MN-DUNE

Python Database Upload Tool

Version 1.0

Marvin Marshak
Hajime Muramatsu
Alex Wagner
Urbas Ekka

Table of Contents

Table of Contents

Introduction

The DUNE HWDB is set up to provide a repository for diverse hardware components and their associated tests, where the data requirements may be drastically different from one component type to the next.

To accommodate this, the database provides free-form JSON/YAML “Specifications” and “Datasheet” fields, where each group can define their own structures. It is therefore helpful for us to provide tools for the various consortia to manage this complexity.

The Python Database Upload Tool is a utility to help with this complexity. It allows consortia to define those structures, map them to tabular data contained in CSV or Excel files, and upload that data into the HWDB.

Setup

Prerequisites

- Linux/MacOS system with Anaconda installed. This tool has not been tested in a pure Windows environment, but it works fine with Windows using WSL/Ubuntu. Anaconda may be downloaded at https://www.anaconda.com/download.
- A Fermilab Services account
- A PKCS12 certificate for the Fermilab account. Visit https://cilogon.org to obtain this certificate.
- The user must have access to the HWDB database. (This may take 24 hours after your Services account is activated.)
- The tool is used in conjunction with the HWDB Web UI. Familiarity with the Web interface is required for setting up hardware items and tests to receive uploaded data. Once these have been set up, subsequent users will not necessarily need to know how to use the Web interface.

  The Web UI can be found at the following URLs:

Development	https://dbweb9.fnal.gov:8443/cdbdev/index
Production	https://dbweb9.fnal.gov:8443/cdb/index

Installation

Currently, the installation package will be supplied as a ZIP file (“Sisyphus-20230615.zip”)

Choose an appropriate directory to extract the contents of the ZIP file. For example: “$HOME/Projects”, thereby creating a directory “$HOME/Projects/Sisyphus-20230615”

If your default shell is BASH, add the following to the end of your “.bashrc” file (or other shell startup script), using, of course, the actual name of the directory of the extracted contents:

```
export PATH=~/Projects/Sisyphus-20230615:$PATH
export PYTHONPATH=~/Projects/Sisyphus-20230615/lib:$PYTHONPATH
```

Modify these instructions as necessary if you are using a different shell.

These paths are not required to run the main scripts in the upload tool, but they do allow those scripts to be invoked without specifying the entire path to those scripts. The remainder of this document will assume that these paths have been added.

User Configuration

Obtaining a PKCS12 Certificate

Before the upload tool can be used, users need to download a PKCS12 certificate for their fermilab account.

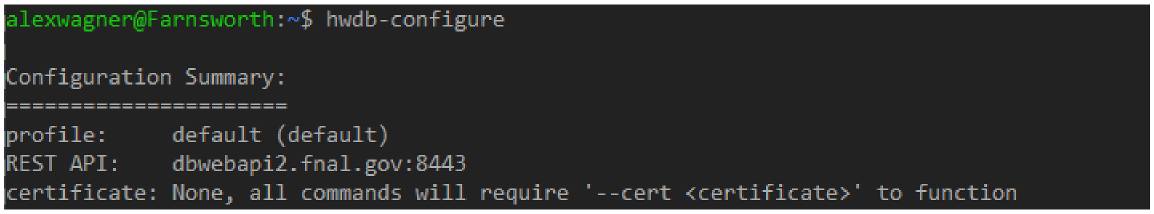
1. In your browser, go to https://cilogon.org
2. If you have never visited CILogon before, it may ask you to select an identity provider. Choose “Fermi National Accelerator Laboratory” and click “Logon”
3. The browser will redirect to the Fermilab authentication page. Authenticate using one of the methods offered.
4. The browser will now return to the CILogon page. Choose “Create Password-Protected Certificate”
5. Choose a password to be associated with your PKCS12 certificate. Note that this does not have to match the password for your Fermilab Services account.
6. Click “Get New Certificate.” A new certificate will be generated for you to download.
7. Click “Download Certificate.” The new certificate will be downloaded to the directory your browser is currently set for downloading files. (On Windows, this will typically be your “Downloads” folder.) The file will be named “usercred.p12” if this is the first certificate you’ve downloaded, or “usercred (<number>).p12” if you have downloaded previous certificates.
8. Make a note of the location of the PKCS12 certificate. If you will be using the Upload Tool on a Linux system, you may need to move the file to an accessible location. You may rename this file to avoid ambiguity if you will have multiple users on your system, e.g., to “<your username>.p12”.

Configuring Access to the HWDB

Use the “hwdb-configure” command to configure access to the HWDB.

Viewing the configuration

To view the current configuration, enter “hwdb-configure” with no parameters. If the configuration utility has never been called before, the tool will automatically create a directory “$HOME/.sisyphus” containing a default configuration using the DEVELOPMENT version of the HWDB.

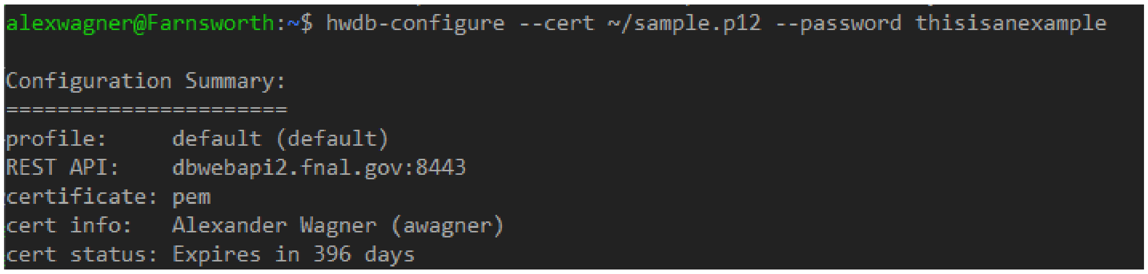


```
alexwagner@Farnsworth:~$ hwdb-configure

Configuration Summary:
======================
profile:     default (default)
REST API:    dbwebapi2.fnal.gov:8443
certificate: None, all commands will require '--cert <certificate>' to function
```

Figure 1: Using hwdb-configure to display the current configuration

Setting a PKCS12 certificate

To use the PKCS12 certificate downloaded from CILogon, supply the location of downloaded certificate and the password you created using “--cert” and “--password” parameters.



```
alexwagner@Farnsworth:~$ hwdb-configure --cert ~/sample.p12 --password thisisanexample

Configuration Summary:
======================
profile:     default (default)
REST API:    dbwebapi2.fnal.gov:8443
certificate: pem
cert info:   Alexander Wagner (awagner)
cert status: Expires in 396 days
```

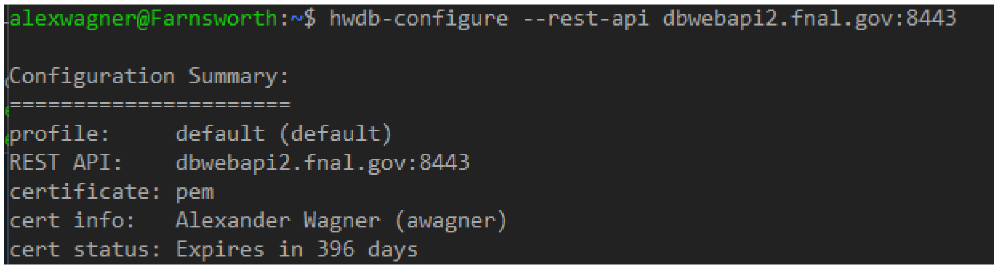
Figure 2: Using hwdb-configure to add a user's certificate

The configuration utility will automatically extract a PEM certificate from the P12 file using the password provided (or raise an error if the password is incorrect) and store it in the “$HOME/.sisyphpus” directory. The password used is not retained by the system after configuration!

WARNING: Please note that the PEM certificate is unencrypted and not password protected, and it can now be used to access the HWDB without a password! The configuration utility does attempt to set the “$HOME/.sisyphus” directory to only be readable by the owner, but it may be worthwhile to check the permissions to see that they were set correctly. In general, use good security practices in handling the PEM file!

Setting the HWDB server

To set the utility to point to a different HWDB server (e.g., production), use the "--rest-api" parameter.



```
alexwagner@Farnsworth:~$ hwdb-configure --rest-api dbwebapi2.fnal.gov:8443

Configuration Summary:
======================
profile:     default (default)
REST API:    dbwebapi2.fnal.gov:8443
certificate: pem
cert info:   Alexander Wagner (awagner)
cert status: Expires in 396 days
```

Figure 3: Using hwdb-configure to change the REST API server location

The addresses for the two currently-available servers are as follows:

Development	dbwebapi2.fnal.gov:8443
Production	dbweb9.fnal.gov:8443

The examples provided in this document are only currently available on the Development server. It is highly recommended that new Component Types are tested on the Development server before being deployed to Production.

Tutorial

The following examples demonstrate the general usage of the Upload Tool. After installing the application, these examples can be found under `${INSTALL_DIR}/Examples/upload-docket/Tutorial`.

For these examples, the HW Item and Test datasheets already exist in the development database. For your own items and tests, you will need to edit these yourself. Please refrain from changing the HW Item and Test Definition datasheets in the Web UI for the Component Types used in these examples!

Note that you will need to change the serial numbers in the spreadsheets between tests!

Example 1 (Simple)

For this example, we will upload a HW Item to the database, with no test data. The HW Item will have a simple datasheet, consisting only of (field, value) pairs.

