



## *Object-Oriented Programming's Building-Block Approach Means Faster Development Times and Increased Quality and Troubleshooting Efficiency*

05/04/2008

---

By Jeff Miller

A common problem with many plants is that they have too many existing systems completed by too many different vendors with too many different standards. Whenever plants expand they tend to either 1) accept the control system standard from the vendor, or 2) pick the system in the plant they like the most, and request the vendor to use that as a guideline more than a standard. In today's highly competitive global market this approach does not maximize value.

Having non-standard programming practices causes increased training burdens and longer troubleshooting and development times. As a result, companies have begun to standardize programming practices; yet few are taking advantage of the latest techniques to maximize the most current controls strategies now available. Recently control systems have increased CPU speeds, memory and bandwidth, and now offer the ability to adopt pieces of the object-oriented methodology that revolutionized the commercial software industry in the 90s.

What is object-oriented programming? It's an approach that uses programming techniques to encapsulate code and support reusability for common recurring functions, promoting code modularity. The terms reusability and modularity should be emphasized because at the control layer, pure object-oriented programming cannot be implemented, but these aspects can be applied by creating an object-based programming methodology. The benefits of using these strategies are faster development times and increased quality and troubleshooting efficiency.

Certain control systems now lend themselves to an object-based approach, allowing the ability to define object templates and instances derived from object templates and giving programmers the ability to manufacture and configure core code as opposed to custom programming. Templates can be developed for each object at each of the software layers: the control layer, the supervisory layer and the Level III software layer. These templates include

standard data structures, properties (alarm conditions, color, historization, etc.), and methods (scripting, logic, etc.) for each of the objects.

At each of these layers, the software can be segregated, but tightly integrated through its data structures. Each layer provides specific functionality within the control system and consists of standard objects that have companion objects in each of the other layers connected by the data structures. This means the data can travel vertically between the layers, creating a powerful model that allows for the separation of business rules from the control system and special processing to be done at the appropriate layer.

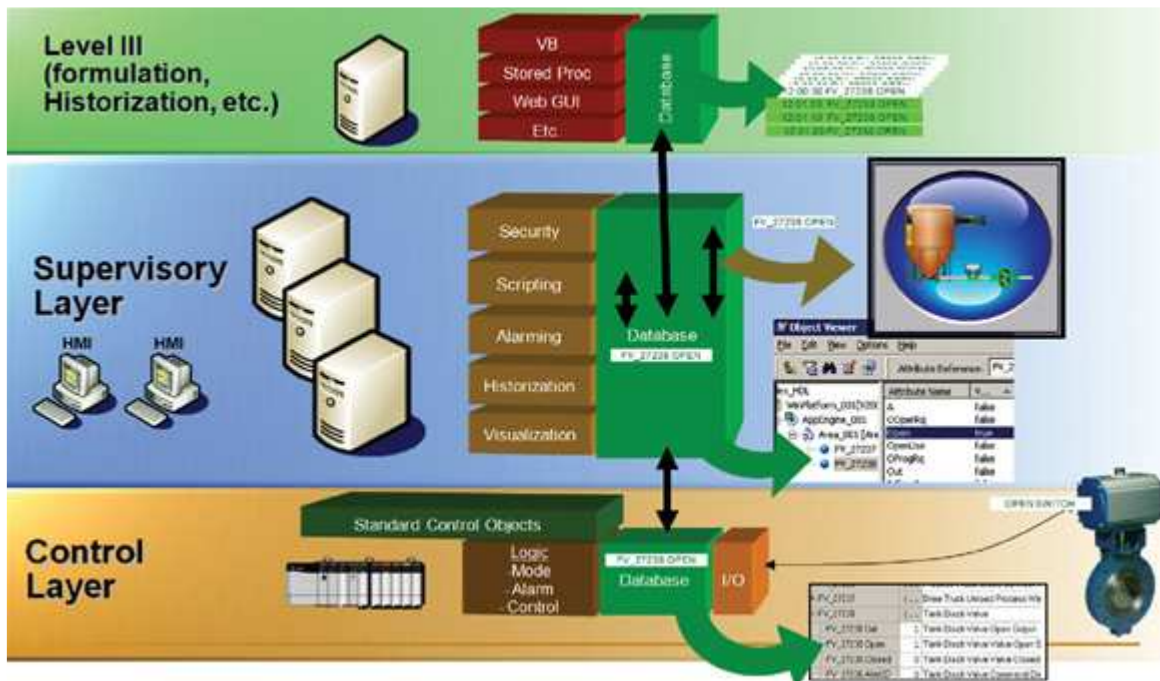


Figure 1. Object-oriented programming templates can be developed for each object at each of the software layers.

## Control Layer

At the control layer, all low-level machine and process control and all the interfaces to the real-world instrumentation, devices and equipment still occurs. The difference now is that a standard object, consisting of standard data structures, logic and I/O interface, is provided for each device and instrument type. Some standard process control system objects include:

- Valve
- Single Speed Motor (SSM)
- Variable Frequency Drive (VFD)

- Device controlled by PID
- Analog Input Device
- Analog Output Device
- Digital Input Device
- Digital Output Device
- Totalizer

So how does this work? Why wouldn't we create temperature, pressure and flowmeter standard objects? In essence we have. Each of these devices can be generalized to fit the analog input device object. The temperature may be an RTD, the level may be a 4-20mA signal, and the flowmeter may be pulses going to a high-speed counter card that ultimately gives a rate. The underlying processing logic for each device is exactly the same:

1. Take a non-scaled value as an input;
2. Provide a scaled value in the engineering units as an output to the process control system;
3. Provide alarming and control limits as an output to the process control system;
4. Provide an interface to the scaling, sample rate, alarming and control limits configuration as inputs.

