

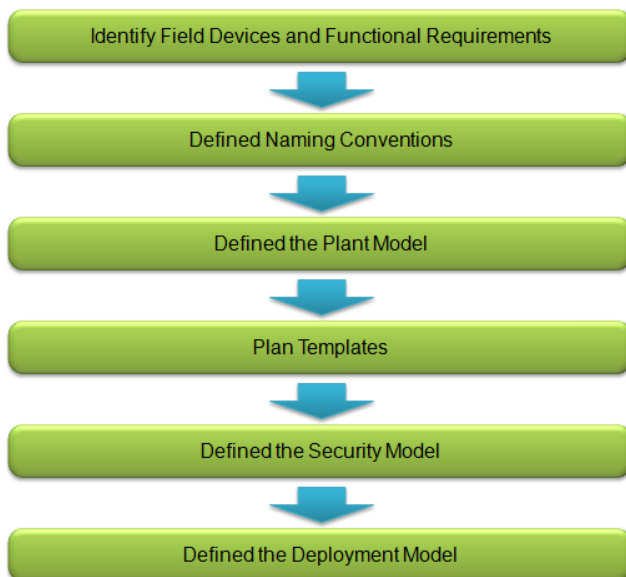
*NOTE: This article and all content are provided on an "as is" basis without any warranties of any kind, whether express or implied, including, but not limited to the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. In no event shall Wonderware North be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of information contained in this article.*

## Introduction

Last issue's TechTip explained the Wonderware Base Template Library (BTL) and how it can be configured to assist in the automatic referencing of I/O in your Wonderware Application Server system. This issue's TechTip will discuss a common architecture challenge and how the BTL can be applied in those situations. If you have not read the first TechTip, it is highly recommended, as the details in this TechTip rely heavily on the groundwork set in Part I.

## Project Workflow

If you have attended the Wonderware System Platform Part I training class, one of the first things covered is the project workflow a developer should consider following:



**Figure 1:** The Application Server Project Workflow

The BTL was created with this workflow in mind. You'll note that the of the two models (or "Views") explained in the workflow, the Plant/Model view is to be defined before the Deployment Model/view. The reason that this workflow is suggested is to allow for objects to be moved from one proposed hardware architecture to another without having to do any major re-engineering of the objects in the galaxy. Consider, for example, a project that was initially specified to be on a single machine. After the project is developed, the end user decides that they would like to implement load-balanced redundancy. If the plant model was developed prior to the

deployment model, the process would not require any re-engineering of the templates or instances, as the project would be built to run independent of the hardware. However, if the deployment model was planned first, there is a good probability that some re-engineering may need to occur to retrofit the architecture into a redundant configuration.

## Set the DIO.BindLevel to "MyArea"

The same philosophy applies to the BTL. Looking at the DIO.BindLevel and DIO.BindLevelEnum, you will see that the default is set to 4, or "MyArea".

Index	Value
1	DisableThisFunction
2	MyPlatform
3	MyEngine
4	MyArea
5	Use DIO.AppObjectName

**Figure 2:** The DIO.BindLevelEnum

While being the default, this is also the most versatile setting for the BTL configuration. Setting the DIO.BindLevel to 4 (or "MyArea") allows the user to have multiple DI Objects per engine. If the BindLevel were set to 2 or 3, and there were multiple DI Objects deployed on an engine, the last DI Object to be deployed would then become the object references in DIO.Name. Thus, anything that was meant to poll from the other DI Objects would be incorrectly referenced. Even if you are only planning on using a single DI Object, it is still recommended to use the "MyArea" DIO.BindLevel in the event that you may add new functionality to the system in the future. All of the following examples utilize the "MyArea" DIO.Bindlevel in both the DI Objects and the Application Objects

## Example Architecture: Using the Area as the Scan Group Name

One of the most popular implementations of the BTL is using the Area an object is in as the scan group name.

**Example:** An end user has a packaging line where each piece of machinery is controlled by a separate Contrologix PLC.

### Implementation:

In the galaxy, you have three options using the BTL:

- 1) Have a separate DI Object for each PLC,
- 2) Have a single object for all PLCs, and manually set the DIO.ScanGroupIndex for each component's objects, or
- 3) Have a single object for all PLCs, and have the objects set the Area name as the scan group name.

## TechTip: Debunking the Base Template Library: Part II

Out of the three options, Option 3 provides the developer with the greatest amount of flexibility matched with ease of development. Option 1 is viable, but may become problematic as you add more PLCs to the mix. Option 2 requires the developer to remember which section he is on as he is creating his instances.

### Extending the DIO.Name

There are two hurdles that need to be overcome to be able to make the architecture work in this manner. The first relates to the Area Model. If we were to set the Plant Model to have an area for the production line and then sub-areas for each component, then the sub-areas would not bind correctly. Since the DI Object would be placed in the main area, and the objects in the sub-areas, then the objects would look to the sub-area's DIO.Name, which would be blank. Thus, the first step is to remedy this by setting the DIO.Name to reference the main area. The best way to do this is to first create a separate template for the sub-areas (example:  $\$aSubArea$ ). This avoids the need to make the subsequent configuration change for each sub-area. The next step is to open the new template and select the Extensions Tab. After selecting the Extensions Tab, set the DIO.Name Input Extension to `mycontainer.DIO.Name`. Since nested areas also share containment, the sub-area's container will be the main area, and thus the sub-area's DIO.Name will reference the main area's DIO.Name. This will also allow you to add more area depth (i.e. more than two levels) to your model if you wish.

