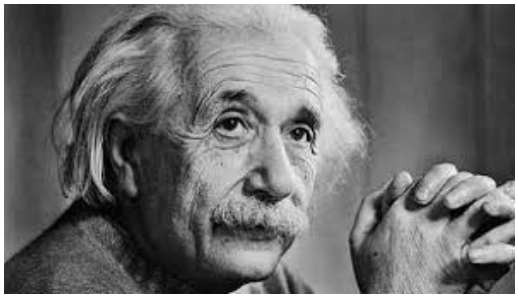
bachelor controls
EXCELLENCE SINCE 1983

Using AOI's Wisely








INTRODUCTION

- ***Can you trust an AOI?***
- ***How do you know that?***
- ***Why use an AOI in the first place?***

These don't seem like outlandish questions so why don't we dive into the subject and see what happens?

First of all, let me acknowledge that anyone interacting with the program whether that be a developer or maybe maintenance personnel, will have to spend a little time learning how to navigate among the UDT's the AOD's and finally the AOI's.

(We'll sort out the alphabet soup in a little bit.)

DIVING IN

Let's begin by recognizing that anything that you're going to use over and over needs to be thoroughly tested (validated). The same is true with an AOI. You have to test the AOI thoroughly enough to trust that it reliably does the job for which it is intended

I think most users of the Rockwell Automation ControlLogix family of controllers are familiar with ***User Defined Types*** (data types or UDT's). It seems to be common practice for developers to define UDT's to organize the properties associated with something that is repeatedly used within an application's program such as locations or an ISA-88 unit or perhaps something as abstract as a 'Finite State Machine' module. (We will save the discussion about the 'Finite Stae Machine' for another time.)

Using AOI's Wisely

DIVING IN continued

AOD's (***Add On Defined***) are a type of UDT where the developer sets up a data structure for the respective AOI's.

The AOI is a collection of instructions, that use controller tags to instantiate the data structure of the respective AOD to accomplish a module of logic that is repeatedly used. ***One example can be alarms***. Alarms need to notify people of their presence both visually and audibly. In addition, the Operator Interface (OIT or HMI) needs to be given a message to display and log (prompt). Alarms must also allow the operator to both silence and acknowledge the alarm. That logic block and its tags must exist for each and every alarm that is in the system.

It seems then that if we can develop a module of logic and tags that will always behave the same and avoid any chance of mis-typing an entry, that would normally be a good idea. So here comes the AOI. The AOI has a distinct appearance when it is used in a section of code within the processor. It is easily identified.

The AOI is set up to have certain tags made visible and these serve to be the inputs and outputs that are available to the programmer. There are also internal tags that can be configured to be accessible by the main application program for things such as HMI prompts, etc. So when programming each alarm, all the programmer has to do is enter the name of the AOI that he needs and then attach logic to it much like the way we deal with blocks of logic such as counters and timers in standard PLC logic.

We just can't 'lift the hood' on the counters and timers that are a standard part of the RSLogix 5000 instruction set. The AOI operates the same way except that it is incumbent upon the developer and user to validate the AOI so that they can absolutely trust how it functions and in addition we can 'lift the hood' on the AOI's that we develop in order to perform that level of testing.

MORE DIVING

You could get the same thing accomplished with a sub-routine, but the big difference between the AOI and a sub-routine is the ability to trouble shoot what is happening. In a sub-routine, data is moving in and out of the sub-routine so rapidly you can't be sure which sub-routine call you're seeing at any given time. ***With an AOI it is unique to the section of logic where it is instantiated.***

SUMMARY

In summary, I believe that the AOI is a very useful tool that can reduce if not eliminate error in dealing with highly repetitive modules of code. ***There is no reason to be afraid of them, but they do deserve respect and validation.*** When something goes wrong it is natural to condemn the thing you understand the least, so if you haven't invested enough time and energy to properly test the AOI, it will always be guilty until proven innocent.

I suggest that user's allow AOI's to be used. They are a great tool for modularity of code and ensuring accuracy of logic within them.

Ray Bachelor
Chairman of the Board
Bachelor Controls, Inc.
https://www.bachelorcontrols.com