WindMil Lua Scripting

2009 Milsoft Users Conference
Greg Shirek
Milsoft Utility Solutions, Inc.

Introduction

WindMil now provides the capability for creating Lua (pronounced LOO-ah) scripts that permit manipulation of the circuit model through the use of LUA scripting language.

Previously, the Updateable Utility Manager was used for new utilities that were created to fix model issues or to provide new tools or features. These utilities were written in C++ and therefore the WindMil code had to be modified, rebuilt, and then run again. With Lua, the code is directly accessible for manipulation at any time. Overall, Lua scripting has become a simpler way to perform the same functions as the Updateable Utilities of the past.

During database conversions, Lua scripting has become a valuable tool; for example, data converted from some other database into WindMil format might contain sporadic locations of consumers or connectivity issues between transformers and consumers. Lua scripting provides much easier repair of such issues when WindMil global editing tools are not robust enough to provide the required results.

What is Lua?

For those wondering if Lua is an acronym, the answer is no; Lua simply means “Moon” in Portuguese, because Lua was designed, implemented, and is now maintained by a team at PUC-Rio (the Pontifical Catholic University of Rio de Janeiro in Brazil).

Lua is free, certified, open-source software. Lua is an "embedded language," in the sense that embedding the interpreter into your program is a simple task. It is very easy to interface Lua with other languages, such as C, C++, or even FORTRAN. Lua is an interpreted language, with dynamic typing and several reflexive facilities. (1)

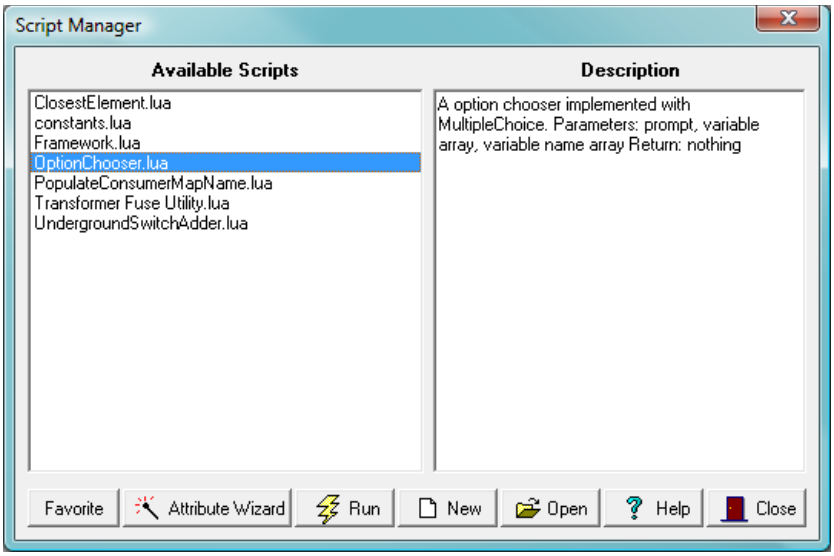
Lua combines simple procedural syntax with powerful data description constructs based on associative arrays and extensible semantics. Lua is dynamically typed, runs by interpreting bytecode, and has automatic memory management with incremental garbage collection, making it ideal for configuration, scripting, and rapid prototyping. (2)

VIEWING SCRIPTS

The available scripts, both *.lua and *.elua types, must be located in the C:\Milsoft\Programs\Scripts folder. Then, when opening the Script Manager, the *.lua list will appear in the available scripts section

