

Kinesix Adopts .NET Architecture The KX EDGE® Application Development Suite

Introduction

Flexibility, speed and rapid response are the keys to being competitive in today's economy. The business critical, real-time data typically only seen in a high end control room must now be instantly available throughout the enterprise. This applies to all industries including: satellite command and control, financial and energy trading, telecommunications, oil and gas transportation, and many others. The problem is how to implement systems that are quick to deploy and that integrate well with other enterprise systems.

Many options are available to help solve this problem, including commercial off-the-shelf solutions and in-house development efforts. Regardless of the tools used, one thing is clear: in order to satisfy this enterprise-wide, real-time demand for information, many companies are choosing Microsoft's .NET framework as their foundation.

What is .NET?

.NET provides a new set of tools and prefabricated components of unprecedented power that can be used to write a new type of software called "managed code." Over time, managed code will become recognized as clearly superior because:

Managed code is more robust. The .NET platform not only runs this new type of code but oversees its execution allowing software errors to be caught and halted before serious problems can occur. "Memory leaks," "memory corruption," and "blue screen of death" problems that freeze older Windows systems are prevented by .NET or curtailed before damage is done.

Side-by-side versions. .NET software minds its own business. It stays together in one place rather than spreading itself all around our systems. And in doing so, it avoids the old Component Object Model (COM) - based mechanisms that allowed software packages to collide with and harm one another. In fact, .NET allows two versions of a managed code application to run side by side on the same computer without any interaction or conflict. .NET thus eliminates the need to remove an old version of a software product just to install and try out a new version.

Better security. Under the old Windows approach to security, software is granted permission to do things on your computer based on the permissions granted to the user who runs the software. Hackers continue to find ways to sneak destructive software onto our systems and trick our computers into running that software under a privileged user account. In contrast, managed code follows a security model that is inherently different; the software is granted permissions on its own merits - not those of the user. Attributes of the software itself such as who wrote the software, where it came from, and where it is located are used to dictate and police what the software is allowed to do. The .NET platform enforces this new regime. Managed code can only run on the .NET platform so must submit to this new system of strict supervision and restriction.

Better connectivity. The new .NET software development tools and the standards-based nature of managed code make it easier to develop systems that employ state-of-the-art connectivity techniques such as XML Web Services. Building a high level of connectivity and the requisite robustness demands new components, tools, and a standards-oriented approach.

Faster software, faster development, easier deployment. .NET is a completely new software technology, created from scratch, both to leverage groundbreaking technologies such as XML and to eliminate the old inefficient internal layers that have built up over the last 20 years in Windows software. Managed code is lean, fast, and lightweight. These traits enable new innovation in application development and deployment. Aspects of managed code coupled with industry standards invite new deployment models including “no-touch” deployment and software that updates itself.

Lower cost of ownership. IT departments struggle daily to fix problems and work around restrictions rooted in the old Windows COM-based software technology. Companies expend vast amounts of corporate energy in their efforts to prevent new debilitating problems as they endeavor to apply system upgrades. Innovation and progress in leveraging new technology is slowed by severe caution and pessimism in rolling out new changes. The problems inherent in old Windows software translate to higher IT costs and lost opportunity costs. Each of the managed code advantages listed above promise to contribute to lower IT costs due to easier software development, easier and more trouble-free deployment, installation, maintenance, and security.

The advantages of managed code are clear and recognition of this fact is gradually growing in the public consciousness and corporate boardrooms. Just as there came a day when the MS-DOS-based software market was pronounced dead, so will come the day when software purchasers will only settle for managed code. .NET is a new and better kind of software and a new set of tools to build it. And yes, .NET makes it easier to build and deploy XML Web Services that seem to dominate any discussion related to enterprise application development.

