**Description of the programming task requirements to import Data Values from the spreadsheet to the WaMDaM database plus to document the previous design**

Adel Abdallah, Jan 25 2017

First, update the previous package to activate the check boxes for loading data based on the selected check box. Currently the Wizard loads all the steps regardless of which check box is selected. This task includes three parts that will finish the last tables to load data values to them with validations and then to better document and modularize all the WaMDaM Wizard script.

**Part 1: Load data to the following tables similar to the previous method used to load data into Parameters and TextFree tables. However, the following tables have more dependencies as explained below.** Each table should be loaded independently from the other one. So the script should be organized in separate modeled to load data for each table. Each Table should have the right validation checks at least for foreign keys

**Text Controlled**

Connect the TextControlledValueCV with its CV table to inforce or control the consistency of entered values.

**Multi-Column Arrays**

The MultiColumnArrays Table combines or relates one attribute that has a type of “MultiColumnArray” type with two or more (at least two and up to six) attributes of type “column”

The Mapping Table will be populated many times (one time for MultiColumnArray and as many times as there are columns) for the same exact MultiColumnArray.

For example, the first Mapping Table will be for the Attribute “Volume Elevation Curve-Reservoir-IdahoWRA”, its Instance, source, method, and scenario. The second and up to six Mapping Tables will be for the attribute with Type called “column” that is entered in the row#4 in Excel and columns from G to L. This column attribute will take the same exact Instance, source, method, and scenario provided for the Attribute “Volume Elevation Curve-Reservoir-IdahoWRA”. The only difference would be to use different AttributeCodes: Elevation-Curve-Reservoir-IdahoWRA, Volume-Curve-Reservoir-IdahoWRA, cost-Reservoir-IdahoWRA

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| MappingID | AttributeID | InstanceID | SourceID | MethodID | DataValuesMapperID |
| 1 | Volume Elevation Curve-Reservoir-IdahoWRA | Soda Reservoir-Reservoir-IdahoWRA | Idaho Department of Water Resources | Idaho Water Rights Accounting | 1000 |
| 2 | Elevation-Curve-Reservoir-IdahoWRA | Soda Reservoir-Reservoir-IdahoWRA | Idaho Department of Water Resources | Idaho Water Rights Accounting | 1001 |
| 3 | Volume-Curve-Reservoir-IdahoWRA | Soda Reservoir-Reservoir-IdahoWRA | Idaho Department of Water Resources | Idaho Water Rights Accounting | 1002 |
| 4 | cost-Reservoir-IdahoWRA | Soda Reservoir-Reservoir-IdahoWRA | Idaho Department of Water Resources | Idaho Water Rights Accounting | 1003 |

**Examples for the MultiColumnArrays:**

|  |  |
| --- | --- |
| MultiColumnID | 1A [I used the A here just to make it easier to track but this key should be integers) |
| ColumnNameID | Get the DataValuesMapperID that correspond to the attribute name “Elevation-Curve-Reservoir-IdahoWRA” |
| DataValuesMapperID | Get the DataValuesMapperID that correspond to the attribute name “Volume Elevation Curve-Reservoir-IdahoWRA” |

|  |  |
| --- | --- |
| MultiColumnID | 2A |
| ColumnNameID | Get the DataValuesMapperID that correspond to the attribute name “Volume-Curve-Reservoir-IdahoWRA” |
| DataValuesMapperID | Get the DataValuesMapperID that correspond to the attribute name “Volume Elevation Curve-Reservoir-IdahoWRA” |

|  |  |
| --- | --- |
| MultiColumnID | 3A |
| ColumnNameID | Get the DataValuesMapperID that correspond to the attribute name “cost-Reservoir-IdahoWRA” |
| DataValuesMapperID | Get the DataValuesMapperID that correspond to the attribute name “Volume Elevation Curve-Reservoir-IdahoWRA” |

For the MultiColumnValues table, use the MultiColumnID unique identifier for each of the three tables above as a primary key to into the MultiColumnValues table. Then enter the Value and ValueOrder for each column attribute. The same ValueOrder will be reused for each column attribute.

|  |  |  |  |
| --- | --- | --- | --- |
| **MultiColumnValueID** | **Value** | **ValueOrder** | **MultiColumnID** |
| 1 | 0 | 1 | 1A |
| 2 | 130 | 2 | 1A |
| 3 | 649 | 3 | 1A |
| 4 | 1739 | 4 | 1A |
| 5 | 3456 | 5 | 1A |
| 6 | 5937 | 6 | 1A |
| 7 | 9236 | 7 | 1A |
| 8 | 13206 | 8 | 1A |

|  |  |  |  |
| --- | --- | --- | --- |
| **MultiColumnValueID** | **Value** | **ValueOrder** | **MultiColumnID** |
| 9 | 4590 | 1 | 2A |
| 10 | 4600 | 2 | 2A |
| 11 | 4610 | 3 | 2A |
| 12 | 4620 | 4 | 2A |
| 13 | 4630 | 5 | 2A |
| 14 | 4640 | 6 | 2A |
| 15 | 4650 | 7 | 2A |
| 16 | 4660 | 8 | 2A |

|  |  |  |  |
| --- | --- | --- | --- |
| **MultiColumnValueID** | **Value** | **ValueOrder** | **MultiColumnID** |
| 9 | 100 | 1 | 3A |
| 10 | 200 | 2 | 3A |
| 11 | 300 | 3 | 3A |
| 12 | 400 | 4 | 3A |
| 13 | 500 | 5 | 3A |
| 14 | 600 | 6 | 3A |
| 15 | 700 | 7 | 3A |
| 16 | 800 | 8 | 3A |

**File Based**

Users will provide the file location (full path including the file name) on their machine, then the Wizard will load this file into the SQLite database as a blob. I think we need to change the physical data type of the FileLocationOnDesk column to be “blob”. No sure about the best way to do this loading.

