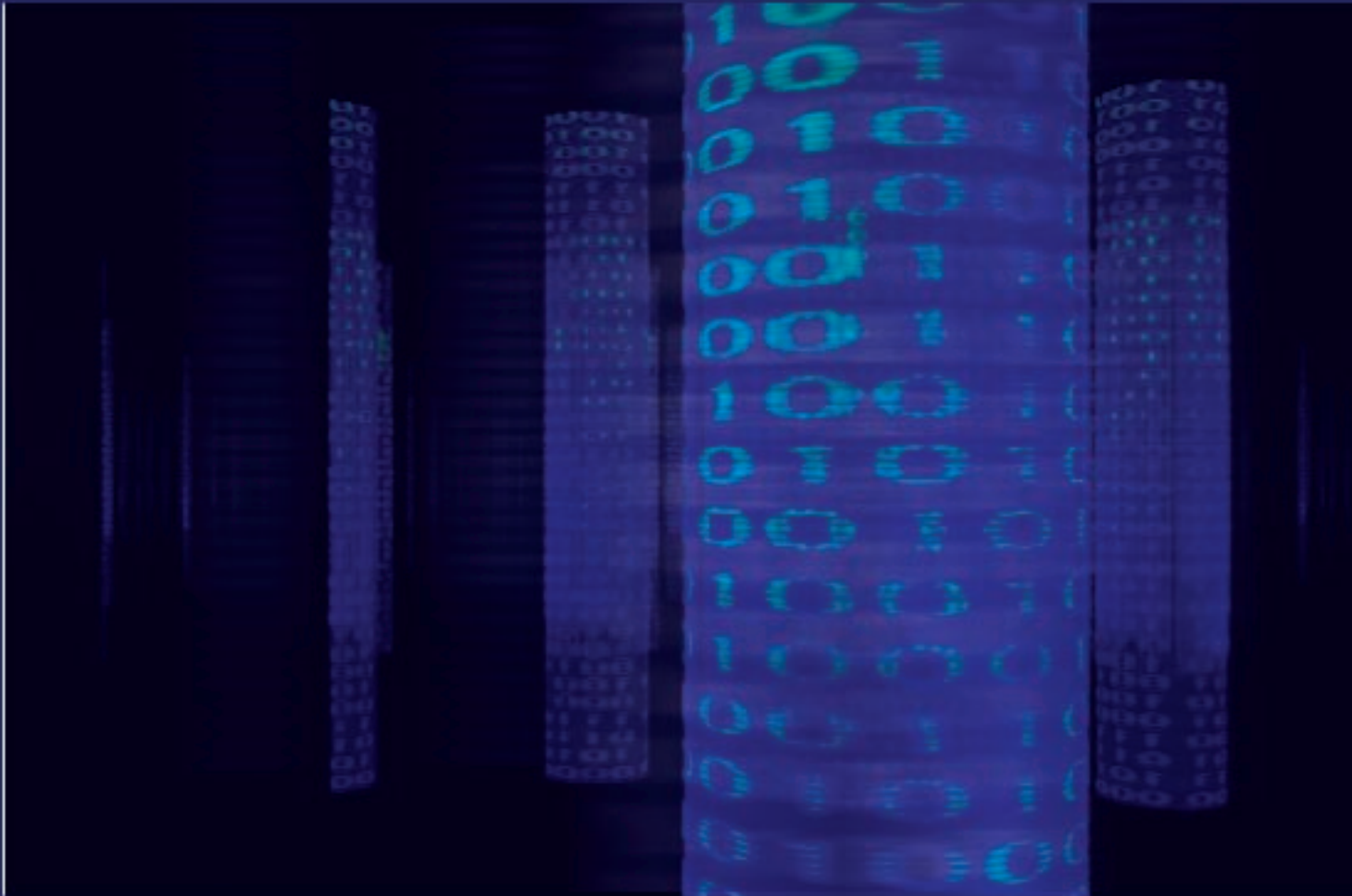


Unlocking the power of .NET for industrial automation



**.NET integration framework helps Transitions
Optical achieve easy, cost-effective assembly of
software objects for high performance visibility
to PLCs and other industrial devices**



Introduction

Industrial application developers have had two main options for interacting with production processes via programmable logic controllers (PLCs): they can buy a pre-programmed monolithic, shrink-wrapped human machine interface (HMI), complete and ready to go or they can customize their own solutions.