Kinesix and .NET

In order to respond to the needs for distributing and displaying real-time data graphically in the control room and throughout the enterprise Kinesix has adopted the Microsoft .NET framework for the development of our next generation application development suite, KX EDGE (Enterprise Development & Graphical Environment). The goal of the KX EDGE solution is to merge the features and benefits of the traditional Sammi product with the advantages of the .NET framework.

First developed in 1991, the original Sammi product line, which is still widely used today, was created as a rapid application development environment for cross-platform, client-server applications with real-time user interface graphics based on X windows and Motif. This was later extended to include support for Microsoft NT, 2000 and XP. Included in this suite of tools was a Runtime component, a display editor and an API. These three components enabled the creation of sophisticated, real-time applications that completely separated the graphics from the application logic and coding. User interface displays could be created and changed independently of the back end application using the drag-and-drop display editor without writing any graphics code. And, the Runtime managed the complexities of a true distributed, multi-process architecture. The API included built-in networking between almost any UNIX and Microsoft based operating systems as well as providing cross-platform communication without the intricacies of low level network protocols.

KX EDGE is the next generation in the long line of Kinesix development products dating back to the original Sammi solutions. Although KX EDGE is a completely new product, written in C# and built entirely on the .NET platform, it still incorporates the primary features that made the original Sammi product so advantageous:

- Easy to use drag-and-drop display builder
- Pre-built Runtime framework for managing the application
- Simple API for interfacing back end applications

The .NET platform has enabled Kinesix to provide a development environment that satisfies the original goals of the Sammi product line but also incorporates the advantages of the .NET architecture. This includes not

only taking advantage of the inherent capabilities of .NET but also by being able to use the Visual Studio.NET to customize the default KX EDGE components, specifically:

- KX EDGE controls are standard .NET controls and can be used outside the KX EDGE Runtime environment.
- Third party .NET controls and ActiveX controls can be imported and used in the KX EDGE Runtime environment.
- Built-in in-memory database management system in Runtime allows heterogeneous data to be exposed to KX EDGE.
- Built-in web services connectivity allow users to add their own client application to consume the Runtime dynamic data (i.e. one case could be using MS Excel to manipulate data from the back-end).
- Extendable Runtime framework allows user to add customizable modules such as commands, Window Application Procedures (WAPs), and data adapters to their own data sources.

The KX EDGE Display Builder

The KX EDGE graphical display editor is called the Display Builder and is used to graphically create the user interface components of a system and to define the link between the user interface components and their controlling applications.

User Interface Components

The top-level user interface component created with the Display Builder is called a display. Each display is graphically displayed in a single window (i.e. WinForm). Displays contain user interface components (.NET controls) called Dynamic Display Objects (DDOs), and graphical primitives, called Static Display Objects (SDOs).

DDOs are graphical .NET controls that process commands and input events, display data, send and process commands, and interact with end users. They dynamically alter their graphical appearance based on events, commands, data values, and user interaction. Examples of DDOs are menus, meters, text fields, plots, and equations.

