

## Itinerary

Monday	AM	1 – Installing StrataFrame 2 – Framework overview 3 – Data access layer
	PM	4 – Tips & tricks #1 5 – Database interaction 6 – Business layer
Tuesday	AM	7 – WinForms development 8 – Role-based security
	PM	9 – Tips & tricks #2 10 – Presentation layer 11 – Database Deployment Toolkit
Wednesday	AM	12 – Enterprise Server 13 – Web development
	PM	14 – Tips & tricks #3 15 – Messaging & localization 16 – Business layer advanced topics
Thursday	AM	17 – Data access advanced topics 18 – Presentation layer advanced topics
	PM	19 – Tips & tricks #4 18 – Presentation layer advanced topics (cont'd)
Friday	AM	20 – Tools classes Q & A
	PM	21 – Tips & tricks #5

## **1 – Installing StrataFrame**

### **1.1 – Getting Installed**

Once StrataFrame has been downloaded from the website it is ready for installation. The following items need to be downloaded before starting: Custom License File and Application Framework Setup. Additionally, SQL Server (2000 or greater) and Visual Studio 2005 must be installed prior to running the StrataFrame setup.

### **1.2 – StrataFrame Activation**

Using the License Activation window, StrataFrame can be activated and deactivated as many times as necessary. There are several different ways to activation StrataFrame. The first is through the standard Activation Client. However, then a proxy prevents the client from accessing the activation server, then a manual activation can be requested. Also, a manual activation will be able to be achieved online without any contact from the StrataFrame team in the near future. Finally, deactivation can be done through the Activation Client or through the website in the My Account area. When using the website deactivation, only a single copy can be de-activated every 10 days. This is not the recommended approach, but rather the use of the Activation Client is the preferred method.

### **1.3 – StrataFrame Databases and IDE Connection**

StrataFrame has a design-time database that is deployed to SQL Server during the install. If the “Install StrataFrame Sample Database” option was selected, another database named StrataFrameSample will exist as well.

### **1.4 – Installing Source Code and Building in Debug Mode**

Since StrataFrame comes with source code, take advantage of it! It will compile immediately after install and is configured to automatically add any new compilations into the GAC. This will automatically show the line of code in error within a debug session. DO NOT ADD STRATAFRAME SOURCE PROJECTS TO EXISTING SOLUTIONS!

### **1.5 – StrataFrame Menu within Visual Studio**

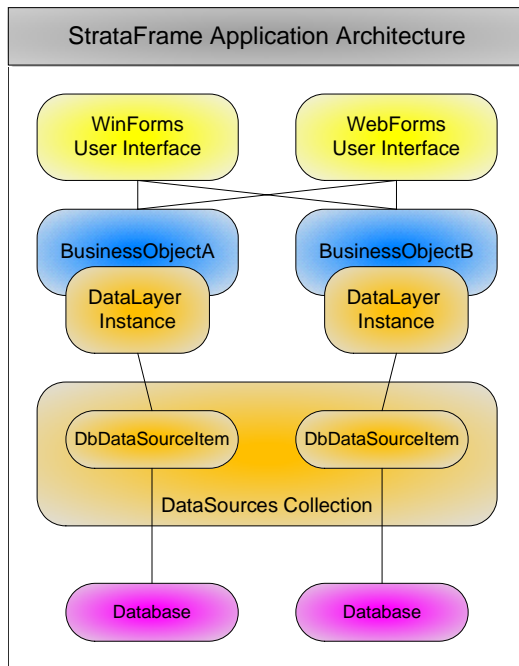
StrataFrame seamlessly integrates into Visual Studio in many different ways. One of the most obvious is the StrataFrame menu within the Visual Studio IDE.

### **1.6 – Working In Multiple Locations**

At times development environments may have more than one office that does not share a VPN connection or a developer may use separate StrataFrame database installations to separate companies or projects.

## 2 – Framework overview

### 2.1 – StrataFrame application architecture



A StrataFrame application consists of 1) one or more databases or data stores, 2) the DbDataSourceItems used to communicate with the data stores, 3) business objects, 4) and a user interface.

### 2.2 – Major classes of a StrataFrame application

#### 2.2.1 – DbDataSourceItem

The DbDataSourceItem classes (concrete implementations of the DbDataSourceItem) provide all of the access to the database for a StrataFrame application. All DbDataSourceItem instances are stored within the MicroFour.StrataFrame.Data.DataLayer.DataSources collection. Each DbDataSourceItem is identified by a unique DataSourceKey. This class provides the “funnel” to centralize data access for the application.

The DbDataSourceItem classes are “stupid;” they only know how to execute one command at a time.

#### 2.2.2 – DataLayer

The DataLayer class is used to provide a DbDataSourceItem independent interface between the BusinessLayer and the DbDataSourceItem objects. Any data formatting and/or data pre-processing is handled within the DataLayer. The DataLayer organizes all of the records that must be saved within a business object and simplifies the job of the DbDataSourceItem classes.

### 2.2.3 – BusinessLayer

The BusinessLayer is the base class for all business objects within a StrataFrame application. The BusinessLayer class houses the base functionality for the business objects to retrieve, present, modify, validate, and persist data.

### 2.2.4 – StandardForm/BasePage

The StandardForm and BasePage classes provide the base functionality for Windows forms and web pages respectively. They provide the base functionality for business objects to communicate with the user interface.

## 2.3 – Namespaces – What does the StrataFrame class library contain?

### 2.3.1 – MicroFour.StrataFrame.AddIns

Contains all of the pieces to the Business Object Mapper, and the functionality needed for StrataFrame to incorporate AddIns into Visual Studio.

### 2.3.2 – MicroFour.StrataFrame.Application

Contains the pieces of the application organization of a StrataFrame application. Examples: StrataFrameApplication and event argument classes for application-level events.

### 2.3.3 – MicroFour.StrataFrame.Business

Contains the classes relating to the business logic layer of a StrataFrame application. Examples: BusinessLayer, BusinessLayerLinkManager, BusinessBindingSource, property descriptor base classes, etc.

### 2.3.4 – MicroFour.StrataFrame.Data

Contains the classes relating to the data access layer of a StrataFrame application. Examples: DataLayer, DbDataSourceItem, DbDataSourceItem concrete implementations, etc.

### 2.3.5 – MicroFour.StrataFrame.DBEngine

Contains the classes relating to the deployment of meta-data to SQL Server. Examples: DatabaseMigrator and deployment dialogs

### 2.3.6 – MicroFour.StrataFrame.Extensibility

Contains all of the type editors, control designers, type converters and related classes used within Visual Studio for the design-time functionality provided by StrataFrame.

### 2.3.7 – MicroFour.StrataFrame.IO.Compression

Contains the classes used to create or access a PackageFile (.pkg file) programmatically. Example: PackageFile

### **2.3.8 – MicroFour.StrataFrame.Messaging**

Contains the classes used to provide the end-user of a StrataFrame application with messaging information. Example: MessageForm, WaitWindow and InfoBox

### **2.3.9 – MicroFour.StrataFrame.Security**

Contains all of the classes used within the StrataFrame role-based security. Examples: LoggedOnUser, Logon, SessionLock, PermissionInfo, and related business objects.

### **2.3.10 – MicroFour.StrataFrame.Tools**

Contains sealed/static, helper classes that contain methods that can be used simplify repeated tasks.

### **2.3.11 – MicroFour.StrataFrame.UI**

Contains all of the classes and related namespaces for both WinForms and WebForms user interface components. Examples: Localization

### **2.3.12 – MicroFour.StrataFrame.UI.Web**

Contains all of the classes relating to the creation of WebForms user interfaces for a StrataFrame ASP.NET application. Examples: WebControls, BasePage, ApplicationBasePage

### **2.3.13 – MicroFour.StrataFrame.UI.Windows.Forms**

Contains all of the classes relating to the creation of WinForms user interfaces for a StrataFrame application. Examples: all WinForms controls and StandardForm

### **2.3.14 – MicroFour.StrataFrame.UI.Windows.Forms.DevEx**

Contains all of the components of the StrataFrame DevExpress wrapper.

### **2.3.15 – MicroFour.StrataFrame.UI.Windows.Forms.Infragistics**

Contains all of the components of the StrataFrame Infragistics wrapper.

### **2.3.16 – MicroFour.StrataFrame.Win32**

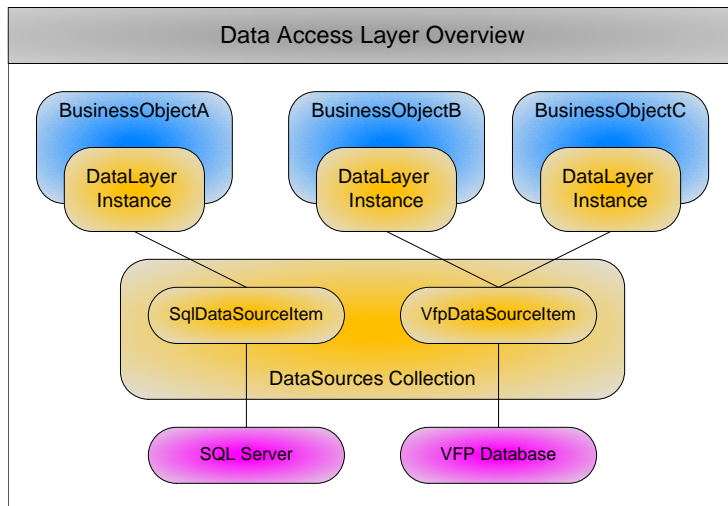
Contains shared/sealed classes that provide Windows API wrappers for use within StrataFrame applications. Examples: RegistryRepository, User32, GDI32, Kernel32, etc.

### **2.3.17 – MicroFour.StrataFrame.Xml**

Contains shared/sealed classes that provide XML processing for StrataFrame applications. Examples: XmlBasics

## 3 – Data access layer

### 3.1 – Overview



The data access layer of any StrataFrame application consists of the DataLayer objects within the business objects of the application and the DbDataSourceItem objects stored within the DataSourcesCollection.

The “connection strings” for the application is stored within the DbDataSourceItem classes; each DbDataSourceItem stores the connection string used to access its associated database.

An application can have an unlimited number of DbDataSourceItem objects, each communicating with a separate database. A single DbDataSourceItem object cannot access more than one database unless its connection string is modified.

### 3.2 – DbDataSourceItem & the DataSources collection

#### 3.2.1 – The DbDataSourceItem class

A DbDataSourceItem implementation provides simple, provider-specific data access to allow commands to be executed against the database using ExecuteNonQuery, ExecuteReader, and ExecuteScalar and DbDataAdapter.Fill(). The DbDataSourceItem is fully abstract, allowing the DbDataSourceItem implementations to be interchanged.

DbDataSourceItems are referred to a “data sources” because they are stored within the DataSources collection. When you hear mention of a “data source,” the reference is most likely to a DbDataSourceItem.

#### 3.2.2 – The DataSourceKey

Each DbDataSourceItem is identified by a DataSourceKey, a unique string that identifies the DbDataSourceItem object within the application. Anywhere that “DataSourceKey” is contained within a property name, the property is referring to the key that will provide the object with the correct data source. For example, the SecurityBasics.SecurityDataSourceKey property tells the security module

which data source to use to communicate with the security database tables; also, each business object has a `DataSourceKey` property that tells the business object which data source to communicate with. The default for all `DataSourceKey` properties is an empty string ("").

### 3.2.3 – The DataSources Collection

The `DataSources` collection is a shared/static collection that contains all of the data sources (`DbDataSourceItem` objects) used by the application. The `DataSources` collection can be found through the `MicroFour.StrataFrame.Data.DataLayer.DataSources` property or the `MicroFour.StrataFrame.Data.DataBasics.DataSources` property. The `DataSources` collection has an indexer that allows you to retrieve a reference to a data source through the `DataSourceKey` or the index.

## 3.3 – DbDataSourceItem implementations

### 3.3.1 – SqlDataSourceItem

The `MicroFour.StrataFrame.Data.SqlDataSourceItem` class communicates with all versions of Microsoft SQL Server.

### 3.3.2 – VfpDataSourceItem

The `MicroFour.StrataFrame.Data.VfpDataSourceItem` class communicates to all versions of Visual FoxPro databases that can be accessed through an OLE DB provider.

### 3.3.3 – OracleDataSourceItem

The `MicroFour.StrataFrame.Data.OracleDataSourceItem` class communicates to all versions of Oracle database that can be accessed through the installed Oracle Client software. The `OracleDataSourceItem` class utilizes the classes within the `System.Data.OracleClient` namespace and requires the Oracle Client to be installed on the machine running the StrataFrame application.

### 3.3.4 – AccessDataSourceItem

The `MicroFour.StrataFrame.Data.AccessDataSourceItem` class communicates to all versions of Microsoft Access databases that can be accessed through the installed Microsoft Jet or OLE DB provider.

## 3.4 – Configuring DataSources

Configuring the `DataSources` for an application requires adding new instances of the correct concrete `DbDataSourceItem` class to the `DataSources` collection. You must supply a unique `DataSourceKey` for each `DbDataSourceItem` that is added to the collection, and the connection string must be supplied when the `DbDataSourceItem` is created.

Adding the `DbDataSourceItem` objects can be done through the StrataFrame Connection String Management (`ConnectionManager` class) or by manually adding the instances if the connection string for the application is hard-coded or pulled from a configuration file.

Configuring the `DataSources` can be done anywhere within the application, but the `SetDataSources()` method is provided to supply the most logical place within the application to set the data sources.