Preparing Component Types For Uploading

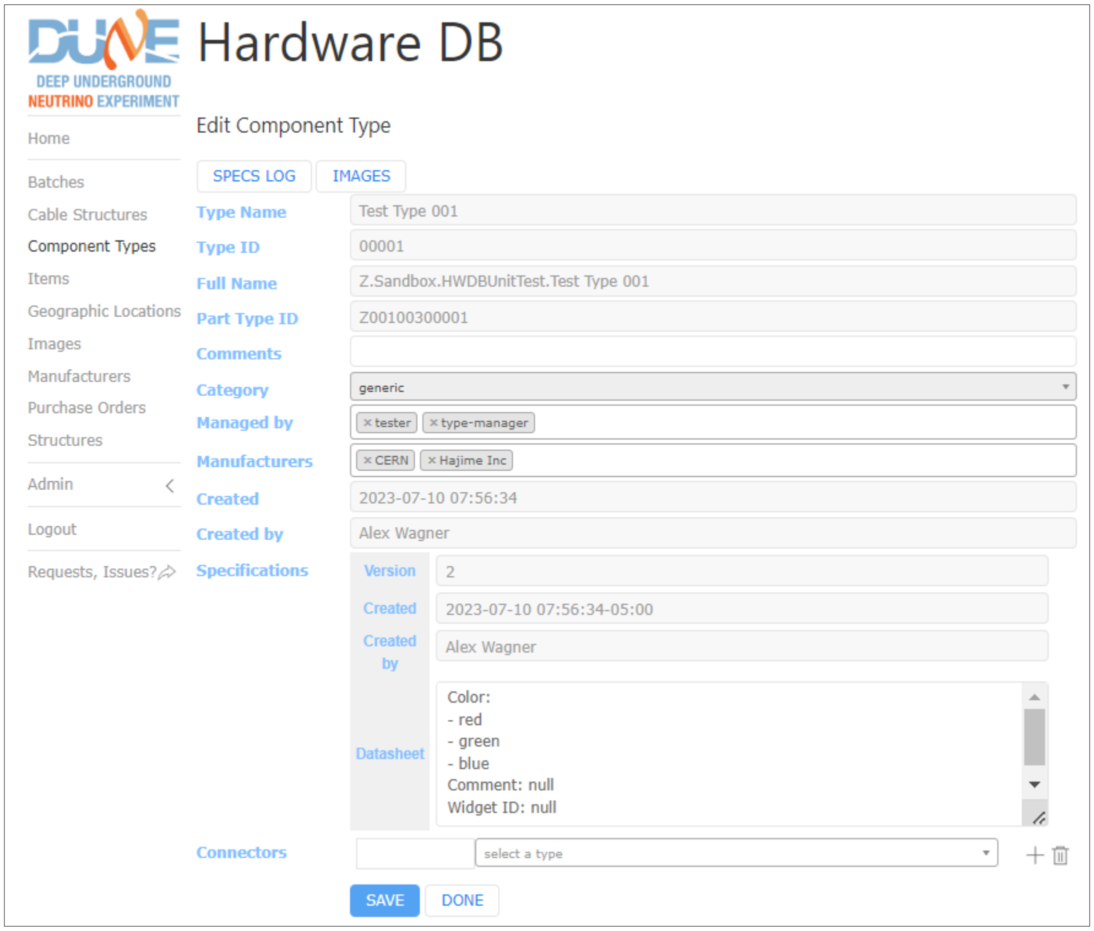
Uploading HW Item data to the database requires four things:

- The Datasheet of the Component Type must be configured in the Web Interface to contain the structure of the data you wish to upload.
- The data to be uploaded must be contained in spreadsheets (CSV or Excel files)
- An encoder must be created in JSON format describing how spreadsheet data is translated into the Datasheet for the item. (If the structure is simple, the encoder is very straightforward.)
- A docket file must be created in JSON format listing the spreadsheets containing data and the encoder(s) to use

Configuring the Datasheet

For this example, the Datasheet has already been configured. The Datasheet has three fields: “Color”, “Comment”, and “Widget ID”.

Note that the intent here was to suppose that the HW Item’s serial number is internally known as “Widget ID”, so it has been added to the Datasheet to reflect that. Note that this is merely an aesthetic choice, and it is not necessary if you are sufficiently happy with it only being known as “Serial Number.”



Type Name	Test Type 001
Type ID	00001
Full Name	Z.Sandbox.HWDBUnitTest.Test Type 001
Part Type ID	Z00100300001
Comments	
Category	generic
Managed by	× tester × type-manager
Manufacturers	× CERN × Hajime Inc
Created	2023-07-10 07:56:34
Created by	Alex Wagner
Specifications	Version: 2; Created: 2023-07-10 07:56:34-05:00; Created by: Alex Wagner
Connectors	select a type

Datasheet:

```
Color:
- red
- green
- blue
Comment: null
Widget ID: null
```

Figure 4: Example 1 HW Item Datasheet

Note the following about datasheets:

- The Datasheet is in YAML format. The same content may be formatted in multiple ways in YAML. Don’t be alarmed if, after saving the Specification, the Datasheet is formatted differently than it was inputted.

- Dictionary-type data is by definition unordered, so the list of fields given will likely not appear in the same order after it is saved.
- Field names are case-sensitive.
- If a field is listed in the Datasheet, it is required to be in the data to be uploaded. (It is acceptable for it to be blank or null in the spreadsheet, but it must be accounted for.)
- The database will accept fields that are not listed. This leaves the possibility of putting leaving the Datasheet empty and still uploading data into it anyway. It is not recommended to do this, as the functionality is subject to change.
- If a field is defined as a list, the database will expect that the uploaded data for that field will be one of the items in that list. E.g., if the field is defined as “Color: [red, green, blue]”, then the submitted data must be one of the three values.
- If a field is expected to contain complex data such as a list or dictionary (or even a nested hierarchical structure of lists and dictionaries, this structure does not need to be defined here (for the reason above that it will be interpreted as multiple-choice). Just use “field_name: null”.

Organizing HW Item Data in Spreadsheets

HW Items to be uploaded must be organized in spreadsheets. These may be in CSV or Excel format. If you use Excel format, the file may contain multiple sheets, because when the docket file is created, it is possible to specify which worksheet in the file to use.

The spreadsheet **must** contain the following column headers. Note that they are case-sensitive and should not contain any extra spaces before or after the name. Item data should be supplied in the second row and beyond.

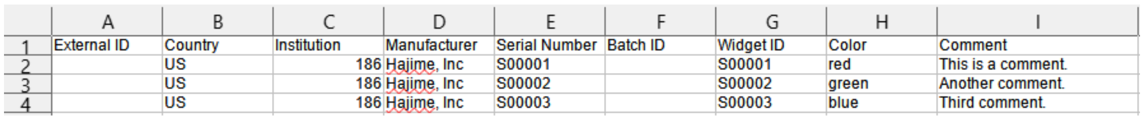
- "External ID" – The database will assign the External ID when the item is uploaded, so leave this blank. (Future: supplying an External ID will make the tool attempt to edit an existing item instead of adding a new one.)
- "Country" – Every item needs to have a country of origin (along with an institution ID) in order for the database to define the full Part ID, e.g. "D00501342003-00002-US125". The country may be supplied as any of the following:
    - The two-letter country code, e.g., "US"
    - The full name of the country, e.g., "United States"
    - The country code in parentheses, followed by the full name, e.g., "(US) United States"

  Please refer to EDMS ID 2505353 for a list of country codes.
- "Institution" – Similarly to Country, the Institution is required for the database to define the full Part ID. The institution may be supplied in the following ways:
    - The numeric institution code, e.g., "186"
    - The full name of the institution, e.g., "University of Minnesota Twin Cities"
    - The institution code in parentheses, followed by the full institution name, e.g., "(186) University of Minnesota Twin Cities"

  Please refer to EDMS ID 2505353 for a list of institution codes.
- "Manufacturer" – The manufacturer must be in the list of manufacturers configured for the Component Type. Similar to Country and Institution, it may be given as the manufacturer's ID, the manufacturer name, or a combination, e.g., "27", "CERN", or "(27) CERN".

- “Serial Number” – This should be some number that uniquely identifies the HW Item (within the given Component Type) prior to it being assigned an External ID. The HWDB itself will allow a serial number to be used multiple times, but the Upload Tool will not. Be warned that editing a serial number through the Web Interface, although allowed, may confuse the Upload Tool when attempting to add secondary information to a HW Item, such as test results.
- The remaining column headers should correspond with the fields listed in the Datasheet. Note that the choice to put “Widget ID” in the Datasheet results in the same serial number data under two differently-named columns.
- It is possible, if you wish, to define structured data for one or more of the fields. In this case, the data can be added in separate sheets or as additional columns in the main sheet. See instructions and examples later in this document to see how to do this.

Here is the HW Item spreadsheet for this example:



	A	B	C	D	E	F	G	H	I
1	External ID	Country	Institution	Manufacturer	Serial Number	Batch ID	Widget ID	Color	Comment
2		US	186	Hajime, Inc	S00001		S00001	red	This is a comment.
3		US	186	Hajime, Inc	S00002		S00002	green	Another comment.
4		US	186	Hajime, Inc	S00003		S00003	blue	Third comment.

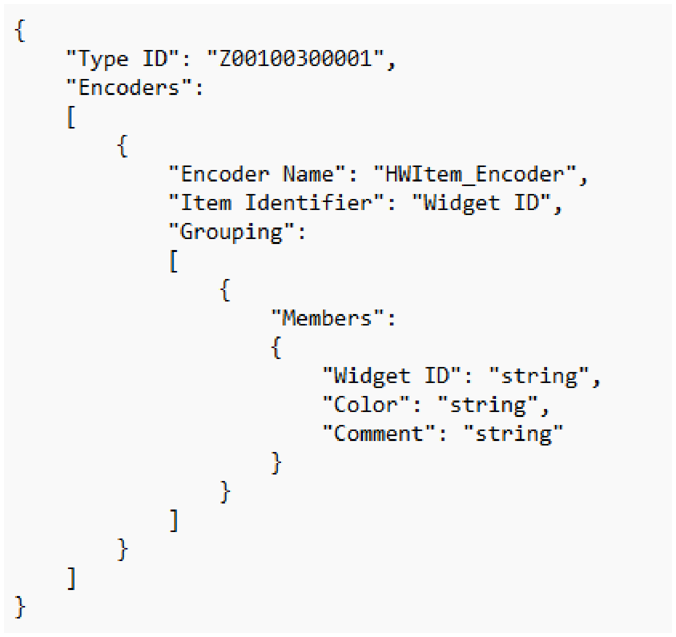
Figure 5: Example 1 HW Item Spreadsheet

Defining Encoders

An Encoder is a JSON document that maps a spreadsheet to a Datasheet in a HW Item or a Test for a HW Item.

It is possible to define an Encoder inline in a Docket file, but since Encoders are intended to be reused, this document will only cover the case where Encoders are isolated into their own files.

Here is the encoder file “encoders.json” for this example:



```
{
    "Type ID": "Z00100300001",
    "Encoders":
    [
        {
            "Encoder Name": "HWItem_Encoder",
            "Item Identifier": "Widget ID",
            "Grouping":
            [
                {
                    "Members":
                    {
                        "Widget ID": "string",
                        "Color": "string",
                        "Comment": "string"
                    }
                }
            ]
        }
    ]
}
```

Figure 6: Example 1 HW Item Encoder

The Encoder file is required to be formatted as follows:

- The top level must be a dictionary
- The top level dictionary must contain a “Type ID” field which specifies the Type ID for the Component Type
- The top level dictionary must contain an “Encoders” field which contains a list
- Each item in the list specifies a different Encoder. For simple cases, this may be a single item.
- Each Encoder is a dictionary which contains the following entries:
    - “Encoder Name” – this is how the Docket file will identify which Encoder to use.
    - “Item Identifier” – if the data in the Datasheet were thought of as an entry in a table, the Item Identifier is the field that would be the key. It may help to explain that this is somewhat of a throwback to before we felt confident that the Serial Number didn’t have some predetermined use outside of the end user’s control. In the future, we may make this field optional and assume “Serial Number” as the default.