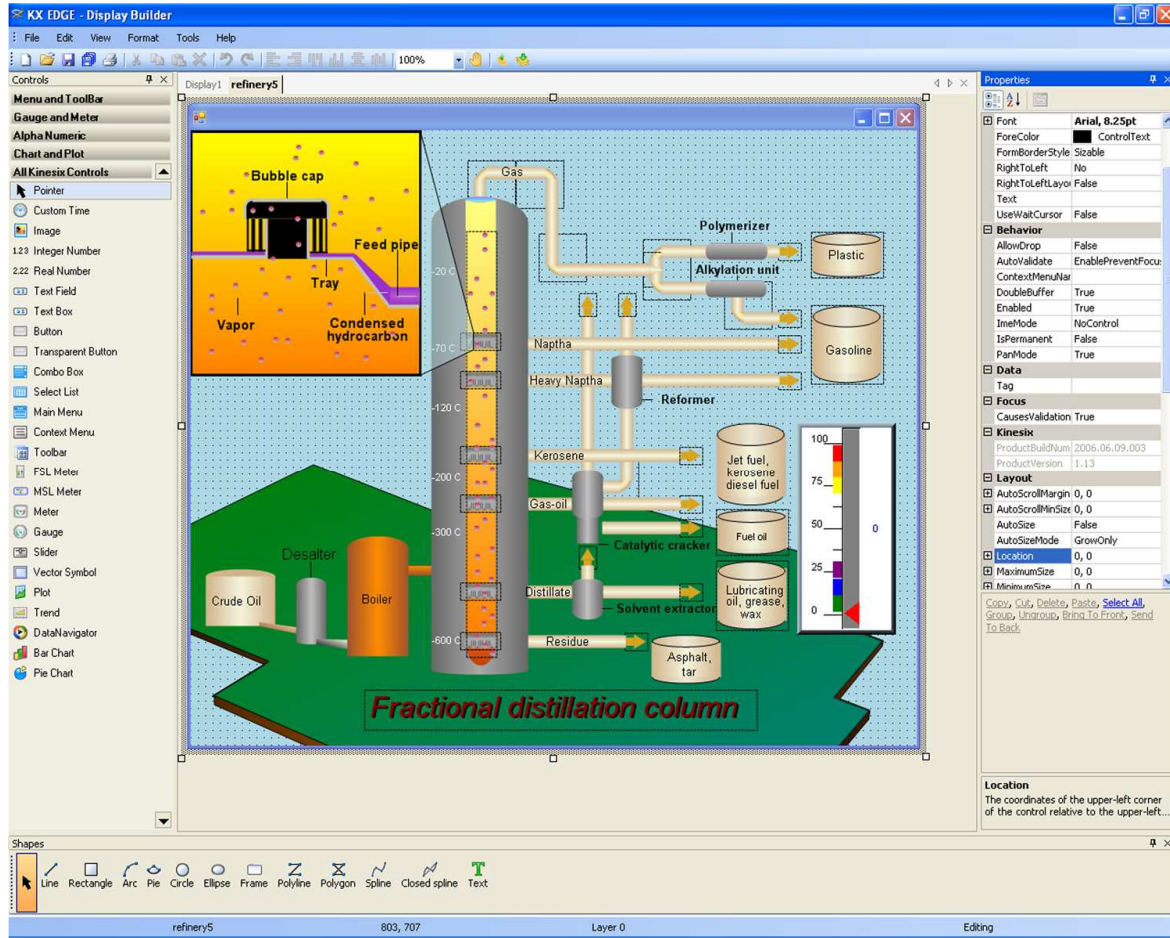
SDOs are primitive graphical objects, such as lines, rectangles, circles and bitmap images. These objects can be animated using a special DDO function, thus providing the ability to create a dynamic display component out of any graphics primitive or composite group. A display can also contain a background image (such as a map) that forms a backdrop for the format's objects.

Also, a KX EDGE display can also be based on a DXF file. The Display Builder includes an import utility that converts AutoCAD DXF files into native KX EDGE objects. Visio file formats are planned to be supported in a future release.

Additionally, each display or display can contain up to 256 layers of information. Entire layers, including any objects contained within, can be displayed or hidden in any combination at any time. A good example of utilizing layers is the presentation of a complex map or manufacturing facility. In either case, the user can zoom in and out while turning on or off various levels of information in order to clarify the view.

The Display Builder also supports composites, consisting of a group of DDOs and SDOs objects. A composite object can be created, saved and reused in symbol libraries.

Displays, and all the objects they contain, can be saved in both binary format and XML format. When a display is added in the Runtime, the Runtime environment will load the binary format, if the binary format is not found; it will load the XML format.



Connecting to Data Sources

Each Dynamic Display Object is linked to a remote data source. This data source may simply be a text file, an SQL database or a custom application. Regardless of the source of the data, the Display Builder provides a mechanism for defining the source and for defining any other relevant data that might be necessary for creating the data connection.