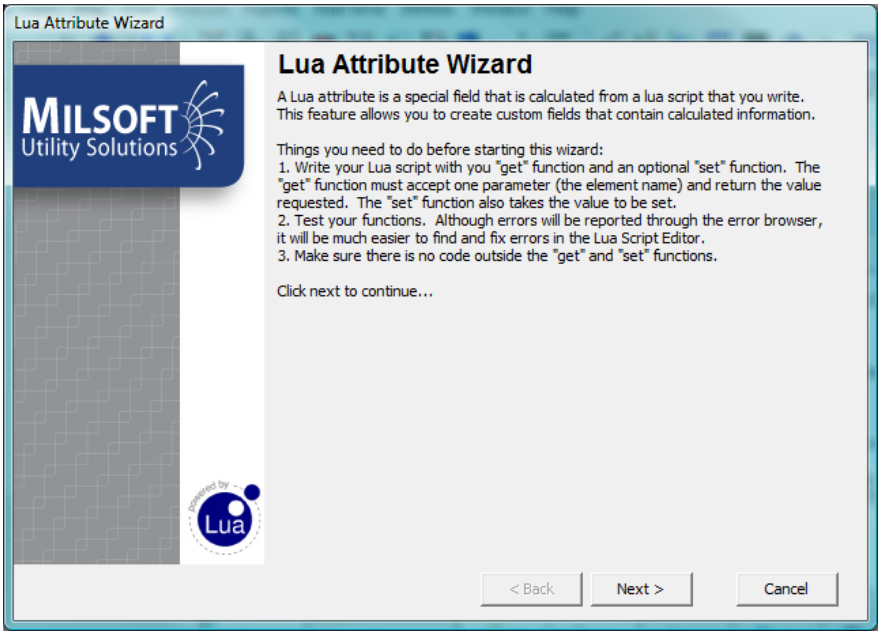
as shown in the figure below. The *.elua scripts will not show up, since they actually create custom element fields.

Figure 1



If desired, *.elua scripts can be modified by stepping through the Attribute Wizard. The description in the screen shot below summarizes what is needed to modify *.elua attribute fields.

Figure 2



EXAMPLE 1 – ADD SOURCE-SIDE DISTRIBUTION TRANSFORMER FUSING

Mainly for purposes of providing more accurate arc flash analysis results, there is a script that searches the entire model for transformers that currently do not have a connected parent protection device, and then inserts one. Since running the arc flash application finds the fastest upline clearing device and returns the clearing time at the calculated arcing current, modeling distribution transformer fuses is important for arc flash scenarios.

Others have found this Lua script useful simply to provide a more accurate model from a system protection/coordination standpoint when running WindMil's coordination analysis. Many utilities fuse transformers according to a fusing schedule and realize after the fact that the upline device, such as a tap recloser or tap fuse, many times has a faster time-current-curve than the transformer fuse, thus possibly creating failed coordination.

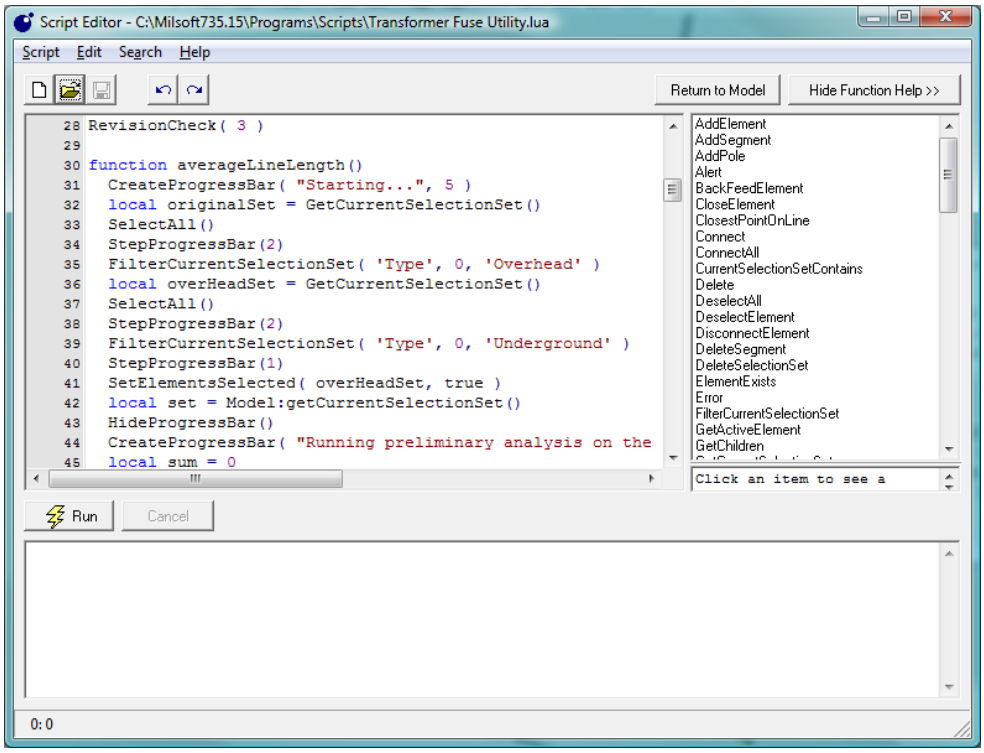
In addition, this transformer fusing Lua script builds selection sets of new overcurrent devices added, based upon the number of transformer equipment definitions used in the model. Keep in mind the script only creates a selection set for each transformer equipment definition actually associated with at least one transformer in the model. The resulting selection sets can then be loaded via the global editor to assign an overcurrent device equipment type to each.

