PLC Solution Development

John C. Glisson

Piedmont Automation Inc.

Table of Contents

Version History

Version	Author	Date	Notes
V1.0	John Glisson	10. February 2016	First Release

Introduction

To facilitate effective solution development, certain protocols, governance, and habits are required from all contributing parties. Since the solutions offered span a wide range of distinct software and hardware platforms, it has become necessary to implement practices that facilitate efficient use of resources, minimize the potential for confusion, and maximize communication between contributing team members. Though no set of rules can deterministically guarantee the best solution to our clients' problems is produced, a uniform set of guidelines can best guarantee a consistent quality and timeliness of our deliverables. Since both the problems that PAI intends to solve as well as the tools that aide us in solving them evolve over time, this document is intended to adapt continuously while supporting a semblance of constancy between projects. The guidelines set forth here are to follow the "Robert's Rules of Order" principle- they are to be used to the extent that they help, but if they become too cumbersome in a specific application they can be bypassed.

Background

Overview

PAI has been known to develop using many different software platforms-

1. Machine controls
   a. B&R Automation Studio (C, C++, ST, Ladder, etc.)
   b. Allen Bradley Controllogix (C, C++, ST, Ladder, etc.)
   c. Visual Components
2. Database & Web Reporting
   a. XAMPP Stack (MySQL, PHP, Java)
   b. Scriptcase (PHP, MySQL)
   c. MSSQL
   d. BIRT
   e. Wavemaker
3. Windows Applications
   a. Python
   b. Visual Studio (VC++, VB)
   c. Eclipse
   d. Source Insight

Since the primary source of our revenue is hardware and software based on B&R's Automation Runtime RTOS, the main focus of this document will be on development within Automation Studio; however, the principles spelled out here are applicable to any and all development environments.

Guidelines

Variable Names & Object Identifiers-

1. Pointers
   a. p<Identifier> should be used to declare a pointer to a variable
      i. i.e. pointer to an axis object **pAxis**
2. Strings
   a. str<> should prepend all string variables
      i. i.e. string of someone's first name **strFirstName**
3. Boolean
   a. b<> should prepend Boolean variables
      i. i.e. Boolean of machine enabled **bMachineEnabled**
4. Iterators
   a. ii_<> should prepend iterators
      i. i.e. iterating through an array of objects **ii_objects**
5. Functions, Function Blocks
   a. fn<> and fbk<> should prepend functions, function blocks respectively
6. Constants
   a. All constants (#define macros, IEC declared constant variables, enumerations) should be in all caps
      i. i.e. global constant for max index of an array **MAX_IDX_OBJECTS**
7. IO
   a. Any intended input/output interfaces should be prepended with i<> or o<>, respectively
      i. i.e. function block with **iEnable** and **oStatus**
8. User-defined types
   a. User-defined types should be appended with <>_typ
      i. i.e. **MachineControl_typ**
9. Enumeration types
   a. User-defined enumerated types should be appended with <>_enum
      i. i.e. **StateMachineStates_Enum**
10. Namespace denotation
    a. Global variables should be prepended with g<>
       i. i.e. **gMachineState**
    b. Task-local scope variables should be prepended with loc<>
       i. i.e. **locMachineState**

IO Mapping

1. For sensors, actuators, etc. based on DI/DO mappings - never use "true"/"false" for value, as it is ambiguous as to what state that represents
    a. i.e. for gate sensor open close, enumerate constant "**INFEED_GATE_OPEN**" / "**INFEED_GATE_CLOSED**"
2. Variables entering the PLC should be prefaced with i_, exiting with o_
    a. i.e. **iDispenseSensor**, **oGateActuator**
3. Variables used in the IO mapping table should be as severable as possible from the main logic to facilitate portability
    a. e.g. use a structure like **gMachine.IOMap.InfeedSensors.iInfeedGateSensor** for IO values and a structure like **gMachine.App.InfeedVariables.InfeedGateSensor_Debounced** for application-use variables