In order to create these data connections the Display Builder user simply specifies which data source is the target for the specific DDO. This is typically done by specifying a logical server name for the data source and by providing any additional data that specific source may need in order to provide the appropriate data. For example, the logical server may simply define an ODBC interface to an SQL database. The additional data required would be the sql command(s) needed to describe the action to be carried out.

KX EDGE also provides an API for interfacing each DDO with a custom application. Again, a logical server can be configured that describes the custom application and this server can be selected for the current DDO. Additionally, there are extra attributes and data that can be provided to help the server understand exactly what data should be provided. Because of the loose connection between the KX EDGE Runtime and the API

the DDO is usually not tied directly to a variable name within the back end code but is simply “described” to the application in such a way that the appropriate data or commands are processed.

Adding Custom .NET Controls

The KX EDGE Display Builder is itself a .NET application and can be extended. One good example of this is adding a third party or custom .NET and ActiveX control to the DDO palette. This is important since our customers will, at some point, need a DDO that is significantly different from the default set provided. Since KX EDGE is based on the .NET framework the process of adding a custom .NET control to the Display Builder is very straightforward. By dragging your custom control into the Display Builder a wizard is started that will walk the user through the steps of fully integrating the custom object with the KX EDGE framework. This includes not only integration with the editor but also with the KX EDGE Runtime application and the KX EDGE API interfaces, providing the same amount of control as a default KX EDGE DDO.

An import wizard in the Display Builder automates the process of importing a third party .NET or ActiveX controls. The wizard walks the user through the steps of fully integrating the custom object with the KX EDGE framework. This includes not only the integration with the Display Builder but also with the KX EDGE Runtime framework. At the end of the process, a new KX EDGE control, adapted from the custom control, is created and will be available to the KX EDGE Display Builder and Runtime.

Customizing KX EDGE Controls

The KX EDGE Display Builder provides a set of pre-built, commonly used controls. These controls are fully customizable using Kinesix provided class libraries. A control can be extended by using the inherent object oriented (OO) paradigm and then added back to KX EDGE with the import wizard.

Globalization

KX EDGE is fully internationalized, enabling the creation of language independent displays. These displays are localized based on the current locale and resource settings.

Alternative User Interface Deployment

The KX EDGE Display Builder, by default, saves a display into a binary format for creation of a WinForm application in the Runtime environment. This type of application is typically deployed on desktops. The Display Builder also allows displays to be saved in the new Microsoft user interface mark-up language XAML. The creation of WebForm based KX EDGE displays through XAML fully extends the KX EDGE Runtime data to the web.

The Display Builder has the exporting ability to allow third party applications to use KX EDGE displays; one example is the integration with ASP.NET pages (.aspx). Additionally, the user can add exporting capabilities of their own using the KX EDGE framework libraries for the Display Builder.

The KX EDGE Client/Runtime Environment

KX EDGE's Client/Runtime Environment loads display files, creates windows, and activates the objects within the window. The Runtime manages the interaction between the users and DDOs, and manages network communications between KX EDGE's client processes and networked applications and data servers.

Multi-Threaded Framework

The KX EDGE Runtime environment (RTE) consists of multiple cooperating threads communicating via an in-memory database based on .NET datasets. Each thread performs a specific function creating a highly efficient and agile application. For example, one thread receives data and routes it to the correct displays and DDOs, another thread causes display objects to blink, while another thread handles input events. This distribution of tasks permits each KX EDGE process to perform its job quickly and efficiently as well as handling user input events simultaneously with data updates. In a monolithic application, the single-process structure cannot utilize a processor's extra CPU cycles. Because a multi-threaded approach allows functionality to be broken down into a set of separate threads, each of which concentrates on a specialized task, CPU cycles can be used more efficiently. Since the system often doesn't need to wait for processing to finish, system performance can be substantially enhanced and response time substantially reduced.