To run the Lua script for source side device insertion, make sure you have the following three files in your Milsoft directory under the C:\Milsoft\Programs\Scripts folder:

1. Transformer Fuse Utility.lua
2. Framework.lua
3. Constants.lua

Open the Script Manager under the Utilities Menu and select Transformer Fuse Utility.lua. From here, select Run or Open. If selecting Open, the following appears, showing both Lua code and call functions.

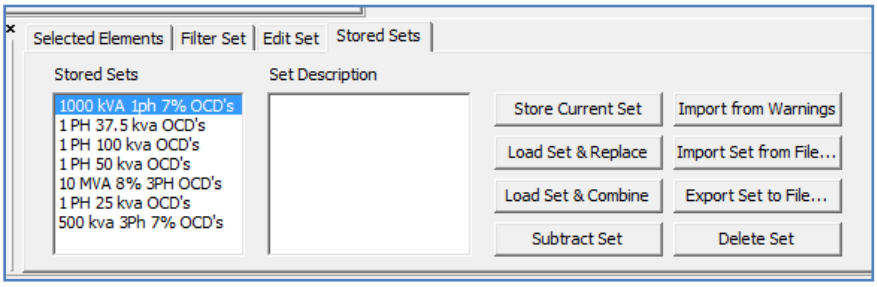
Figure 3



Available functions are listed in the right window; the left window shows the Lua code.

Select Run from the Script Editor window. When complete, there will be stored sets created according to all the used transformer equipment definitions in the model. Each transformer EQDB definition will show up along with an *OCD's term.