Shrink-wrapped HMI software packages are appealing because many complex tasks are hidden from you. Purchase the development software from an authorized distributor, load it into your development PC and then configure, debug and test. Then, just deploy the necessary runtime applications, data servers and configuration files on to your target PC or PCs. What could be easier?

But cookie-cutter HMI software solutions might not necessarily be the best or most practical approach for your specific industrial applications.

For one thing, while the shrink-wrapped HMI software packages enable connections to other vendors' devices, software, and systems via OPC or other standards, such connectivity is seldom adequate for high security or real-time control. And no matter how advanced the integration technology the package uses, you will end up lagging behind the technology curve. For example, if you had bought a package using the distributed common object model (DCOM) and wanted to benefit from advances in security and robustness that Microsoft had made since you bought the package, you would have to buy a new package. Moreover, the monolithic nature of the shrink-wrapped offerings often makes it difficult to embed third-party capabilities directly into your solution, thus limiting your options further.

Then there's training. Because the development environment and behavior of each HMI vendor's software varies, you'll need to acquire specialized skills to accomplish similar tasks. Training courses, material costs and schedules also vary by HMI publisher and many times are offered only through exclusive distributor channels. You could consider hiring outside help, but because of the specialized training and experience, the talent pool can be relatively shallow and therefore proportionately expensive.

And for many, cost of multiple deployments is an even bigger issue. Before you can actually deploy your solution to PCs, portable devices, or Web servers, you must typically have to pay for additional runtime software licenses. If you have more than a couple of users, this could amount to a considerable expense, often making this approach cost-prohibitive, especially if you are paying for more functionality than you actually never need.

Finally, there are the intangibles. As well-designed and flexible as these shrink-wrapped solutions might be, they almost always force compromises that would not be necessary if the solution were custom built for your specific applications. Whether that is a matter of function or just pride, it can be significant determining your satisfaction with the resulting interface.

The benefits of a custom solution

Building your own HMI and communications can overcome the limitations of shrink-wrapped offerings, especially if you have resources available in house. You may also save by building only the functionality that you need. And once built, you own it, so you can deploy to as many users as you want, with a look, feel and final design that are totally unique to you and your company and totally consistent with your best practices. On top of that, you can customize and adapt at will for subsequent applications.

But many companies have missed out on the benefits of custom HMIs because they just did not have the time or resources to devote and had to settle for a shrink-wrapped solution. Today, however, modern application development environments and component-based toolkits are enabling companies to enjoy the flexibility and affordability of a custom application, without sacrificing the ease of use and polish of a shrink-wrapped solution.

Unlike previous “Build-it-yourself” solutions, which only promised to enable customization but in the end required significant investment of time and effort, engineers and operations specialists can assemble the new interfaces by entering parameters into dialog boxes, with little, if any, need to program code. The secret is in using open .NET software development tools to craft inexpensive pre-designed HMI components and communications drivers into an HMI interface that fits your applications and sensibilities perfectly.

.NET has fast become the platform of choice for IT, with many companies already staffed and trained for implementation and long term support. Those who need the .NET technology for HMI development only, can download Visual Studio.NET Express edition from Microsoft at no charge.

This white paper will look at these components-based solutions in more depth, specifically as they relate to building HMI and communications for Programmable Logic Controller (PLC)-based systems within the .NET environment.

A brief history of PLC human interfaces

The computer-based Programmable Logic Controller (PLC) was introduced in 1968. It replaced the inflexible, hard-wired machine relay controls then in use. Early PLCs initially relied on basic, panel-mounted push buttons and lights as a rudimentary “human machine interface” or HMI.

The first PC-based HMIs for PLCs appeared in the 1980s. These were text-based and often proprietary to the respective PLC manufacturers.