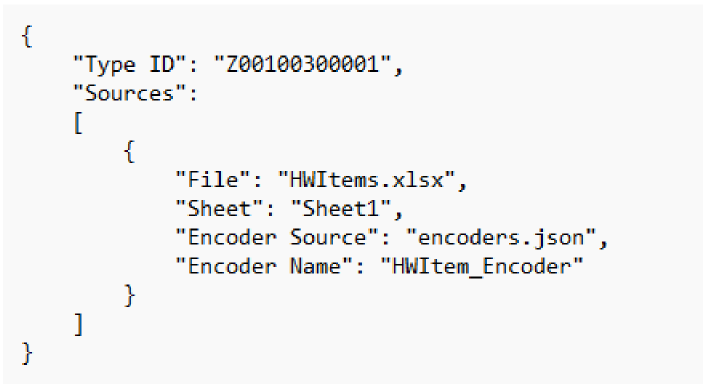
    - “Grouping” – this contains a list of “group” objects. All the user-defined columns in the spreadsheet must belong to a group. The groups define the hierarchy represented by the columns in the spreadsheet. For simple cases, the data would most likely be “flat,” and would only require a single group, but a more complicated item may contain fields that are themselves lists of records with their own fields. This can be achieved by using multiple groups. This will be illustrated later in this document.

- Each “group” must contain a “Members” field, which contains the user-defined columns for the HW Item.
- A “group” is also allowed to define a “Name”, a “Key” containing one or more of the members in that group, and a “Lock Keys” boolean value. These are more useful for Test Encoders than for Item Encoders, and will be explained there.

Defining a Docket

A Docket is a JSON document that specifies which spreadsheets should be encoded and uploaded to the HWDB.

Here is the docket file “docket.json” for this example:



```
{
    "Type ID": "Z00100300001",
    "Sources":
    [
        {
            "File": "HWItems.xlsx",
            "Sheet": "Sheet1",
            "Encoder Source": "encoders.json",
            "Encoder Name": "HWItem_Encoder"
        }
    ]
}
```

Figure 7: Example 1 Docket File

The Docket file is required to be formatted as follows:

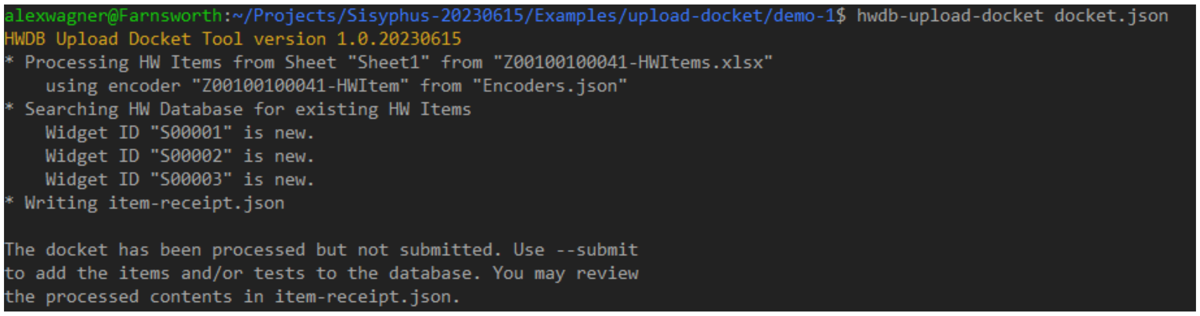
- The top level must be a dictionary
- The top level dictionary must contain a “Type ID” field which specifies the Type ID for the Component Type
- The top level dictionary must contain a “Sources” field which contains a list.
- Each item in the list is a dictionary object that specifies a source
- Each source may contain the following fields:
    - “File” – the absolute or relative location of a CSV or Excel spreadsheet. If the location is relative, the tool will first check relative to the directory containing the Docket file. If not found, it will then check relative to the current working directory in the shell invoking the tool. File globbing is allowed. If the pattern matches more than one file, the order in which the files will be processed is not guaranteed.
    - “Sheet” – if the file(s) are Excel, “Sheet” specifies which worksheet name or number will be used. If no sheet is specified, only the first sheet will be used. If the file is CSV, “Sheet” is ignored.

    - “Encoder Source” – the absolute or relative location of an Encoder file containing the Encoder to be used. If the Encoder is specified later in the Docket file, “Encoder Source” is not needed.
    - “Encoder Name” – the name of the Encoder specified in either “Encoder Source” or later in the Docket file
- The top level dictionary may also contain an “Encoders” field governed by the same rules as the Encoder file.

Uploading a HW Item

Once you have created a Docket and Encoder, have defined the Datasheet in the HWDB Web Interface, and have a spreadsheet containing HW Item information to be uploaded, you are now ready to use the Upload Tool to upload your HW Items.

Invoke the Upload tool in your shell with “hwdb-upload-docket”. The Upload Tool will scan the Docket for sources and process them using the specified Encoders. At this point, it will NOT upload the results to the database. Instead, it will output its results to “item-receipt.json” for you to review to make sure that the encoding produced what you expected.



```
alexwagner@Farnsworth:~/Projects/Sisyphus-20230615/Examples/upload-docket/demo-1$ hwdb-upload-docket docket.json
HWDB Upload Docket Tool version 1.0.20230615
* Processing HW Items from Sheet "Sheet1" from "Z00100100041-HWItems.xlsx"
    using encoder "Z00100100041-HWItem" from "Encoders.json"
* Searching HW Database for existing HW Items
    Widget ID "S00001" is new.
    Widget ID "S00002" is new.
    Widget ID "S00003" is new.
* Writing item-receipt.json

The docket has been processed but not submitted. Use --submit
to add the items and/or tests to the database. You may review
the processed contents in item-receipt.json.
```

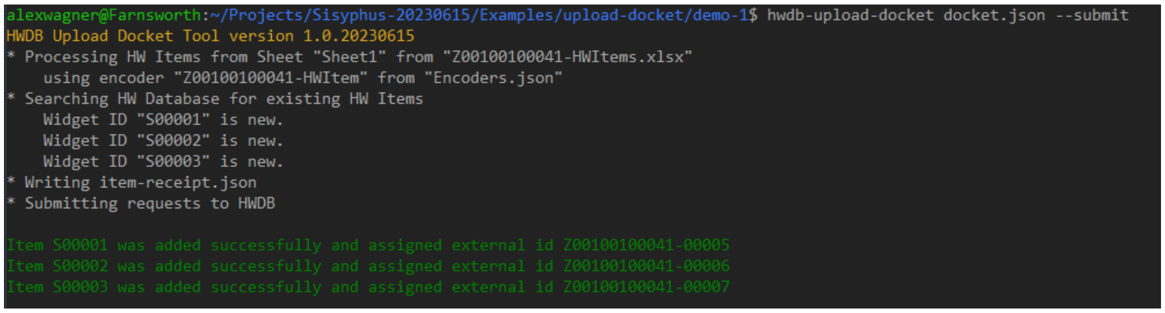
Figure 8: Using hwdb-upload-docket to preview upload

Here is the contents of the “item-receipt.json” file:

```
{
    "S00001": {
        "_in_manifest": true,
        "External ID": null,
        "Country": "US",
        "Institution": 186,
        "Manufacturer": "Hajime Inc",
        "Serial Number": "S00001",
        "Batch ID": null,
        "Specifications": {
            "_meta": {
                "_column_order": [
                    "Widget ID",
                    "Color",
                    "Comment"
                ]
            },
            "Widget ID": "S00001",
            "Color": "red",
            "Comment": "This is a comment."
        }
    },
    … (two more items not shown) …
}
```

Figure 9: item-receipt.json

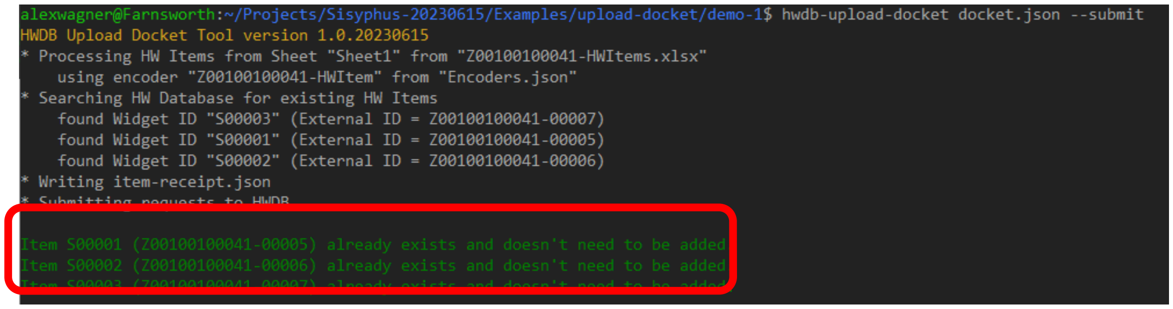
If the contents of “item-receipt.json” looks correct, you may then invoke “hwdb-upload-docket” again, this time adding the parameter “--submit”. The Upload Tool will attempt to upload the items to the database and it will report whether the attempt was successful. PLEASE NOTE that when using “--submit”, the tool processes the docket from the beginning. It does not read or use what was previously written in “item-receipt.json”.



```
alexwagner@Farnsworth:~/Projects/Sisyphus-20230615/Examples/upload-docket/demo-1$ hwdb-upload-docket docket.json --submit
HWDB Upload Docket Tool version 1.0.20230615
* Processing HW Items from Sheet "Sheet1" from "Z00100100041-HWItems.xlsx"
    using encoder "Z00100100041-HWItem" from "Encoders.json"
* Searching HW Database for existing HW Items
    Widget ID "S00001" is new.
    Widget ID "S00002" is new.
    Widget ID "S00003" is new.
* Writing item-receipt.json
* Submitting requests to HWDB

Item S00001 was added successfully and assigned external id Z00100100041-00005
Item S00002 was added successfully and assigned external id Z00100100041-00006
Item S00003 was added successfully and assigned external id Z00100100041-00007
```

Figure 10: Using hwdb-upload-docket to upload data

The tool will automatically check for existing serial numbers and will not attempt to add them again if you invoke the tool a second time:



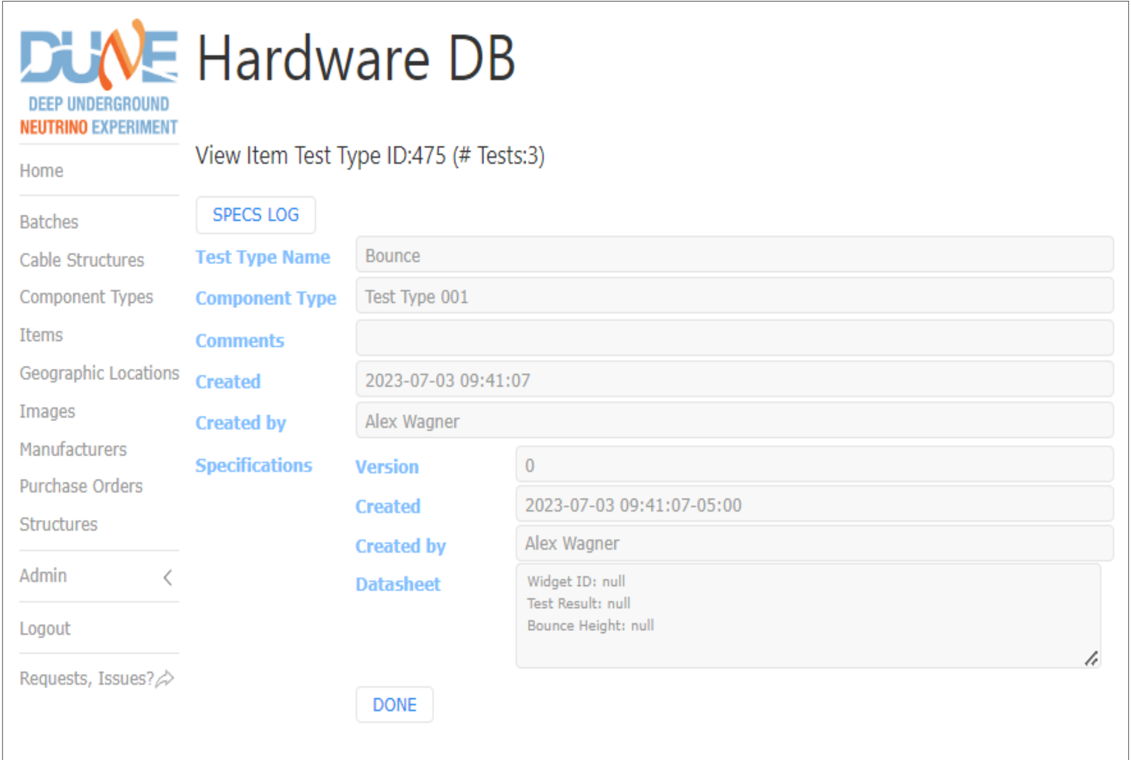
```
alexwagner@Farnsworth:~/Projects/Sisyphus-20230615/Examples/upload-docket/demo-1$ hwdb-upload-docket docket.json --submit
HWDB Upload Docket Tool version 1.0.20230615
* Processing HW Items from Sheet "Sheet1" from "Z00100100041-HWItems.xlsx"
    using encoder "Z00100100041-HWItem" from "Encoders.json"
* Searching HW Database for existing HW Items
    found Widget ID "S00003" (External ID = Z00100100041-00007)
    found Widget ID "S00001" (External ID = Z00100100041-00005)
    found Widget ID "S00002" (External ID = Z00100100041-00006)
* Writing item-receipt.json
* Submitting requests to HWDB

Item S00001 (Z00100100041-00005) already exists and doesn't need to be added
Item S00002 (Z00100100041-00006) already exists and doesn't need to be added
Item S00003 (Z00100100041-00007) already exists and doesn't need to be added
```

Figure 11: Attempting to add HW Items a second time

Example 2 (Adding a Test)

Let’s add a new Test Type for our Component Type.

We will create a test named “Bounce”, with the Datasheet as shown:



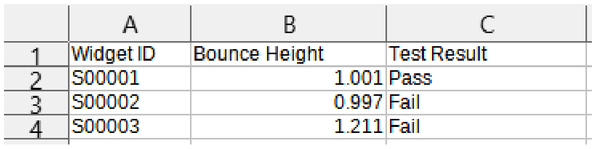
View Item Test Type ID:475 (# Tests:3)

Test Type Name	Bounce
Component Type	Test Type 001
Comments	
Created	2023-07-03 09:41:07
Created by	Alex Wagner
Specifications	Version: 0 Created: 2023-07-03 09:41:07-05:00 Created by: Alex Wagner Datasheet: Widget ID: null Test Result: null Bounce Height: null

Figure 12: Example 2 HW Item Test Specification

.

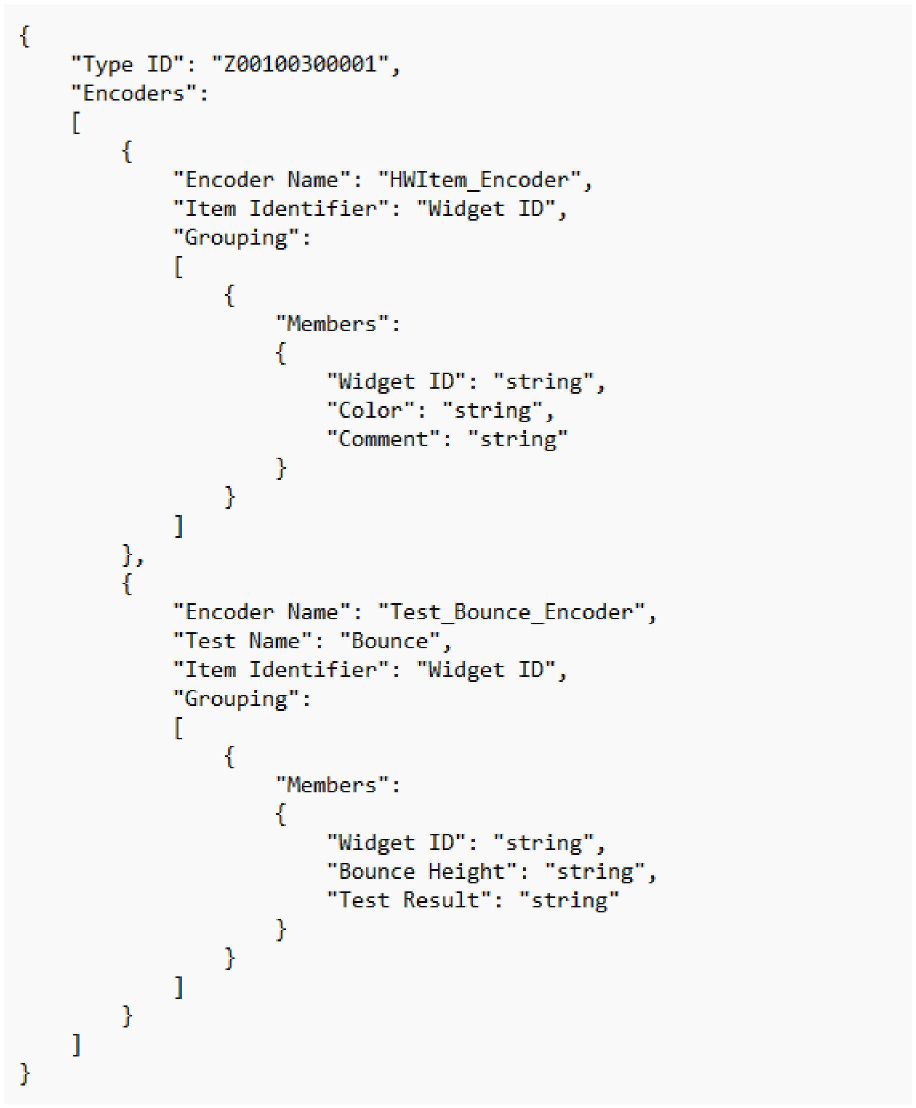
The spreadsheet “Test_Bounce.xlsx” is as follows:



	A	B	C
1	Widget ID	Bounce Height	Test Result
2	S00001	1.001	Pass
3	S00002	0.997	Fail
4	S00003	1.211	Fail

Figure 13: Example 2 Test Spreadsheet

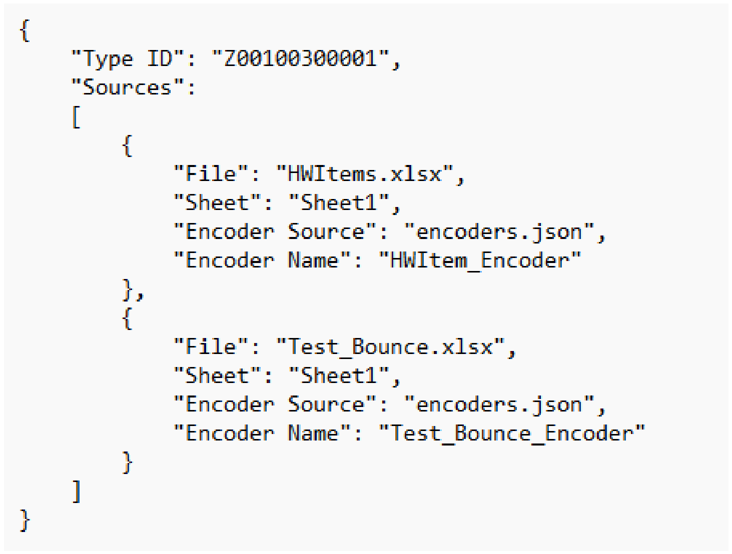
We will add a second encoder to “encoders.json”:



```
{
    "Type ID": "Z00100300001",
    "Encoders":
    [
        {
            "Encoder Name": "HWItem_Encoder",
            "Item Identifier": "Widget ID",
            "Grouping":
            [
                {
                    "Members":
                    {
                        "Widget ID": "string",
                        "Color": "string",
                        "Comment": "string"
                    }
                }
            ]
        },
        {
            "Encoder Name": "Test_Bounce_Encoder",
            "Test Name": "Bounce",
            "Item Identifier": "Widget ID",
            "Grouping":
            [
                {
                    "Members":
                    {
                        "Widget ID": "string",
                        "Bounce Height": "string",
                        "Test Result": "string"
                    }
                }
            ]
        }
    ]
}
```

Figure 14: Example 2 Encoder File

Finally, we’ll add the test spreadsheet to “docket.json”:

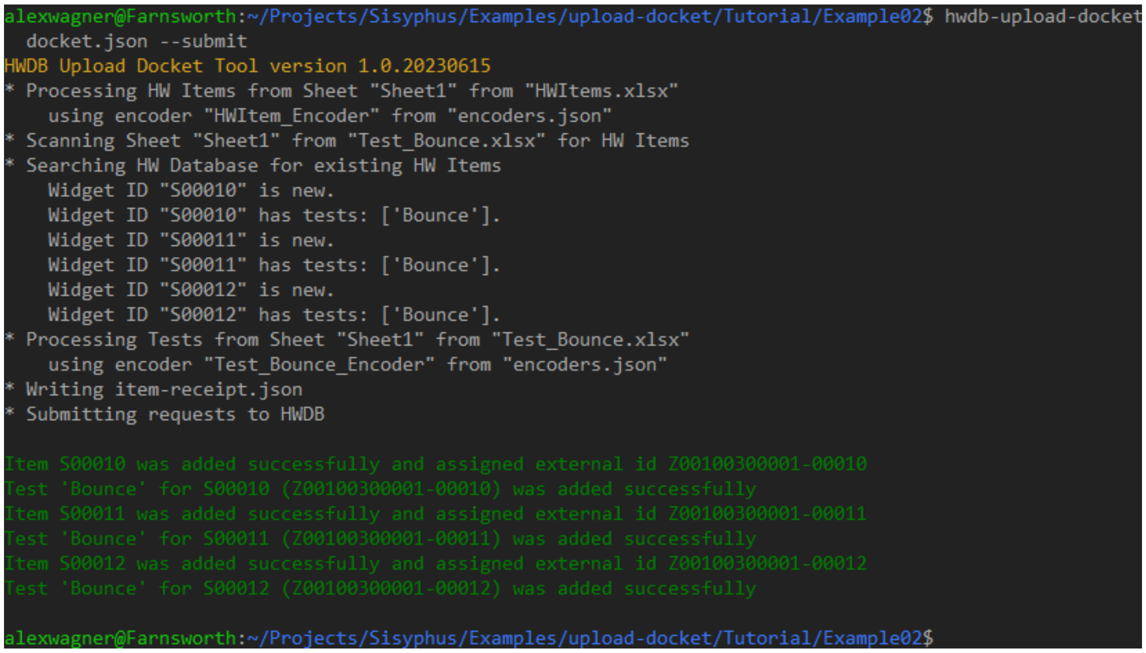


```
{
    "Type ID": "Z00100300001",
    "Sources":
    [
        {
            "File": "HWItems.xlsx",
            "Sheet": "Sheet1",
            "Encoder Source": "encoders.json",
            "Encoder Name": "HWItem_Encoder"
        },
        {
            "File": "Test_Bounce.xlsx",
            "Sheet": "Sheet1",
            "Encoder Source": "encoders.json",
            "Encoder Name": "Test_Bounce_Encoder"
        }
    ]
}
```

