



SL-GMS[®] and Microsoft[®] .NET Technologies

Executive Summary

This white paper will provide a very high level view of Microsoft .NET and include typical reasons why the Microsoft .NET platform is being chosen to architect distributed enterprise solutions. As part of this solution, Microsoft Visual Studio .NET is described, referencing the unique features the tool has for creating rich client applications. Finally, an introduction to the SL-GMS Developer for Microsoft .NET product is provided to demonstrate how it enhances the use of Visual Studio .NET in creating high-performance dynamic graphic displays for client applications.

Introduction

SL-GMS is known worldwide as the most complete, high performance dynamic graphics system available for real-time monitoring and control applications. SL-GMS is widely used in various industries such as process automation, network monitoring, power distribution and traffic control. Providing products since 1985, SL has maintained the goal of protecting its customers from shifting trends in user interface technologies. SL has helped its customer base retain their investment in user interface implementations making the transition smooth between such varied technologies as proprietary graphics systems, XWindows, Motif, Microsoft Windows, MFC, Internet applications, ActiveX, Java and Microsoft .NET. As technologies change, the SL customer doesn't have to think about entire re-writes of user interfaces. SL-GMS Developer for Microsoft .NET is a graphical development system specifically designed for use with Microsoft's Visual Studio .NET development environment.

Why Microsoft .NET?

Software product architects in the past often focused on solving a particular focused business problem and created applications highly specified to addressing that point solution. This was effective in that it enabled developers to deliver applications that would be successful at providing needed functionality to a well-defined user. Recognizing that many of these point solutions use common building blocks gave rise to the concept of having common components which would promote re-use of software and reduction of development costs. Technologies such as Microsoft COM, CORBA, and Enterprise Java Beans were developed to address this need for component object brokering between applications. This has been a

powerful architectural model for a number of years; however, it can create “application stovepipes” or “islands of information” that cause significant problems in architectural reuse when trying to integrate these applications to implement enterprise-wide business solutions.

Today, complex enterprise solutions are being constructed that integrate existing application services which may reside on the local network or are only accessible on the Internet. Older common object model technologies are often not suited for this new type of solution because they are difficult to implement, not scalable, not suited for Internet communication across firewalls and only integrate with common implementation technologies and platforms. Microsoft .NET resolves many of these issues with .NET server technologies, the facilities for creating Web services and the ease of building distributed clients with Visual Studio .NET.

The trend in enterprise solutions is to move from a strictly server-client paradigm to a distributed-computing paradigm composed of loosely coupled services that could reside on any smart device anywhere on a local network or on the Internet. The Microsoft .NET platform is clearly a very competitive choice for an implementation technology for new solutions in which either software services, Web services or clients can be restricted to the Microsoft .NET platform. ISV's who have already implemented legacy products using Microsoft technologies must struggle with the decision over which products must be migrated from strictly COM based technologies to .NET, to be commercially viable in a world that seeks open standards and ease of integration in a distributed enterprise environment.

How does Visual Studio .NET help?

While Visual Studio .NET provides tools that assist in creating new Web services or exposing existing code as a Web service, we want to focus on the features which make it optimal for creating distributed user interface clients. When building Web service clients, Visual Studio .NET provides classes to support writing, reading and validation of XML as well as the capability of recognizing existing Web services and automatically generating a proxy class which exposes the Web service interfaces as simple methods. Other ease of use features for data connectivity including classes to support TCP/UDP, HTTP, and sockets.

Visual Studio .NET also has many features which make it easier to develop applications including:

- Common language run-time - supports many languages with a consistent type system and class hierarchy
- Assemblies - manages units of deployment and version control
- Easy form building features for both standard applications and web-based applications

-
- Network deployment - in a variety of service and client configurations
 - Security - built into the run-time environment
 - Features to support wireless client development

How does SL-GMS Developer for Microsoft .NET fit in this environment?