### 3.4.1 – Manually adding new DbDataSourceItem objects

#### Adding a new SqlDataSourceItem [C#]

```
private static void SetDataSources()
{
    DataLayer.DataSources.Add(new SqlDataSourceItem("",
        @"Server=MyServer;User Id=sa;Password=password;Database=MyDatabase;"));
}
```

#### Adding a new SqlDataSourceItem [Visual Basic]

```
Private Shared Sub SetDataSources()
    DataLayer.DataSources.Add(New SqlDataSourceItem("", _
        "Server=MyServer;User Id=sa;Password=password;Database=MyDatabase;"))
End Sub
```

### 3.4.2 – Setting the data sources through the ConnectionStringManager

For the times when storing the connection string in a configuration file or hard-coding the connection string are not feasible, StrataFrame provides the ConnectionManager class to set, store, and retrieve the application connection string.

You use the ConnectionManager by setting the ApplicationKey for the application. This is a unique key that is used to identify the groups of connection strings used by an application. The key is generally unique to an application, but multiple applications can use the same ApplicationKey if they are going to use the same connection string.

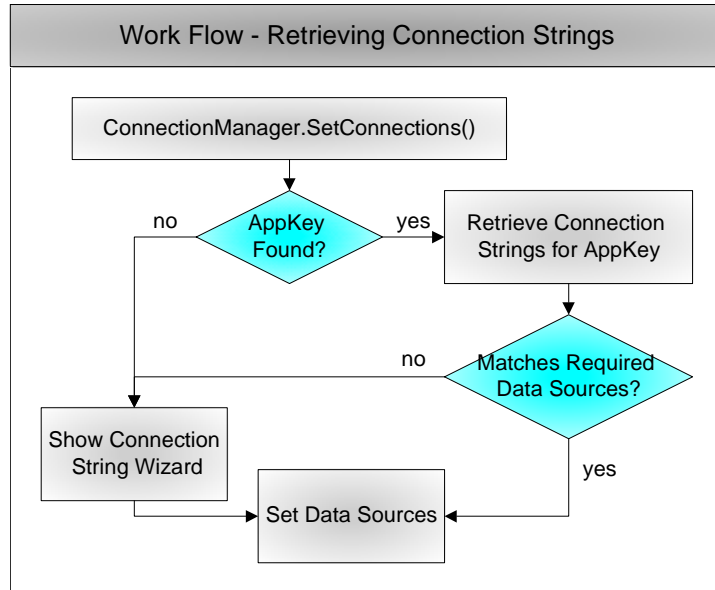
Once the application settings are set, you can add the required data source items. A single required data source item corresponds to a DbDataSourceItem that is required by the application. The required data source item contains information that the ConnectionManager and ConnectionStringWizard will use to gather the connection string from the end-user.

When the ConnectionManager.SetConnections() method is called the ConnectionManager will either retrieve the connection string from storage or gather it from the end-user. When the connection string(s) is acquired, the ConnectionManager will create the DbDataSourceItem object, set its connection string and add it to the DataSources collection.

All Connection information is stored within the C:\Documents and Settings\All Users\Application Data\MicroFour\ConnectionData\ folder in the AppKeys.dat file and the Connections.dat file.



### 3.4.3 – Work Flow – Retrieving the connection string



### 3.4.4 – Shared Settings File

The ConnectionManager supports the use of a Shared Settings File. The Shared Settings File is a configuration file that can be stored in a central location to allow the connection string to be changed on all computers that use the shared settings file at the same time. The use of the Shared Settings File is covered in Section 10.

### 3.5 – Purpose of the DataLayer class

The DataLayer class is used to provide a buffer between the BusinessLayer and the DbDataSourceItem classes. The DataLayer is provider independent and communicates directly with the specified DbDataSourceItem. It works closely with its containing BusinessLayer object to format commands and create QueryInformation objects that act as instructions for the DbDataSourceItem. The DataLayer also contains the necessary logic to update the entire data type within a business object.

## **4 – Tips & tricks #1**

### **4.1 – Organizing an Application**

#### **4.1.1 – Overview**

A common question that comes up, especially once developers get entrenched in the process of developing their applications, is what is the best way to organize an application or solution as it relates to the assemblies, project, and solutions? Many times developers feel the need to have all of the answers the day that they begin, but this will only overwhelm the development cycle even more. As an application grows then so will the project. At some point the project may begin to feel somewhat sluggish and then all of a sudden a realization occurs that the project may need to be split up.

Don't misunderstand, good design up front will always reduce confusion and problems down the line. But to aide in the "getting started" process are a few good pointers that can help reduce the stress of making all of the "perfect" decisions day one.

#### **4.1.2 – The solution and projects**

When starting the development cycle on a project, there are always a few constants that can be expected and that should always be created for your application. First, any application should always have base, business, and UI assemblies which will provide a foundation for the entire application and will more than likely be referenced by most downstream assemblies within the application. Next, there will always be an entry point or client access assembly. This is generally the assembly that is the executable associated with the application. Last, are any modularized or utility assemblies. It is generally good practice to separate modules. For example, accounting classes and dialogs would reside in one assembly while point of sale may reside in another. This is not an exact science but at least provides a good rule of thumb.

##### **4.1.2.1 – Terminology**

For the purposes of this section when we discuss an assembly we are generally referring to a .NET project. One .NET project results in a single assembly. This assembly may be an executable or a class library, but in either case they will be addressed as assemblies within this section. The assembly is a project in a compiled or completed state whereas a project is the design-time collection of source files that will be compiled to create a single assembly. A solution, on the other hand, is just a collection of projects that can be managed, maintained, and referenced within a single designer session.

One other term used throughout this section is "application project." This is not a .NET project or solution, but rather a reference to the overall and entire application that is being developed. This includes any and all projects, solutions, and assemblies that go into creating the end result.

##### **4.1.2.2 – The Base assembly**

The base assembly in many cases is an afterthought but should really be one of the first assemblies created for any application project. This assembly will contain any classes or tools that are common to an application at large. Additionally, this assembly should reference no other assemblies for the application. Since this is the base, all other assemblies within the application should be able to safely reference this assembly without the concern of creating any type of circular reference.

#### 4.1.2.3 – The Business assembly

Aside from the base, the business assembly is generally one of the most commonly referenced assemblies for any application. The business assembly should reference a minimum number of application assemblies and it is generally good practice that the business assembly only reference the base assembly as it relates to the application assemblies. This project houses all of the business objects that are associated with an application. A common question is, “Can there be more than one business assembly?” The answer is, yes. However, the fewer business projects that exist the easier maintenance and deployment will be.

#### 4.1.2.4 – The UI assembly

The UI assembly is often overlooked but plays an important role within the application. This assembly should contain all controls and components that are specific to the application. This is also the assembly where any inherited StrataFrame controls should reside. It is rare to write an application without creating some custom controls or inheriting controls and adding additional functionality. That is the purpose of this assembly and every application project should have one.

#### 4.1.2.5 – The Executable assembly

The executable assembly is generally where most developers start. Though this is a critical component to the application, it is important that this assembly not be overloaded with so many classes that it becomes top-heavy and difficult to follow and maintain. There are several schools of thought when dealing with the executable assembly. The first is that only startup and dialogs specific to the executable are stored within this assembly and all other classes fall into other categorized assemblies specific to application modules. The second is similar but has allowance for more dialogs that operate as part of the application. For example, a point of sale form may reside in the executable since there is only one form and it will never be referenced by any other assembly. Either of these approaches is acceptable because they both have one thing in common, all startup and environment initialization takes place within this assembly. The data sources are specified and security authenticated within this assembly.

#### 4.1.2.6 – The Modular Assemblies

The modular assemblies contain all of the classes and dialogs that pertain to more specific aspects of an application. For example, all accounting classes and dialogs may have their own modular assembly. This creates a clean delineation between each of the modules of an application and makes maintenance, deployment, and support much easier. Additionally, this can increase overall performance of an application by separating the assemblies and preventing any one assembly from being too large.

### 4.1.3 – Namespaces

Namespaces provide an excellent way to further organize an application. Each time an assembly is created another piece of the overall application puzzle is added. However, some assemblies may contain many classes and cover a broad spectrum of functionality, such as a base assembly. The better organized an application is the easier it is to maintain, deploy, support, and develop. There are some good guidelines that can be following when setting up namespaces.

First, all assemblies should share a common root namespace. A common format is the company name followed by the product name. In this example let's assume that MicroFour is creating an accounting package named AccountIt. In this example the root namespace would look like this:

MicroFour.AccountIt

To set the root namespace on an assembly, right-click the project and go to the properties. Once open, select the Application tab and set the root namespace. Keep in mind that this is just the root for the assembly, so every class created within this project will have at least a namespace that begins with the root namespace provided here. To add to a namespace within a class the Namespace definition can be used within the class definition.

The Root Namespace within the project properties only applies to a Visual Basic project. Within C#, all class declarations must be nested within the full namespace. The Root Namespace within C# is only used to determine the namespace when creating a new code file from a template.

#### Namespace example [C#]

```
namespace MicroFour.AccountIt.UI
{
    public class MyUIClass
    {

    }
}
```

#### Namespace example [Visual Basic]

```
Namespace UI
    Public Class MyUIClass

    End Class
End Namespace
```

The above code sample would result in a class with a namespace like MicroFour.AccountIt.UI.MyUIClass. The following table provides some good standards as that relates to namespaces and assemblies.

Assembly Type	Recommended Namespace
Base	Company.Product.Base
UI (User Interface)	Company.Product.UI
Business	Company.Product.Business
Modules	Company.Product.ModuleName
Common Classes (may reside in Base assembly)	Company.Product.Common

#### 4.1.4 – Working with Multiple Solutions

A common misconception is that an entire application project can only have a single solution. Another misconception is that a .NET project can only belong to a single .NET solution. Neither of these are true. Many times it makes perfect sense to create another solution when one starts to get too large. An example of when this may be done is when working on different modules. Pulling from the examples that were used above, it may make sense to create a solution just for the accounting module. When this is done, it is a good idea to add any common projects, such as the base project, to the new solution as well so it can be referenced as a project and maintained from the new solution without requiring the original solution from be opened and maintained before it becomes effective in the new modular solution.

When developing in VB.NET the compiler runs at all times. As a solution gets larger, the design-time environment can become almost unbearable to work in. When this happens it is a sure sign that a new solution should be created and the work environment split up between multiple solutions. C# does not have the same design-time problem since the compiler is not constantly running.

### 4.2 – Programming in a Team environment

#### 4.2.1 – Overview

One of the most common things that we are asked during and after training classes is how to develop in a team environment with StrataFrame. So we have since incorporated this into our training guidelines and have dedicated a section solely to setting up this type of environment. Now keep in mind that this does not only to multi-developer teams, but anyone who may be developing from more than a single location.

So first it is important to define the problem so that there can be better understanding as we discuss each section. StrataFrame requires that a SQL Server connection be present which contains a single database used to house meta-data, localization information, solution settings, BO Mapper assignments, etc. So when working in a distributed or multi-developer environment this database needs to be accessible. It is best that common projects all reference a common StrataFrame database.

Let's assume that we are working in a 5 developer environment and we are all working on the same project. There are several issues that we need to address. First, when accessing the project, the source files need to reside on our local computer to properly interact with the .NET security policies, but moreover, to prevent sharing violations and corruption of the source files. Next, we all need to have access to the StrataFrame database. Last, let's assume that one of these developers is working from a remote location (i.e. not on the internal network). So what components are needed in order to make this an ideal situation?

There are really only two necessary components in order to achieve this type of functionality: a Source Control implementation and a VPN. However, there are a few "gotchas" that must be addressed to avoid any type of network contention and slowness.

#### 4.2.2 – Team Environment Software and Selection

The first step is determining which type of team environment or source control software that will be used. There are a number of different options in this area, including some 3<sup>rd</sup> party vendors. These include SourceGear Vault/Dragnet, Dynamsoft SourceAnywhere/Issue Tracking Anywhere, ionForge,

Microsoft Team Foundation, and Visual SourceSafe. Out of these there is one that stands head and shoulders above the rest, Microsoft Team Foundation Server. But this is not the only solution that may suit your needs. Below is a description for each of the mentioned packages.

#### 4.2.2.1 – Visual SourceSafe (Rating: D)

**Overview:** Visual SourceSafe is compatible with Visual Studio.NET and provides only a source control solution. However, there are a number of pitfalls of Visual SourceSafe. VSS is a file handle based source control solution which may create hundreds of handles when pulling a single file from source control. This is the slowest and most cumbersome of all source control solutions.

**Pros:** VSS is free and will work as a source control solution when the development environment will be run on an internal network or local computer.

**Cons:** Since VSS is file handle based it is not a viable solution for a remote connection and really cannot be used from a remote location. Also, VSS is an older technology and is prone to causing issues within the newer .NET development environment such as missing a check-in or having a corrupt source file. Though infrequent, when this happens it can cost hours or more to get back up to speed. The larger a project becomes the more issues that arise and the slower performance becomes. Last, there is no bug or task tracking system.

#### 4.2.2.2 – SourceGear Vault/Dragnet (Rating: B-)

**Overview:** SourceGear has two products that can be used as a team development solution. Vault is the source control product and Dragnet is the team collaboration support (i.e. bugs, tasks, etc.).

**Pros:** SourceGear is a SQL Server based source control solution which provides increased security. This will allow developers to effectively connect and work remotely. Provides the most functionality by any 3<sup>rd</sup> party vendor as it relates to team development software. Good alternative to Team Foundation and cost effective.

**Cons:** Competes directly with Team Foundation and falls short. The .NET interface and dialogs are not as clean as they could be, especially when compared to Team Foundation.

