**DEVELOPER USER GUIDE:**

This is for developers who wants to build or use existing use cases/models in the API server. Feel free to branch and contribute, everyone in SAS benefits with progress :)

1. Checkout/Branch a copy of the stable version on Enterprise Git. Otherwise just use this copy if Git isn’t set up.

2. Change auth details specific to your VIYA instance, or point it to your own server (for Singapore, that’s our AFT box). Note that each server configuration differs (Race vs Global vs Local instance), and a slight change in network settings or IP affects things in alot of ways. Best is to use the same instance/environment!

3. The following files abstract away unnecessary CAS interactions so demo-building only needs perhaps 2-3 lines and some customization (i.e. load the data you want, change it to run a model to that specific dataset). For more advanced customization, you are likely to use the Models.py file - this is where you can link it to run models from a HTTP POST request, which are all found in Sassy.py. Just make sure you write an additional if/else to cater to your new model and returns it as per the TemplateFunction.

The files can be summarized as such:

1. Sassy.py - everything for HTTP and calling the right CAS functions. Treat this as the main connectivity file

2. Models.py - everything for your use cases and demos. In Sassy.py, the function Run() will call specific models from Models.py, so if you intend to do a new or more elaborate customization, do create another Model function. This keeps the rest of the models and linkage with Sassy intact, and somewhat more re-usable. Of course over time, there may be many other models, but these can be organized in other ways later on with python packages.  
  
Simply put in an IF/ELSE statement to recognize the ‘Model’ param passed in via JSON, and call the right function to run a model!  
  
Individual models are segmented and handled by its own function. If I have RunLoans\_V1 and RunLoans\_V2, I will create two separate functions and put two if/else statements to handle both. Additionally, Sassy passes in the connection object to each function if you want to do more custom stuff, just like interacting with CAS directly (read: SAS Viya, The Python Perspective by Kevin Smith & Xiangxiang Meng). This is likely needed in the future if processes are different from two clients with a similar use case.

3. DataHandler.py - This is meant to serve as a way to plumb data from one place to another quickly, and not meant as a permanent pipeline. A sample use case is piping on-premise, PostGre data into CAS once, then running models against the stored data for...ever. This also implies Sassy is deployed on a local instance, so there may be some breakage due to network/server config here. NOT COMPLETE NOR TESTED. Here be dragons!

4. The most important thing to take note is the formatting of what is returned. We do our best to return a json object with as much information as possible for front-end users to use. At the same time, they parameters/keywords used must be standardized according to CAS such that there is no need to keep a separate mapping or dictionary of these words used.

That being said, a rudimentary function to convert strings into readable output can be found in utility.py. This is to convert any json in xhttp form encoded into a dictionary (incoming, processing), and for unformatted CASOutput to be converted into JSON (outgoing, for consumption).

Thus, what will always be returned is a JSON, with important information accessible to frontend users via the keywords. Sassy.py should not be used to make dictionaries, only JSON - thus it is important to note that prior to returning a model's output, it should transform all intended outputs into string/int, assign it to as a key/value pair in default dictionary, and then returning this dictionary to Sassy.py to convert into JSON. This effecitvely abstracts data output definitions from http request/response logic.

An example can be found in Models.py, named TemplateModel. Use 'link','msg' and 'dFrame' as these keywords do not change from whichever output users would need to specify on their frontend. If you require additional parameters, simply add a new key/value pair in modelOutput before it returns.

For quick reference, refer to the example below or in Models.py. You should see something similar:  
def TemplateModel(conn):

link = "www.sas.com.sg"

msg = "Template model successfully executed!"

dFrame = None

modelOutput = defaultdict() #initialization of dictionary (aka Objects in JS)

modelOutput['link'] = link #always has a default link output. Can be used by #frontend to redirect

modelOutput['msg'] = msg #a default response to let user or developer know response is OK or not.

modelOutput['dFrame'] = dFrame #The data to be visualized

return modelOutput #Returning the dictionary. This will be parsed into JSON

#in Sassy.py before returning to frontend.

