IO Formatting UDO

3/23/2024

The IO Formatting UDO enhances the basic abilities of the User Defined application object. It provides the ability to insert metadata parsing command into the InputSource and/or OutputDest on the IO Feature Extension. Scripted functionality has been added to allow IO reference formatting commands or rules to be added to any IO reference. The Syntax of the rule definition closely resembles the System.String.Format statement from .Net.

The object contains an Object Wizard that assists with the configuration of where the IO source for this object will come from.

Object Wizard

- Enable IO Formatting – Selecting this option adds in the Attributes and Scripts needed to make this feature work. Unselecting this option will remove all the attributes and scripts from the deployed instances.

- IO Target – this is a choice group that allows you to choose the target of the object's IO references that begin with the place holder **xDIO.SG.** any IO reference that does use this placeholder will not be directed to this target. If you have multiple targets on the same object, then any valid DIO.ScanGroup(Topic) can be used as a literal prefix to the MxReference. Entering an actual DIO.ScanGroup(Topic) will direct that reference to a different target. The reference can still have formatting rules after this literal target to complete the Item (Attribute) name construction as required. Settings on the wizard will cause the following behavior.

    - IO Devices Tree – will detect which DIClient and ScanGroup (Topic) the object is assigned to on the IO Devices Tree. This will be used to replace all **xDIO.SG.** prefixes on the InputSource or OutputDest.

    - PCS IO Source – Allows you to assign a PCS Scope name to the hidden attribute Me._DIO.Target. The wizard prompts you for this, but you can also write to the attribute directly. The PCS Scope name will replace all **xDIO.SG.** prefixes on the InputSource or OutputDest followed by a colon (:).

    - App Object - Allows you to assign an Application Object name to the hidden attribute Me._DIO.Target. The wizard prompts you for this, but you can also write to the attribute directly. The object must exist in the same galaxy. The ObjectName will replace all **xDIO.SG.** prefixes on the InputSource or OutputDest followed by a dot (.).

    - Multi-Galaxy – Allows you to assign an external Galaxy name to the hidden attribute **Me._DIO.Target**. The wizard prompts you for this, but you can also write to the attribute directly. The Galaxy name will replace all **xDIO.SG.** prefixes on the InputSource or OutputDest followed by a colon (:).

- Propagate Template Changes – Allows you to use the wizard capability to assign your attributes

to this Option and then those unlocked formatting rules will propagate to the instances that have this option selected. The InputSource and/or OutputDest must be **Bolded** on the Settings pane for the value to propagate. Only changes to the settings pane will be propagated. Any changes to the IO Feature Dialog will not be propagated to existing templates and instances as per normal unlocked attribute behavior. For the rule to be properly evaluated the InputSource and/or OutputDest must not be **locked** in a template.

There are seven (7) hidden attributes on this object to support the scripted functionality.

Me._DIO.ArrayElement	Integer	Element in the UDT Array to link to, ENTER [x] in reference format to use. x will be replaced with the value of this attribute. This can be set via the Wizard Settings; Scripting can also be used to set this value.
Me. _DIO.BaseReg	Integer	Base Register to use to offset Attributes reference from. The {OFF:xxx} rule will add xxx to this value. This can be set via the Wizard Settings; Scripting can also be used to set this value.
Me_DIO.ConfigCmd	Boolean	Command to assign IO References. Setting to true will trigger the script to run. This can be set via the Wizard Settings, It can also be triggered manually.
Me. _DIO.SG.Ref	Boolean	Attribute that holds the DIClient and ScanGroup(Topic) assigned via the IO Device Tree. This will be set to "---" after the scripting has been triggered.
Me _DIO.State	String	State of the Object scripting as it progresses through the IO assignment process. A state of Done means the scripting has completed. State can be the following Prep, Queued, Connect, Connecting, Select, WaitAsync, Process, Dispose, Done
Me_DIO.Target	String	Actual Name of Target for IO Connection. Contain the DIO.SG(Topic) from the IO Device Tree after the logic completes.
Me._DIO.TargetType	String	Type of IO Connection (IO DeviceTree , PCS, AppObject, MultiGalaxy). The wizard sets this value.