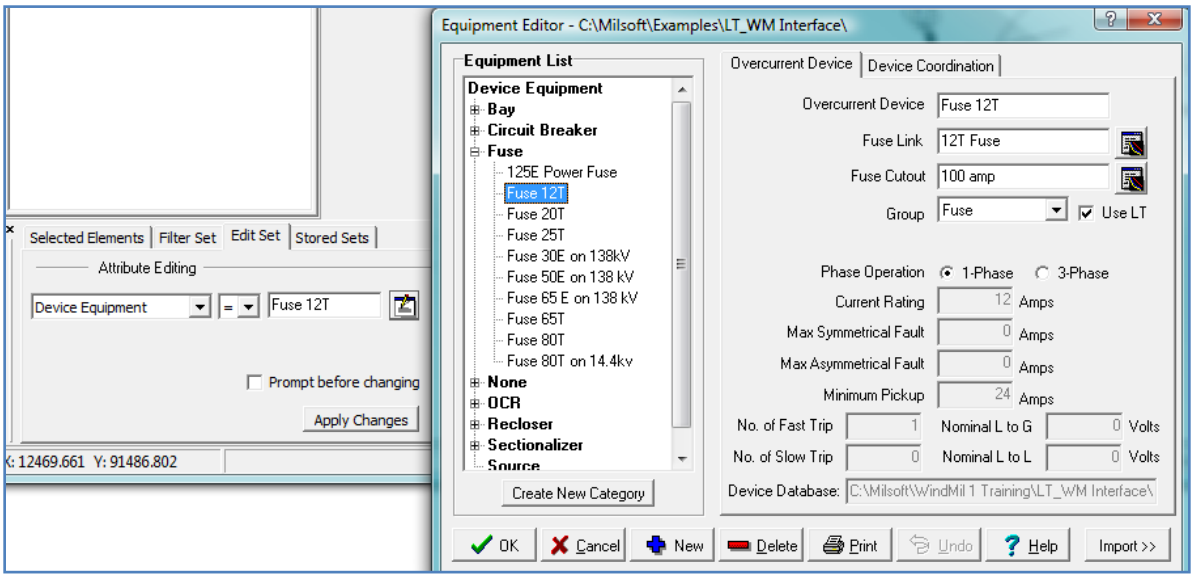
Figure 4



At this point, select "load set and replace" to load the stored set. Then the global editing tab can be used to assign an overcurrent device definition to the set, such as from a transformer fusing schedule used at the utility.

Continue with the load set and replace to assign device definitions to each set until all the OCDs are defined as shown in the following figure:

Figure 5



As a side note, another comparable script has been written that inserts overcurrent devices on the **LOAD-SIDE** of transformers; it's what's needed for CSP (completely self-protected) transformers, since they have a load side breaker device for downline faults.