Figure 15: Example 2 Docket File

Uploading an HW Item with Tests

We may now upload this example:



```
alexwagner@Farnsworth:~/Projects/Sisyphus/Examples/upload-docket/Tutorial/Example02$ hwdb-upload-docket
  docket.json --submit
HWDB Upload Docket Tool version 1.0.20230615
* Processing HW Items from Sheet "Sheet1" from "HWItems.xlsx"
    using encoder "HWItem_Encoder" from "encoders.json"
* Scanning Sheet "Sheet1" from "Test_Bounce.xlsx" for HW Items
* Searching HW Database for existing HW Items
    Widget ID "S00010" is new.
    Widget ID "S00010" has tests: ['Bounce'].
    Widget ID "S00011" is new.
    Widget ID "S00011" has tests: ['Bounce'].
    Widget ID "S00012" is new.
    Widget ID "S00012" has tests: ['Bounce'].
* Processing Tests from Sheet "Sheet1" from "Test_Bounce.xlsx"
    using encoder "Test_Bounce_Encoder" from "encoders.json"
* Writing item-receipt.json
* Submitting requests to HWDB

Item S00010 was added successfully and assigned external id Z00100300001-00010
Test 'Bounce' for S00010 (Z00100300001-00010) was added successfully
Item S00011 was added successfully and assigned external id Z00100300001-00011
Test 'Bounce' for S00011 (Z00100300001-00011) was added successfully
Item S00012 was added successfully and assigned external id Z00100300001-00012
Test 'Bounce' for S00012 (Z00100300001-00012) was added successfully

alexwagner@Farnsworth:~/Projects/Sisyphus/Examples/upload-docket/Tutorial/Example02$
```

Figure 16: Example 2 Uploading the Data

Example 3 (Complex Case)

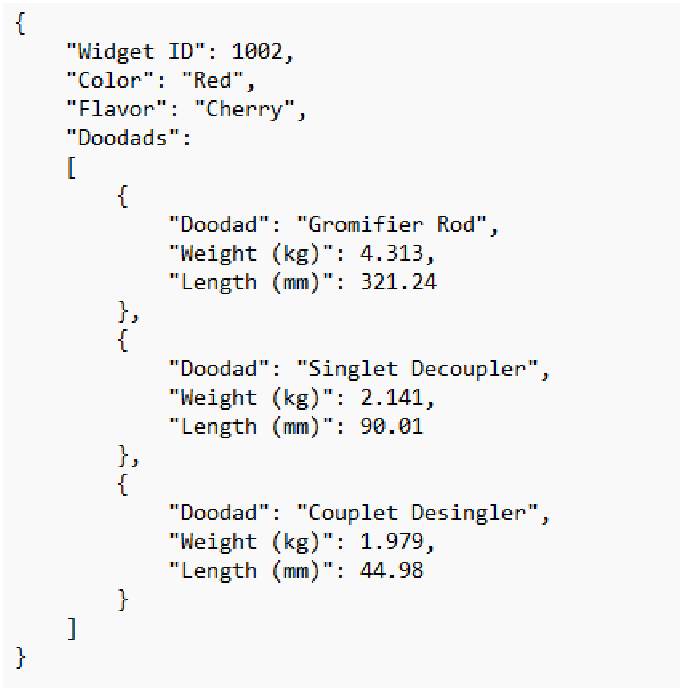
The Upload Tool is able to handle more complex situations than the ones previously described.

Let's consider the following example.

HW Item Datasheet

Suppose we have a component type representing a "widget," and each widget contains several "doodads." One possibility, of course, is to make "doodad" a completely different component type in the database, and then link them as subcomponents, but let's suppose that the doodads are intrinsic enough to the widget that it doesn't make sense to consider them as entirely separate entities, but important enough to be represented individually in the widget specification.

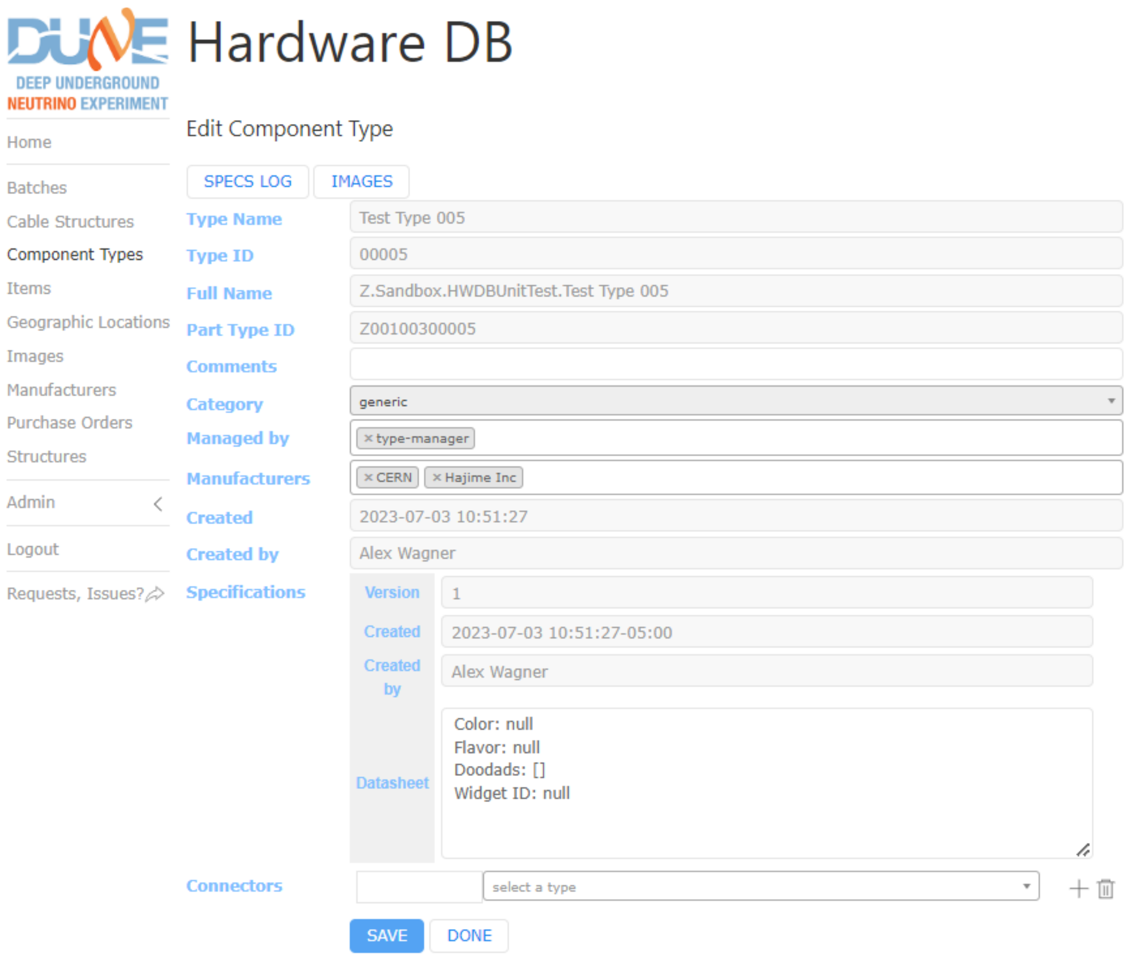
The specification, therefore, might be best represented as a hierarchical structure instead of a simple list of fields and values, e.g.,



```
{
    "Widget ID": 1002,
    "Color": "Red",
    "Flavor": "Cherry",
    "Doodads":
    [
        {
            "Doodad": "Gromifier Rod",
            "Weight (kg)": 4.313,
            "Length (mm)": 321.24
        },
        {
            "Doodad": "Singlet Decoupler",
            "Weight (kg)": 2.141,
            "Length (mm)": 90.01
        },
        {
            "Doodad": "Couplet Desingler",
            "Weight (kg)": 1.979,
            "Length (mm)": 44.98
        }
    ]
}
```

Figure 17: Example 3 HW Item Structure

For this sort of hierarchical structure, only the top-level fields need to be represented in the Datasheet for the HW Item:



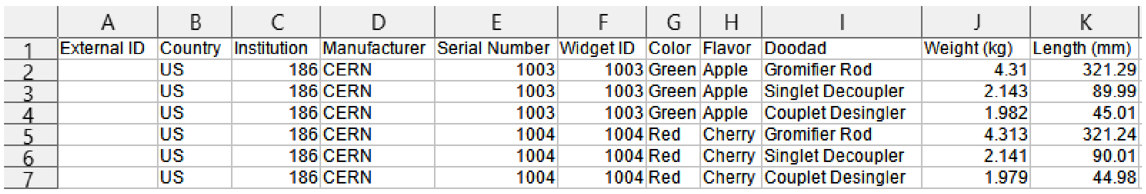
Edit Component Type

Type Name	Test Type 005
Type ID	00005
Full Name	Z.Sandbox.HWDBUnitTest.Test Type 005
Part Type ID	Z00100300005
Comments	
Category	generic
Managed by	× type-manager
Manufacturers	× CERN × Hajime Inc
Created	2023-07-03 10:51:27
Created by	Alex Wagner
Specifications	Version: 1 Created: 2023-07-03 10:51:27-05:00 Created by: Alex Wagner Datasheet: Color: null Flavor: null Doodads: [] Widget ID: null
Connectors	select a type

Figure 18: Example 3 HW Item Specifications

HW Item Data

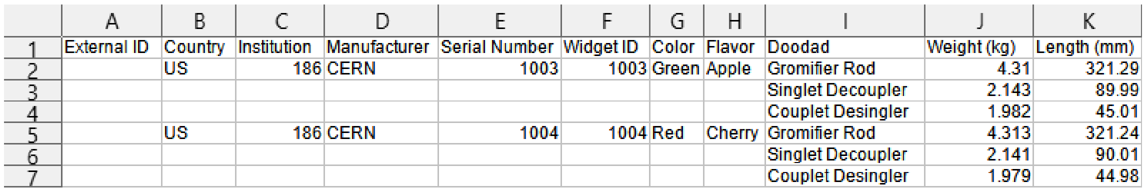
The HW Item spreadsheet needs to reflect the hierarchical structure of the data. One fairly intuitive way is to have the top-level and secondary-level fields as columns in the same sheet, repeating the top-level values as needed to indicate that the secondary-level values are subordinate to them:



	A	B	C	D	E	F	G	H	I	J	K
1	External ID	Country	Institution	Manufacturer	Serial Number	Widget ID	Color	Flavor	Doodad	Weight (kg)	Length (mm)
2		US	186	CERN	1003	1003	Green	Apple	Gromifier Rod	4.31	321.29
3		US	186	CERN	1003	1003	Green	Apple	Singlet Decoupler	2.143	89.99
4		US	186	CERN	1003	1003	Green	Apple	Couplet Desingler	1.982	45.01
5		US	186	CERN	1004	1004	Red	Cherry	Gromifier Rod	4.313	321.24
6		US	186	CERN	1004	1004	Red	Cherry	Singlet Decoupler	2.141	90.01
7		US	186	CERN	1004	1004	Red	Cherry	Couplet Desingler	1.979	44.98

Figure 19: Example 3 HW Item Data

The utility does allow, however, for duplicate information for the top-level data to be eliminated in subsequent rows:



	A	B	C	D	E	F	G	H	I	J	K
1	External ID	Country	Institution	Manufacturer	Serial Number	Widget ID	Color	Flavor	Doodad	Weight (kg)	Length (mm)
2		US	186	CERN	1003	1003	Green	Apple	Gromifier Rod	4.31	321.29
3									Singlet Decoupler	2.143	89.99
4									Couplet Desingler	1.982	45.01
5		US	186	CERN	1004	1004	Red	Cherry	Gromifier Rod	4.313	321.24
6									Singlet Decoupler	2.141	90.01
7									Couplet Desingler	1.979	44.98

Figure 20: Example 3 HW Item Data, "sparse" format

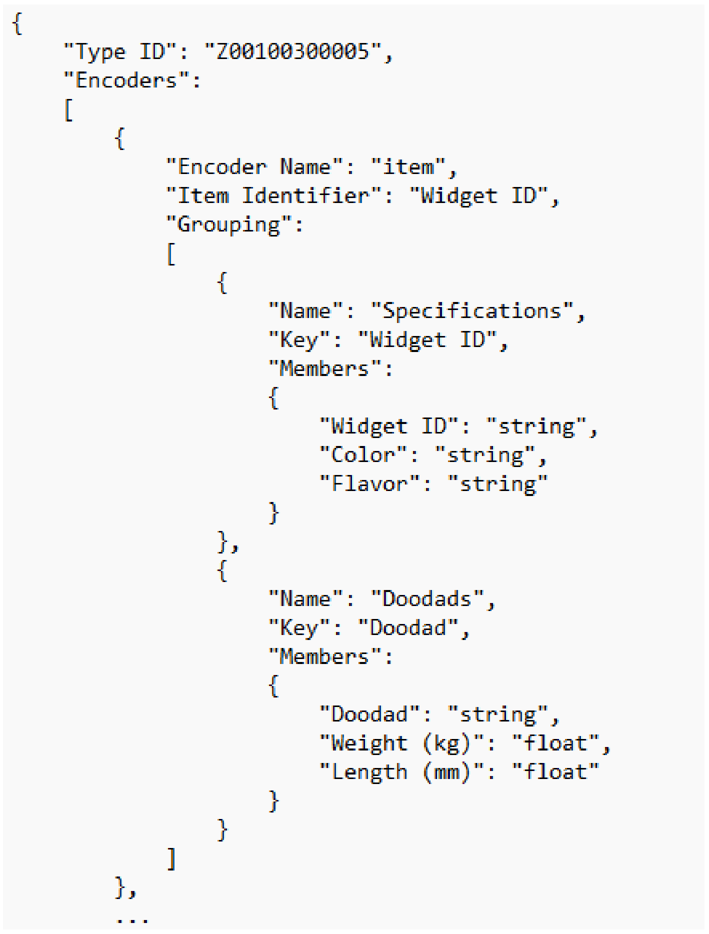
Either format may be used (with no changes to the encoder), but the code in the Tutorials directory uses the second.

HW Item Encoder

It is possibly confusing that the field “Doodads” (plural) from the Datasheet does not appear in our spreadsheet. “Doodads” here specifies a container or “Grouping” for the secondary-level data.

Because of the two-level hierarchy, our HW Item encoder for this example must contain two groups. The first group specifies which fields belong to the top-level structure, and which of those fields is best suited to act as a “key” for those data. In this case, “Color” and “Flavor” do not appreciably contribute to uniquely identifying a set of data, so “Widget ID” is the only reasonable choice. We have given the first group a name of “Specifications,” because that is where the data belongs in the HW Item record in the HWDB, but it is completely optional to give the first group a name, and giving it a different name will not affect the functionality of the upload tool.

Because our Datasheet has the secondary-level data under “Doodads”, the second group must be named that as well. Because this is the final grouping for this example, it is not strictly necessary to identify a key, because there are no further levels of data that need it to determine their place in the structure.



```
{
    "Type ID": "Z00100300005",
    "Encoders":
    [
        {
            "Encoder Name": "item",
            "Item Identifier": "Widget ID",
            "Grouping":
            [
                {
                    "Name": "Specifications",
                    "Key": "Widget ID",
                    "Members":
                    {
                        "Widget ID": "string",
                        "Color": "string",
                        "Flavor": "string"
                    }
                },
                {
                    "Name": "Doodads",
                    "Key": "Doodad",
                    "Members":
                    {
                        "Doodad": "string",
                        "Weight (kg)": "float",
                        "Length (mm)": "float"
                    }
                }
            ]
        },
        ...
```

Figure 21: Example 3 HW Item Encoder

Test Datasheet

For this example, let’s make the “Bounce” test more complex.

Let’s suppose that the widget is dropped at several different temperatures at a number of different heights. At each temperature, we are summarizing the results of the various heights into a single average value. We therefore have two different sheets, one with the average value, and another containing the details that were summarized. (We could still have done these as a single sheet, but we’ll do it as separate sheets for the sake of an example.)

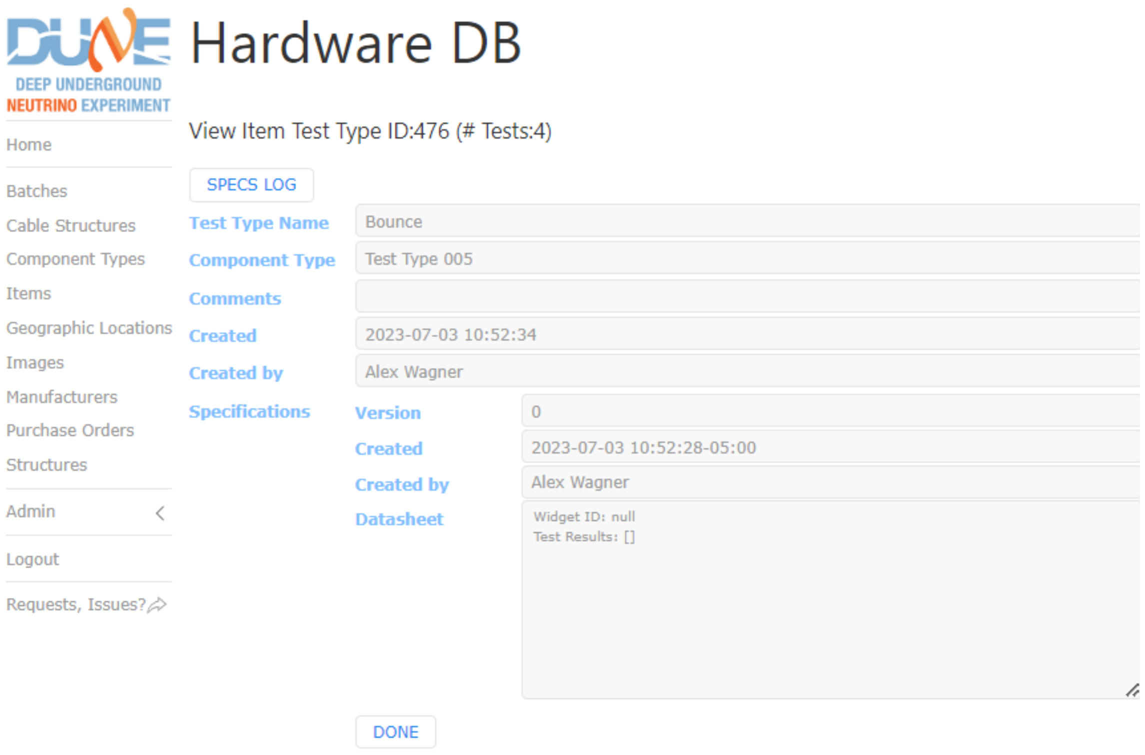
To further complicate matters, let’s suppose that our detail sheets contain only one HW Item each, while our summary sheet contains multiple HW Items.

The desired resulting data structure may appear as follows:

```
{
    "Widget ID": 1003,
    "Test Results":
    [
        {
            "Test ID": "20221106-01",
            "Operator": "Alex",
            "Timestamp": "11/06/22 01:48:12 PM",
            "Temperature (K)": 295,
            "Average Elasticity": 0.8493,
            "Details":
            [
                {
                    "Timestamp": "2022-11-06 01:45:06 PM",
                    "Drop Height (cm)": 25,
                    "Bounce Height (cm)": 21.16,
                    "Elasticity": 0.8464
                },
                ... (more detail rows) ...
            ]
        },
        ... (more summary rows) ...
    ]
}
```

Figure 22: Example 3 Test Data Structure

In the Web UI, the Datasheet for this is very simple. We only need the top-level fields; in this case, "Widget ID" and "Test Results".