Functionality of the IO Assignment is contained within a single script named **scp.IOConfig**. The script has been designed to load the application engine (AE) that it is being hosted on in a way to prevent overloading the scan period and script execution timeout limit of the AE. There are two local memory variables assigned during the OnScan section of the script **scp.IOConfig**. These are:

objPerScan = 50; 'Number of objects allowed to execute at a time. If more than this number are deployed to a single AE, then the objects greater this number will be queued to be executed once the number executing drops below this number. A default value of 50 seems to work well.

ioExecTime = 5; 'mSec of engine scan time allowed to process an IO reference. Uses the AE Engine.ScriptExecutionTimeout.Limit to determine how many attributes of a single object can be processed each scan of the AE.

The combination of these two values allows large objects to complete without error and having many objects deployed to a single AE at the same time to not all execute at the same time which could also overload the AE. This mean that there is a slight delay between deployment and completion of the IO

Formatting Rule processing. As an example, during testing 400 objects with 200 IO attributes each took about 4 minutes to complete processing on a single AE. The status of the Object can be monitored with **Me._DIO.State** this will indicate where in the progression of execution the object is. A value of "Done" means that everything has completed.

For the MxReference of the InputSource and/or OutputDest to validate and not put warnings on the Instances. A valid target needs to be present in the galaxy to point to. This is the purpose of the **xDIO.SG.** prefix being needed. There also must be a DIClient object in the galaxy with this name (**xDIO**) and scangroup/topic (**SG**) configured in it. One was included in the aaPKG file but if this is not the case then simply create an instance of **$DDESuitelinkClient** and add a topic to this called **SG**. You can leave the **ServerNode** and **ServerName** fields blank since this is never to be deployed and doing so will prevent it from being deployed. If this object **xDIO** does not exist in the galaxy then all the instances will validate with warnings, they will still work but the warnings are annoying.

Rule syntax

All rules are enclosed in {} delimiters and are case sensitive for the rule part. The first 2 or 3 characters of the rule must be in UPPERCASE. The subsequent rule definition after the ":" or"." are not case sensitive.

Relative Metadata

Metadata from the model/deployment organization of the object can be included in the MxReference. These rules are simple 2- or 3-CHARACTER mnemonics enclosed in curly braces {}. The following table lists the rule syntax and example results that can be expected.

Rule	Meaning	Example Result
{PTN}	MyPlatform.Tagname	AOS02Platform
{ETN}	MyEngine.Tagname	AppEngine03
{ATN}	MyArea.Tagname	TankFarmArea
{AHN}	MyArea.HierarchicalName	CorpArea.SiteA.Tanks
{ACN}	MyArea.ContainedName	Tanks
{CTN}	MyContainer.Tagname	Mxr40536
{CHN}	MyContainer.HierarchicalName	Tank20304.Mixer
{CCN}	MyContainer.ContainedName	Mixer
{OTN}	Me.Tagname	VFD50745
{OHN}	Me.HierarchicalName	Tank20304.Mixer.VFD
{OCN}	Me.ContainedName	VFD
{AN}	Name of Attribute	FlowRate

Rule Enhancements

To facilitate the use of Arrayed UDT symbols in the controller (PLC) configuration a place holder rule of "[x]" can be inserted in the MxReference. For register table-based devices such as RTU, PowerMeters, and Older PLCs where the items in the table are based on an offset from a defined base register. The {OFF:x} rule can be used to set an increment from the starting point in **Me._DIO.BaseReg**.

Rule	Meaning	Example Result

[x]	Insert the value of **Me._DIO.ArrayElement** (ex:12) between the [ ] brackets	[12]
{OFF:7}	Add a numerical value to the **Me._DIO.BaseReg** (eg:40100)	40107

Model Relative Attributes

Inclusion of Feature Attributes added to this attribute, inclusion of attribute data from the other attributes on the Object, or inclusion of attribute data from other objects in the modeled hierarchy. This set of rules allow the formatting to reference any attribute from the Attribute, Object, Container, and Area. Using this rule allows formatting information to be placed within the modeled hierarchy as appropriate eliminating the need to repeat this information on every instance. For example, a common value can be placed on the Area object and referenced with {ATN.AttribtueName}.