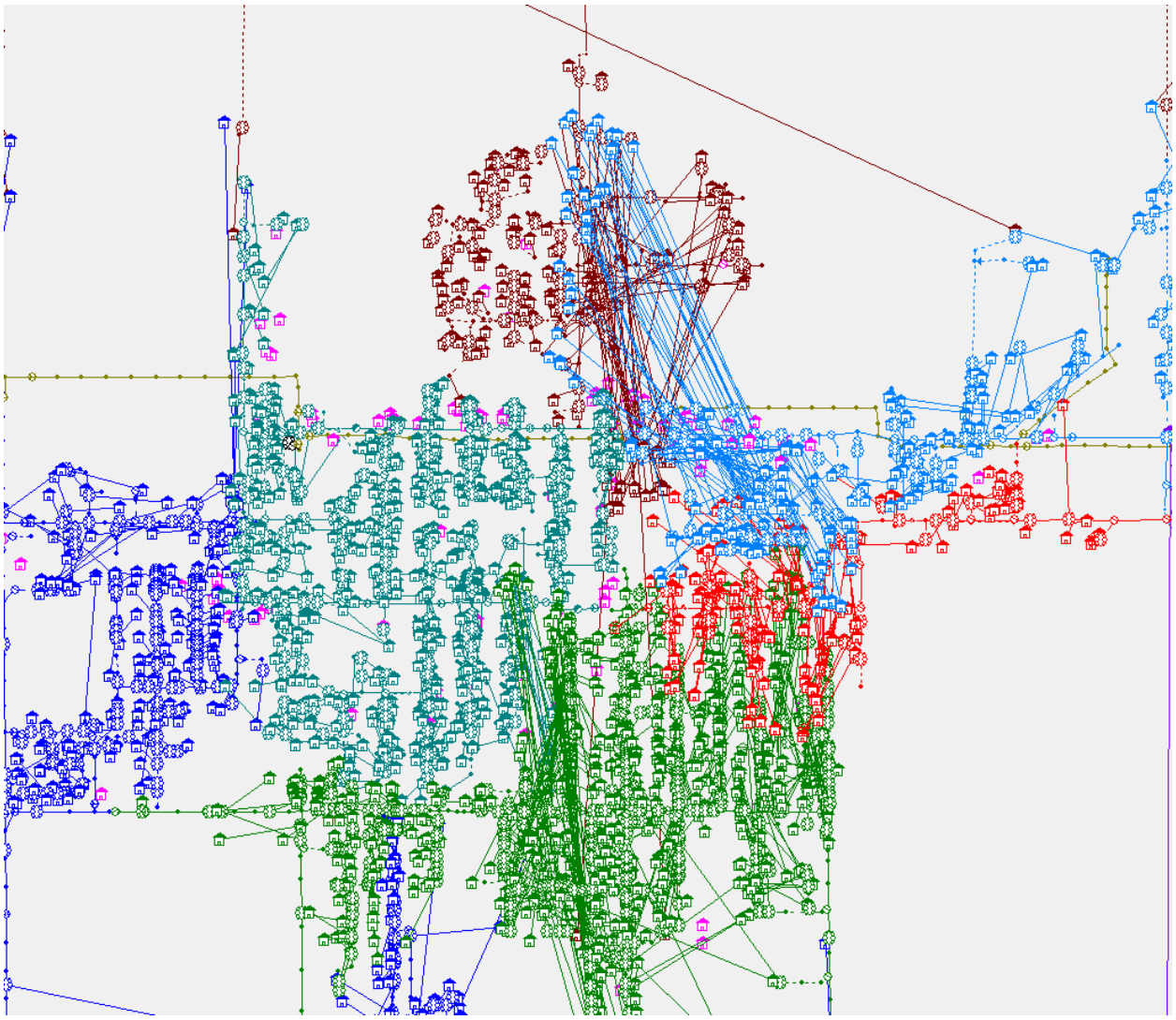
EXAMPLE 2 – FIX TRANSFORMER-CONSUMER CONNECTIVITY & ADD SECONDARY

As previously explained, Lua scripting decreases time and effort when correcting modeling issues, such as when repairing disconnected consumers. Further, many users have found it extremely useful to model secondary lines between distribution transformers and consumers (both will be demonstrated).

This Lua script is called "SecondaryUtility.lua." Originally it was written to correct a consumer-transformer connectivity problem that was occurring from a GIS export. It was deemed appropriate that consumers