SL-GMS Developer for Microsoft .NET contains the following:

- SL-GMSDraw Dynamic Graphic Editor
- SL-GMS Microsoft .NET Viewer control
- Symbol libraries
- Application examples
- Documentation

SL-GMS Developer for Microsoft .NET allows users to rapidly create dynamic graphic displays using the SL-GMSDraw Dynamic Graphic Editor. Non-programmers can configure displays with pre-defined components or design and animate their own components. These displays can then be previewed with simulated data or attached to real-time data from within SL-GMSDraw.

SL-GMS Developer for Microsoft .NET also provides a custom .NET control which can be used in Visual Studio .NET to create .NET applications that render these displays in either a Windows Forms based application or a browser-based application for Internet use.

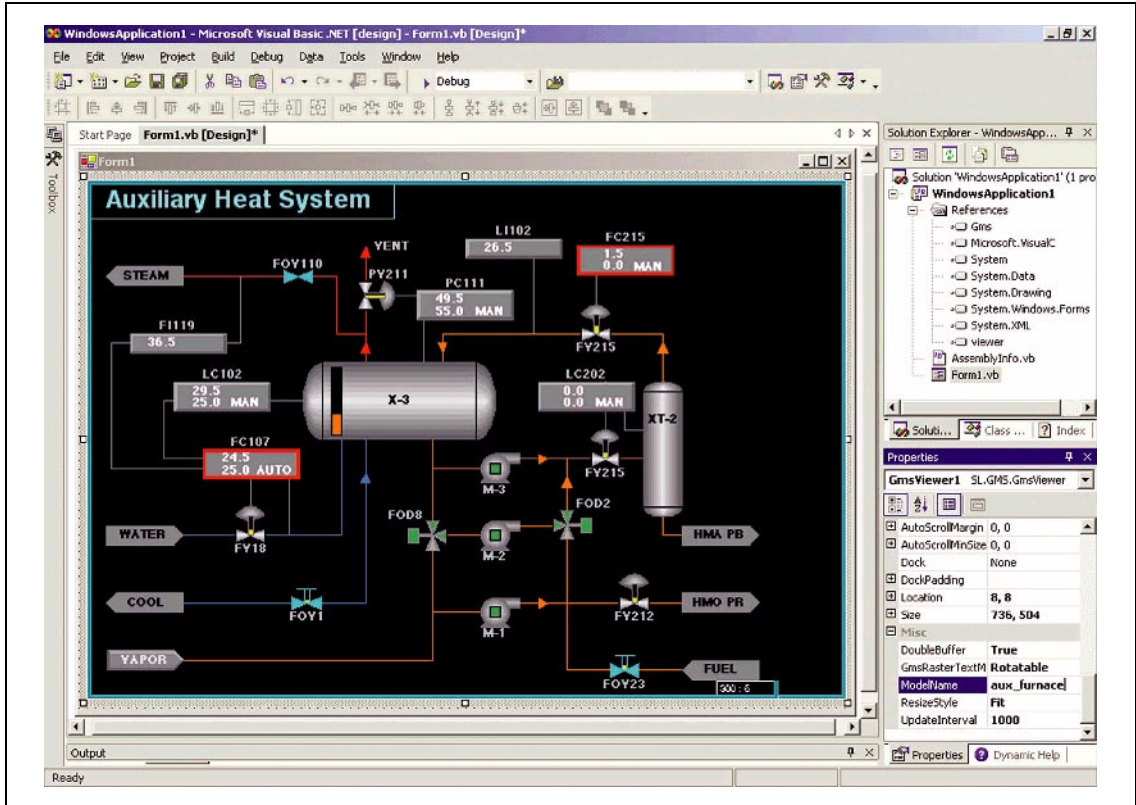


Figure 1-1: Furnace Model in Visual Studio .NET

SL-GMS Developer for Microsoft .NET takes advantage of the powerful development environment provided in Visual Studio .NET which allows for rapid development of both standard applications and web-based applications. Because the custom .NET control can be easily used in this development environment, these applications can also take advantage of the rich support .NET has for driving these displays from distributed data services. For example a plant floor operations control application could be driving the display from an OPC server, an SQL database, or from proprietary SCADA or DCS data systems. Another web-based application could be using the same user interface, with data from an OPC-XML or other Web service.

