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

Elegant XML - Manager v1.3.2

GENERAL INFORMATION

SIMPL WINDOWS NAME	Elegant XML - Manager v1.3
CATEGORY	Elegant Modules
VERSION	1.3.2
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. Version 1.3.2 is a patch and all modules are still versioned 1.3. This allows a direct copy-and-replace method, which will replace all the existing modules, but keep compatibility with projects. If you have specific projects that rely on the behavior pre-1.3.2, you should create copies of these files in the project directory.
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.
PathDelimiter	S	A single character that defines the delimiter to use when

		separating path elements in the path parameters.
DefaultValueDelimiter	S	A single character that defines the delimiter to use when separating the path from the default value in the path parameters.

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 v1.1 – Fixes to error messages related to parsing when loading a file. Updated the file saving routine to be non-destructive. When loading an XML file with attributes that aren't brought into Simpl Windows, these attributes are retained when the file is subsequently saved. v1.2 • Added optional final delimiters and the ability to write a complete file using default values, if no file exists. ○ Can use default values for non-existent elements, while still reading other values

	from a file. • Added a DefaultValueDelimiter parameter, which by default is the pipe character. This allows for setting default values in the path, instead of initializing them elsewhere in the program. • Added a PathDelimiter parameter, why by default is the period character. This allows a user to customize the character looked for when specifying path separators. • Fixed a bug preventing programs without at least one of each processor type from correctly saving a file. v1.3 • Performed various minor fixes. • Significantly improved both loading and saving of files. A test file with 9000 values being read loads in around 30 – 40 seconds on a CP3 and is saved in around 10 to 15 seconds. • Fixed a small number of issues that could cause a processor to lock-up, especially when dealing with large numbers of value modules and signals. • Improved reporting of progress. The Simpl# code will now only notify the Simpl+ modules of progress percentages that equate to whole numbers, instead of notifying on every fractional percent. This reduces the number of task switches required to ensure the SIMPL Windows code gets a chance to display the value change. v1.3.1 • Fixed a bug preventing DefaultValueDelimiters from being correctly set/called in the C# code. v1.3.2 • Fixed a problem with values being passed through processors regardless of whether the value was new. Now a value only propagates to an output if the value on the input is different than the current output.
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. **Notes** • You do NOT need to place the RootElement on each path. This helps shorten the length of the paths needed. • Also note the last optional delimiter in the path. If you place this last delimiter on the path you will be telling the parser to read or write the value inside of a pair of tags, the opening and closing tags. (Like: <Quantity>2</Quantity>) This is useful for XML files that use unique individual elements for all properties, instead of placing multiple values on elements. • If a value doesn't exist in the XML file and no default value is defined in the path, the value of the corresponding input is used. So, if you time your system startup to initialize the inputs of all your element module and then load the file, if any value isn't found it will use that initialized value instead. The exact format for specifying paths is as follows: Element[Identifier]{PathDelimiter}Element[Identifier]{PathDelimiter}Element[Identifier][{PathDelimiter}][{DefaultValueDelimiter}DefaultValue]

The following file snippet and paths demonstrate how this works:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Config>
  <Switchers>
  <Quantity>2</Quantity>
    <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|DefaultName
    a. This outputs “Input 1” if the file is present, or “DefaultName” if not.
2. Switchers.Switcher Id=”2”.Inputs.Input Id=”1”.Patch|1
    a. This outputs “9” if the file is present, or “1” if not.
3. Switchers.Switcher Id=”1”.Inputs.Input Id=”1”.IsInstalled|true
    a. This would return “true” if the file is present, or “true” if not.
4. Switchers.Quantity.|0
    a. This outputs “2” if the file is present, or “0” if not.

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”.