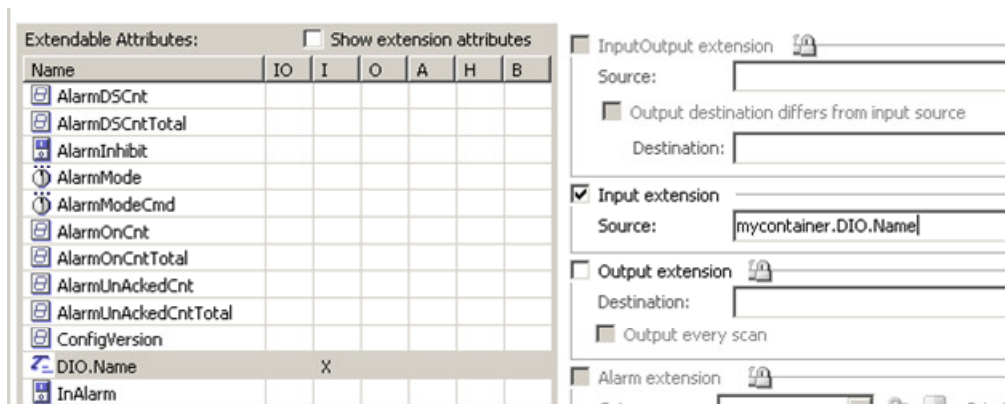


Figure 3: Setting the  $\$aSubArea$ 's DIO.Name extension

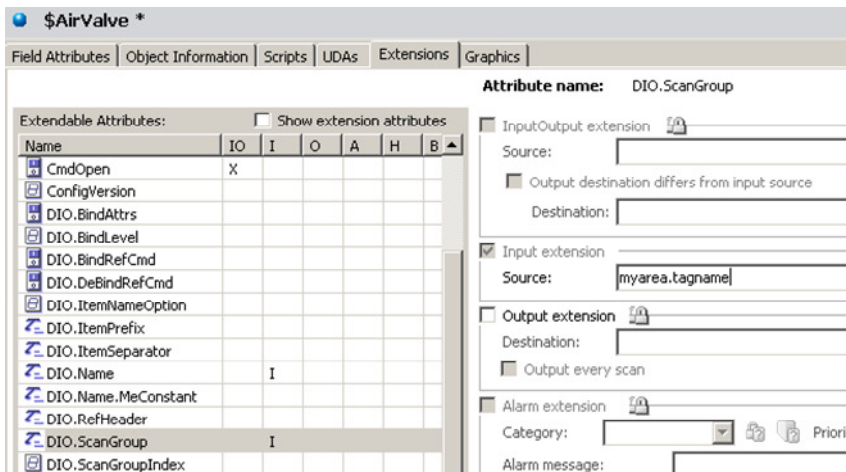
### Extending the DIO.ScanGroup

Once the area model is done, the next hurdle to get over is to set up the DIO.ScanGroupIndex in the Application Object Templates. By default, the DIO.ScanGroupIndex of Application objects is 1, which means that the objects would take the first scan group/topic listed in the DIObject's ScanGroupList. Thus, without setting some sort of automatic ScanGroup configuration in the BTL, you would need to change this for each instance group so that the first group would access the first topic, the second group the second topic, and so on.

Instead, set the DIO.ScanGroupIndex to 0, which tells the BTL that it is going to be fed its Scan Group Name in a different manner. Now that the Index is set, open the Extensions tab. While the DIO.ScanGroupIndex provides the BTL the information to GET the desired scan group, the ScanGroup is actually STORED in the DIO.ScanGroup attribute. Thus, by setting the DIO.ScanGroup attribute to have an InputSource, we can dynamically provide the ScanGroup to the BTL objects. In the case of this example, the sub-area's name will match the ScanGroup, so extend the DIO.ScanGroup to `myarea.tagname`. Upon deployment, the objects will then get the DI Object from

## TechTip: Debunking the Base Template Library: Part II

the main area, use the ScanGroup that matches the name as the sub-area, and whatever item naming convention you configure.



**Figure 4:** Configuring the DIO.ScanGroup extension in the AppObject Templates

### Summary:

(In all objects, *DIO.BindLevel* = 4)

- 1) Set the DIO object's *DIO.BindLevel* to 4 (MyArea)
- 2) Make a new template off of \$aArea called \$aSubArea
- 3) Extend the *DIO.Name* to in \$aSubArea to `mycontainer.DIO.Name`
- 4) Set the *DIO.ScanGroupIndex* in AppObject templates to 0
- 5) Extend the *DIO.ScanGroup* in AppObject to `myarea.tagname`