SL-GMS Developer for Microsoft .NET is a fully .NET compliant real-time graphics development system.

- Enabled with classes, methods, properties, and events from within Microsoft Visual Studio .NET

- Usable with any .NET supported language including C++, C#, or Visual Basic
- Provides current SL-GMS users a supported migration path to .NET

SL-GMSDraw Dynamic Graphic Editor

SL-GMSDraw is a sophisticated graphic editor that facilitates the construction of dynamic graphic models. SL-GMSDraw provides tools that allow the user to specify an object's dynamic behavior and the variables that drive that behavior. Dynamic behaviors can be tested within the editor to ensure proper operation before using them in an application.

SL-GMSDraw is an Windows/MFC-based editor that provides advanced features, such as snap and point modes, alignment tools, object ordering tools, object reference point tools, and point editing. It also provides standard editing features, such as cut, copy, paste, undo, redo, toolbars, tooltips, and pop-up menus.

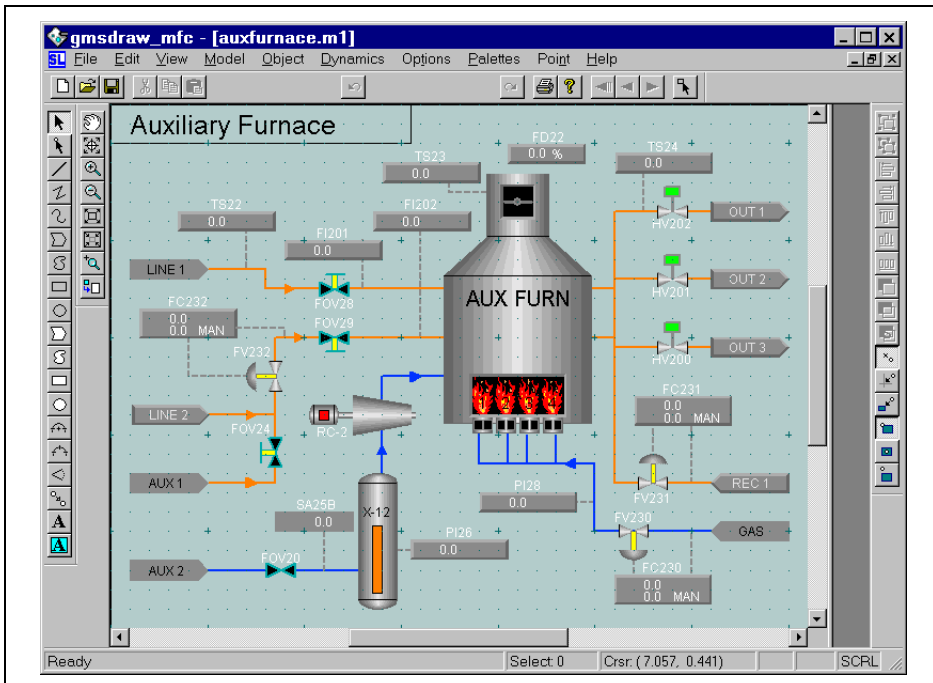


Figure 1-2: SL-GMSDraw Displaying Furnace Model

SL-GMSDraw is often used to develop libraries of dynamic symbols and graphic elements which are collected in palettes for easy use. Such symbols and graphic elements can be constructed from simple graphic primitives and shapes. There is

no limit to the number of hierarchical levels that can be used to build a model. An object can be part of a group that is also a part of another group, and a model can be used as a part in another model. SL-GMSDraw can also import graphics from a number of different sources including AutoCAD and Windows metafiles such as Visio vector drawings, as well as JPEG and BMP image formats.