State Machines

1. All State Machine State variables should be of an enumerated type
    a. i.e. **gMachineState = STATE_GLOBAL_MACHINE_RUNNING** rather than **gMachineState = 30**
2. All SM states enumerated types should be prepended with **STATE_**
    a. i.e. **STATE_GLOBAL_MACHINE_RUNNING**
3. All state machines need an "error" state
    a. I.e. **STATE_GLOAL_MACHINE_ERROR**
4. Whenever a SM is put into its error state, a corresponding Boolean VC error datapoint should be latched

AS/AR specific rules

1. -FOR LOOP iteration must iterate to and from constant defined values(with 0 and 1 being the exceptions) (i.e. FOR ii_systems := 0 to MAX_IDX_SYSTEMS BY 1 DO)
    a. -As such, the loop runs the same amount of times each cycle
    b. -As such, iterating through an array outside of appropriate bounds should not be possible
2. -Reusable code interfaces should be declared via IEC/C library declaration OR be its own task.
3. -No defining of variable-length strings (though in AS this may be compiler enforced?)
4. -Defined macros that are hardware-specific should be clearly marked
    a. -i.e. if an ETH interface identifier is to be used on two hw types where it is "IF2" on the first and "SS1.IF1" on the second-
        i. -In a task, there should be a clearly marked location like <#define ETHERNET_INTERFACE_TO_BE_USED "IF2">
        ii. -In a library, use library's .var file to define constant similar to define macro.

Motion/Mechatronics

1. -To avoid ambiguity regarding units, precison, and rollover, observe the following in regards to axis setup-
2. In the axis initialization file, use the convention that one unit here corresponds to the smallest measurable change in axis position
    i. -e.g. for an actuator that moves 1000 mm for every 35 rotations of a servo with desired resolution of 0.1 mm and desired units of mm, the numbers would be axis units : 10000, motor revs: 35
3. In the ACP10 Mapping file, add the designated scaling factor under "additional information"
    i. -for previous example, this would be "PLCOpen_ModPos = 0, 10" to allow PLCOpen and MAPP programming of movements in mm with resolution feedback of 0.1 mm

Comments & Changelog

1. The following ALWAYS needs to be noted SPECIFICALLY in the change log-
    a. Any changes to Axis initialization files or Acopos Par tables
    b. Any changes affecting the configuration/physical views (as these are not always easy to track with a diff/merge tool)
    c. Any changes to VC object files (as these can be very difficult to keep track of by using diff/merge tools)
    d. Any changes made to non-textual programming language tasks (i.e. ladder diagram, function block diagram, as these can be difficult to keep track of by using diff/merge tools)
2. The following changes ALWAYS need to be noted via code comments with a specific tag (i.e. TODO, JCG020916 to facilitate finding these places later)
    a. Whenever you have made any changes to a file created by someone else
    b. Whenever code is functional but incomplete (e.g. handling returns from a function that are not implemented yet)

Versions of Development Tools

1. Ask project manager

Other

1. Unless no other option, never use "negative logic"- (i.e. use "SystemIsEnabled", never "SystemIsNotEnabled" or even "SystemDisabled")
2. Unless no other option, never implicitly typecast integer to pointer
    a. -If the reason here isn't obvious, go study up on pointers. Think about ambiguity between function returning a UDINT value vs. function returning a pointer to a UDINT value.

3. If the notation rules leave ambiguity as to what the type or usage of the object is, it is ok to spell it out
4. When iterating through an array, never hardcode iteration max value, as array can become shorter and then you are iterating outside of array bounds!
    a. -Use global constant for both array length AND iterator max value
5. When using #ifdef <> macros based on hw configuration name, ensure an #else #error(configuration not defined) case is included, such that someone else who makes a new configuration can be notified at compile-time that there are configuration-specific options that need to be specified.
6. Whenever a rule needs to be broken (i.e. two rules conflict) make a note!
7. Except in trivial cases, NEVER hardcode an integer value in the code! Always either use a #define macro or use a constant, **ESPECIALLY WHEN CHANGING THIS VARIABLE IN ONE PLACE NEEDS TO BE REFLECTED IN ANOTHER**!
8. Overall, **CONSISTENCY IS KEY**.