**Binarys**

There is a design issue on how to handle this table. This table is for dual text values (e.g., [active or inactive] or [open, closed]) which corresponds to binary 0, 1 values. The BinaryValueMeaningCV table restricts the entries to be either one value.

One solution is to have two columns (Value1, Value2) in the Binarys table then each column will be controlled in the BinaryValueMeaningCV table using two foreign keys (one for each value). But it will be a bit harder to query and merge both query results into one column.

**Part 2: Modularize the script around smaller data loading steps as follows**

The current script calls multiple data loading steps together. I guess each sub step should have a separate function or class name to be called by-itself. Then the major step calls all the sub-steps together in the end. Users in the future should be able to call one sub-step by itself. The code should be organized into modules as listed below for each step. Let's leave the controlled vocabulary (CVs) tables a side for now. But each table should be called on its own. There is a chance to update the whole script to load all the CVs tables together at once at the beginning. Will come back to them later.

**Under Step 1:**

Organizations

People

Sources

Methods

**Under Step 2:**

Dataset

ObjectTypes

Network Attributes

Attributes

**Under Step 3:**

MasterNetworks

Scenarios

Nodes

Links

**Under Step 4:**

Binarys

Parameters

SeasonalParameters

TextControlled

TextFree

FileBased

TimeSeriesSites

MultiColumnArrays

**Part 3: documentations for files, functions, and classes**

I would like to follow a very consistent and extensive documentation throughout the Wizard scripts. The documentation's target is for someone who is not a computer scientist and they want to get an inside high level understanding of how the Wizard script works.

Below I provide a template for how each python file should include. This template and descriptions could be updated or adjusted for a more professional looking style. It should include 1. File General Info, 2. Imported Python Libraries used in this file, 3., Imported Python modules from within the Wizard used in this file, 4., Script begins here (the body of the script).

The last section below "Script begins here" follows similar documentation you used here for all the classes and functions in the wizard. Please make sure to first describe the logic of the class or function in high level plain English and then specify how the logic work by referring the variables used in it.

Throughout the documentation, delete any unused imported libraries or script pieces. For example, the "Load\_step.py" files still have the name of the spreadsheet input file hard coded like this below. So delete them if not used anymore.

**if**\_\_name\_\_ == '\_\_main\_\_':  
    instance = Load\_Networks\_Data('./WaMDaM\_InputData.xlsx')  
    instance.load\_data()

**# File General Info**

#e.g., this file is the main WaMDaM Wizard launcher and it calls the main GUI form called "frm\_Home".

or something more detailed like this:

'''  
    The load\_step\_4.py is a python script that is used

    to load some CV's of Data Values to the sqlite  
    database.The file has a class called "Load\_Data\_Values()" that   
     uses xlrd library to handle the xlsx files, load, read  
    data and workbook sheets from the file.  
'''

#-------------------------------------------------------------------------------------------------------------------

**# Imported Python Libraries used in this file (examples below)**

import xlrd

#This xlrd package is for reading data and formatting information from older Excel files (ie: .xls)

from openpyxl import load\_workbook

# this package does xyz

Import  datetime

#The datetime library supplies classes for manipulating dates and times in both simple and complex ways

#-------------------------------------------------------------------------------------------------------------------

**# Imported Python modules within the Wizard used in this file** **(examples below)**

from .step\_variables import Network\_sheets\_ordered, row\_start\_network

# this module is imported to do xyz in this place

#-------------------------------------------------------------------------------------------------------------------

**# Script begins here**

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#  
#                                                                                  #  
#                      This is the class to load CV data                           #  
#                                                                                  #  
# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* #

for simpler classes, just add one comment line on top of the class

**class Load\_CV\_To\_DB**(Parse\_Excel\_File, DB\_Setup):

*"""  
        This class "Load\_CV\_To\_DB" inherits methods from Parse\_Excel\_Files  
        and DB\_Setup classes so as to parse excel file and load  
        subsequent data received from parser  
    """*

for simpler functions, just add one comment line on top of the function

**def load\_data**(self, sheet\_names):  
    *"""  
        This function "load\_data()" takes a list of sheet names and using  
        the dictionary of rows return from the excel parser  
        it load the data in the appropriate table* ***:param****sheet\_names:* ***:return****: None  
    """*

**Add a comment to explain the purpose of key variables like this one in the Load\_steps\_2.py file**

#this foreing\_key variable does x,y and z

foreing\_key = -1