META Material Library User Manual

Vanderbilt University

February 5, 2014

1 Introduction

The material library is a set of data which fully describes the materials used for components in the META tools. This library is contained within a JSON file on the user's local computer, and it contains relevant data about these materials, such as its density, electrical resistivity, and shear strength. This document allows the users to access whatever properties they need for their programs. A Python Application Programming Interface (API) has been provided. The META Material Library is included in the META tools installer. The files are installed in the Public Documents\META Documents\MaterialLibrary folder. Batch files are installed along with the META Material Library to perform the API functions from the command line.

2 Material Definitions for Creo®

Creo® .mtl files are installed with the META tools installer to make the process of assigning material names and properties in Creo® simpler. All components, to be analyzed in the META tools, must be assigned a material name from the META Material Library. The .mtl files are installed in Public Documents\META Documents\MaterialLibrary\MATERIALS_CREO_MTL. META Link automatically sets the material file location in Creo® making the process of setting materials easier. If you open Creo® outside of META Link, you may navigate to the MATERIALS_CREO_MTL folder each time you assign a material. Alternatively, you may change the default location for materials in Creo® by using the following procedure:

1. Select FILE:OPTIONS from the pull-down menu
2. Select CONFIGURATION EDITOR
3. Select FIND
4. In blank, type MATERIAL and hit FIND NOW
5. Select PRO_MATERIAL_DIR
6. Under SET VALUE, select BROWSE and navigate to:
   Public Documents\META Documents\MaterialLibrary\MATERIALS_CREO_MTL
7. Select ADD/CHANGE
8. Select CLOSE
9. To make it permanent, select IMPORT/EXPORT
10. Choose EXPORT
11. Browse to your existing CONFIG.PRO file, select it, and hit OK
12. Select OK, and you are done

3 Python API

The Demo.py program, co-located with the material_library.json file, provides example Python code for accessing the META Material Library from Python. Batch files are installed along with the META Material Library to perform the API functions from the command line. From Python, you need to first import the LibraryManager from the MaterialLibraryInterface package:

```
from MaterialLibraryInterface import LibraryManager
```

3.1 Constructor

To retrieve the data from the JSON file, construct the LibraryManager object. The constructor receives one argument - the location of the JSON file (relative or absolute), and it returns an instance of the Library-Manager class. If you don't specify an argument, the default argument is the location of the material library JSON included in the META tools installation.

```
library_manager = LibraryManager()
```

3.2 List Properties of a Specific Material

Perhaps the most useful of the functions provided, this function returns a dictionary of a material's properties. It's easiest to demonstrate how this function works:

```
steelData = library_manager.materialData("steel_stainless_316_l")
```

The variable "steelData" now contains all the data related to the "steel_stainless_316_l" material. How is this data represented? Take a quick look at the text representation (if you try this, be sure to import pprint):

```
pprint(steelData)
```

This gives the following output:

```
{u'capacity_specific_heat': {u'unit': u'J/K.kg', u'value': 510.0},
 u'coefficient_expansion_linear': {u'unit': u'1/K', u'value': u'None'},
 u'coefficient_expansion_volumetric': {u'unit': u'1/K', u'value': u'None'},
 u'conductivity': {u'unit': u'W/K.m', u'value': 15.0},
 u'density': {u'unit': u'kg/m3', u'value': 7970.0},
 u'elongation': {u'value': u'0.4'},
 u'enthalpy_of_fusion': {u'unit': u'J/kg', u'value': u'None'},
...
...
 u'weldable': {u'value': u'None'}}
```

The data is represented as a dictionary, where the keys are strings representing the properties of the material. The keys are mapped to dictionaries, which contain the property's value and (if applicable) its units. Note that the `u'` in front of the strings signifies that they're unicode strings. To access the data, index into the dictionary:

```
pprint(steelData["density"])
```

This returns another dictionary:

```
{u'unit': u'kg/m3', u'value': 7970.0}
```

Index into this dictionary to retrive the value of the density.

```
densityValue = steelData["density"]["value"]
densityUnit = steelData["density"]["unit"]
```

Be sure to keep a few ideas in mind:

- A value can be "None,' meaning that you can receive a string with the value "None" when you were expecting a float. This usually only happens when a property is not applicable for a certain material. Especially for this reason, always make sure the output has the datatype you expect. You can even do the following:

  ```
  if type(densityValue) is not float:
      print "Not expected data type!"
  ```

- Every property has a “value” entry, but not every property has a “unit” entry.
- Always check that a dictionary entry exists before attempting to index into it, if you want to avoid raising exceptions.

If you want to iterate through the dictionary, printing the value and unit of every entry whose value is not “None,” you might try the following code:

```
for keys in steelData:
    unit = ""
    if "unit" in steelData[keys]:
        unit = steelData[keys]["unit"]
        if steelData[keys]["value"] != "None":
            print keys + " = " + str(steelData[keys]["value"]) + " " + unit
```

The materialData function is intended to be flexible so that users can retrieve the data they need in whatever fashion or format they desire.

3.3 List All Properties

The method listAllProperties returns a list of all properties that a material could possess. Not every property applies to every material (like weldability), but a user can make a list of properties such that for each property on the list, at least one material in the library has that property. Call this method as follows:

```
propList = material_library.listAllProperties()
```

The return value, assigned to the variable ”propList,” is a list of strings with the material properties in the order they are listed in the library. Iterate through them as follows:

```
numProperties = 0
for prop in propList:
    print prop
    numProperties += 1
```

You can also check to see if a property exists:

```
if "density" in propList:
        print "Density is in the list of properties"
        return True
```

As you would expect, you can operate it as you would any list of strings.

3.4 List All Materials

You’re also able to list all the materials in the material library. Just like the last example, this function returns a list of strings.

```
allMaterials = material_library.listAllMaterials()
```

You can iterate through those strings, just as in the last example.

```
for mat in allMaterials:
    print mat
```

3.5 Checking for Updates

To see if a new version is available, call the checkVersion function. Specify a username and password to authenticate access to the VehicleFORGE servers. Returns the version number of the online material library. An optional parameter lets the user specify the URL, in case the default URL (which poitns to beta.vehicleforge.org) isn't the correct one.

```
print "Checking version number: "
version = library_manager.checkVersion(username, password)
print "Version: " + str(version)
```

3.6 Updating to a New Version

To update to the newest version of the material library, use the updateJSON function. Again, specify a username and password to authenticate access. This function first calls the checkVersion function to find the version of online material library. Then, if the online version of the library is newer, it downloads the newest version and saves it over the local version of the library. An optional parameter lets the user specify the URL, in case the default URL (which poitns to beta.vehicleforge.org) isn't the correct one.

```
library_manager.updateJSON(username, password)
```