#### 4.2.2.3 – Dynamsoft SourceAnywhere / Issue Tracking Anywhere (Rating: C+)

**Overview:** Dynamsoft has two products like SourceGear which provide source control and issue tracking. SourceAnywhere is the source control software and Issue Tracking Anywhere is the developer collaboration tools.

**Pros:** The interface is clean and easy to use. SQL Server based source control solution which provides increased security. Remote connection is relatively fast and has a decent compression rate. Provides far more functionality than SourceSafe and is a good alternative to VSS and has some of the Team Foundation features such as branches. Simple, easy to use, and gets the job done.

**Cons:** .NET integration is limited. Though there is much more functionality than SourceSafe it does not have as quite much depth as SourceGear, primarily at the interface to .NET level which ultimately makes development more cumbersome.

#### 4.2.2.4 – ionForge (Rating: C-)

**Overview:** ionForge is a step up from Visual SourceSafe and provides only a source control product. There is no team collaboration product that goes along with ionForge so another 3<sup>rd</sup> party product would have to be used for issue tracking.

**Pros:** SQL Server based source control solution and supports remote connections. Inexpensive solution and is better than SourceSafe.

**Cons:** Limited functionality and is more like SourceSafe on steroids. For the price would be better off going with a more commercial solution such as Dynamsoft or SourceGear for not much more. Clumsy interface and is somewhat cumbersome to navigate. Though some .NET integration exists it is more limited than any of the other 3<sup>rd</sup> party source control tools.

#### 4.2.2.5 – Team Foundation and VS 2005 Team Editions (Rating: A)

**Overview:** Team Foundation provides a tool for every aspect of software development including source control, load and order testing, architect tools (i.e. database architect), team collaboration, project management, and more. Though not all of these tools are required for most development environments there is definitely a replete selection to choose from.

**Pros:** SQL Server based which provides the additional security and speed for remote connections. Extremely fast remote downloads due to enhanced compression. Work remotely nearly as fast as when in the office! Source control and collaboration tools are both integrated seamlessly into Visual Studio. Workgroup edition available through elevated MSDN subscription. Overall the absolute best choice when cost is not part of the formula.

**Cons:** Expensive when needed more than source control and Workgroup edition (5 users). Team Suite alone over \$10,000 with an MSDN subscription. Setting up the server can be somewhat tedious and frustrating, but once setup it works great.

### 4.2.3 – VPN (Virtual Private Network)

One requirement that is a must when remote access is required is a VPN connection back to the source location. This can be either a software or hardware VPN tunnel depending upon the circumstances. For example, a Linksys VPN router can be setup at a home location which has a hardware tunnel back to the home office. However, when on the road a software tunnel can be established and used just as effectively as long as the source or host location has an adequate router and bandwidth.

#### 4.2.3.1 – Home office router/firewall

The best router and firewall solution would be a Cisco PIX, however, for smaller offices with 5 or less developers, the Linksys RVS4000 is a more than adequate router which supports up to 5 hardware tunnels and 5 software tunnels and is very cost effective at around \$130. Since Cisco owns Linksys, the core of the router is very strong and even comes with the VPN software.

#### 4.2.3.2 – Home office Internet connection

A stable internet connection is required. The bandwidth required is influenced by the number of remote developers. For example, if a single remote developer will exist then an up speed of 512 Kbps

would probably be sufficient. However, the more up speed, regardless of the number of developers, the faster check-ins, check-outs, and source retrieval will be. It is recommended that an up speed of 1 Mbps or more be available for optimum performance.



## 5 – Database interaction

The developer never communicates directly with the DataLayer internal to a business object, and generally, the developer will never need to communicate directly with a DbDataSourceItem object (although you can if necessary). Instead, façade (wrapper) methods have been created on the BusinessLayer class to provide the necessary database interaction.

Database interaction is controlled through the methods on the BusinessLayer class that retrieve data from the database and the methods and properties that control the persistence of changes back to the database.

### 5.1 – Fill methods

The BusinessLayer class contains several methods that allow you to retrieve records from the database and populate the internal table of the business object with the retrieved results.

#### 5.1.1 – FillDataTable()

The FillDataTable method is the most commonly used Fill method on the BusinessLayer class. It allows you to populate the business object with the results retrieved from a string query or a parameterized command. The FillDataTable() method is public to provide the necessary access within the framework itself; however, it is best used within a public method on the business object class to prevent the embedding of SQL anywhere the business object is used.

##### **FillDataTable() [C#]**

```
public void FillAll()
{
    this.FillDataTable("SELECT * FROM MyTable ORDER BY MyField");
}

public void FillByMyValue(int MyValue)
{
    //-- Establish locals
    SqlCommand loCommand = new SqlCommand();

    //-- Create the command
    loCommand.CommandText = "SELECT * FROM MyTable WHERE MyField = @MyField";

    //-- Add the parameter
    loCommand.Parameters.Add("@MyField", SqlDbType.Int);
    loCommand.Parameters["@MyField"].Value = MyValue;

    //-- Execute the command
    this.FillDataTable(loCommand);
}
```

**FillDataTable() [Visual Basic]**

```

Public Sub FillAll()
    Me.FillDataTable("SELECT * FROM MyTable ORDER BY MyField")
End Sub

Public Sub FillByMyValue(ByVal MyValue As Integer)
    '-- Establish locals
    Dim loCommand As New SqlCommand()

    '-- Create the command
    loCommand.CommandText = "SELECT * FROM MyTable WHERE MyField = @MyField"

    '-- Add the parameter
    loCommand.Parameters.Add("@MyField", SqlDbType.Int)
    loCommand.Parameters("@MyField").Value = MyValue

    '-- Execute the command
    Me.FillDataTable(loCommand)
End Sub

```

**5.1.2 – Other common Fill methods**

**FillByPrimaryKey()** – This method allows you to retrieve a single record or a group of records by specifying the primary key(s) of the record(s) to retrieve.

**FillByParentPrimaryKey()** – This method allows you to retrieve all of the child records by supplying the primary key of the parent record. This method requires the ParentRelationship property to be set.

**FillByParent()** – This method allows you to retrieve all of the child records for each parent contained within the supplied parent business object. If an explicit parent business object is not supplied, then the ParentBusinessObject of the business object is assumed. This method requires the ParentRelationship property to be set.

**FillByStoredProcedure()** – This method allows you to populate the business object with the results returned from a stored procedure.

**5.2 – Get methods**

For every Fill method that exists on a business object, there also exists a corresponding Get method. The Get methods retrieve the same results as the Fill methods, but rather than populating the business object with the returned results, the results are returned by the method, and the CurrentDataTable of the business object is not modified.

**GetDataTable() [C#]**

```

public DataTable GetAll()
{
    return this.GetDataTable("SELECT * FROM MyTable ORDER BY MyField");
}

public DataTable GetByMyValue(int MyValue)
{
    //-- Establish locals
    SqlCommand loCommand = new SqlCommand();

    //-- Create the command
    loCommand.CommandText = "SELECT * FROM MyTable WHERE MyField = @MyField";

    //-- Add the parameter
    loCommand.Parameters.Add("@MyField", SqlDbType.Int);
    loCommand.Parameters["@MyField"].Value = MyValue;

    //-- Execute the command
    return this.GetDataTable(loCommand);
}

```

**GetDataTable() [Visual Basic]**

```

Public Function GetAll() As DataTable
    Return Me.GetDataTable("SELECT * FROM MyTable ORDER BY MyField")
End Function

Public Function GetByMyValue(ByVal MyValue As Integer) As DataTable
    '-- Establish locals
    Dim loCommand As New SqlCommand()

    '-- Create the command
    loCommand.CommandText = "SELECT * FROM MyTable WHERE MyField = @MyField"

    '-- Add the parameter
    loCommand.Parameters.Add("@MyField", SqlDbType.Int)
    loCommand.Parameters("@MyField").Value = MyValue

    '-- Execute the command
    Return Me.GetDataTable(loCommand)
End Function

```

## 5.3 – Execute methods

The BusinessLayer class contains wrapper methods for both ExecuteScalar and ExecuteNonQuery. Each method has overloads that allow you to execute either a string statement or a parameterized command. The ExecuteScalar command is very commonly used during business rules checking to retrieve duplicate record counts and other single values.

### ExecuteScalar() [C#]

```
public bool EmailDuplicateExists()
{
    //-- Establish locals
    SqlCommand loCommand = new SqlCommand();

    //-- Create the command
    loCommand.CommandText = "SELECT COUNT(*) FROM " +
        "MyTable WHERE MyEmailAddress = @MyEmailAddress AND MyPK != @MyPK";

    //-- Add the parameters
    loCommand.Parameters.Add("@MyEmailAddress", SqlDbType.NVarChar);
    loCommand.Parameters["@MyEmailAddress"].Value = Me.MyEmailAddress;
    loCommand.Parameters.Add("@MyPK", SqlDbType.Int);
    loCommand.Parameters["@MyPK"].Value = Me.MyPK;

    //-- Execute the command
    return ((int)Me.ExecuteScalar(loCommand)) > 0;
}
```

### ExecuteScalar() [Visual Basic]

```
Public Function EmailDuplicateExists() As Boolean
    '-- Establish locals
    Dim loCommand As New SqlCommand()

    '-- Create the command
    loCommand.CommandText = "SELECT COUNT(*) FROM " & _
        "MyTable WHERE MyEmailAddress = @MyEmailAddress AND MyPK != @MyPK"

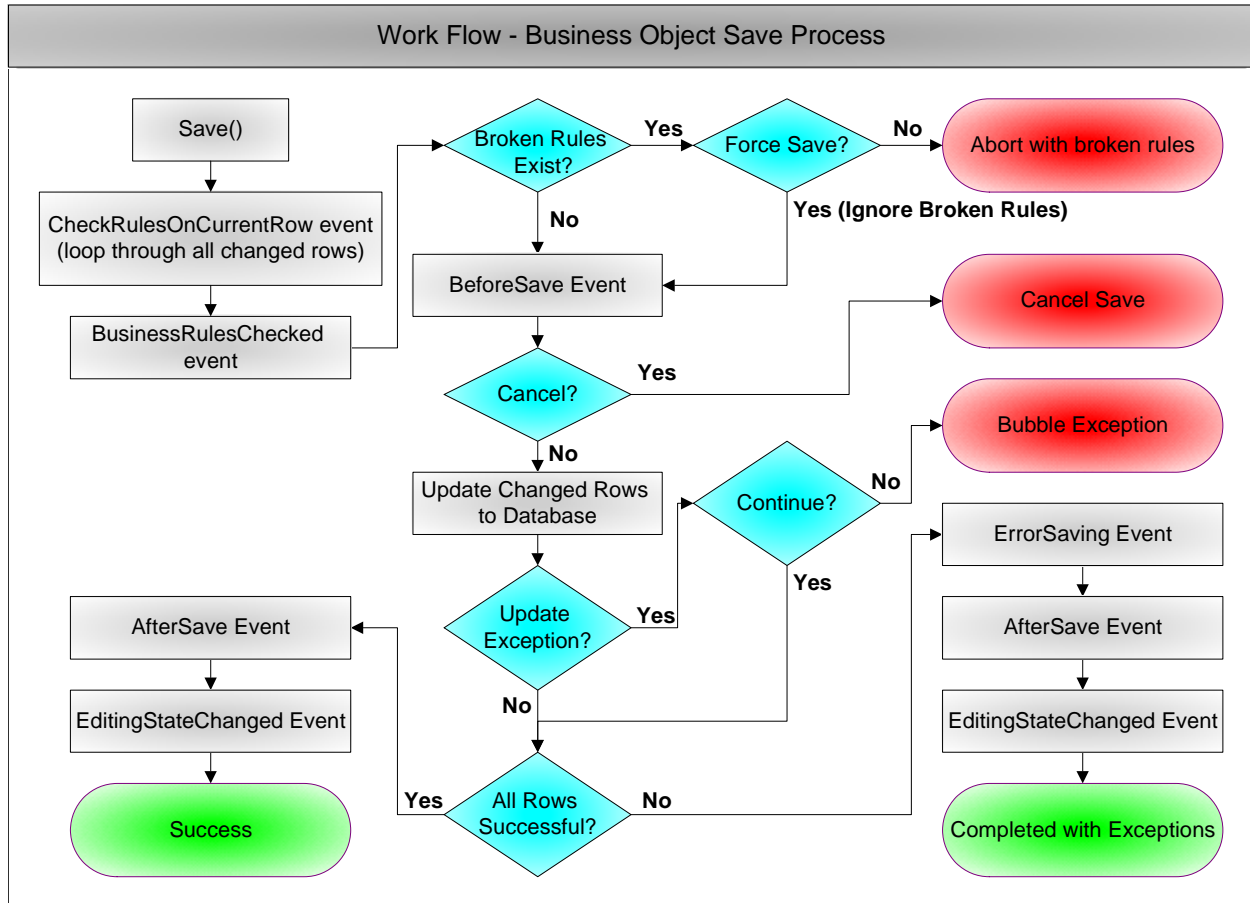
    '-- Add the parameters
    loCommand.Parameters.Add("@MyEmailAddress", SqlDbType.NVarChar)
    loCommand.Parameters("@MyEmailAddress").Value = Me.MyEmailAddress
    loCommand.Parameters.Add("@MyPK", SqlDbType.Int)
    loCommand.Parameters("@MyPK").Value = Me.MyPK

    '-- Execute the command
    Return (CType(Me.ExecuteScalar(loCommand), Integer) > 0)
End Function
```

## 5.4 – Saving a business object

A Save() method exists on the business object to allow the changes within the business object to be persisted back to the database. Overloads exist on the Save() method that allow the business object to be saved on a transaction.