With those assumptions, we can produce a model at the control layer where the core contains all of the standard code. Inside the core you will find the PLC logic to handle HMI interfacing, alarming, control limits, scaling and filtering. In an object-based system this code exists only once, and all instances reuse this code. There may be 1,000 devices that qualify as analog input standard objects, but only one set of the core logic exists in the processor. This has the effect of reducing the overall programming footprint within the PLC and decreases troubleshooting times because the technician doesn't have to look at extraneous code that has no effect on the actual sequencing of the process or system. It allows the engineers and/or maintenance people to focus on the critical sequencing of the system and not wade through logic that has been tested and verified inter-mixed with sequencing logic.

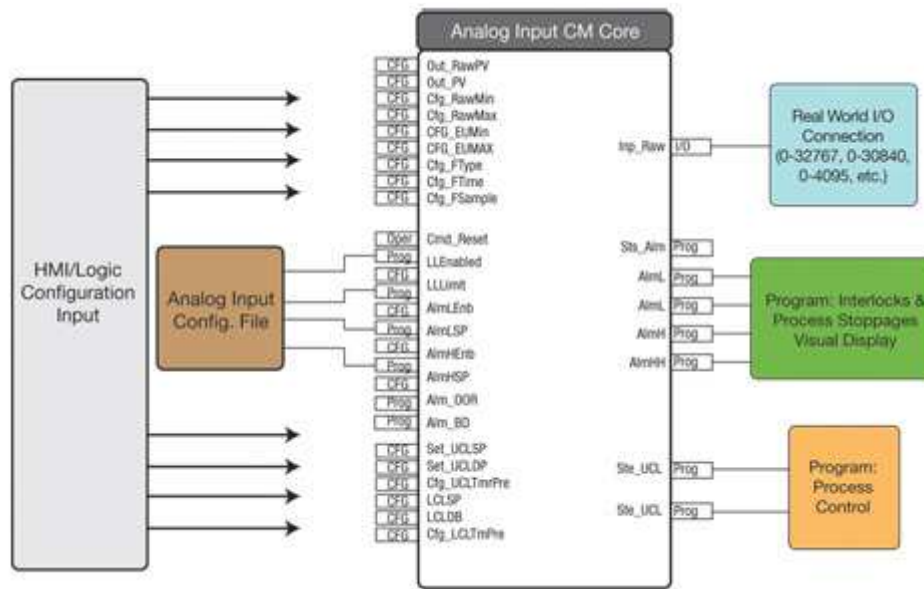


Figure 2. Object-oriented programming can produce a model at the control layer where the core contains all of the standard code to handle HMI interfacing, alarming, control limits.

## Supervisory Layer

At the supervisory layer, standard graphical objects are created that allow instances of master templates to be implemented with the concept of “one-change-one-place.” These objects are built to visually represent devices and contain scripting and animations to represent real-world status through the namespace, and provide connections for other graphic controls for configuration and control. Since this layer can be comprised of many computers, a single namespace exists within this layer to allow data to flow without needing to know where the data is located. Within this layer, some systems have or are moving towards having the ability to add/configure alarming, historization, security, scripting, advanced programming (.NET Applications) and I/O interfacing to other systems for each object. Since the graphical objects expose the properties of the control layer objects for configuration, many of the process changes that normally would require a PLC logic change can now occur with a mouse click or touch point. An example of this is enabling a high alarm on an analog transmitter. This can now be done by checking a box and entering a value.

## Level III Layer

Within Layer III, data historization and collection, formula management, material tracking, downtime tracking, inventory management and manufacturing execution systems reside. Detailed discussion of this layer is beyond the scope of this article, except to say that building

objects within it that maximize the lower-level data structures continues moving the data upward.

## Object Creation

Finally, we must answer one other important question: When is a device different enough to warrant a separate standard object? That depends on the plant, personal theory and the complexity or footprint of the differences. For example, a solenoid-operated valve may have one limit switch to detect the open position, or it may have two limit switches to detect both closed and open positions, or it may have no feedback. In these examples, the creation of a separate standard object is not necessary; the best approach would be to have a configuration bit that enabled each limit switch that is used; the reason being is that it takes minimal under-the-hood logic to make that happen.

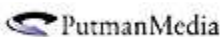
Let's look at another example. A conveyor system may have the following devices associated with it:

- SSM
- Two photo-eye switches

If there were more than three of these conveyor systems, and additional conveyors are expected to be added in the future, then the creation of a standard object for this type of conveyor system may be warranted. If this was a one-off system or rarity, then the proper choice would be to use a SSM standard object and two digital input standard objects.

In conclusion, every device should be either an object or built from objects at every layer. By implementing properly engineered standard objects in your system, you will achieve faster development times with better quality. Furthermore, by creating an object-based toolkit you can achieve the same look and feel for all systems within your organization. Someone trained on the toolkit can have no knowledge of the process, but feel comfortable troubleshooting the control system. Our organization has had many successes creating these toolkits and have helped our customers revolutionize their control systems.

*Jeff Miller is a group leader with systems integrator, [E-Technologies](#).*



[Contact Us](#) | [Advertise](#) | [Privacy Policy](#) | [Legal Disclaimers, Terms and Conditions](#)

Copyright © 2004 - 2010 Control Global [All rights reserved](#)

P: 630-467-1300 | 555 West Pierce Rd., Suite 301, Itasca, IL 60143