In the 1980s, several small independent industrial software companies began developing more open, PC-based HMIs. Following the rollout of the Microsoft Windows operating system later in the 1980s, the first PC-based, graphical user interfaces for PLCs were introduced. As a class, these graphical interfaces provided both HMI and supervisory control and data acquisition (SCADA) functionality. Although far from perfect, Windows proved to be an excellent platform upon which to build HMIs and other industrial applications. Not surprisingly, this approach proliferated. As Microsoft added new capabilities to the Windows platform, these capabilities were gradually incorporated into the third-party industrial software offerings. Embedded Web portals are one example.

Then, around 2001, Microsoft introduced the .NET programming technology, the company’s next-generation application development and operating environment for Windows. The Microsoft .NET Framework makes full use of Web services and other open integration technologies to provide an extremely powerful platform for developing and deploying applications that simplify information sharing between computers and across operating systems.

One downside to the Microsoft.NET framework is that it has left many leading HMI software vendors lagging behind in support for current technology and unable to keep pace with rapid advances in operating systems and platforms. Unlike the monolithic HMI software offerings, today’s “do-it-yourself” solutions can take full advantage of everything the .NET framework has to offer through third-party plug-ins and components.

A closer look at .NET

Microsoft .NET is both a strategy and a software architecture. According to Microsoft’s Chairman, Bill Gates, the .NET strategy is “as significant as the move from DOS to Windows.” In fact, .NET represents Microsoft’s effort to finally “open up” Windows as a development platform, improve interoperability with non-Microsoft-based products and applications, and improve overall reliability and security.

As explained on the Microsoft Web site, “Microsoft .NET is a set of Microsoft software technologies for connecting information, people, systems and devices. It enables a high level of software integration through the use of Web services — small, discrete building block applications that connect to each other as well as to other larger applications over the Internet.” Written in extensible markup language (XML), a universal language for data exchange, these Web services allow data communication across the Internet (or within intranets) between otherwise unconnected sources.

XML, upon which all Web services are based, is an open industry standard managed by the World Wide Web Consortium. XML

separates the actual data itself from the display of that data, making it much easier to work with the data and move it between applications. XML enables data from multiple sources to be aggregated into a single unit of information.

The .NET Framework as an application development platform

The .NET Framework, which is at the heart of the .NET development platform, provides software developers with a single approach to building applications that can run on PC desktops, in mobile devices, and across the Web. Applications developed using the .NET framework also offer improved reliability, scalability, performance and security — all of which are critical for industrial applications. The .NET Framework enables developers to access the features of the common language runtime and offers many high-level services to minimize the recoding of commonly accessed features.

Thanks to a simple, consistent programming model and a number of built-in services, the .NET Framework provides a managed environment for building more robust, industrial-grade solutions than developers had previously come to expect from Microsoft. These services include garbage collection, cross-language integration and exception handling, enhanced security, versioning and deployment support, debugging and profiling services, and so on. Significantly, the .NET Framework simplifies application deployment and avoids the all-too-common version conflicts that resulting in the “DLL Hell” that developers experienced all too often with earlier Windows deployment models.

Reusable code means that application developers can “write less, and reuse more.” The .NET environment also supports the re-use of legacy code; legacy C/C++ code can be recompiled into managed .NET code. This helps to preserve critical intellectual property.

.NET fosters reusability through an Object-Oriented Programming (OOP) software design and construction method that enables creation of individual software components (classes), assignment of specific behaviors to each (methods), within specified limitations (properties). They can be grouped into discrete variables contained within a class in your program, enabling partitioning of groups from each other, and preventing any interaction that is not controlled by a specific operation in that class.

In this way, each object’s functionality is expressed as a collection of properties and methods to which it responds. Other code can summon these methods and use them to retrieve or change some information, but the other code cannot affect the information or processes of other objects themselves.

This enables .NET Framework application developers to build executable programs in .NET and then deploy them on different computers and computing devices. This platform independence makes the specific programming language used irrelevant, since all become equal for most applications.

Visual Studio.NET

Microsoft CEO, Steve Ballmer, claims that the company spent more than three quarters of a billion dollars developing Visual Studio.NET, an off-the-shelf toolkit for developing .NET-enabled applications. Smart application developers will take advantage of this massive investment on Microsoft’s part to improve the quality of and reduce development costs for their own applications. With Visual Studio.NET, you pay for these powerful development tools once and can then use them over and over again to develop and deploy applications.