The Runtime Environment provides the framework for the targeted distributed applications. It creates windows, loads display files, and activates the objects in the windows. It manages the interaction between KX EDGE users and DDOs, and manages network communications between the Runtime threads and application/data servers. Essentially, the Runtime removes the burden from the developer from having to create the framework for managing the graphics and interfacing with data over the network.

KX EDGE also utilizes an in-memory display cache which allows displays to be dynamically loaded and displayed at Runtime instead of compiled and linked into the application code. This permits hundreds or even thousands of displays to be developed and used without any increase in memory required. The cache also allows frequently used displays to be stored in memory and displayed without requiring any disk access resulting in very quick display drawing. Testing requirements are also minimized since displays are serialized binary files instead of code fragments. Only the new display needs to be tested as opposed to the entire application, in the case of re-compiling and linking of new code.

In-Memory Database

KX EDGE exploits .NET's Dataset technologies to manage heterogeneous data from different data sources. Data updates in each display are fully automated by the .NET runtime environment through data bindings. An added feature is that custom applications can access this same database through the KX EDGE data manger interfaces. Built-in web services simplify the integration of custom applications with the Runtime by removing the need to handle the underlying complex sets of data.

Custom Data Drivers

The KX EDGE Runtime provides an adapter for connecting to Sammi legacy applications (peers) through the traditional Sammi APIs. It also provides a built-in adapter for connecting to ODBC compliant databases such as SQL Server and Oracle. Custom adapters can be implemented using the KX EDGE framework libraries and can be added to the KX EDGE Runtime on the fly. Because of the design of the Runtime, any displays connecting to a specific driver will immediately receive data updates. One example of such a custom adapter would be an interface to pull data from a legacy application running on a mainframe.

Custom Commands

The KX EDGE Runtime provides a set of commands that can be invoked by the user, controls or applications. Although this set of commands is extensive it is also extensible providing the user with the ability to add custom commands specific to their needs. Custom commands are built using the KX EDGE framework class libraries, built it into an assembly, and then added into KX EDGE Runtime. This new, custom command is immediately available to be used like any other default KX EDGE command.

Custom Window Application Procedures (WAP)

Window Applications Procedures (WAPs) contain custom application logic to handle certain window events for a particular display. A WAP is called before the display is visible in the Runtime or after certain user interaction such as a button mouse click or keyboard press has occurred. For example, a custom WAP can be created that gives the end user the option of having data sent to the target data source after each field in a form is entered or only after a “submit” button is pressed. With the KX EDGE framework class libraries, users can create custom WAPs of their own and dynamically add them into the KX EDGE Runtime.

Failover and Server Redundancy

Application/data server redundancy is provided in the RTE to provide built-in support for automatic or manual fail-over of the primary server. This means that if the primary server fails, continuous interaction with the user interface is supported by the redundant server. This makes it possible for users to continue interaction with the application, reducing the risk of operational down-time.

Working Within the Runtime Environment

Interaction with the Runtime is typically accomplished in one of two ways: through the standard KX EDGE Command Window or through the application code. The command window is provided for the user's convenience and is simply a KX EDGE display created with the Display Builder and therefore can be modified and customized as necessary. Specific modifications might be to include or exclude certain functionality that your users may or may not need.

The Runtime is running in the background and is automatically managing the majority of tasks associated with the overall application. Because of this to call up a display you simply issue a command (add-window) either through the command window, a control or programmatically. Similarly, you can manage connectivity to remote data sources, including the configuration of failover servers. Also, accessible within the Runtime environment are modules for managing alarms, reports and schedules.

An additional feature of the Runtime is the ability to embed application logic. Rules can be created that control the user interface components and are stored in look-up tables or in memory. This avoids the need to embed certain application and user interface logic in the back end application. The largest advantage to this is that the logic can be changed without having to modify the back end code.

