Developer: Christopher McNeely
Source: https://github.com/professoraire/ElegantXML-Crestron

Elegant XML - Manager v1.0

GENERAL INFORMATION

SIMPL WINDOWS NAME	Elegant XML - Manager v1.0
CATEGORY	Elegant Modules
VERSION	1.0
SUMMARY	Provides the core functionality for XML configuration file processing.
GENERAL NOTES	This module works with the Elegant XML – Analog Values, Signed Analog Values, Serial Values, and Digital Values modules to provide parsing of XML files into usable Simpl Windows data.
CRESTRON HARDWARE REQUIRED	3-Series Processor

PARAMETERS

FileName	S	The file name on the processor to load. By default if this is just a file name this will check for a file in the directory User/Config/App# (where # is equal to the program slot the program is running in). First it will check the ROMDISK/Config/App# path to see if the file exists there. If it doesn't exist in either of the two spaces, an empty file is created in User/Config/App#. If you prefix the string with a \ you can provide your own full path. For instance: \\RM\\MyFiles\\Config.xml

RootElement	S	This is the RootElement of the XML file. For example, if you use the word Config, then the processor will expect a file that looks like:
		`<?xml version="1.0" encoding="UTF-8" ?>`
		`<Config>`
		`  <DATA />`
		`</Config>`

ManagerID	S	The unique ID of this Manager instance. All modules that are related to this manager will use this ID. Only one manager in a program should have this ID.

CONTROL

LoadFile	D	On the rising edge, this attempts to load the file. This shouldn't be called until all dependent modules have initialized.

SaveFile	D	On the rising edge, this attempts to save the file. This shouldn't be called until all dependent modules have initialized.
EnableDebug	D	Hold this value high to enable debug messages to print to the console.

FEEDBACK

IsLoading	D	Indicates if the module is in the process of loading data from the XML file.
LoadSuccessPulse	D	Pulses for 0.1 seconds when the file is loaded successfully.
LoadFailurePulse	D	Pulses for 0.1 seconds when the file fails to load.
IsSaveNeeded	D	Transitions to high when a value has been changed. Note that the Simpl# code only tracks if any value has changed at least once. If a value is changed and then returned to its previous value, this will remain high.
IsSaving	D	Indicates if the module is in the process of saving data to the XML file.
SaveSuccessPulse	D	Pulses for 0.1 seconds when the file saves successfully.
SaveFailurePulse	D	Pulses for 0.1 seconds when the file fails to save.
Progress	A	Provides a value of 0% (0d) to 100% (65535d) to track the progress of the current save or load operation.
ErrorMessage	S	Provides a readout of the error encountered while saving or loading. This is updated when the LoadFailurePulse or SaveFailurePulse outputs are pulsed.

ADDITIONAL DATA

Revision History	v1.0 – Initial Release
Additional Details	This set of modules is designed to allow loading an XML file into Crestron Simpl Windows code. A single copy of this module is required for every XML file that needs to be loaded. The Elegant XML – Analog/Signed Analog/Digital/Serial Values modules that are provided with this are used to define the path to each element in the file that needs to be pulled from it. Loading from the file should not happen until all associated modules have been initialized. The easiest way to make sure this happens, is to AND all the IsInitialized signals from the Values modules together and use the resulting AND to pulse LoadFile. Note: This module ignores unused data elements and when the file is saved using the processor none of those elements will be saved to the new file. The ONLY elements that are saved to the file are those that are called out in the Simpl Windows program. If you want to keep unused elements in the file, you will need to place their path in a Values module and just comment out the input and output for that path. The format for specifying paths is as follows: Element[Identifier].Element[Identifier].Element[Identifier]

Note: You do NOT need to place the RootElement on each path. This helps shorten the length of the paths needed.

The following file snippet and paths demonstrate how this works:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Config>
  <Switchers>
    <Switcher Id=”1”>
      <Inputs>
        <Input Id=”1” Name=”Input 1” Patch=”1” IsInstalled=”true” />
        <Input Id=”2” Name=”Input 2” Patch=”2” IsInstalled=”false” />
      </Inputs>
    </Switcher>
    <Switcher Id=”2”>
      <Inputs>
        <Input Id=”1” Name=”Laptop” Patch=”9” IsInstalled=”false” />
      </Inputs>
    </Switcher>
  </Switchers>
</Config>
```

Paths

1. Switchers.Switcher Id=”1”.Inputs.Input Id=”1”.Name
    a. This outputs “Input 1”
2. Switchers.Switcher Id=”2”.Inputs.Input Id=”1”.Patch
    a. This outputs the value 9
3. Switchers.Switcher Id=”1”.Inputs.Input Id=”1”.IsInstalled
    a. This digital value would be high.

You can see in this example that the optional identifier MUST occur immediately following the name of the element. For example, if you attempted to write the path: Switchers.Switcher Id=”1”.Inputs.Input Name=”Laptop” this would fail, since the Name=”Laptop” attribute isn’t immediately following the name of the element, which is Input.

You can also see that the identifier isn’t required. Be wary though! If you place multiple elements with the same name inside another and you don’t provide identifiers, the first element that matches this will ALWAYS be used. The path: Switchers.Switcher Id=”1”.Inputs.Input.IsInstalled would always return a true (or Digital High) value, because it would always use the value of the element Input Id=”1”.