should be connected to the most nearby transformer. Later, the option of adding secondary conductors was included in order to increase the accuracy of the model for engineering applications, such as the calculation of secondary line losses. Even though the problems and feature additions were provided through this Lua script, it is always recommended to resort to the source of the problem and to fix the issues in the GIS.

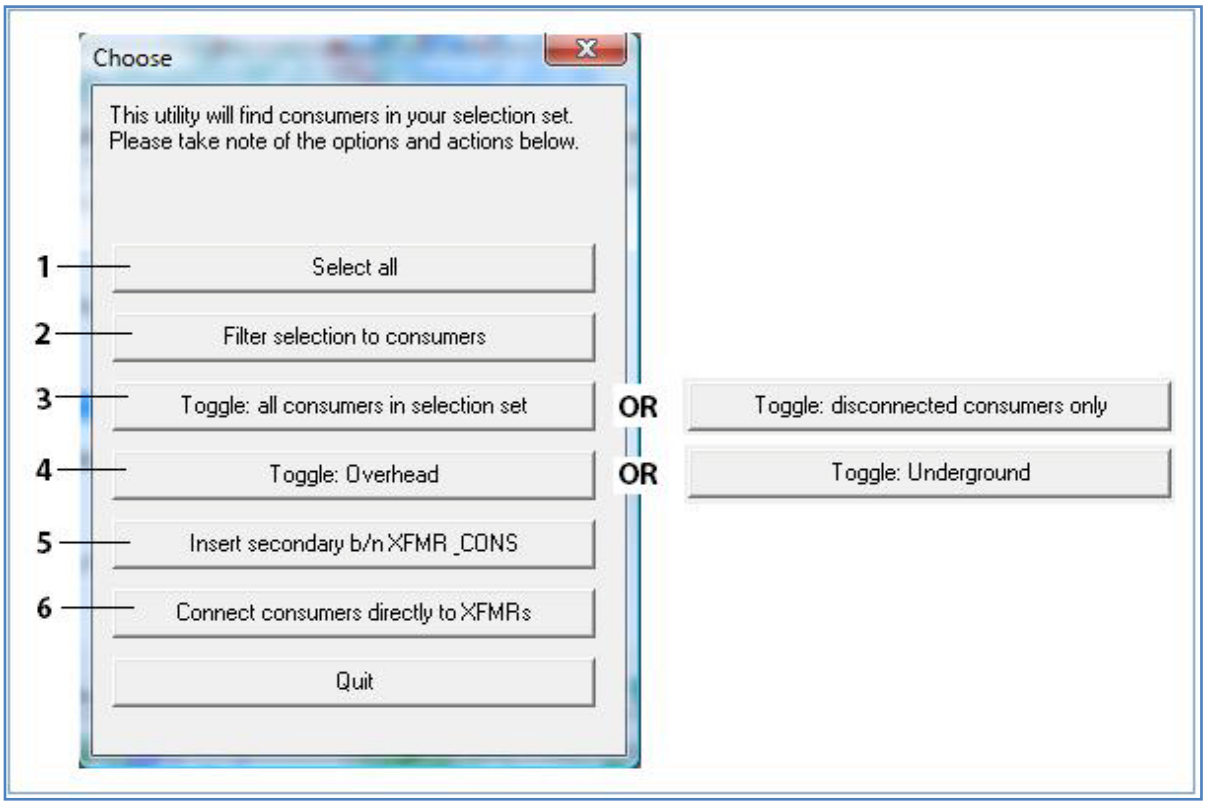
The following figure reflects the original model. Notice many consumers are either disconnected or else connected to the incorrect transformer.