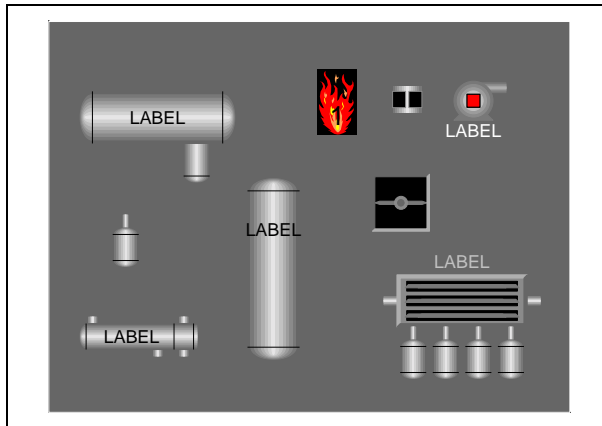


Figure 1-3: Custom Equipment Palette

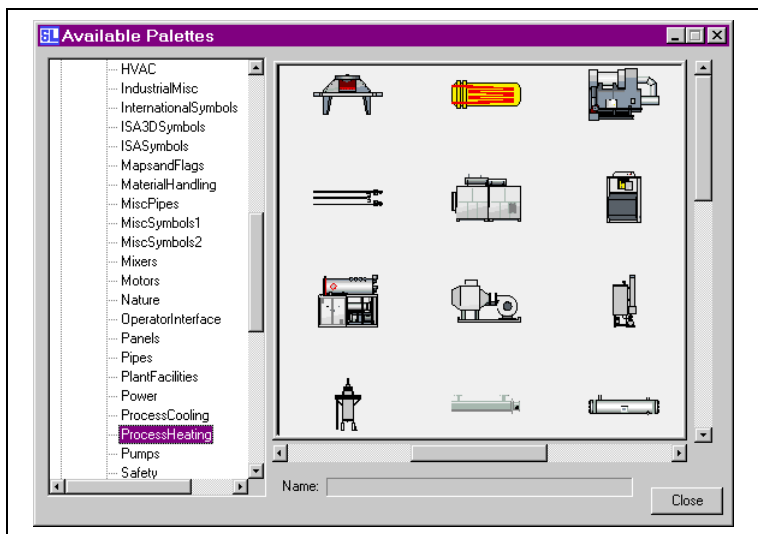


Figure 1-4: Palettes of Symbol Factory Objects Included with SL-GMSDraw

Dynamic Graphics

SL-GMSDraw also provides the ability to define dynamic changes to graphic objects in response to changes in data variables. A robust library of basic functionality is provided for the management of dynamic graphics and the control of user interaction. Objects in the graphical model are automatically changed at runtime to reflect the state of application-specific data variables. The dynamic behavior library uses proven, tested, and efficient techniques for storage of model files, object extent processing, execution of dynamic properties, redraw operations, and much more.

Dozens of dynamic behaviors are available that are controlled through application variables and events including:

- real-time movement, rotation, scaling, and path traversal
- percent fill of irregular objects
- visibility and detectability change
- text label, formatting, and alignment change
- fill color and style change
- edge color, width, and style change

Dynamic behaviors that change object properties are triggered by data thresholds that are defined using constants or variables.

Using the SL-GMS Microsoft .NET Viewer Control

The SL-GMS Microsoft .NET Viewer control can be instantiated from the Visual Studio .NET control toolbar and inserted into Windows Forms or WebForms based applications using any .NET supported language. The control is then sized and positioned in the Windows form and will be used to display the graphic models built with SL-GMSDraw and animate them with live data. Changing the value of the control's **ModelName** property changes the model displayed. Thus a single control can display many different models. The variables which drive a particular model's dynamic behaviors are obtained using the control's methods. These variables can then be driven by other .NET controls or data acquired by the container application. Some of the key properties and methods are described below followed by an example scenario of their use.