These are examples of the formatted output worked on so far:

a) TableInfo, in a json format:

"{'status': None, '\_OrderedDict\_\_root': [[[...], [...], u'TableInfo'], [[...], [...], u'TableInfo'], None], 'updateflags': set([]), 'severity': 0, 'status\_code': 0, '\_OrderedDict\_\_map': {u'TableInfo': [[[...], [...], None], [[...], [...], None], u'TableInfo']}, 'messages': [], 'reason': None, 'session': u'f55851c3-5f5a-a346-aa48-d430689cb1a1', 'sessionname': u'py-session-1', 'signature': {u'autoRetry': True, u'desc': u'Shows information about a table', u'params': [{u'name': u'name', u'isTableName': True, u'value': u'IRIS', u'label': u'Table', u'parmType': u'string', u'desc': u'specifies the table name.', u'type': 4, u'aliases': [u'table']}, {u'name': u'caslib', u'parmType': u'string', u'value': u'CASUSER(sasdemo01)', u'label': u'Caslib', u'isCasLib': True, u'type': 4, u'desc': u'specifies the caslib containing the table that you want to use with the action. By default, the active caslib is used. Specify a value only if you need to access a table from a different caslib.'}, {u'name': u'quiet', u'parmType': u'boolean', u'default': False, u'label': u'Silent', u'desc': u'when set to True, attempting to show information for a table that does not exist returns an OK status and severity. When set to False, attempting to show information for a table that does not exist returns an error.', u'type': 5, u'aliases': [u'silent']}], u'name': u'table.tableinfo', u'label': u'Table information'}, 'debug': None, 'performance': CASPerformance(cpu\_system\_time=0, cpu\_user\_time=0.00092, data\_movement\_bytes=0, data\_movement\_time=None, elapsed\_time=0.0007, memory=73184, memory\_os=None, memory\_quota=25546752, system\_cores=8, system\_nodes=1, system\_total\_memory=67389382656), 'events': OrderedDict()}"

**TBD. (Note: not possible until CAS standardizes all table outputs)**

b) Dataframe, in a json format

...

c) A link to redirect frontend, in a json format:

{

"link": "http://172.27.217.62/links/resources/report/?uri=/reports/reports/6c2b3c1c-d598-4f8f-ac0c-f84e59978093&page=vi931"

}

For developers, remember to commit and request merges to the main source, this is so everyone can share an organized copy of all use cases - this benefits the entire SAS community in pushing out new use cases and demos for your sales. It's like demo-iterations on steroids!

**Common Problems Faced… the ones we spent the most time on :(**

1. pip install failing

i) If you are already on Linux, you may need to do a pip install -U pip to upgrade the pip intaller first before doing anything else

ii) If you are getting an error related to setup tools, do a pip install setup tools

2. Cannot connect through SAS SWAT

Well you shouldn't face this problem but if you do, it can be isolated to two issues:

i) Swat is installed on a Windows, but it actually needs a Linux but does not tell you explicitly. All you get are 400-404 errors, but you just need it hosted on Linux!

ii) When connecting to CAS: swat.CAS(...), you need to explicitly state " protocol='http' " as one of the arguments to use the RESTFUL connection. Otherwise it will default to binary, which isn't in this package.

iii) Port issues. Ensure you are hosting the server with ports > 1024 and that other apps are not using the same port as you are.

3. Params are not going through smoothly:

Especially for web links such as https://...., ensure that it is not wrapped with single-quotes ('') or double-quotes ("") as this may mess around with JSON syntax and conversion into readable outputs internally.

Otherwise, please check if parameters/arguments have been fully stated both in your POST request, and Model logic.

Note: Tested on both Python 3.5 & 2.7

*For inquiries, POST them (hurhurhur) to thomas.thio@hotmail.com*