Figure 6



First, open the “SecondaryUtility” script through the Script Manager and select “Run.” The following dialog opens, showing some options to set before executing.

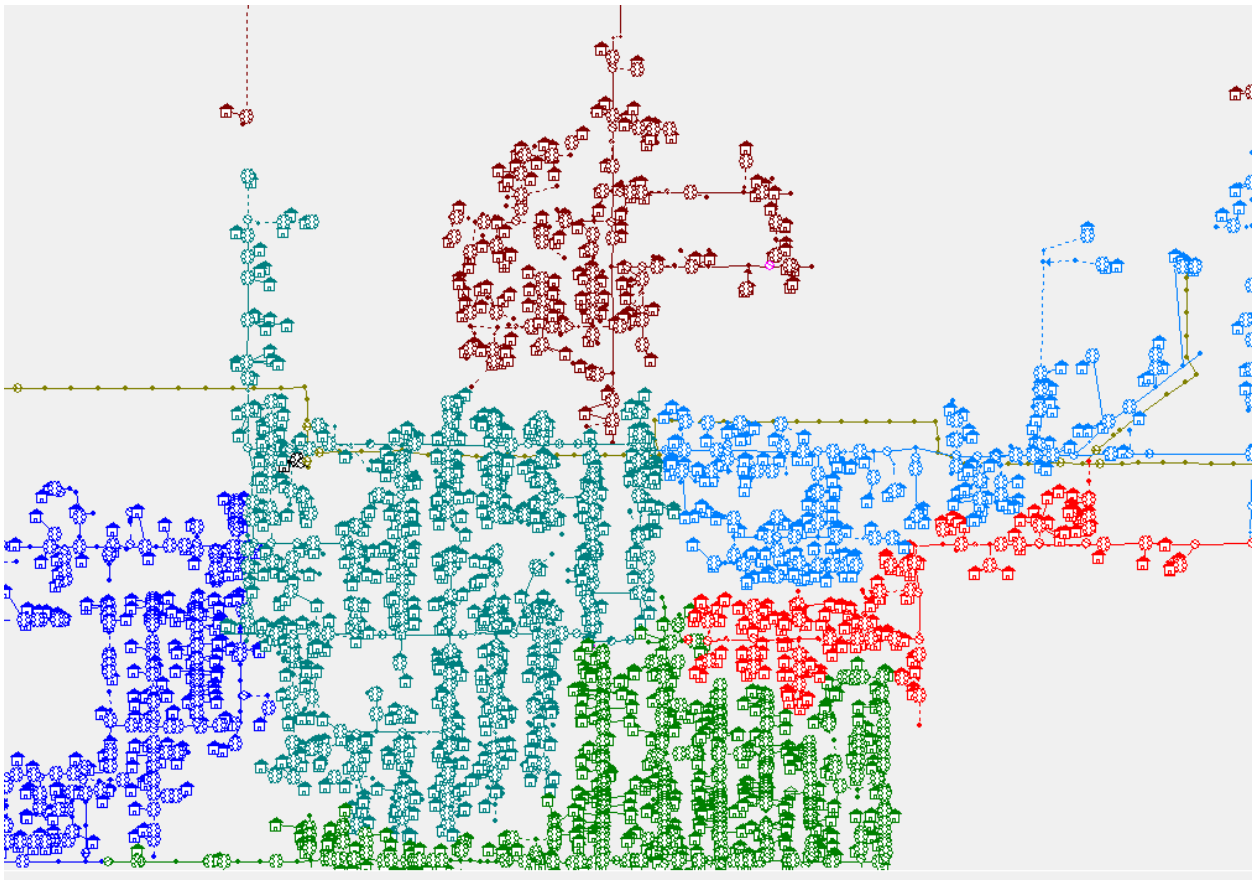
Figure 7



1. Selects all the elements in the model while running the Lua script rather than requiring you to exit and use the global editor.
2. Filters the selection set to only consumers while running the Lua script so that you do not have to exit the script and use the global editor.
3. Press this button to set whether you would like this script to run on all consumers or only the disconnected ones.
4. Press this button to set whether you will be inserting Overhead or Underground as your secondary.
5. Inserts secondary wire (OH or UG based on selection 4) between transformers and selected consumers (from 3). The phasing will be set to the same phasing as consumer.
6. Connects selected consumers (from 3) to the nearest transformer. *Note: This option does not insert secondary wire.*

Two actions were performed to clean up this model. First, “connect consumers directly” was selected, which created a model with the correct connectivity shown below. Compare it to figure 6 and notice the difference.

Figure 8

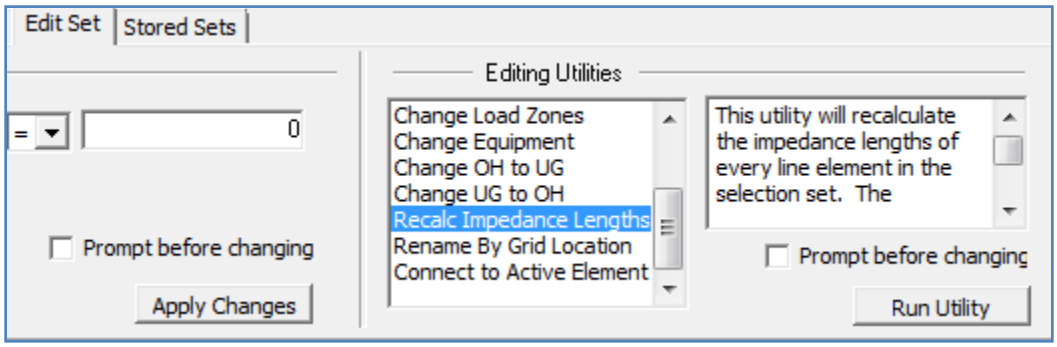


To add secondary overhead conductor elements, the “insert secondary” button was selected. This process may take a few minutes, depending upon how many elements are in the selection set or the entire model. For example, the above area in Figure 8 contains 2,100 consumers, and it took approximately two minutes to complete the add-secondary function. You may want to test this script on only smaller areas of your model at first.

The script is written so that the new secondary conductor equipment definitions will be assigned the same as the first upline overhead or underground line. If no upline element is found that is the same element type being added (overhead or underground), the equipment definition will remain blank and have to be assigned by the user. In reality, for either situation, the user will most likely want to edit these conductor definitions, since the secondary will be different than the first upline primary conductor type.

This script does not populate the impedance length of the secondary. However, there is another quick editing tool called “Recalc Impedance Lengths” that assigns lengths to the newly-created secondary elements, as shown below.

Figure 9



LUA ATTRIBUTE SCRIPTS

With the advent of the global editor, WindMil users are developing many more complex model queries and/or filters, and then sometimes editing those filtered sets. As a result, new element fields may be needed, or simply desired, to reduce the number of query rules or filtering steps. Think of these new element fields as fields that show calculated data.

As an example, in the past, if the **upline device equipment type** was desired, the "Element Name," "Upline Device," and "Device Equipment" element fields were needed. From there, some sort of lookup command to find the "Device Equipment" associated with the "Upline Device" was performed in a separate application such as Microsoft Excel. Thus, the element field "Upline Device Equipment" was created to eliminate this extra step.