Properties

ModelName - Specifies the name of a file containing an SL-GMS graphical model.

UpdateInterval - Optional time interval in milliseconds. At the end of each time interval, dynamic behaviors triggered by data changes are updated and their associated graphics are redrawn. Alternatively, the container application can provide event-driven data updates by executing the **SetGmsVar()** method described below.

Methods

GetGmsVarNames - Returns the list of variable names on the currently loaded model

GetGmsVarTypes - Returns the list of variable types on the currently loaded model

GetGmsVar - Gets the current variable value

SetGmsVarType - Sets a variable type

SetGmsVar - Sets a variable value

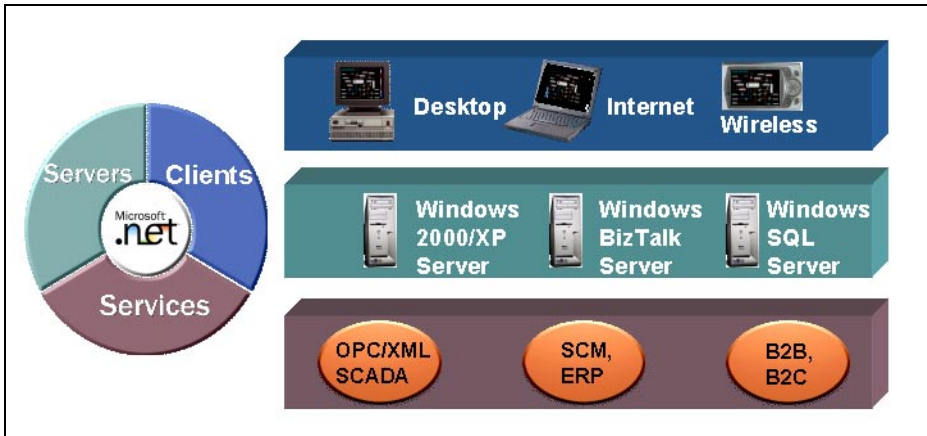
Example Scenario

With the wide variety of servers and support for creating Web services and clients with the Microsoft .NET environment, there can be any number of architectural designs for creating a distributed system. One possible scenario for process control might unfold as follows:

- End-users build up displays with SL-GMSDraw, using palettes of common process symbols. Each symbol has one or more variables that drive its dynamic behavior.
- In SL-GMSDraw, variables for each symbol are “re-named” to specific real data tagnames. The tagnames differentiate the symbol from each other and associate each symbol with a particular piece of real equipment.
- As each display is loaded into the SL-GMS Microsoft .NET Viewer control using the **ModelName** property, the variable tagnames are retrieved using **GetGmsVarNames()**.
- Using the tagnames as keys, the container application then queries the data source for the current values.
- The application uses **SetGmsVar()** to update the value of each variable in the display.
- The Viewer control automatically determines the necessary graphic updates based on the new values.

Summary

While Microsoft .NET and Visual Studio .NET resolve many enterprise development issues with their .NET server technologies, there are still challenges to developing commercially viable applications in a world that is coming to expect open standards and ease of integration in a distributed enterprise environment.



SL-GMS Developer for Microsoft .NET:

- enables the creation of rich user interface displays by non-programmers
- offers high performance
- allows for the distribution of these displays with multiple data sources across the enterprise

Development of a system that meets these requirements would take many man-years of development effort with Visual Studio .NET alone. SL-GMS Developer for Microsoft .NET can reduce this effort to a matter of man-months rather than years, expediting time-to-market.

Microsoft, ActiveX and Visual Studio are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. They are mentioned in this document for identification purposes only.

SL, SL-GMS, GMS, SL Corporation and the SL logo are trademarks or registered trademarks of Sherrill-Lubinski Corporation in the United States and other countries. ©2002 Sherrill-Lubinski Corporation. All rights reserved.