Visual Studio.NET provides today’s application developers with a common, easy-to-use toolset for writing managed code for Windows Forms, PDAs, and Web applications. These applications can range from simple HMI and blind data acquisition applications, to more sophisticated supervisory control, batch management, performance monitoring, and supply chain applications at the factory or plant level, on up to intra- and inter-enterprise applications.

Many manufacturing companies — large and small — have adopted Microsoft as their standard for front office productivity software, cross-platform integration, and application development. With so many companies already using Visual Studio.NET to develop plant and enterprise business applications, it makes a lot sense for them to also take advantage of their investment in Microsoft development tools and training to develop plant- or factory-floor applications.

.NET as an enterprise integration enabler

In today’s competitive environment, manufacturers can’t afford to allow their PLCs and other real-time machine and process control systems to function as isolated islands; independent of maintenance, supply chain, and other manufacturing and information systems. Microsoft’s integrated .NET solution approach for manufacturing eliminates the need to cobble together different point solutions. Instead, Visual Studio.NET, SQL Server, Sharepoint Portal Server, and BizTalk Server are some of the Microsoft components available to help application developers create plant and enterprise applications that can work together as a unified solution.

An off-the-shelf solution for HMI software companies and independent application developers alike

Literally tens of thousands of components for the .NET Framework are available off-the-shelf (not to mention hundreds of “how-to” books and other support materials), these very same benefits are also accessible to developers of industrial custom HMI and data acquisition applications. Unlike companies that chose to go the monolithic, shrink-wrapped HMI solution route, developers of custom applications are also free to take advantage of a huge assortment of third-party, .NET-compliant industrial components and communications drivers that can be easily plugged into their custom applications.

Low-cost, .NET communications libraries and industrial components provide the “missing link”

While the .NET Framework and Visual Studio.NET application provide an ideal development and deployment environment, these off-the-shelf tools can only take you so far when it comes to developing application-specific solutions for industrial HMI and SCADA. The good news is that there are high-quality, low-cost .NET communications libraries for PLC’s, industrial graphics components, and third party tools in the form of dynamic link libraries (.dlls) to complete your solution. A Windows dynamic-link library is a software file containing a collection of subroutines or classes used for application development. Dynamic-link libraries provide programming services that facilitate code and data sharing in a modular fashion.

A variety of different licensing arrangements are available. These provide great pricing flexibility, ranging from single-developer/single-seat licenses, to source-code-provided and unlimited, runtime-free licenses.

Third-party libraries, built with .NET Framework managed code, can also be easily plugged into your own custom .NET HMI, SCADA, or other industrial solutions. These can provide a direct Ethernet communication channel with Allen-Bradley, GE Fanuc, Schneider and other popular PLCs. By using direct Ethernet connectivity, these .NET communications libraries eliminate the need to purchase a communication driver from the PLC manufacturer or use an OPC server with its associated hardware and software costs. While OPC plays an important role within today’s automation and integration strategies, it’s important that the technology be used in an appropriate manner. Not only is OPC cumbersome to implement in the .NET framework, it may also be too slow for many real-time applications. According to recent survey results of 113 OPC users from Fortune 500 companies reported in a white paper jointly produced by the cyber-security experts at British Columbia Institute of Technology and Byres Research, inappropriate use of OPC technology might well be putting industry at risk. Over a quarter of the end-users surveyed reported that loss of OPC communications would result in a shutdown of their company’s production.

Many excellent pre-defined instrumentation and dynamic graphics components developed for the .NET Framework are also available to plug into your .NET Windows Forms, Web Forms, and Embedded HMI and SCADA applications. These include components such as electronic dials, sliders and knobs and communications (figure 1) all pre-designed for interchangeability within the .NET environment. These minimize or completely eliminate the need to tediously draw your own display objects or code the appropriate connections to dynamic or static data sources.

.NET data visualization components (such as dynamic charts and graphs) and complete dashboard packages (with dynamic performance gauges) are also available to make it easy for developers to create customized data visualization and real-time decision support applications.