Four new fields with "first," "second," "third," and "fourth" terms preceding the element field names now exist, significantly helping with overcurrent protection studies to find consumers within various zones of protection. The functionality of each of these fields will simply search the radial path from any element to the source and then provide the device equipment name for all (up to four) upline overcurrent devices found.

The following is a small sample of the Lua attribute element fields that have been created.

1. Base kV LL – Voltage in L-L rather than L-G as in the Base kV field
2. Closest Transformer – Finds closest transformer based on straight-line distance
3. Closest Pole – Finds closest pole based on straight-line distance
4. First through Fourth Upline Device – Returns first, second, third, or fourth Upline Device Elements' Name
5. First through Fourth Upline Device Equipment - Returns first, second, third, or fourth Upline Device Elements' Equipment Name
6. Minimum of Max Ph-Gd Amps – Created for Arc Flash purposes
7. Minimum of Three Phase Amps – Created for Arc Flash purposes
8. Thru kW Loss – Sums selected and downline elements thru kW losses

9. Upline Impedance Length – From selected element to the source
10. Upline OCR – Finds first upline device defined in the OCR group
11. Upline transformer kVA – Finds first upline transformer and returns total bank kVA

FOURTH UPLINE DEVICE EQUIPMENT ELUA

To demonstrate some of the most commonly used call functions better and how they are interpreted in the code, the following attribute element field for the simple "fourth upline device equipment" is displayed below, along with a description of each line of code and call functions.

Figure 10

```
1  -- BEGIN META DATA
2  --<luaelementfield>
3  -- <name>Fourth Upline Device Equipment</name>
4  -- <type>stringTypeEnum</type>
5  -- <category>fieldCategoryGeneral</category>
6  -- <getfunction>FourthUplineDeviceEquipment</getfunction>
7  -- <setfunction></setfunction>
8  --</luaelementfield>
9  -- END META DATA
10
11
12 function FourthUplineDeviceEquipment( element )
13   local count = 1
14   local parent = GetParent( element )
15   while true do
16
17     if GetElementField( parent, 'Type' ) == 'Device' then
18       if count == 4 then
19         if GetElementField( parent, 'Device Equipment' ) ~= '' then
20           return GetElementField( parent, 'Device Equipment' )
21         else
22           return 'none'
23         end
24       end
25       count = count + 1
26     end
27     parent = GetParent( parent )
28     if parent == nil then return 'none' end
29   end
30 end
```

Refer to rows in above figure

1-9 -- Setting up the meta data, classifying it a Lua Attribute
12 -- The *element* variable holds the name of the currently selected element
13 -- *count* keeps track of the number of devices found upline
14 -- Starts by getting the parent of *element* and giving the value of the parent name to the variable *parent*
15 -- The *while* loop continues until a *return* statement breaks the loop
17 -- This *if* statement checks to see if *parent's* type is a device element type

18 -- If it's a device, then this *if* statement will check to see if this is the fourth upline device by checking *count's* value
19 -- If this is the fourth upline device, then the statement checks whether or not the element's parent has equipment attached to it
20 -- This *return* statement will return the name of the device equipment if it does have equipment attached to it
22 --If it turns out that parent does not have equipment, then this *return* statement returns 'none'
25 -- *count* is updated each time a device is found, so it's placed in the *if* statement that checks whether parent is a device or not
27 -- At the end of the *while* loop *parent* becomes the next upline device and keeps the loop running until it reaches the 4th device or the root element (source) in the model
28 -- If the statement returns 'none,' then there are no more elements upline

As WindMil updates are released, these element field Lua Attribute scripts will be provided in your scripts folder. You may wish to review the contents of this folder after each update to see which fields have been added to take advantage of them for making queries and filters easier.

SUMMARY

The Lua scripting feature in WindMil is a very powerful feature that accomplishes what would normally take much manual effort when using standard WindMil editing tools.

The user must take the necessary precautions when creating scripts, since the model can be manipulated drastically if the script is not written correctly. Even if it appears the script works as designed, changes to the model may occur that aren't apparent to the user. It is recommended that the user contact Milsoft to define the requirements of what you need and then Milsoft will write the script, test, and verify to ensure correct functionality.

References

(1) *http://www.lua.inf.puc-rio.br/*
(2) *lua.org*