View Item Test Type ID:476 (# Tests:4)

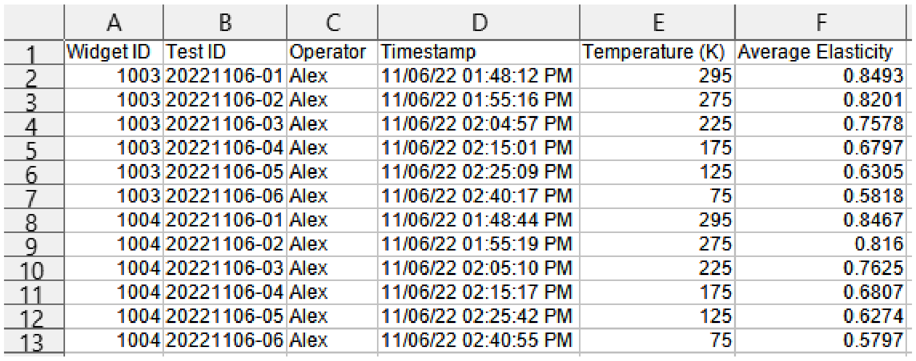
Test Type Name	Bounce	
Component Type	Test Type 005	
Comments		
Created	2023-07-03 10:52:34	
Created by	Alex Wagner	
Specifications	Version	0
	Created	2023-07-03 10:52:28-05:00
	Created by	Alex Wagner
	Datasheet	Widget ID: null Test Results: []

Figure 23: Example 3 Test Specification

Test Data Spreadsheets

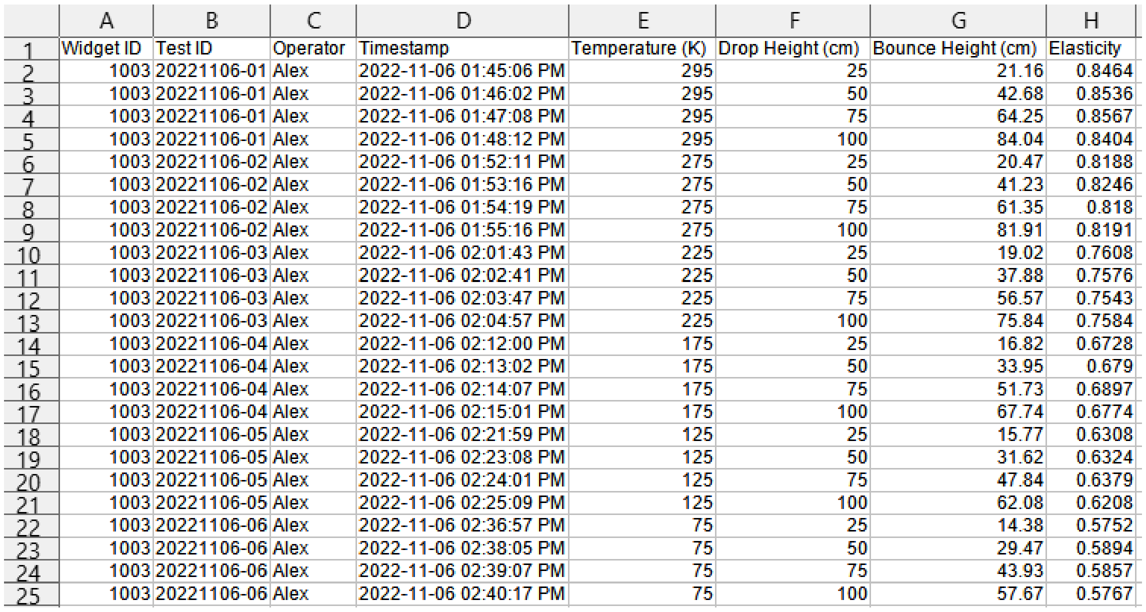
The test data for this example is contained in three spreadsheets. The format for the two subtables is the same.

Note that since the subtable rows need to attach to the main table, enough of the main table columns need to be duplicated in the subtable to indicate ownership.



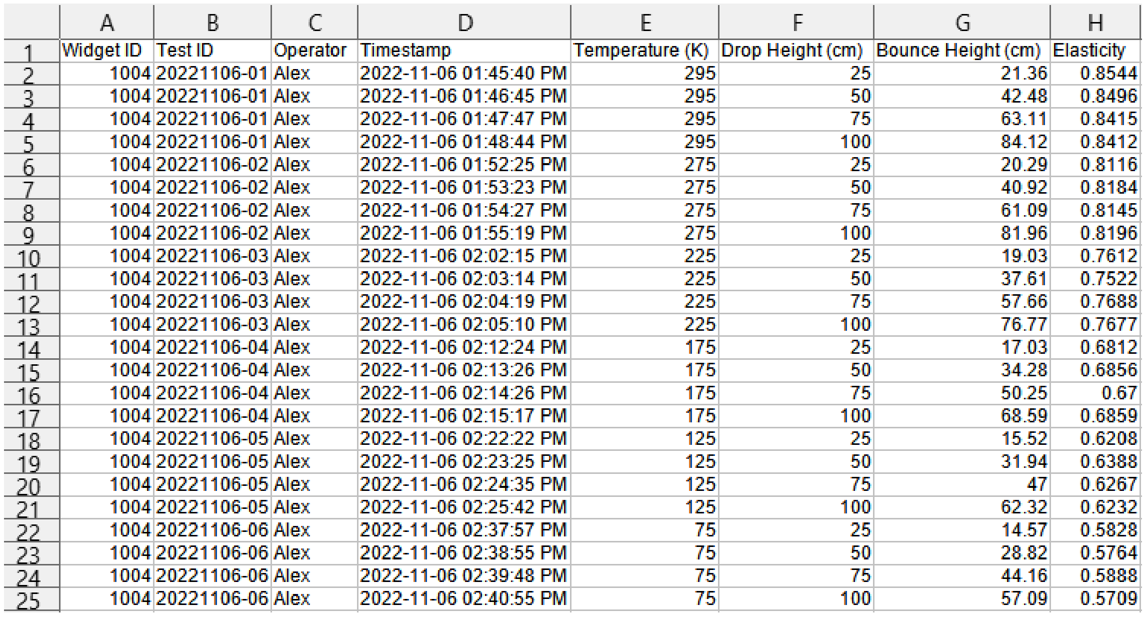
	A	B	C	D	E	F
1	Widget ID	Test ID	Operator	Timestamp	Temperature (K)	Average Elasticity
2	1003	20221106-01	Alex	11/06/22 01:48:12 PM	295	0.8493
3	1003	20221106-02	Alex	11/06/22 01:55:16 PM	275	0.8201
4	1003	20221106-03	Alex	11/06/22 02:04:57 PM	225	0.7578
5	1003	20221106-04	Alex	11/06/22 02:15:01 PM	175	0.6797
6	1003	20221106-05	Alex	11/06/22 02:25:09 PM	125	0.6305
7	1003	20221106-06	Alex	11/06/22 02:40:17 PM	75	0.5818
8	1004	20221106-01	Alex	11/06/22 01:48:44 PM	295	0.8467
9	1004	20221106-02	Alex	11/06/22 01:55:19 PM	275	0.816
10	1004	20221106-03	Alex	11/06/22 02:05:10 PM	225	0.7625
11	1004	20221106-04	Alex	11/06/22 02:15:17 PM	175	0.6807
12	1004	20221106-05	Alex	11/06/22 02:25:42 PM	125	0.6274
13	1004	20221106-06	Alex	11/06/22 02:40:55 PM	75	0.5797

Figure 24: test-bounce.xlsx



	A	B	C	D	E	F	G	H
1	Widget ID	Test ID	Operator	Timestamp	Temperature (K)	Drop Height (cm)	Bounce Height (cm)	Elasticity
2	1003	20221106-01	Alex	2022-11-06 01:45:06 PM	295	25	21.16	0.8464
3	1003	20221106-01	Alex	2022-11-06 01:46:02 PM	295	50	42.68	0.8536
4	1003	20221106-01	Alex	2022-11-06 01:47:08 PM	295	75	64.25	0.8567
5	1003	20221106-01	Alex	2022-11-06 01:48:12 PM	295	100	84.04	0.8404
6	1003	20221106-02	Alex	2022-11-06 01:52:11 PM	275	25	20.47	0.8188
7	1003	20221106-02	Alex	2022-11-06 01:53:16 PM	275	50	41.23	0.8246
8	1003	20221106-02	Alex	2022-11-06 01:54:19 PM	275	75	61.35	0.818
9	1003	20221106-02	Alex	2022-11-06 01:55:16 PM	275	100	81.91	0.8191
10	1003	20221106-03	Alex	2022-11-06 02:01:43 PM	225	25	19.02	0.7608
11	1003	20221106-03	Alex	2022-11-06 02:02:41 PM	225	50	37.88	0.7576
12	1003	20221106-03	Alex	2022-11-06 02:03:47 PM	225	75	56.57	0.7543
13	1003	20221106-03	Alex	2022-11-06 02:04:57 PM	225	100	75.84	0.7584
14	1003	20221106-04	Alex	2022-11-06 02:12:00 PM	175	25	16.82	0.6728
15	1003	20221106-04	Alex	2022-11-06 02:13:02 PM	175	50	33.95	0.679
16	1003	20221106-04	Alex	2022-11-06 02:14:07 PM	175	75	51.73	0.6897
17	1003	20221106-04	Alex	2022-11-06 02:15:01 PM	175	100	67.74	0.6774
18	1003	20221106-05	Alex	2022-11-06 02:21:59 PM	125	25	15.77	0.6308
19	1003	20221106-05	Alex	2022-11-06 02:23:08 PM	125	50	31.62	0.6324
20	1003	20221106-05	Alex	2022-11-06 02:24:01 PM	125	75	47.84	0.6379
21	1003	20221106-05	Alex	2022-11-06 02:25:09 PM	125	100	62.08	0.6208
22	1003	20221106-06	Alex	2022-11-06 02:36:57 PM	75	25	14.38	0.5752
23	1003	20221106-06	Alex	2022-11-06 02:38:05 PM	75	50	29.47	0.5894
24	1003	20221106-06	Alex	2022-11-06 02:39:07 PM	75	75	43.93	0.5857
25	1003	20221106-06	Alex	2022-11-06 02:40:17 PM	75	100	57.67	0.5767

Figure 25: test-bounce-subtable-1003.xlsx



	A	B	C	D	E	F	G	H
1	Widget ID	Test ID	Operator	Timestamp	Temperature (K)	Drop Height (cm)	Bounce Height (cm)	Elasticity
2	1004	20221106-01	Alex	2022-11-06 01:45:40 PM	295	25	21.36	0.8544
3	1004	20221106-01	Alex	2022-11-06 01:46:45 PM	295	50	42.48	0.8496
4	1004	20221106-01	Alex	2022-11-06 01:47:47 PM	295	75	63.11	0.8415
5	1004	20221106-01	Alex	2022-11-06 01:48:44 PM	295	100	84.12	0.8412
6	1004	20221106-02	Alex	2022-11-06 01:52:25 PM	275	25	20.29	0.8116
7	1004	20221106-02	Alex	2022-11-06 01:53:23 PM	275	50	40.92	0.8184
8	1004	20221106-02	Alex	2022-11-06 01:54:27 PM	275	75	61.09	0.8145
9	1004	20221106-02	Alex	2022-11-06 01:55:19 PM	275	100	81.96	0.8196
10	1004	20221106-03	Alex	2022-11-06 02:02:15 PM	225	25	19.03	0.7612
11	1004	20221106-03	Alex	2022-11-06 02:03:14 PM	225	50	37.61	0.7522
12	1004	20221106-03	Alex	2022-11-06 02:04:19 PM	225	75	57.66	0.7688
13	1004	20221106-03	Alex	2022-11-06 02:05:10 PM	225	100	76.77	0.7677
14	1004	20221106-04	Alex	2022-11-06 02:12:24 PM	175	25	17.03	0.6812
15	1004	20221106-04	Alex	2022-11-06 02:13:26 PM	175	50	34.28	0.6856
16	1004	20221106-04	Alex	2022-11-06 02:14:26 PM	175	75	50.25	0.67
17	1004	20221106-04	Alex	2022-11-06 02:15:17 PM	175	100	68.59	0.6859
18	1004	20221106-05	Alex	2022-11-06 02:22:22 PM	125	25	15.52	0.6208
19	1004	20221106-05	Alex	2022-11-06 02:23:25 PM	125	50	31.94	0.6388
20	1004	20221106-05	Alex	2022-11-06 02:24:35 PM	125	75	47	0.6267
21	1004	20221106-05	Alex	2022-11-06 02:25:42 PM	125	100	62.32	0.6232
22	1004	20221106-06	Alex	2022-11-06 02:37:57 PM	75	25	14.57	0.5828
23	1004	20221106-06	Alex	2022-11-06 02:38:55 PM	75	50	28.82	0.5764
24	1004	20221106-06	Alex	2022-11-06 02:39:48 PM	75	75	44.16	0.5888
25	1004	20221106-06	Alex	2022-11-06 02:40:55 PM	75	100	57.09	0.5709