### 5.4.1 – Work Flow – Business object save process



## 5.5 – Customizing the saving process – error handling

### 5.5.1 – AcceptRowChangesMode

The **AcceptRowChangesMode** property controls when the changes to the business object's **DataRow**s are changed. The possible values are: 1) **AcceptAfterRowSaved** – (default) The changes are accepted if the row successfully updates 2) **AcceptAfterTableSaved** – The changes are saved only if the entire table successfully saves (this value is assumed if saving the business object on a transaction).

### 5.5.2 – ErrorSavingMode

The **ErrorSavingMode** property controls how the business object will handle SQL errors that occur when saving. The possible values are: 1) **FailOnError** – (default) the business object will bubble the exception up to the method that called **Save()** on the business object. Any rows not saved will not be saved (the process is aborted). 2) **ContinueOnError** – the business object will store the exception, but continue to

save the rest of the rows. At the end of the saving process, the `ErrorSaving` event will be raised to provide the caller to `Save()` with the saving exceptions.

## 5.6 – Customizing the saving process – CRUD settings & concurrency

### 5.6.1 – Primary Key Specifications

The `PrimaryKeysAutoIncremented` property determines how the business object will handle the assigning of a primary key through the `IDENTITY` specification on the primary key column. If the property is set to `True`, the newly inserted `IDENTITY` value is retrieved from the server. If the value is `False`, the business object assumes that you will assign the primary key before the record is saved.

### 5.6.2 – Using stored procedures to save

Business objects can use stored procedures to save. The `DeleteUsingStoredProcedure`, `InsertUsingStoredProcedure` and `UpdateUsingStoredProcedure` properties control whether the business object will perform each of the actions using a stored procedure. The names of the stored procedures are supplied through the `DeleteStoredProcedureName`, `InsertStoredProcedureName`, and `UpdateStoredProcedureName` properties.

When using stored procedures, you can specify the string to be prefixed to each parameter through the `StoredProcedureParameterPrefix` property.

### 5.6.3 – Controlling concurrency

The `DeleteConcurrencyType`, `UpdateConcurrencyType`, and `RowVersionOrTimestampColumn` properties allow you to control the concurrency type that will be used during the saving process. If either of the `ConcurrencyType` properties is set to `OptimisticRowVersion` or `OptimisticTimestamp`, the `RowVersionOrTimestampColumn` must be specified. The concurrency type used within the stored procedures must match the concurrency type specified by the `ConcurrencyType` properties.

## **6 – Business layer**

### **6.1 – Arguments for using business objects**

The purpose of business objects must first be understood before they can be properly used. The most important reason for a business object is a separation of layers and encapsulation. By encapsulating the business and separating the logic from the UI, the business objects can be reused over and over again while maintaining all logic in a single location. Make a change to the business object and all sources that use that business object reflect the change. Additionally, this structure inherently allows an application to be much more scalable.

### **6.2 – Purpose of business objects**

Business objects form the core of any StrataFrame application. Each business object contains a DataTable that holds a disconnected record set that has been retrieved from the database or created on the local computer. The data within this DataTable is accessed through strong typed properties that are created by the Business Object Mapper. Business objects contain methods that both manipulate the data stored within the DataTable and persist the changes made back to the data store.

### **6.3 – Basics of the Business Object Mapper functionality**

The Business Object Mapper is a tool within StrataFrame that strong types business objects with properties that reflect the schema of a database object such as a table or view. To open the Business Object Mapper, open the StrataFrame menu and click “Business Object Mapper.”

#### **6.3.1 Creating a BOMapper project for a Visual Studio project**

All Visual Studio Projects that contain business objects require a corresponding BOMapper project. The BOMapper project contains the mappings and customizations for all business objects within the corresponding VSProject. A BOMapper project is linked to the full path of a VSProject on a development computer; a separate link is created for each development computer that accesses the BOMapper project.

#### **6.3.2 Mapping a database object to a business object**

This is the primary reason the Business Object Mapper exists which is to pull structure information from a source and create strong-typed properties. Assigning a source structure will give the Business Object Mapper a reference from which to pull field property information which can be reflected directly through the business object within code.

#### **6.3.3 Examining the partial class**

The Business Object Mapper places all generated code in a partial class, a separate physical file from the business object definition, which contains all of the strong-type field properties as well as field properties descriptor classes which speed the data-binding by removing much of the reflection that is inherently used within .NET data-binding.

## 6.4 – Extending the strong typing of business objects

The Business Object Mapper allows field properties to be customized strong typed beyond their original or declared type. A common example of changing the return type of a property is replacing an integer value with an enumeration.

### Enum property example [C#]

```
if (Customers.cust_phonedaytype == PhoneDayType.Work)
{ MessageBox.Show("Work Phone Found."); }
```

### Enum property example [Visual Basic]

```
If Customers.cust_phonedaytype = PhoneDayType.Work Then
    MsgBox("Work Phone Found.")
End If
```

## 6.5 – CurrentRowIndex

A business object can contain one record or a large set of records within its internal DataTable. A record pointer navigation system is used to allow the business object to work with one record at a time. The CurrentRowIndex determines the DataRow within the DataTable that is the CurrentRow of the business object. The strong typed properties created by the BOMapper do not require an index since they access the CurrentRow property.

### Non-indexed property example [C#]

```
//-- A non-indexed property does not require an index to access the property
Customers.MyField;

//-- An indexed property requires an index to access the property
//Customers.MyField[1];
```

### Non-indexed property example [Visual Basic]

```
'-- A non-indexed property does not require an index to access the property
Customers.MyField

'-- An indexed property requires an index to access the property
'Customers.MyField(1)
```



## 6.6 – Move and Navigate methods

StrataFrame provides a number of different methods to move through the data within the business object. Two types of methods commonly used are the Move and Navigate methods. The primary difference between the two types of methods is that the Navigate method will refresh any bound controls when the record position is moved while the Move methods will not refresh any bound controls.

### Scanning example [C#]

```
if (Customers.MoveFirst())
{
    do
    {
        //-- Place code here
    } while (Customers.MoveNext());
}
```

### Scanning example [Visual Basic]

```
If Customers.MoveFirst() Then
    Do
        '-- Place code here
    Loop While Customers.MoveNext()
End If
```

## 6.7 – GetEnumerator()

The If MoveFirst() Do While MoveNext() construct allows you to cycle through all of the records within a business object's CurrentView from top to bottom. However, a new feature has been added to allow business objects to be enumerated so that a For Each loop can be used rather than the If MoveFirst() Do While MoveNext() construct.

The BusinessLayer class cannot directly implement the IEnumerable interface because BindingContext would change the BindingManager used for the business object from a PropertyManager to a CurrencyManager. Therefore, the GetEnumerator() method has been added to the BusinessLayer class to return an IEnumerable object that can be referenced by the For Each loop. Internally the enumerator will save the CurrentRowIndex, call MoveFirst() and then MoveNext() and when the end of the records are reached, restore the CurrentRowIndex.

### Iterating through the records of a business object [C#]

```
//-- Using a foreach to iterate through the records of a business object
foreach (BoType bo in myBo.GetEnumerator())
{
    //-- Process the record
    MessageBox.Show(bo.PkField.ToString());
}
```

**Iterating through the records of a business object [Visual Basic]**

```
'-- Using a For Each to iterate through the records of a business object
For Each bo As BoType In myBo.GetEnumerable()
    '-- Process the record
    MsgBox(bo.PkField.ToString())
Next
```

## 6.8 – Seek methods

All business objects have several seek methods that make record position very easy. First is the SeekToPrimaryKey method which accepts a parameter that positions the business object on the record associated with the passed key. This method does not refresh any bound controls. The method with the same functionality that refreshes bound controls is the NavigateToPrimaryKey(). Next is the Seek() method which accepts a WHERE clause. The business object is positioned on the first record that matches the specified criteria.

**Seek example [Visual Basic]**

```
If Customers.Seek("cust_lname LIKE 'tay%'") Then
    '-- Place code here.
End If
```

**Seek Example [C#]**

```
if (Customers.Seek("cust_lname LIKE 'tay%'"))
{ /* Place code here */ }
```

## 6.9 – Adding, editing, and deleting records

Business objects intrinsically have methods to add, edit, and delete records. Additionally, each business object has an EditingState property that determines if a business object is currently in an Idle, Adding, or Editing state.

### 6.9.1 – Add()

The Add() method will add a new record the business object and set the editing state to “Adding.” This method will also update any bound controls so this is the method that should be used when adding a new record on a form when the associated controls need to reflect the new record for entry.

### 6.9.2 – NewRow()

The NewRow() method has the same functionality as the Add with two distinct differences. First, the editing state of the business object will not be changed. Second, any bound controls will not be refreshed.

### 6.9.3 - Edit()

The Edit() method places the current record in the business object in an “edit” state by changing the EditingState. Additionally any bound controls are made ready for modification purposes by changing the “Enabled” state.

### 6.9.4 – DeleteByPrimaryKey() and DeleteCurrentRow()

The DeleteCurrentRow() method deletes the currently active row, which is reflected by the CurrentRow and CurrentRowIndex properties. The deletion can be immediately forced back to the server or just deleted in the business object and not deleted from the server until the Save() method is called. The DeleteByPrimaryKey immediately deletes the record on the server. Additionally, the record being deleted does not need to be present within the populated business object in order to be deleted.

## 6.10 – Basics of business rules

Business rules checking and data validation within StrataFrame is handled through the use of events and the generated strong-typed properties. There are two distinct ways to define business rules which include the RequiredFields property and the custom defined business rules.

### 6.10.1 RequiredFields property

All business objects have a RequiredFields property which allows fields that require entry to be defined by simply checking a box next to each required field. StrataFrame will automatically verify these rules without any coding from the developer.

### 6.10.2 Defining custom business rules

All business objects have an event named CheckRulesOnCurrentRow which is raised for each row within the business object that requires a rules check (unmodified rows are skipped). The strong typed properties can be used within the CheckRulesOnCurrentRow event since the CurrentRowIndex of the business object is automatically moved to correct row before the event is raised.

## 6.11 – Sort, Filter, and Count

The BusinessLayer class stores the data in a DataTable within the business object, but business objects do not navigate the records within the business object from that DataTable, but rather through the DefaultView of the DataTable. This allows business objects to be sorted and filtered. The CurrentView property exposes the DefaultView of the CurrentDataTable. Setting the Sort or Filter property on the business object directly modifies the sort or the filter on the CurrentView. The Count property of the BusinessLayer class reflects the number of records visible within the CurrentView, not the number of records contained within the CurrentDataTable.

## 6.12 – Exposed ADO.NET components

In addition to the strong typed properties, all StrataFrame business objects expose the internal data through object references to the internal ADO.NET components. The BusinessLayer class exposes the CurrentDataTable (System.Data.DataTable), CurrentView (System.Data.DataView), and CurrentRow (System.Data.DataRow) properties to provide access to these ADO.NET components. Additionally, each business object has a Count property which reflects the current number of records visible. This property respects the filter of the business object.

### Exposed property example [C#]

```
MyComboBox.DataSource = Customers.CurrentView;
```

### Exposed property example [Visual Basic]

```
MyComboBox.DataSource = Customers.CurrentView
```

### 6.12.1 – CurrentDataTable property

This property is an ADO.NET data table and contains all of the records for the populated business object. This property does not respect the Filter or Sort properties.

### 6.12.2 – CurrentView property

All other properties, including the strong-typed properties, ultimately reference the current row active on the view, which is reflected in the CurrentRowIndex property. Obviously this property respects the Filter and Sort of the business object.

### 6.12.3 – CurrentRow property

The current row is the most commonly used ADO.NET property used within the business object itself. All strong-typed properties communicate with this property when retrieving a value for a particular field. This value is retrieved by returning the CurrentRowIndex from the CurrentView property.

## 6.13 – Clear()

The Clear() method removes all of the records that are currently populated within the business object. Additionally all records in a dirty state are ignored and changes are not committed back to the server. The EditingState and IsDirty state are reverted back to Idle and False.

## 6.14 – The Item property (indexer/this[])

The BusinessLayer class contains the default property Item, which allows you to access any field within the business object by specifying its string name. Unlike accessing the CurrentRow("field"), accessing BusinessObject("field") will access the column through the strong typed properties rather than directly from the CurrentRow. The Item property uses the internal dictionary of the FieldPropertyDescriptors to get or set the accessed property.

The Item property is used internally for the BusinessLayer base class to access the strong typed properties of the subclasses (the actual business objects).

### Setting a field property through the this[] Property [C#]

```
public void SetFirstName(string NewName)
{ this.Customers["cust_fname"] = NewName; }
```

### Setting a field property through the Item() property [Visual Basic]

```
Public Sub SetFirstName(ByVal NewName As String)
    Me.Customers("cust_fname") = NewName
End Sub
```

## 6.15 - Basic parent child relationship

There are a number of different ways to manage relationships within StrataFrame, but the most common and most basic is a one-to-many parent-child relationship. StrataFrame automatically manages the foreign key relationships between the parent and child, including the assignment of the parent primary key value to the child. The definition of the relationship needs to be setup one time on the child business object class (ParentRelationship property); after that, the specific instance of the parent business object type must be specified for each desired child business object instance (ParentBusinessObject property).

## 6.16 – Business object events

There are many business object events that allow developers to insert code when certain actions occur within the business object. Events are used for hooks into the framework both to follow the standard conventions of .NET and to allow developers to attach to the hooks from more than one location within the application.

