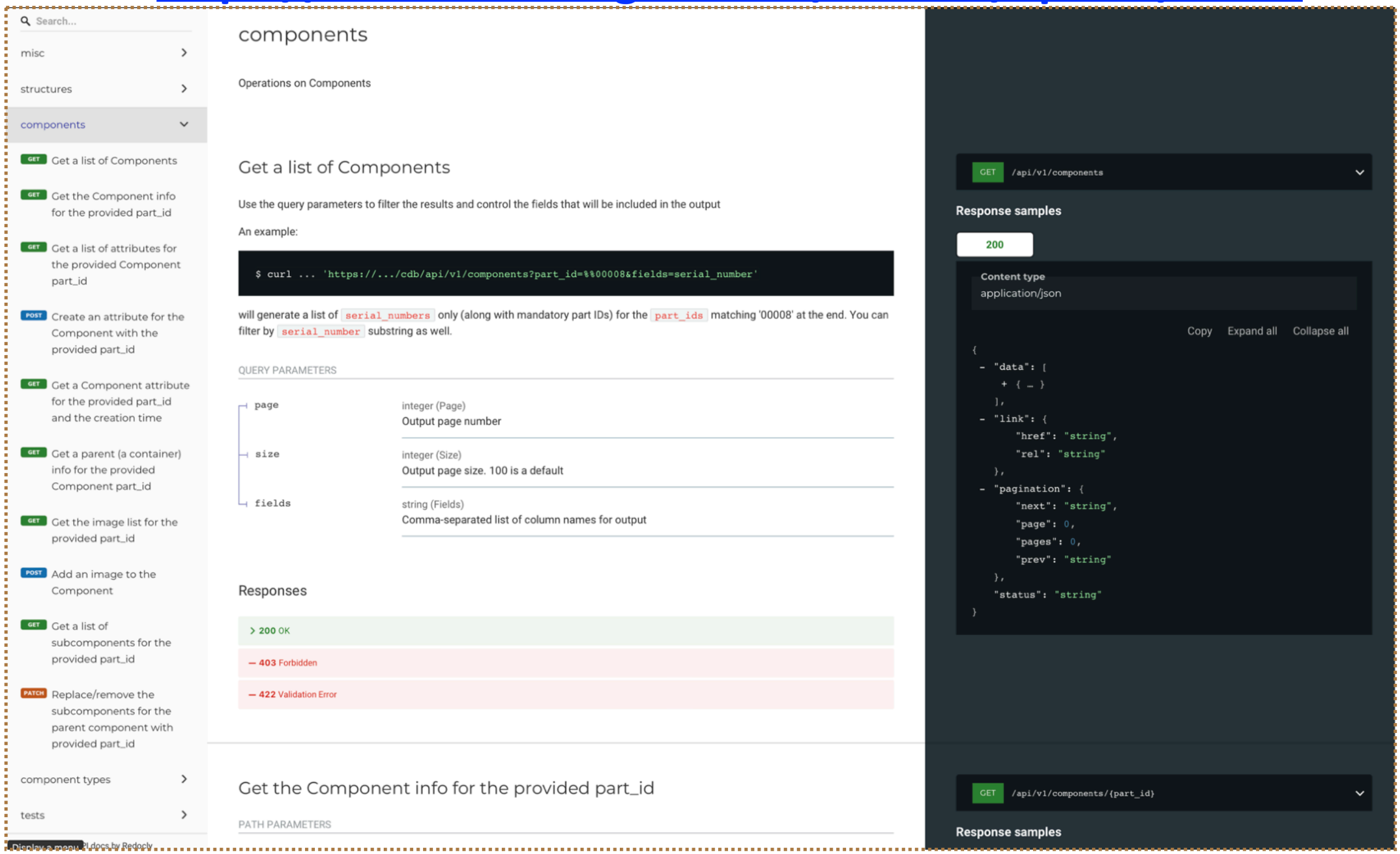
  CURL 'APIPATH/users'

  CURL ‘APIPATH/users/whoami’