Rule	Meaning	Example Result
{AN.FeatureAttr}	Insert the Feature Value for this Attribute. Any value from the features added to this attribute can be added to the MxReference. Place the FeatureAttrName after {AN. And before the closing curly brace (})	{AN.RawMax} 2000.0
{Me.AttrName}	Insert the value of Me.AttrName as a string into the MxReference. Any Attribute name on this object can be placed after the {Me. and before the closing curly brace.	{Me.OpcUaHdr} MCSConnection.PLC
{OTN.AttrName}	Insert the value of OTN.AttrName as a string into the MxReference. This is identical to the {Me.xx} rule. Any Attribute name on this object can be placed after the {OTN. and before the closing curly brace.	{OTN.OpcUaHdr} MCSConnection.PLC
{CTN.AttrName}	Insert the attribute value from the Container Object as a string into the MxReference. Any Attribute name on the container object can be placed after the {CTN. and before the closing curly brace.	{CTN.MQTTPrefix}
{ATN.AttrName}	Insert the attribute value from the Area Object as a string into the MxReference. Any Attribute name on the area object can be placed after the {ATN. and before the closing curly brace.	{ATN.OutStation} MainOS.Substation

Relative Metadata Parsing

Relative metadata commands allow you to parse the individual metadata available to a specific object attribute. The basic format of the parsing rule is {AAA:x(,y)}Where:

AAA is the 2- or 3-character name of the metadata value.
x will return the first x characters, -x will return the last x characters from the string.
y will return the number of characters from the x position within the string. This value is optional. y can also be the keyword END which means include everything from the x position to the end of the string.

Given a Tagname of Reactor40020_002 the following rules would yield the results of:

{OTN:7} - Reactor
{OTN:-3} - 002
{OTN:8,5} - 40020
{OTN:-6,2} - 20

Given an Area Hierarchy Name of USArea.West.California

{AHN:7,END} - West.California

This function has been enabled for

Rule	Meaning
{ATN:x(,y)}	MyArea.Tagname
{AHN:x(,y)}	MyArea.HierarchicalName
{ACN:x(,y)}	MyArea.ContainedName
{CTN:x(,y)}	MyContainer.Tagname
{CHN:x(,y)}	MyContainer.HierarchicalName
{CCN:x(,y)}	MyContainer.ContainedName
{OTN:x(,y)}	Me.Tagname
{OHN:x(,y)}	Me.HierarchicalName
{OCN:x(,y)}	Me.ContainedName
{AN:x(,y)}	Me.AttributeName

Deployment considerations and cascading changes from templates

Deploying the instances to existing runtime instances that were previously deployed will need to have to option of **Preserve Runtime Changes** be unchecked if a new rule is to be applied to the attribute. New attributes will have their rules process correctly, but existing ones will use the existing references that the previous rule resolved to if **Preserve Runtime Changes** is checked on deployment.

Since the InputSource and the OutputDest contents must be writable at runtime they cannot be locked on the $Templates. As such when changes are made to the rules of existing attributes on the **IO Feature** dialog these changes will not propagate down the derivation tree to the existing child objects. New Templates or Instances created after the change will have the current setting, but existing Templates or Instances will have the setting that existed when they were created.

A Wizard Option: **Propagate Template Changes,** will facilitate the propagation of unlocked attributes to child instances. This works by associating the attribute to this wizard option. In the Settings pane the attribute values that need to be propagated can be bolded with an X on them. An enter key typed on the setting value will flag this setting to be propagated. Remove the trimming checkbox to prevent these associated attributes from being trimmed from the deployed instance.

Examples

Examples of InputSource and OutputDest rules from given senarios:

1. The device has a fixed modbus register as in an ION Power Meter. The Target is IO Device Tree

and the DIClient Object is PMSvr with a Topic of MixerPM. The rule would be as follows:

a. Item Needed:

    i. 40011

b. Object Model:

    i. Mixer
        PowerMeter
            VIn_a – Attribute

c. Target Type: IO Device Tree
   Target: PMSvr.MixerPM

d. IO Format Rule

    i. xDIO.SG.40011

e. Result:

    i. PMSvr.MixerPM.40011

2. The device has a flat symbol naming definition. The symbol we want is Tank04MixerStartCmd. The object model is hierarchical. The Target is ABCIP.TankPLC. The rule would be as follows:

    a. Item Needed:

        i. Tank04MixerStartCmd

    b. Object Model:

        i. Tank04 – Container object.
            MTR004[Mixer] – Contained object.
                StartCmd – Attribute name.