### 6.16.1 – CheckRulesOnCurrentRow event

This event is where all custom business rules checking will be placed within each business object.

#### CheckRulesOnCurrentRow example [C#]

```
private void CustomersBO_CheckRulesOnCurrentRow(CheckRulesEventArgs e)
{
    if (this.cust_lname.Length < 5)
    { this.AddBrokenRule(CustomersBOFieldNames.cust_lname, "Must be at least 5 characters long."); }
}
```

**CheckRulesOnCurrentRow example [Visual Basic]**

```
Private Sub CustomersBO_CheckRulesOnCurrentRow(ByVal e As CheckRulesEventArgs)
    If Me.cust_lname.Length < 5 Then
        Me.AddBrokenRule(CustomersBOFieldNames.cust_lname, "Must be at least 5 characters long.")
    End If
End Sub
```

**6.16.2 – Navigated event**

The Navigated is raised each time a Navigate() is called on the business object and the record pointer changes with a UI update. A handler for the Navigated event is the most common place for code to be placed that will requery lists and perform any action needed to update the UI due to a record pointer change.

**Navigated Example [C#]**

```
private void Customers_Navigated(Business.NavigatedEventArgs e)
{ MessageBox.Show(e.NavigatedPosition.ToString()); }
```

**Navigated Example [Visual Basic]**

```
Private Sub Customers_Navigated(ByVal e As Business.NavigatedEventArgs)
    MsgBox(e.NavigatedPosition.ToString())
End Sub
```

**6.16.3 – BusinessRulesChecked event**

This event is raised after all business rules have been checked. All information related to broken rules can be gathered from within this event.

**BusinessRulesChecked Example [C#]**

```
private void Customers_BusinessRulesChecked(BusinessRulesCheckedEventArgs e)
{
    // Cycle through all of the broken rules
    foreach (BrokenRule loRule in this.Customers.BrokenRules.ToArray())
    { MessageBox.Show(loRule.Description, loRule.FieldName); }
}
```

**BusinessRulesChecked Example [Visual Basic]**

```
Private Sub Customers_BusinessRulesChecked(Byval e As BusinessRulesCheckedEventArgs)
    '-- Cycle through all of the broken rules
    For Each loRule As BrokenRule In Me.Customers.BrokenRules.ToArray()
        MsgBox(loRule.Description, MsgBoxStyle.OkOnly, loRule.FieldName)
    Next
End Sub
```

## **7 – WinForms development**

### **7.1 – Creating a WinForms project**

StrataFrame includes project templates for the creation of C# and VB.NET WinForms applications. Creating a StrataFrame WinForms project is as simple as selecting the appropriate template when creating a new project.

### **7.2 – Understanding the AppMain.vb (Program.cs) file**

The AppMain.vb (Program.cs) file contains the entry point for a StrataFrame application. The entry point is the shared (static) Main() method. This method is configured by the template to be the first method called by the .NET runtime for the application.

The Main() method makes a call to StrataFrameApplication.Run() at the bottom of the method body. This starts the execution of the application through the rest of the methods in the AppMain.vb file.

#### **7.2.1 – SetDataSources()**

The SetDataSources() method is the first method to be called after StrataFrameApplication.Run(). This method is provided to configure the data sources for the application. Within this method, you can add the data sources manually, or you can use the ConnectionManager class to configure the data sources.

The data sources for the application can be altered at runtime, but this is the most logical place to initial configure the data sources.

#### **7.2.2 – InitApplication()**

The InitApplication() method is called immediately after the SetDataSources() method has exited. The InitApplication method is used to configure any additional settings for the application that are not considered data sources. Localization and security settings are usually configured within this method. The possible main form types for the application are also provided to the StrataFrameApplication runtime through this method.

#### **7.2.3 – ShowGateway()**

The ShowGateway() method is used to show a “gateway” form to the end-user. A gateway form is a menu form that allows the end-user to select from a list of possible main forms that can be opened or exit the application. Showing a gateway form is entirely optional.

#### **7.2.4 – ShowLoginAndInitMainForm()**

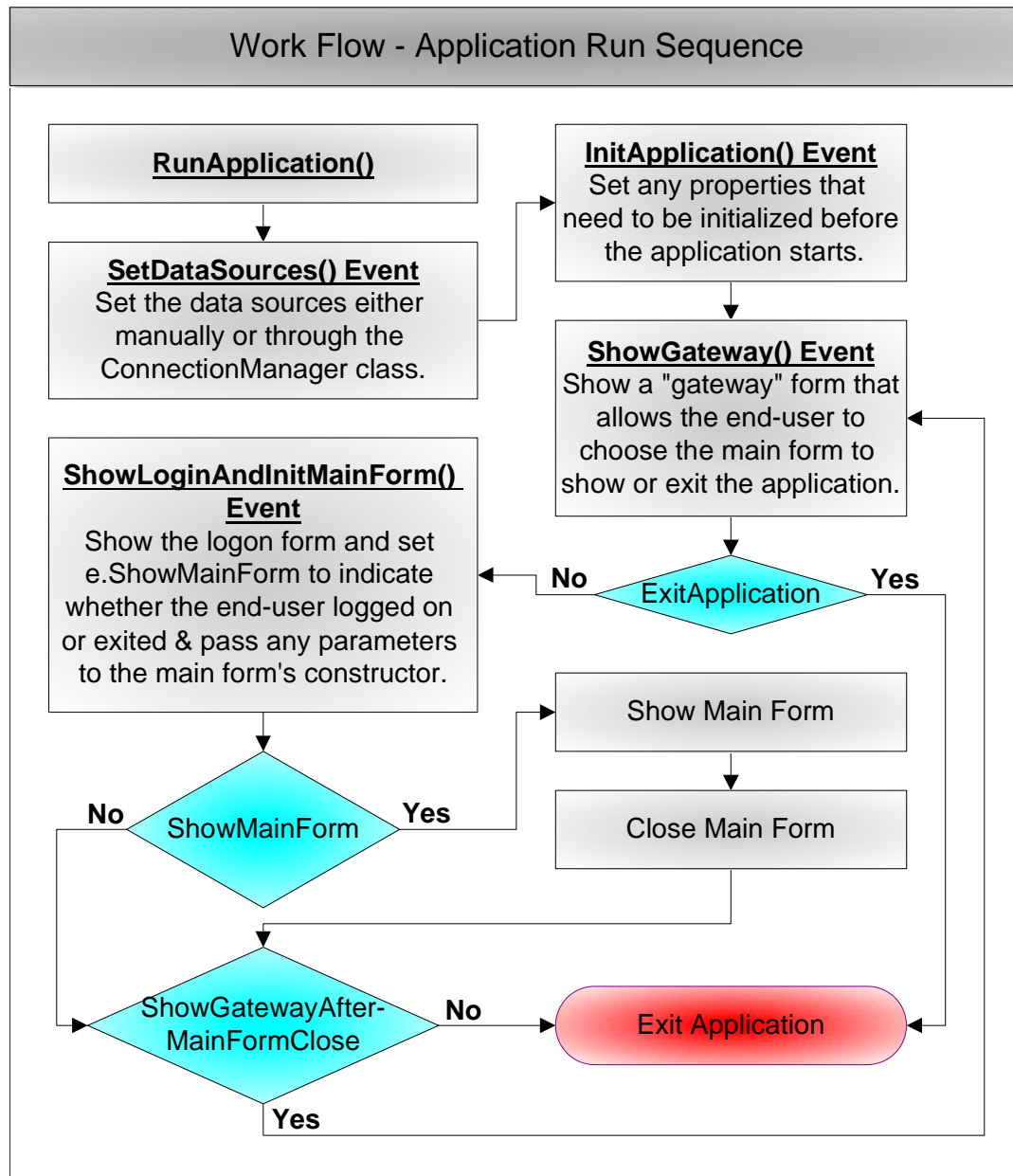
The ShowLoginAndInitMainForm method is called after the application has determined the main form to show. It is responsible for showing a logon form to the end-user (optional) and passing any parameters to the constructor of the main form through the event arguments.

#### **7.2.5 – UnhandledExceptionFound()**

The UnhandledExceptionFound() method is called if the StrataFrameApplication class catches an exception that is not handled within the code. This method provides a central place to log any

unhandled exceptions thrown by the application. To prevent the ApplicationExceptionDialog (red error window) from showing, set e.Handled = true within this method.

### 7.2.6 – Work Flow – Application execution order



## 7.3 – Creating business object libraries

A Business Object Library is a class library that contains business object classes. It can contain other classes besides the business objects, since it is a basic class library. The template for the Business Object Library is the exact same as the template for a .NET Class Library, but instead of having the initial object as Class1, it is BusinessObject1. Business Object Libraries can be referenced in the same way as any other .NET assembly.



Business objects that are to be used by an ASP.NET application must be contained within a class library. They cannot be contained directly within the ASP.NET application.

## **8 – Role-based security**

### **8.1 – Permission Keys**

The StrataFrame role-based security module is based around the concept of a “permission key” (aka “security key”). Objects within a StrataFrame application are assigned a permission key. Permissions are defined for the application and each is identified by a unique permission key. The permission key on an object within the application indicates the permission that is required to view/edit the object. Permissions are assigned to roles and users through the Security Maintenance Dialog. At runtime, the permissions of the current user are evaluated each object checks the permissions of the current user to determine whether the user has access to that object.

### **8.2 – Security database tables**

The StrataFrame security module requires that a specified set of tables exist on SQL Server. These tables all begin with the prefix “SFS” and contain all of the data used by the security module.

### **8.3 – Security maintenance**

The Security Maintenance Dialog is used to maintain the preferences, users, roles, permissions, and restrictions sets used by the security subsystem of the application. The same security dialog is shown at both design-time and runtime with the only difference between the two being that users cannot edit, add or delete permissions (permission can only be defined and created by the application developer).

#### **8.3.1 – Security projects**

Like a DDT profile or a Messaging & Localization project, a Security project contains all of the security settings specific to an application. The Security projects are used as “containers” to separate the settings of one application from the settings of another application. The projects are only needed at design-time (because at runtime, only one application will exist); in fact, the SFSPROJECTS tables does not need to be deployed to SQL Server.

#### **8.3.2 – Preferences**

The Global Preferences within the Security Maintenance Dialog contain all of the settings that affect all users: password length and strength, password history, invalid logon attempts handling, session timeout time, etc.

#### **8.3.3 – Users**

The users node within the Security Maintenance Dialog contains all of the users defined within the application. All user information is stored within the database in encrypted fields. The encryption key used to view the data is stored within the project settings and must be matched through the `SecurityBasics.SetSecurityKeyAndVectorForUserAuthentication()` method. Each user has several settings that can be used to override the global preferences for just that user.

Permissions can be assigned to a user either through the roles or by assigning a permission directly to a user.

### 8.3.4 – Roles

The roles node within the Security Maintenance Dialog contains all of the roles defined within the application. Permissions can be assigned to roles which can then be assigned to users to allow the users to inherit the assigned permissions.

### 8.3.5 – Permissions

The permissions are the heart of the security module within a StrataFrame application. Permissions are grouped logically by categories. When assigning permissions to a user or role, the category nodes can be used to simplify the configuration of the permissions by cascading the changes to the permissions listed beneath the category. When you create a permission, you must assign an action that will be taken if that permission is denied to a user.

### 8.3.6 – Restriction Sets

Restriction sets allow you and the end-user to define sets of time and workstation restrictions that can be assigned to a permission when the permission is assigned to a user and/or role. The restriction sets allow an unlimited number of line items to be added to the restriction set. Wild cards can be used within the workstation name to set match multiple computers (\* for multiple characters, ? for a single character).

### 8.3.7 – Assigning permissions to users/roles

When permissions are assigned to a user/role, an action is specified to indicate what should happen when the user attempts to access an object within the application that requires that permission (grant, deny, or block). A default action is specified programmatically within the application. The default action is used for any permission that is not assigned to the user at runtime. A restriction set can also be assigned to the assigned permission to restrict or open the access granted to the user.

A user's apparent permissions within the application are a combined list of the permissions granted to the user by the user's assigned roles and the user's assigned permissions. Any permission assigned directly to a user overrides whatever action has been granted to the user for that permission by any assigned roles. When multiple roles that have the same permission assigned are assigned to a user, the user's inherited permission is the greatest of the permission actions assigned to the role.

## 8.4 – Assigning permissions to objects

A type editor is provided to allow you to search the projects defined permissions for a specific key to assign to a SecurityKey property on an object. Security keys can be assigned to forms to block access to the form, to business objects to prevent access to adding, editing, or deleting the records within the business object, and to individual fields on a business object to provide field-level security.

## 8.5 – Programmatic access

### 8.5.1 – CurrentUser

The `MicroFour.StrataFrame.Security.CurrentUser` property contains the `ISecurityUser` object that is currently logged into the application. The `CurrentUser` property exposes the username, primary key, and permissions of the currently logged on user.

### 8.5.2 – Retrieving permissions programmatically

Permissions can be retrieved programmatically by using the `GetPermission()` method on the `CurrentUser` object. This method will return a `PermissionInfo` object that contains the permission information describing the action granted to the user for the requested permission.

### 8.5.3 – Retrieving Global Preferences

Global preferences are stored within the `SFSPreferences` table in the database. At runtime, the preferences are accessed through public properties on the `SecurityBasics` class. To retrieve the preferences from the database and assign them to the properties on the `SecurityBasics` class, the `MicroFour.StrataFrame.Security.BusinessObjects.SFSPreferencesBO.RetrieveSecurityPreferences()` method is provided.