# DUNE HWDB REST API Online Documentation

## All available commands are documented here (maintained by Redocly):

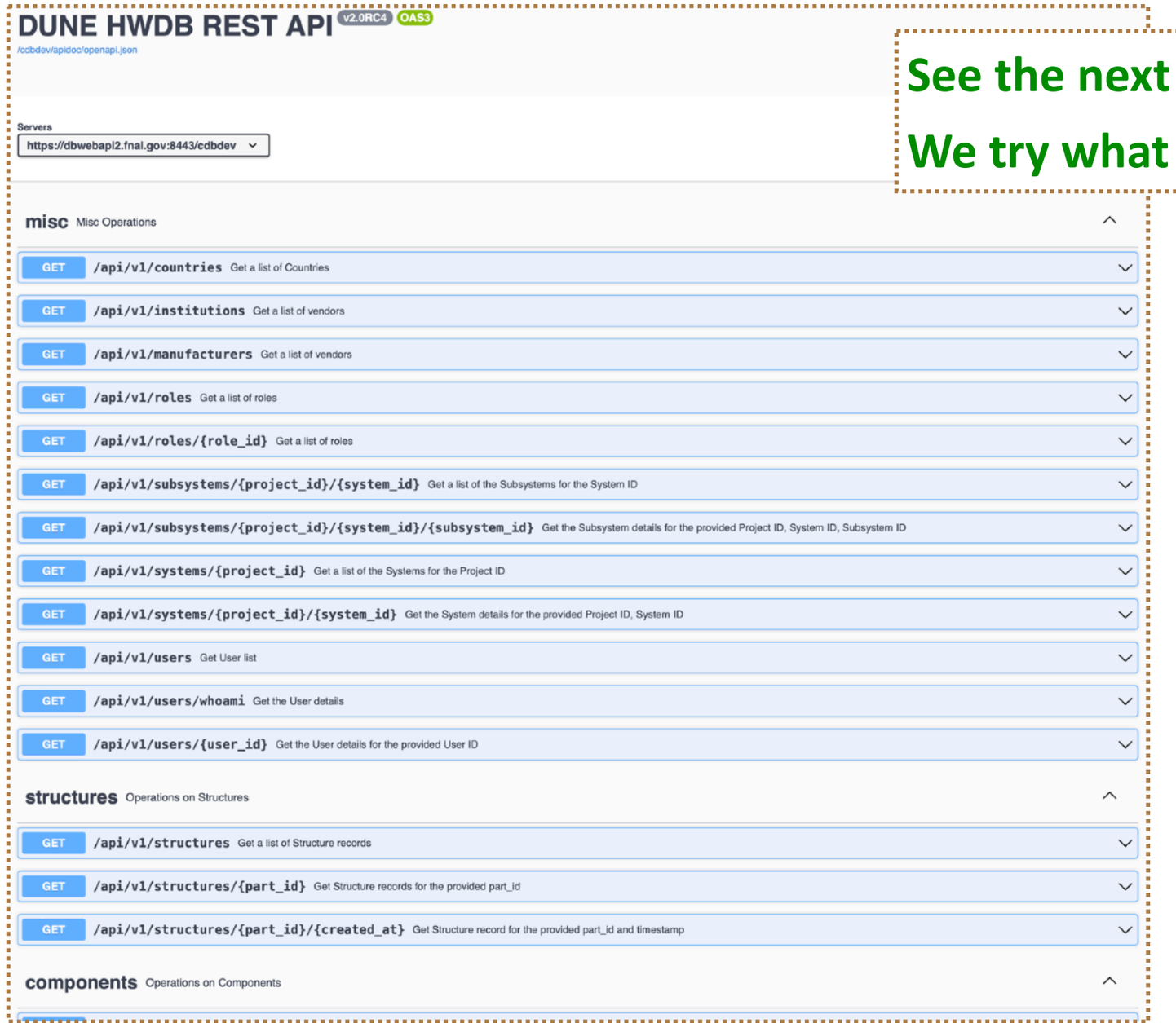
## https://dbweb9.fnal.gov:8443/cdbdev/apidoc/redoc



# Swagger!

## A place where you can actually try API commands and see their responses:

## https://dbweb9.fnal.gov:8443/cdbdev/apidoc/swagger




See the next page.

We try what we did on page 9.

# Checking a Component Type (GET) via Swagger

GET /api/v1/components/{part_id} Get the Component info for the provided part_id

Parameters

Try it out

Name Description

part_id * required
string
(path)

part_id

One can try this command here.

See the next page.

Responses

| Code | Description | Links |
|---|---|---|
| 200 | OK | *No links* |

Media type
application/json
Controls Accept header.

Example Value | Schema


```
{
  "data": {
    "batch": {
      "id": 0,
      "received": "2023-03-27"
    },
    "component_id": 0,
    "component_type": {
      "name": "string",
      "part_type_id": "string"
    },
    "country_code": "string",
    "created": "2023-03-27T19:41:27.194Z",
    "creator": {
      "id": 0,
      "name": "string",
      "username": "string"
    },
    "institution": {
      "id": 0,
      "name": "string"
    },
    "link": {
      "href": "string",
      "rel": "string"
    },
```


Its schema is displayed

| | | |
|---|---|---|
| 403 | Forbidden | *No links* |
| 404 | Not Found | *No links* |
| 422 | Validation Error | *No links* |

GET /api/v1/components/{part_id} Get the Component info for the provided part_id

Parameters Cancel

Name Description

part_id * required
string
(path)

Z00100100048-00031

Enter an EID…

Execute Clear

Responses

Curl

```
curl -X 'GET' \
  'https://dbwebapi2.fnal.gov:8443/cdbdev/api/v1/components/Z00100100048-00031' \
  -H 'accept: application/json'
```

Request URL

```
https://dbwebapi2.fnal.gov:8443/cdbdev/api/v1/components/Z00100100048-00031
```

Server response

Code Details

200

Response body


```
{
  "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",
      "username": "hajime3"
    },
    "institution": {
      "id": 186,
      "name": "University of Minnesota Twin Cities"
    },
    "manufacturer": {
      "id": 7,
      "name": "Hajime Inc"
    },
    "part_id": "Z00100100048-00031",
    "serial_number": null
```


Actual response can be seen here!

Download

Response headers

```
content-length: 1171
content-type: application/json
```