    c. Target Type: IO Device Tree
       Target: ABCIP.TankPLC

    d. IO Format Rule

        i. xDIO.SG.{CTN}{OCN}{AN}

    e. Result:

        i. ABCIP.TankPLC.Tank04MixerStartCmd

3. The Device has a Function Block that has a Base Register of 40100. This attribute is offset from the base by 17 and is a 2-register integer. The Target is IO Device Tree and the DIClient Object is MBTCP with a Topic of PLC4 The rule would be as follows:

a. Item Needed:

    i. 40117 I

b. Object Model:

    i. FuncBlk
        Attribute

c. Target Type: IO Device Tree
Target: MBTCP.PLC4

d. IO FormatRule

    i. xDIO.SG.{OFF:17} I

e. Result:

    i. MBTCP.PLC4.40117 I

4. The Device has an UDT with the Name of MixerVFD and the Element we want is SpeedRef.

a. Item Needed:

    i. MixerVFD.SpeedRef

b. Object Model:

    i. Mixer005
        VFD40305[VFD]
            SpeedRef - Attribute Name

c. Target Type: IO Device Tree
Target: ABCIP.PLC2

d. IO Format Rule

    i. xDIO.SG.{CTN:5}{OCN}.{AN}

        1. First 5 characters of the Container Tagname (Mixer}
        Object ContainedName (VFD)
        . as a literal character
        AttributeName (SpeedRef)

e. Result:

    i. ABCIP.PLC2.MixerVFD.SpeedRef

5. The Device has an UDT Array with the Name of Mixers. There is a sub UDT called VFD and the Element we want is SpeedRef. The Array Element is 7. Here the object model names are close but

not exactly correct. Parsing is used to fix the naming. Literal characters are inserted to give the correct item name.

a. Item Needed:

   i. Mixers[007].VFD.SpeedRef

b. Object Model:

   i. Mixer007
      VFD40305[VFD]
      SpeedRef - Attribute Name

c. Target Type: IO Device Tree
   Target: ABCIP.PLC2

d. IO Format Rule

   i. xDIO.SG.{CTN:5}s[{CTN:-3}].{OCN}.{AN}

e. Result:

   i. ABCIP.PLC2.Mixers[007].VFD.SpeedRef

6. The PLC has an array of Power Meters with a UDT structure, one of the elements in the UDT is PhaseAVoltage. The array element we need is 27. The symbol name in the PLC is PowerMeter. The array element must be explicitly configured. The attribute name needs to be rearranged and characters removed.

   a. Item Needed:

      i. PM[27].PhaseAVoltage

   b. Object Model:

      i. Mixer – Container Asset.
         PM20321[PM] – Contained object.
         Voltage.PhaseA – Attribute name.
         _DIO.ArrayElement = 27

   c. Target Type: IO Device Tree
      Target: M580.Fast

   d. IO Format Rule

      i. xDIO.SG.{OCN}[x]{AN:7}{AN.-6}

   e. Result:

      i. M580.Fast.PM[27].PhaseAVoltage

7. The Device is linked to the AVEVA Telemetry Server on PCS with a scope name of Telem_Server_TSNODE. The data point is a dynamic one and is an Analog Input. It has scaling of ZS:0.0, FS:2000.0, DB:2.0. Outstation Full Name in the Telemetry Server is West.California.LakeOrville Point Number is 3.

   To create an dynamic Analog Input (AI) reference with:

   - a point number 3,
   - a Zero Scale (ZS) equals to 0,
   - a Full Scale (FS) equals to 2000,
   - a Deadband (DB) of 10 percent,
   - and reference ([Point-Name]) as Level

   ```
   Telem_Server_<Scope-Name>:%<Outstation-Full-Name>;<Point-Type>;<Point-
   Number>;ZS<Value>;FS<Value>;DB<Value>;[<Point-Name>]
   ```

   a. Item Needed:

      i. %West.California.LakeOrville;AI;3;ZS0.0;FS2000.0;DB2.0;[Level]

   b. Object Model:

      i. USArea - Area
         WestArea[West] - Area
         CaliforniaArea[California] - Area
         Orville - Object
         Dam203[Dam] - Object
         Level – Attribute
         Scaling – Feature
         RawMin = 0.0
         RawMax = 2000.0
         Deadband = 2.0

   c. Target Type: PCS Source
      Target: Telem_Server_TSNODE

   d. IO Format Rule

      i. xDIO.SG.%{AHN:7,END}.Lake{CTN};AI;3;ZS{AN.RawMin};FS {AN.RawMax};DB{AN.Deadband};[{AN}]

   e. Result:

      i. Telem_Server_TSNode:%West.California.LakeOrville;AI;3;ZS0.0;FS2000.0;DB2.0; [Level]