## 8.6 – The Logon class

The `Logon` class is a sealed/static class that contains methods that can be used to show logon forms and authenticate users into the application. The `ShowLogonAndAuthUser()` method shows the logon form specified by the `LogonFormType` property and authenticates the user. A Boolean value is returned to indicate whether the end-user authenticated into the system and the `CurrentUser` value was changed.

## 8.7 – The SessionLock class

The StrataFrame role-based security module contains extra functionality to allow an application to be “locked” when the user’s session times out or the user chooses to explicitly lock the application. When an application is locked, all forms are minimized and the logon form is displayed to allow a user to re-authenticate into the system. When a user logs back onto the system, the permissions are re-evaluated and forms, fields, and objects are hidden or displayed to the user depending upon the permissions assigned to the user.

## 8.8 – Creating a custom logon form

Most developers will prefer to use a custom logon form over the one that is included within the StrataFrame security module. To create a custom logon form, you must implement the `ILogonForm` interface on a Windows form and set the `LogonFormType` to a type reference of the custom logon form type. The Security subsystem will then use that logon form type instead of the default logon form type. The source code for the included logon form can be used as a template for a custom logon form.

## 9 – Tips & tricks #2

### 9.1 – Events

#### 9.1.1 – What is an event?

An event is a special field within a class that contains a delegate. In VB, the delegate is created behind the scenes to hold the pointers to the methods that are configured to “handle” the event; in C#, the delegate is explicitly created rather than handled behind the scenes. When the event is “raised” the delegate behind the scenes is invoked and calls the methods that have been configured to handle the event.

##### Declaring an event [C#]

```
public event EventHandler MyEvent;
```

##### Declaring an event [Visual Basic]

```
Public Event MyEvent As EventHandler;
Public Event MyEvent2(ByVal sender As Object, ByVal e As EventArgs)
```

#### 9.1.2 – What is a delegate?

A delegate is a special data type that is used to describe a method; the description includes the parameters and the return type of the method. An instance of a delegate contains a pointer (reference) to a method (or methods) that have the same signature as the delegate; whereas an instance of a class is a reference that points to a location on the application’s memory heap, a delegate points to a location in the application code within memory.

A delegate instance can contain references to 0 or more methods; this list of methods that are referenced is called the delegate’s “invocation list.” Any delegate that contains more than 1 reference is considered to be a multicast delegate. When a delegate is invoked, the invocation list is “walked” from top to bottom (first in first out) and each referenced method is invoked in turn.

Delegates are used when external code needs to be called and the code that invokes the delegate has no reference to the method that should be called.

##### Declaring and using a delegate [C#]

```
//-- Declaring the delegate (just like declaring a new type)
public delegate void MyDelegate(object param);

//-- Declaring a method that has the same signature of the delegate
public void MyMethod(object param)
{ //-- Do some work }

//-- Creating a new instance of the delegate
MyDelegate var = new MyDelegate(MyMethod);

//-- Invoking the delegate
var(value);
// or
var.Invoke(value);
```

**Declaring and using a delegate [Visual Basic]**

```
'-- Declaring the delegate
Public Delegate Sub MyDelegate(ByVal param As Object)

'-- Declaring a method that has the same signature of the delegate
Public Sub MyMethod(ByVal param As Object)
    '-- Do some work
End Sub

'-- Creating a new instance of the delegate
Dim var As New MyDelegate(AddressOf MyMethod)

'-- Invoking the delegate
var(value)
' or
var.Invoke(value)
```

**9.1.3 – Common System.Delegate methods**

The System.Delegate class contains several methods and properties that are useful when working with delegates programmatically.

**Combine()** – The Combine() method is a static (shared) method that is used to combine 2 delegates of the same type into one delegate. Since a delegate can have more than 1 method reference, calling Combine() and passing 2 delegates returns a single delegate that contains a combined invocation list of the two delegates. When adding a new method to an existing delegate, the Combine() method is used on the existing and new delegate and the result is set back to the existing delegate.

**Combining 2 delegates [C#]**

```
MyDelegate myNewDeleg = (MyDelegate)Delegate.Combine(myDeleg1, myDeleg2);
```

**Combining 2 delegates [Visual Basic]**

```
Dim myNewDeleg As MyDelegate = CType([Delegate].Combine(myDeleg1, myDeleg2), MyDelegate)
```

**CreateDelegate()** – The CreateDelegate() method is a static (shared) method that is used to dynamically create a delegate at runtime that matches the method signature of the specified method. Several overloads exist to allow you to specify the method for which you want to create the delegate. This method does not create a new delegate that points to the specified method, but an empty delegate that matches the specified method's signature.

**Using Delegate.CreateDelegate() [C#]**

```
//-- Dynamically creating a delegate to a target method
Delegate deleg = Delegate.CreateDelegate(typeof(Form1), this, "TargetMethod");

//-- The target method of the delegate
private void TargetMethod(object param)
{
    //-- Do some work
}
```

**Using Delegate.CreateDelegate [Visual Basic]**

```
'-- Dynamically creating a delegate to a target method
Dim deleg As [Delegate] = [Delegate].CreateDelegate(GetType(Form1), Me, "TargetMethod")

'-- The target method of the delegate
Private Sub TargetMethod(ByVal param As Object)
    '-- Do some work
End Sub
```

***DynamicInvoke()*** – The DynamicInvoke() method is an instance method that is used to dynamically invoke a delegate. If you don't know the signature of the delegate or you programmatically created the delegate and you don't know its type, then you can call DynamicInvoke() and pass the parameters as an param array to invoke the delegate.

**Dynamically invoking a delegate [C#]**

```
deleg.DynamicInvoke(sender, e);
```

**Dynamically invoking a delegate [Visual Basic]**

```
deleg.DynamicInvoke(sender, e)
```

***GetInvocationList()*** – The GetInvocationList() method is an instance method that returns an array of single-cast delegates for the delegate on which it was called. If your delegate contains the references to 5 methods and you call GetInvocationList(), then it will return an array containing 5 separate delegates of the same type, each containing one of the method references.

**Getting the invocation list from a delegate [C#]**

```
EventHandler deleg;
EventHandler[] list = (EventHandler[])deleg.GetInvocationList();
```

**Getting the invocation list from a delegate [Visual Basic]**

```
Dim deleg As EventHandler
Dim list As EventHandler() = CType(deleg.GetInvocationList(), EventHandler())
```

***Remove()*** – The Remove() method is a static method that removes a method pointer from a delegate and returns a new delegate containing the invocation list without the removed delegate.

**Removing one delegate from another [C#]**

```
//-- Removing delegate #2 from delegate #1
deleg1 = (EventHandler)Delegate.Remove(deleg1, deleg2);
```

**Removing one delegate from another [Visual Basic]**

```
'-- Removing delegate #2 from delegate #1
deleg1 = CType([Delegate].Remove(deleg1, deleg2), EventHandler)
```

***RemoveAll()*** – The RemoveAll() method is a static method that removes all occurrences of a method reference from a delegate.

#### **Using Delegate.RemoveAll() [C#]**

```
//-- Removing all occurrences of delegate #2 from delegate #1
deleg1 = (EventHandler)Delegate.RemoveAll(deleg1, deleg2);
```

#### **Using Delegate.RemoveAll() [Visual Basic]**

```
'-- Removing all occurrences of delegate #2 from delegate #1
deleg1 = CType([Delegate].RemoveAll(deleg1, deleg2), EventHandler)
```

## 9.1.4 – Creating events

### 9.1.4.1 – Conforming to standards

Microsoft has published a set of conventions for programming classes to conform to the .NET standards. One of these conventions is using delegates with 2 parameters for events. The first parameter is always of type System.Object and is a reference to the object that raised the event (the “sender” of the event). The second parameter is the “event args” of the event. The event args should be an instance of a class that inherits from System.EventArgs and contains all of the properties that should be passed to the handler of the event.

This convention even applies to events that do not have any data to pass to the handlers. In the case where no data is required by the handler the System.EventHandler delegate should be used as the event type.

In VB, you have more control over the delegate used by the event if you explicitly declare the delegate rather than declare the event’s parameters on the event. By explicitly creating the delegate, you can use the same delegate for more than one event. If you create the event using the “short-hand” form, the delegate is created behind the scenes for the event.

### 9.1.4.2 – OnEvent() methods

Another convention recommended by .NET is the creation of an OnEvent() method. Every event should have an OnEvent() method that corresponds to the event (a “TextChanged” event will have an “OnTextChanged()” method that raises the event. The event is then never raised directly, but instead, the OnEvent() method is called to raise the event. The event args are passed to the OnEvent() method which uses the current object reference (Me or this) as the first parameter of the event and the supplied event args as the second parameter.

When the System.EventHandler type is used, the OnEvent() can be parameterless and EventArgs.Empty is used as the second parameter when raising the event (to cut down on instantiating a new instance of the EventArgs class every time, even though it won’t be used).

The OnEvent() should be of protected scope and virtual (overridable). This allows an inheriting class to raise the event if necessary and override the OnEvent() method to perform custom processing if necessary.



**OnEvent example [C#]**

```
//-- Declaring the OnEvent() for an EventHandler event
protected virtual void OnMyEvent()
{
    if (this.MyEvent != null)
    { this.MyEvent(this, EventArgs.Empty); }
}

//-- Declaring the OnEvent() for an event with arguments
protected virtual void OnMyEvent2(MouseEventArgs e)
{
    if (this.MyEvent2 != null)
    { this.MyEvent2(this, e); }
}
```

**OnEvent example [Visual Basic]**

```
'-- Declaring the OnEvent() for an EventHandler event
Protected Overridable Sub OnMyEvent()
    RaiseEvent MyEvent(Me, EventArgs.Empty)
End Sub

'-- Declaring the OnEvent() for an event with arguments
Protected Overridable Sub OnMyEvent(ByVal e As MouseEventArgs)
    RaiseEvent MyEvent2(Me, e)
End Sub
```

**9.1.5 – Overriding an OnEvent() method**

When a subclass needs to handle an event defined in a base class, it is recommended to override the OnEvent() method rather than handle the event. This is recommended for 2 reasons:

While invoking a delegate is “fast,” it is slow compared to overriding a method.

When handling an event, you have no control over the order in which your handler will be called in relation to the other handlers on that event. By overriding the OnEvent() method, you can force your logic to be called either before or after the event is actually raised.

Don’t forget to call the base.OnEvent() (MyBase.OnEvent()) or the event will never be raised.

**OnEvent() override example [C#]**

```
//-- Overriding an OnEvent() method in a subclass
protected override void OnLoad(EventArgs e)
{
    //-- Process work that should be done before the event is raised
    //-- Call the base method to raise the event
    base.OnLoad(e);
    //-- Process work that should be done after the event is raised
}
```

**OnEvent() override example [Visual Basic]**

```

'-- Overriding an OnEvent() method in a subclass
Protected Overrides Sub OnLoad(ByVal e As EventArgs)
    '-- Process work that should be done before the event is raised

    '-- Call the base method to raise the event
    MyBase.OnLoad(e)

    '-- Process work that should be done after the event is raised
End Sub

```

### 9.1.6 – Illegal cross-thread calls

Event handlers (and all methods in the invocation list of a delegate) are invoked on the thread that calls the invoke of the delegate. So, if an event is raised on a background thread, your handler is going to be called on the same background thread.

In .NET 2.0, Microsoft added the functionality to test for illegal cross-thread calls. An illegal cross-thread call is a call to a control's method from a thread other than the thread that created the control. So, if a form is created on the main thread (almost always is) and you attempt to call a method on the form from a background thread, an `InvalidOperationException` is thrown.

The reason a cross-thread call is illegal is that the calling thread has no idea what the state is of the thread that created the control. The control owning thread might be executing something at the same time that is also modifying the control.

### 9.1.7 – `ISynchronizeInvoke` and threading

The .NET framework provides a way to avoid cross-thread calls: the `ISynchronizeInvoke` interface. This interface is implemented on the `System.Windows.Forms.Control` object, so you can synchronize a delegate on any control.

The `ISynchronizeInvoke` interface contains the `InvokeRequired` property that determines whether an invoke is required (it checks the thread context of the property's get caller and compares it with the thread context of its creator, if they're the same, an invoke is not required because it's the same thread.)

If the `InvokeRequired` property returns `True`, then you can use the `Invoke()` method to cause a delegate (event) to be invoked on the control's thread rather than the current thread. The `ISynchronizeInvoke.Invoke()` method waits until the control's thread has a break in processing and then calls the method.

**Using an ISynchronizeInvoke object [C#]**

```
//-- Testing the InvokeRequired to determine whether
// the ISynchronizeInvoke.Invoke() should be used
private void InvokeIfNecessary(MouseEventArgs e)
{
    if (this.syncObj.InvokeRequired)
    { this.syncObj.Invoke(delegate, new object[] { this, e }); }
    else
    { this.delegate(this, e); }
}
```

**Using an ISynchronizeInvoke object [Visual Basic]**

```
'-- Testing the InvokeRequired to determine whether
' the ISynchronizeInvoke.Invoke() should be used
Private Sub InvokeIfNecessary(ByVal e As MouseEventArgs)
    If Me.syncObj.InvokeRequired Then
        Me.syncObj.Invoke(delegate, New Object() { this, e })
    Else
        Me.delegate(this, e)
    End If
End Sub
```

### 9.1.8 – Getting a reference to an ISynchronizeInvoke object

Using the ISynchronizeInvoke interface requires a reference to an object that implements that interface. That generally means that you need a reference to the form or a reference to a control on the form. Rather than remembering to manually specify a reference to the form when you create a new instance of your component, you can use the following code to “automagically” get a reference to the parent form.