API Applications and Data Servers

The KX EDGE Runtime environment receives data and control logic from back end applications connected through data drivers or adapters. The underlying protocol for connecting to a data source via a KX EDGE data driver/adaptor can be RPCs, .NET remoting, XML SOAP or user specific. Regardless of the communication mechanism this back end program is the source for data and commands from the developer's overall application. Essentially, there are two parts to the KX EDGE API, the underlying networking protocol and the actual methods implemented in the back end application.

Network Protocols

The default set of communication protocols include .NET remoting, web services and the legacy Sammi ONC RPC interface. The legacy Sammi interface is included to provide seamless integration between an existing legacy Sammi application and the KX EDGE Runtime and user interface components. Additional adapters can be added for custom or third party communication needs such as CORBA, TIBCO, BEA, NDDS and other middleware systems.

The KX EDGE API methods themselves are based on an independent abstraction level above the underlying networking protocols. This enables the developer to concentrate on data manipulation and application needs regardless of the communication layer underneath. The KX EDGE API methods are based on an event driven programming model enabling the back end application to respond to events from the Runtime and to send data and commands as needed in an asynchronous manner.

APIs

Two different sets of API methods are provided with KX EDGE. One set is based on the legacy API calls found in previous Sammi releases and the other set provides a newer, .NET-based, object oriented framework for building .NET based back-end applications.

The legacy based API methods are primarily designed to handle events and to send data to remote Runtimes and not to include embedded graphics or networking code. This is a very high level API and the result is a very distinct separation between the graphics managed by the Runtime and the data managed by the back end API application. This set of methods is provided primarily to support legacy customers who want to interface with the KX EDGE Runtime or for those customers who simply need built-in, cross-platform networking, KX EDGE also provides this same API.

The new KX EDGE API methods enable the developer to achieve ultimate control over the user interface from the back end application. Each of the KX EDGE DDOs is actually a .NET control with all of its properties and attributes exposed through the KX EDGE API. Not only does this give the developer the ability to send data and respond to events in the same way as the legacy Sammi API, but also provides very fine control of the graphics at Runtime. Essentially, through the KX EDGE API, the developer can write an application in any .NET language and interface with the KX EDGE Runtime as well as access all attributes of the displays and the individual DDOs.

Summary

KX EDGE is the next generation of application development environments provided by Kinesix Corporation. It incorporates the expertise garnered by the Kinesix development staff over the past 14 years in delivering software tools for deploying real-time, distributed applications with the inherent advantages of the Microsoft .NET architecture.

As the .NET architecture evolves so will the KX EDGE solution. Kinesix is already at work implementing the future technologies included in the next Microsoft Windows release code-named Longhorn. This includes taking advantage of:

- XAML as the XML language for describing KX EDGE displays
- Avalon as the new presentation subsystem
- Indigo as the new communications infrastructure built around web services

Connecting the control room or factory floor to all other aspects of the enterprise is no longer optional. Deploying applications based on KX EDGE solves this problem and enables today's businesses to compete by providing a framework for building applications to serve the needs of the entire enterprise.

Kinesix plans to have a KX EDGE early release in the 4th quarter of 2005 with a general availability upcoming in 1st quarter 2006.

Kinesix Software of Houston, Texas, develops and markets software tools for creating client/server and Web-enabled applications that graphically, dynamically and interactively display real-time data. Kinesix' products have been field-proven in the most demanding environments and applications where the graphical presentation and management of multiple sources of data are critical. Kinesix is a full service organization providing product development, marketing, sales, technical support, consulting and training.

For additional information contact:

Kinesix Software
Russ Jamerson
(713) 953-8344
russ.jamerson@kinesix.com

Copyright © 2005 Kinesix Software All rights reserved. Microsoft and .NET are registered trademarks of Microsoft Corporation in the U.S. and other countries. All other trademarks are the property of their respective owners.

The information contained in this paper pertaining to Kinesix products is proprietary and confidential. Additionally, the information is based on a pre-release version of KX EDGE and features and functionality are subject to change.