Figure 1: Pre-designed graphics can eliminate need to build electronic dials, sliders, knobs and other display objects from scratch.

Using the tools

Transitions Optical is an example of a company that has successfully applied off the shelf .NET technologies in manufacturing. Transitions Optical has production facilities in five global locations and uses Allen Bradley PLCs in the execution of all shop floor systems. Transitions Optical, Inc. is the manufacturer of Transitions® lenses, the #1-recommended photochromic lenses worldwide. Transitions Optical has developed the world’s most advanced photochromic lens technology which adjusts automatically to changing light conditions — going from clear indoors to as dark as sunglasses in bright, glaring sunlight. Data from the PLCs are critical to many critical management functions.

Data from its lot tracking and tracing system, for example, are used by the plant MES system to generate lens packaging labels that conform to FDA regulations.

Getting this data requires polling the PLC that controls an auto verification system and Transitions has built custom interfaces to do this using programming tools from CimQuest INGEAR. These tools package all the necessary Allen Bradley drivers with Microsoft Visual Basic or Visual Studio.NET routines and procedures that are commonly used to provide industrial services.

According to Patrick LaFerriere, who is responsible for the manufacturing execution systems for Transition's shop floor operations, this kind of programming requires knowledge of Microsoft programming languages and of the industrial processes that are being programmed.

"It's relatively easy to find Visual Basic or .NET programmers, I can have the Microsoft programmers writing for the PLC very quickly," said LaFerriere.

CimQuest INGEAR provided Transitions with collection of runtime free class libraries that simplify building, developing and deploying connected systems for manufacturing applications using Visual Basic or Visual Studio .NET. INGEAR.NET provides a direct Ethernet communication channel to programmable logic controllers needed to write HMI interfaces to PLC controls or to acquire data from them. It requires no additional third-party components, drivers, APIs or tools, such as OPC Servers.

The tool uses only three primary classes of code, one class manages the PLC, one manages the data that are read from or written to that PLC, and one manages groupings of methods, properties and events to optimize read/write operations on a collection of tag classes.



Figure 2: Only three primary classes of .NET code manage all communications with the PLC.

The properties, methods and events in the PLC controller class, represented by the lower portion of figure 2, establish the connection to the Allen-Bradley controllers, executes communication transactions, such as individual Tag Read/Write and TagGroup Read/Write operations, and manage error reporting and notification.

The properties, methods and events of the tag classes, represented by the green rectangles in figure 2, represent data values to be read and written to the AB programmable logic controller device. This covers operations such as addressing data tables, defining data types, setting format for data values, time-stamping operations, etc. And the TagGroup class properties, methods and events perform optimized read/write operations on a collection of Tag classes, including adding and removing tags from groups.

"With .NET we can simply define everything we need up front, make one call to the PLC and receive everything we need in a single array," said LaFerriere.

LaFerriere says also that accessing the PLC more efficiently in this way also puts less strain on the system, which reduces the chance of failure, especially critical for an operation which must run 24/7 to meet production goals.

Using such capabilities, LaFerriere says that he is able to integrate easily with applications that he might never have been able to access cost-effectively. In addition to the integration of track and trace data from the PLCs, as mentioned above, there are numerous possibility for data mining to drive enterprise resource planning (ERP) functions.

Indeed, Transitions is now literally transitioning the communications to the web-based framework, so that all global locations can share a common database, common key performance indicators (KPIs) and can report data consistently.

Whether you choose to go with a shrink-wrapped over a custom HMI, of course, depends on many factors, including available budget, need for customized interfaces, number of potential users, maintenance requirements and availability of qualified personnel. But if you need a professional, perfectly tailored interface for your PLC application, there are very few instances in which using .NET technology to customize an interface will not pay for itself many times over, while leading you to an industrial HMI that is truly your own.

For a step-by-step demonstration of how to create your own custom HMI and SCADA applications for Allen-Bradley, GE Fanuc, and Schneider PLCs using Visual Studio.NET and downloadable, runtime-free industrial communications drivers and .NET graphics components, visit www.ingeardrivers.com.