#### Getting a reference to an ISynchronizeInvoke object [C#]

```
private ISynchronizeInvoke _SyncObj;
[Browsable(false)]
public ISynchronizeInvoke SyncObj
{
    get
    {
        //-- Only process this code if the internal
        //  reference is null and we are at design-time
        if ((this._SyncObj == null) && (this.DesignMode))
        {
            //-- Get a reference to the Visual Studio designer
            IDesignerHost loDesigner =
                (IDesignerHost)this.GetService(typeof(IDesignerHost));

            //-- If the designer was retrieved, the root component
            //  will be the form or user control, which will
            //  implement ISynchronizeInvoke
            //-- If the root component is this object, then don't set it
            if (loDesigner != null)
            {
                if (loDesigner.RootComponent == this)
                { this._SyncObj = null; }
                else
                { this._SyncObj =
                    (ISynchronizeInvoke)loDesigner.RootComponent; }
            }
        }

        //-- Return the internal reference
        return this._SyncObj;
    }
    set
    {
        {
            this._SyncObj = value;
        }
    }
}
```

**Getting a reference to an ISynchronizeInvoke object [Visual Basic]**

```

Private _SyncObj As ISynchronizeInvoke
<Browsable(False)> _
Public Property SyncObj() As ISynchronizeInvoke
    Get
        '-- Only process this code if the internal reference
        ' is null and we are at design-time
        If (Me._SyncObj Is Nothing) AndAlso (Me.DesignMode) Then
            '-- Get a reference to the Visual Studio designer
            Dim loDesigner As IDesignerHost = _
                CType(Me.GetService(GetType(IDesignerHost)), IDesignerHost)

            '-- If the designer was retrieved, the root component will be
            ' the form or user control, which will implement
            ' ISynchronizeInvoke
            '-- If the root component is this object, then don't set it
            If (loDesigner IsNot Nothing) Then
                If loDesigner.RootComponent Is Me Then
                    Me._SyncObj = Nothing
                Else
                    Me._SyncObj = _
                        CType(loDesigner.RootComponent, ISynchronizeInvoke)
                End If
            End If
        End If

        '-- Return the internal reference
        Return Me._SyncObj
    End Get
    Set(ByVal value As ISynchronizeInvoke)
        Me._SyncObj = value
    End Set
End Property

```

**9.1.9 – When to create a custom, thread-safe event**

If you have a class that will generally be used on a background thread and it needs to raise events to work with the UI, you will want to create a custom event to synchronize the event with the main thread. Otherwise, don't create "thread-safe" events because they are even slower than regular events (the testing on the `InvokeRequired` and calling `Invoke()` waste a lot of processor cycles).

A perfect example of when to use a custom, thread-safe event is a background worker. If you create a component that processes on a background thread and raises progress and/or completed events, you want to raise those events on the main thread, not the thread that is processing, because you generally want to update the UI with the progress.

Creating a custom event requires several steps:

1. Define a private field to store a synchronizing object that implements the `ISynchronizeInvoke` interface (using the above code above or manually).
2. Define a field that will store the delegate for the event.

3. Create custom Add and Remove methods to add and remove handlers from the event's field.
4. (VB only) Create a custom RaiseEvent that tests the InvokeRequired to either directly call the event or to Invoke() on the synchronizing object.
5. (C# only) Modify the OnEvent() to test the InvokeRequired to either directly call the event or to Invoke() on the synchronizing object.

#### **Creating a custom, thread-safe event [C#]**

```
private EventHandler _MyEvent;
public event EventHandler MyEvent
{
    add
    {
        this._MyEvent += value;
    }
    remove
    {
        this._MyEvent -= value;
    }
}
protected virtual void OnMyEvent()
{
    if (this._MyEvent != null)
    {
        if (this._SyncObj.InvokeRequired)
        { this._SyncObj.Invoke(this._MyEvent, new object[] { this, EventArgs.Empty }); }
        else
        { this._MyEvent(this, EventArgs.Empty); }
    }
}
```

**Creating a custom, thread-safe event [Visual Basic]**

```

Private _MyEvent As EventHandler
Public Custom Event MyEvent As EventHandler
    AddHandler(ByVal value As EventHandler)
        Me._MyEvent = CType([Delegate].Combine(Me._MyEvent, value), EventHandler)
    End AddHandler
    RemoveHandler(ByVal value As EventHandler)
        Me._MyEvent = CType([Delegate].Remove(Me._MyEvent, value), EventHandler)
    End RemoveHandler
    RaiseEvent(ByVal sender As Object, ByVal e As System.EventArgs)
        If Me._MyEvent IsNot Nothing Then
            If Me._SyncObj.InvokeRequired Then
                Me._SyncObj.Invoke(Me._MyEvent, New Object() {sender, e})
            Else
                Me._MyEvent(sender, e)
            End If
        End If
    End RaiseEvent
End Event
Protected Overridable Sub OnMyEvent()
    RaiseEvent MyEvent(Me, EventArgs.Empty)
End Sub

```

## 9.2 – Debugging Tips

### 9.2.1 – Watch windows

The watch windows are one of the most widely used debugging tools in .NET (and many other programming languages, for that matter). A watch window displays the current state or return value of any property, reference or method.

### 9.2.2 – Visualizers

Several debugger visualizers exist within .NET that can help you can a better picture of the state of an application. A visualizer can be opened by clicking the small magnifying glass next to a value within the watch window.

### 9.2.3 – Locals & Autos windows

The Locals and Autos windows are watch windows that “automatically maintain” a list of variables to watch. The Locals window always displays the local variables that are currently in scope. The Autos window displays variables that are referenced by the current line of code and the executing line of code.

### 9.2.4 – Call Stack & Threads windows and “Just My Code”

The Call Stack window and the Threads window are generally used together. The Call Stack windows shows the current execution and the nested method calls for the current thread. The Threads window shows the threads that currently belong to your application. When a breakpoint is reached on a thread, the entire application breaks including all other threads. You can then switch between threads using the Threads window and the Call Stack window will change to reflect the current thread. You can double-

click any method within the call-stack to move to that method to check variables within the Watch windows; the green-highlighted line shows where the execution left that method.

Sometimes it is necessary to view the call stack of .NET, for example, to view the execution sequence that caused an event to be fired. To view the entire call stack, including methods that are part of the .NET framework, you will need to turn off the “Just My Code” option. It is found within Tools -> Options -> Debugging -> General; uncheck the “Enable Just My Code” to turn it off.

### 9.2.5 – Dynamically loading an assembly and enabling debugging

Assemblies can be dynamically loaded using static methods on the System.Reflection.Assembly class. Often, you will need to dynamically load an assembly that is not directly referenced by your application.

The above code is great for loading the assembly from the raw bytes; however, if you wrote the DLL, you will most likely also want to be able to debug into that dynamically loaded assembly. To do this, you will need to use the Assembly.Load() method that also includes the debugging symbols which are found in the .pdb file for the assembly.

#### **Dynamically loading .NET assemblies [C#]**

```
//-- Dynamically loading an assembly
private void DynamicallyLoadAssembly()
{
    Assembly.Load(File.ReadAllBytes(@"C:\MyAssembly.dll"));
}

//-- Dynamically loading an assembly with debug symbols
private void DynamicallyLoadAssemblyForDebug()
{
    Assembly.Load(File.ReadAllBytes(@"C:\MyAssembly.dll"),
        File.ReadAllBytes(@"C:\MyAssembly.pdb"));
}
```

#### **Dynamically loading .NET assemblies [Visual Basic]**

```
'-- Dynamically loading an assembly
Private Sub DynamicallyLoadAssembly()
    Assembly.Load(File.ReadAllBytes("C:\MyAssembly.dll"))
End Sub

'-- Dynamically loading an assembly with debug symbols
Private Sub DynamicallyLoadAssemblyForDebug()
    Assembly.Load(File.ReadAllBytes("C:\MyAssembly.dll"), _
        File.ReadAllBytes("C:\MyAssembly.pdb"))
End Sub
```

### 9.2.6 – Debugging in other solutions

When you debug your application, the .pdb files of other assemblies are automatically loaded when available. This allows you to step into any assembly that is built in debug and is not part of your



solution. If you have the source code, you can also open a source code file from a project that is not part of the current solution and set a break point.

## 10 – Presentation layer

### 10.1 – StrataFrame data binding explained

The data-binding within StrataFrame is designed to allow developers to assign data fields to controls through the visual form designer. The data-binding in StrataFrame relies on a combination of the `IBusinessBindable` interface and several type editors that are contained within the `MicroFour.StrataFrame.Extensibility.dll` assembly.

Data-binding to a business object makes extensive use of the `FieldPropertyDescriptor` classes that are created within the partial class for each business object. Therefore, a `FieldPropertyDescriptor` is required for all bindable properties on a business object (including custom field properties).

#### **IBusinessBindable declaration sample [C#]**

```
public class MyTextBox
    : System.Windows.Forms.TextBox, MicroFour.StrataFrame.UI.Windows.Forms.IBusinessBindable
{
    bool MicroFour.StrataFrame.UI.Windows.Forms.IBusinessBindable.BindingEditable
    {
        get{ return this.Enabled; }
        set{ this.Enabled = value; }
    }
}
```

#### **IBusinessBindable declaration sample [Visual Basic]**

```
Public Class MyTextBox
    Inherits System.Windows.Forms.TextBox
    Implements MicroFour.StrataFrame.UI.Windows.Forms.IBusinessBindable

    Public Property BindingEditable As Boolean Implements IBusinessBindable.BindingEditable
        Get
            Return Me.Enabled
        End Get
        Set(ByVal value As Boolean)
            Me.Enabled = value
        End Set
    End Property
End Class
```

### 10.2 – IncludeInForm properties

The `BaseForm` class has properties that determine how the business objects will be accessed when form methods are called. The `BaseForm` class has several methods, such as `Add()`, `Edit()`, `Delete()`, `Navigate()`, etc. that can be called on the form to call the corresponding methods on one or more business objects on the form. The `IncludeInForm*` properties can be set to `AllBusinessObjects`, `PrimaryBusinessObject`, or `DeterminedByBusinessObject`. The `AllBusinessObjects` setting indicates that all business objects should be included in the method; `PrimaryBusinessObject` indicates that only the `PrimaryBusinessObject` of the form should be included in the method. `DeterminedByBusinessObject` allows each business

object to determine whether it should be included within the action through corresponding IncludeInForm\* Boolean properties on the business objects themselves.

### 10.3 – UI automation

StrataFrame has many properties on a form that instruct the forms to automatically handle certain events and how to handle those events. Some examples of this are the AutoShow properties, error provider settings, and the RememberForm properties. It is important to keep in mind that all automated features of StrataFrame can be turned off and handled manually by the developer.

### 10.4 – EditingState

As mentioned earlier in the Business Object Basics section, a business object has an editing state which can affect the “Enabled” state of a bound control. There are two ways to work around the editing state: on each control or at the business object level.

**ManageUIReadOnlyState** – This property exists on each business object and when set to False, no controls will be automatically placed into an altered “Enabled” state based on the state of the business object.

**IgnoreUIReadOnlyState** – This property resides on each control. To exclude a single control from being managed by a business object, simply set this property to True.

Note: if you set a control to ignore the ReadOnly state specified by a business object, several field-level security options will not work properly (i.e. if a field is blocked, the control will not be disabled by the business object if the control is ignoring the business object’s attempts to set it’s BindingEditable state).

### 10.5 – Corresponding form events

Many of the events on a business object are reflected within a form as well. A form can contain multiple business objects which may or may not be included in certain form wide events, such as a save or a broken rules notification.

#### **BusinessRulesChecked (Form Event) Example [C#]**

```
private void MyForm_BusinessRulesChecked(BusinessRulesCheckedEventArgs e)
{
    MessageBox.Show(e.CountOfBrokenRulesTotal.ToString());
}
```

#### **BusinessRulesChecked (Form Event) Example [Visual Basic]**

```
Private Sub MyForm_BusinessRulesChecked(Byval e As BusinessRulesCheckedEventArgs)
    MsgBox(e.CountOfBrokenRulesTotal.ToString())
End Sub
```

### 10.6 – Controls overview

StrataFrame inherits all of the basic .NET controls, but has added additional functionality to many of the controls. In addition to the standard controls, StrataFrame has many exclusive controls as well that really help in creating an aesthetically pleasing end-user environment. This section introduces some of the StrataFrame exclusive controls and touches on the basic controls. We will talk more in-depth in later sessions about some of the more advanced controls.

## 10.7 – List population

List population is generally one of those annoying things that have to be done and requires a fair amount of code each time population is necessary. StrataFrame lists have made short work of most list population circumstances. In this section we will discuss three controls: ComboBox, ListBox, and ListView. We will also discuss the three types of list population which are Manual, Enumeration, and Business Object.

### 10.7.1 – Population via enumeration

Enumerations are a very useful method of taming properties and fields that may represent more than a single value. The most recognizable benefit of enumerations is the ability to code using a readable reference rather than using a constant or being required to remember what a value of “1” actually represents. Since enums are commonly used, and may even be reflected on a business object as an extended strong-typed property, it only makes sense that a list should be able to populate from that same enumeration without writing any code.

There are two required properties when using an enumeration to populate a list control: PopulationType must be set to Enumeration and PopulationEnumName must be set with the desired enum type.

### 10.7.2 – Populating via business object

Another common population method is via a business object. By providing the business object type, method to use, and field properties, the list will do the rest and populate according to the specified properties. This is extremely nice since it allows complex populations to be made while still requiring no coding by the developer.

