thinknode™ Examples

These examples provide a starting point for issuing http connections and requests to the dicom app on the thinknode™ framework. They are provided as is, and are written in python. Any further dependencies are listed along with the provided scripts.

Python

Python: Overview

The provided python scripts and libraries are meant to be a foundation and starting point for using the astroid apps on the thinknode™ framework. The provided scripts outline the basics of using ISS to store objects, as well as constructing and making calculation requests to the calculation provider. The below sections detail the basic usage for each script.

Download: The python astroid_script_library can be downloaded from the .decimal GitHub repository.

thinknode.cfg

There is a simple configuration file (thinknode.cfg) that is used to store user data for connecting to the astroid app on the thinknode™ framework. This file is required by all scripts in the python astroid_script_library to authenticate and use the app. A sample file with no user data is available in the repository and the details of the information to include in the file are provided below.

- *basic_user* being a base64 encoded username and password. Refer to the thinknode documentation for more information.
- *api_url* being the connection string to the thinknode™ framework.
- *apps*
  - *app_name* being the current app name (e.g. dosimetry or dicom).
    - *app_version* being the current version of the app existing on the thinknode™ framework being used. If left blank the thinknode_worker will select the first app's version returned by the Realm Versions GET request.
    - *branch_name* not currently implemented
- *realm_name* thinknode realm
- *account_name* thinknode account name

thinknode.cfg

```
{
    "basic_user": "<Base64 encoded thinknode username:password>",
    "api_url": "https://<thinknode_account>.thinknode.io/api/v1.0",
```

```
    "apps":
    {
        "dosimetry":
        {
            "app_version": "1.0.0-beta1",
            "branch_name": "master"
        },
        "dicom":
        {
            "app_version": "",
            "branch_name": "master"
        },
        "rt_types":
        {
            "app_version": "",
            "branch_name": "master"
        }
    },
    "realm_name": "<thinknode realm>",
    "account_name": "<thinknode account>"
}
```

Python: decimal Libraries

rt_types

The *rt_types* module is a reconstruction of all astroid types in python class format. This includes interdependencies between types (e.g. the class “polyset” requires the class “polygon2”).

Each data type detailed in the astroid Manifest Guide has a corresponding class in this python module.

Below you will see a snippet from the rt_types module that shows the class for the *polyset* rt_type along with its default initialization, *expand_data* and *from_json* functions.

```
class polygon2(object):

    #Initialize
    def __init__(self):
        blob = blob_type()
        self.vertices = blob.toStr()

    def expand_data(self):
        data = {}
        data['vertices'] =
```

```
parse_bytes_2d(base64.b64decode(self.vertices['blob']))
        return data

    def from_json(self, jdict):
        for k, v in jdict.items():
            if hasattr(self,k):
                setattr(self, k, v)

class polyset(object):

    #Initialize
    def __init__(self):
        self.polygons = []
        self.holes = []

    def expand_data(self):
        data = {}
        polygon = []
        for x in self.polygons:
            s = polygon2()
            s.from_json(x)
            polygon.append(s.expand_data())
        data['polygons'] = polygon
        hole = []
        for x in self.holes:
            s = polygon2()
            s.from_json(x)
            hole.append(s.expand_data())
        data['holes'] = hole
        return data

    def from_json(self, jdict):
        for k, v in jdict.items():
            if hasattr(self,k):
                setattr(self, k, v)
```

- **Interdependence:** When rt_types are constructed of other or multiple named types, they will be constructed as such in each class as seen in the *polygons* parameter of the *polyset* in the above example.
- ***expand_data* function:** Each class's *expand_data* function returns a python dictionary containing each of the values in the class, with all data values expanded out to remove compression or other encodings (i.e. providing results in a format more useful for send to other applications or for human-readability).
- ***from_json* function:** Each class's *from_json* function provides a method to turn a raw json string (e.g. a result from a thinknode calculation or ISS object) into an rt_type data type. Proper use is to first construct an empty class instance, then to call the *from_json* method on that instance, passing in the desired json data string.

Below is an example usage of getting a thinknode dose image (image_3d data type in the astroid manifest) and turning it into a rt_types image_3d data type, so that it can be expanded and then used to output the image into a VTK graphics file:

```
def dose_to_vtk(dose_id):
    img_data = json.loads(thinknode.get_immutable(iam, 'dicom', dose_id))

    img = rt_types.image_3d()
    img.from_json(img_data)
    img2 = img.expand_data()

    vtk.write_vtk_image3('E:/dicom/dose.vtk', img2)
```

dicom_worker