Figure 26: test-bounce-subtable-1004.xlsx

Test Encoders

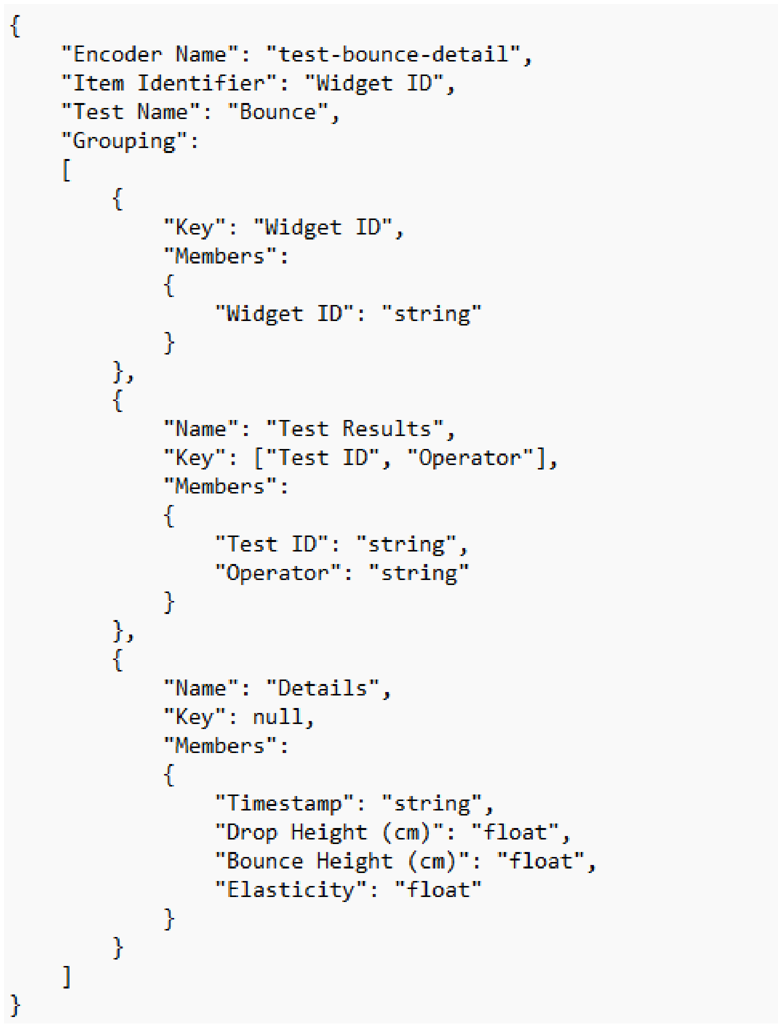
We need two encoders for tests for this example.

The first encoder manages the main sheet.

```
{
    "Encoder Name": "test-bounce",
    "Item Identifier": "Widget ID",
    "Test Name": "Bounce",
    "Grouping":
    [
        {
            "Name": "<Main>",
            "Key": "Widget ID",
            "Members":
            {
                "Widget ID": "string"
            }
        },
        {
            "Name": "Test Results",
            "Key": ["Test ID", "Operator"],
            "Members":
            {
                "Test ID": "string",
                "Operator": "string",
                "Timestamp": "string",
                "Temperature (K)": "float",
                "Average Elasticity": "float"
            }
        }
    ]
},
```

Figure 27: "test-bounce" encoder

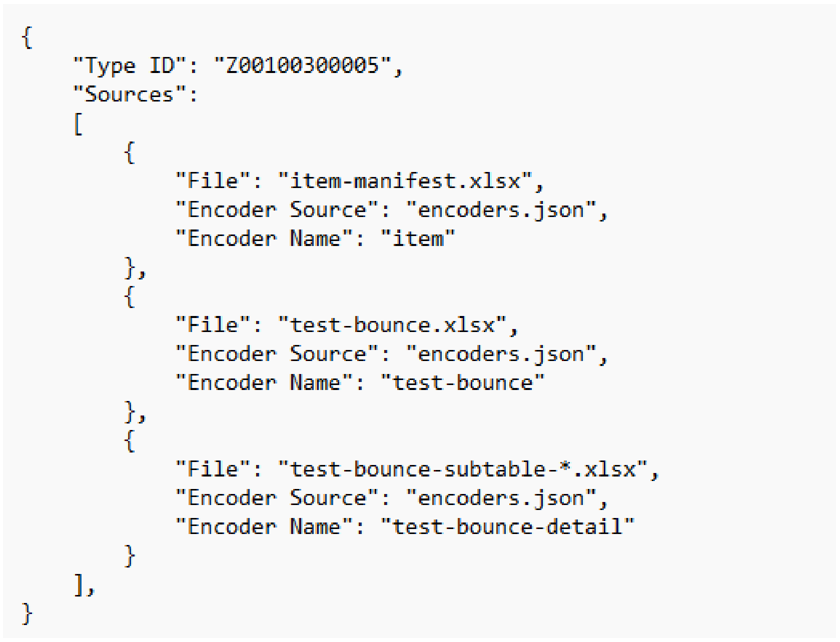
The second encoder manages the detail. Note that the “grouping” of the main sheet is duplicated, at least as far as the “key” fields go. This is necessary for the “detail” contained in the rows to find the place in the structure hierarchy where it belongs.



```
{
    "Encoder Name": "test-bounce-detail",
    "Item Identifier": "Widget ID",
    "Test Name": "Bounce",
    "Grouping":
    [
        {
            "Key": "Widget ID",
            "Members":
            {
                "Widget ID": "string"
            }
        },
        {
            "Name": "Test Results",
            "Key": ["Test ID", "Operator"],
            "Members":
            {
                "Test ID": "string",
                "Operator": "string"
            }
        },
        {
            "Name": "Details",
            "Key": null,
            "Members":
            {
                "Timestamp": "string",
                "Drop Height (cm)": "float",
                "Bounce Height (cm)": "float",
                "Elasticity": "float"
            }
        }
    ]
}
```

Figure 28: "test-bounce-detail" encoder

Docket



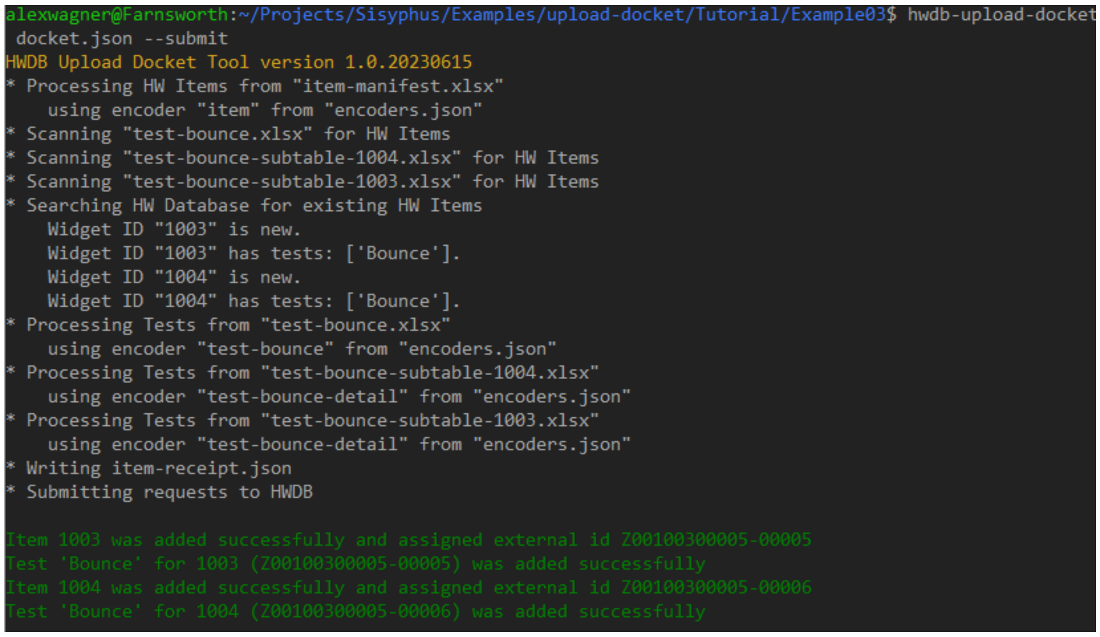
```
{
    "Type ID": "Z00100300005",
    "Sources":
    [
        {
            "File": "item-manifest.xlsx",
            "Encoder Source": "encoders.json",
            "Encoder Name": "item"
        },
        {
            "File": "test-bounce.xlsx",
            "Encoder Source": "encoders.json",
            "Encoder Name": "test-bounce"
        },
        {
            "File": "test-bounce-subtable-*.xlsx",
            "Encoder Source": "encoders.json",
            "Encoder Name": "test-bounce-detail"
        }
    ],
}
```

Figure 29: docket.json

Note that since there is more than one detail sheet, and they are similarly-named, we are able to use file globbing to include all files matching a pattern.

Uploading the Docket

We are now ready to upload the data to the database!



```
alexwagner@Farnsworth:~/Projects/Sisyphus/Examples/upload-docket/Tutorial/Example03$ hwdb-upload-docket
 docket.json --submit
HWDB Upload Docket Tool version 1.0.20230615
* Processing HW Items from "item-manifest.xlsx"
    using encoder "item" from "encoders.json"
* Scanning "test-bounce.xlsx" for HW Items
* Scanning "test-bounce-subtable-1004.xlsx" for HW Items
* Scanning "test-bounce-subtable-1003.xlsx" for HW Items
* Searching HW Database for existing HW Items
    Widget ID "1003" is new.
    Widget ID "1003" has tests: ['Bounce'].
    Widget ID "1004" is new.
    Widget ID "1004" has tests: ['Bounce'].
* Processing Tests from "test-bounce.xlsx"
    using encoder "test-bounce" from "encoders.json"
* Processing Tests from "test-bounce-subtable-1004.xlsx"
    using encoder "test-bounce-detail" from "encoders.json"
* Processing Tests from "test-bounce-subtable-1003.xlsx"
    using encoder "test-bounce-detail" from "encoders.json"
* Writing item-receipt.json
* Submitting requests to HWDB

Item 1003 was added successfully and assigned external id Z00100300005-00005
Test 'Bounce' for 1003 (Z00100300005-00005) was added successfully
Item 1004 was added successfully and assigned external id Z00100300005-00006
Test 'Bounce' for 1004 (Z00100300005-00006) was added successfully
```

Figure 30: Example 3 Uploading Data