There are two required properties when using a business object to populate a list control: PopulationType must be set to BusinessObject and PopulationDataSourceSettings must be configured with the desired population settings.

### 10.7.3 – Requery()

The Requery() method exists on any StrataFrame list control that supports dynamic population. It allows you to repopulate the list control due to a parent record change or other event. This can be done in a single line of code by calling the Requery() method.

## **11 – Database Deployment Toolkit**

### **11.1 – Benefits of the Database Deployment Toolkit**

To understand the benefits of the DDT, it is first important to understand what it is and ultimately what it does for you. The DDT is a tool that allows SQL Server database schema to be defined using meta-data. Using the meta-data, those structures can then be deployed to an infinite number of SQL Server instances. In addition to schemas, stored procedures, CLR assemblies, static or initial data, advanced file groups, and views can be distributed as well.

Other major features of the DDT include the automatic creation of INSERT, UPDATE, and DELETE stored procedures with the check of a single box. The ultimate benefit of the DDT is that using a single package file SQL Server databases can have their structures automatically deployed and updated in a single line of code while never worrying about what version an end-user's database may be on.

### **11.2 – Concepts of meta-data**

Meta-data is basically data that represents data. The DDT stores all of the design time meta-data information in the SQL Server database named StrataFrame that is actually installed using the DDT itself (seems kind of strange, huh?). When a DDT profile is packaged, a single compressed file is created which is the only file required for deployment. All of the information pertaining to the meta-data is contained in this one file.

The file can be viewed using the Package-It! application that comes with StrataFrame. The package file is created using the compression tools within StrataFrame which is a proprietary “ZIP” file. The major benefit is that it requires no 3<sup>rd</sup> party DLLs and may be distributed freely without the requirement of royalties or licenses.

### **11.3 – Database Deployment Toolkit editor**

The Database Deployment Toolkit editor can be launched several ways, via the “All Programs” menu item in Windows or through the Visual Studio environment and the StrataFrame menu. This section will cover the basic use of the editor as we create a data structure that will represent the CRM system that will be created throughout the course.

#### **11.3.1 – Creating a new profile**

To get started, a profile needs to be created that will represent the collection of databases needed by your application. There are a number of settings that can be configured at the profile level.

#### **11.3.2 – Understanding the profile**

The profile is the top most entity of a DDT schema. This is important because it allows multiple databases to be deployed from within a single package. That is the primary reason for a profile rather than a database being at the top of the hierarchy.

### 11.3.3 – Importing existing structures

The DDT has the ability to import existing structures from SQL Server, Access, Visual FoxPro, or an exported DDT package file. This is extremely beneficial since allows existing structures to be brought forward but then updated via the DDT from that point forward.

### 11.3.4 – Packaging and deploying meta-data

When a profile is packaged, a single compressed file that represents the entire profile is created. The DDT comes with a deployment tool that allows a package file to be deployed to a server. In most commercial applications, developers will create their own deployment dialogs that are generally part of a setup process. In this case, we are going to deploy our CRM structure to SQL Server.

### 11.3.5 – Deployment data

In addition to schemas, initial or static data can be packaged up and distributed through the DDT as well. When a package is created, the deployment data definitions gather the data at that point which are stored in a binary format within the package file and installed with the schema. For example, data such as postal codes, countries, and states may be included so the end-user's database is pre-populated with the information it needs. Additionally, there are options to add new records and keep existing data in sync when updating a schema on future updates.

### 11.3.6 – Object name history

Every time a field, table, or database is renamed or deleted, this modification is stored. This allows existing fields to be renamed on existing deployments, regardless of the database deployment version, while maintaining the data within the column.

## 11.4 – Deploying packages programmatically

After the meta-data has been created and packaged up eventually the time comes where the application needs to be distributed to the end-users machine. This can be achieved programmatically using the DatabaseMigrator class. In fact this is the same class and tool used to deploy the StrataFrame and StrataFrameSample databases when installing StrataFrame.

Note: For code samples, refer to the DatabaseInstaller sample that comes with the framework.

### 11.4.1 – Using the pre-built cialogs

StrataFrame comes with pre-built dialogs which allow a package file to be selected and deployed to a specified server. These dialogs are the very same dialogs that are used within the Database Deployment Toolkit when deploying a package.

### 11.4.2 – Creating custom deployment dialogs

Custom dialogs can also be created using the DatabaseMigrator class and handling the events associated with the class. Also, deployment data and meta-data can be deployed separately or together depending on the needs of the developer. The DatabaseMigrator class provides much more in-depth and extensible functionality to the developer.

### 11.4.3 – Deploying using installer classes

Depending upon the installation techniques being used, one method may incorporate the use of installer classes and the Windows Installer. Using installer classes, .NET code can be used, including the DatabaseMigrator class, to create custom installation dialogs and deployment functionality.

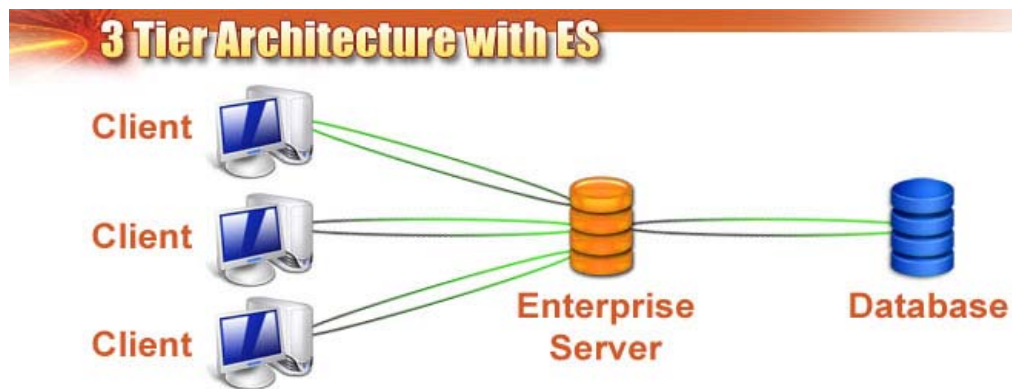
## 12 – Enterprise Server

### 12.1 – Overview & technical details

The StrataFrame Enterprise Server provides that method of connection to the data that can be easily implemented in an existing StrataFrame application without code changes, provides remote access to the database through HTTP, and increases the scalability of the application by reducing the load on the database server.

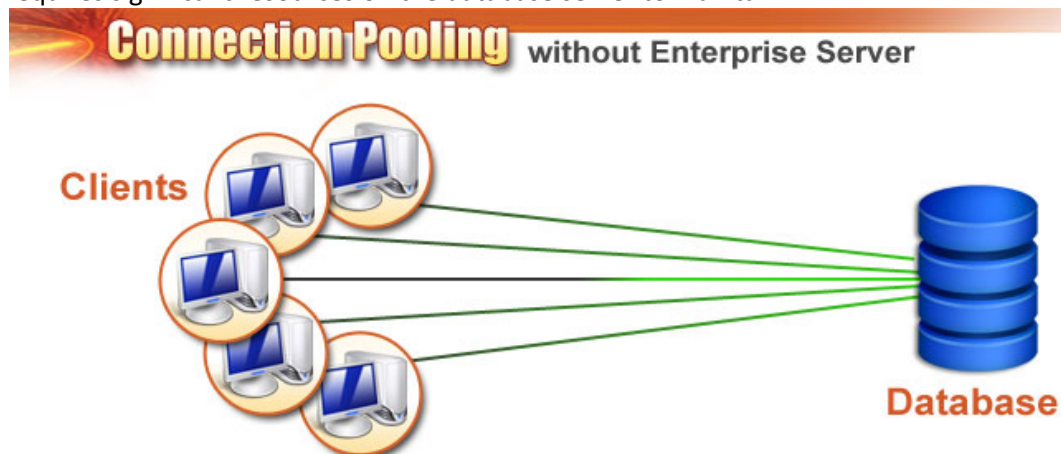
#### 12.1.1 – How it works

The Enterprise Server is a web site hosted within Microsoft® IIS that can be consumed as a data source by a StrataFrame application. Clients connect to the Enterprise Server using HTTP and the Enterprise Server proxies their database requests to the database server. The Enterprise Server allows clients to access the data within the database from a remote network through the use of the HTTP protocol, and uses encryption and compression to improve the security and performance to remote clients.



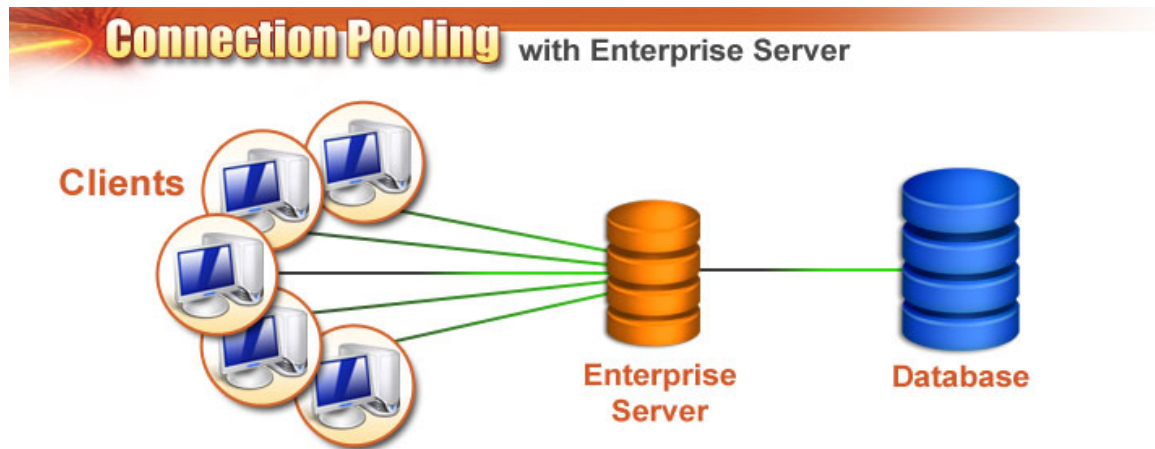
#### 12.1.2 – Benefits

**Connection Pooling** - The Enterprise Server (or Enterprise Servers in a server farm) provide a centralized location to pool connections to the database server. One of the larger drains on database server resources is connection maintenance. Even with .NET connection pooling, if clients are allowed to connect directly to the database, each client will open several connections and every connection requires significant resources on the database server to maintain.





When the Enterprise Server is incorporated, it becomes the only "client" of the database server (the only computer that opens connections to the database). This allows the ES to pool the connections from all of the client computers and greatly reduces the needed database server resources. The database server can spend more of its resources running queries and serving data rather than maintaining client connections.

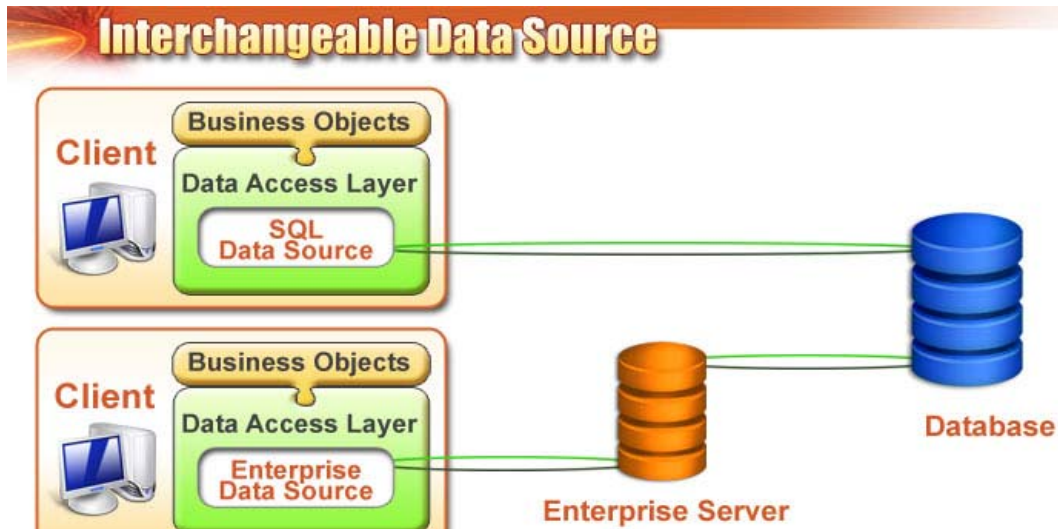


This functionality is the largest contributor to the increased application scalability that is gained from using the Enterprise Server. Additional Enterprise Servers can be added to handle client connections without greatly increasing the number of connections to the database.

**Encryption** - The Enterprise Server uses the symmetric 3DES encryption algorithm to encrypt all data that is transferred between the client computers and the Enterprise Server. 3DES is a universally accepted method for encrypting mission-critical data and is used commonly by secure VPN connections.

**Compression** - The Enterprise Server can optionally compress all data sent between the Enterprise Server and the client computers. Compression can reduce the amount of traffic between the client computers and the Enterprise Server by as much as 80%. The CPU resource cost of compressing the data is marginal, and when a client connects to the Enterprise Server using a low-bandwidth/high-latency connection the performance is greatly improved by compressing the communications.

**Interchangeable** - A StrataFrame application uses a transparent data source class for accessing data. The same methodology is used for the Enterprise Server; an Enterprise Server specific data source replaces the SQL Server (or other database) specific data source within the client application. These data sources are transparent to the upper levels of the application, and can be hot-swapped at runtime.



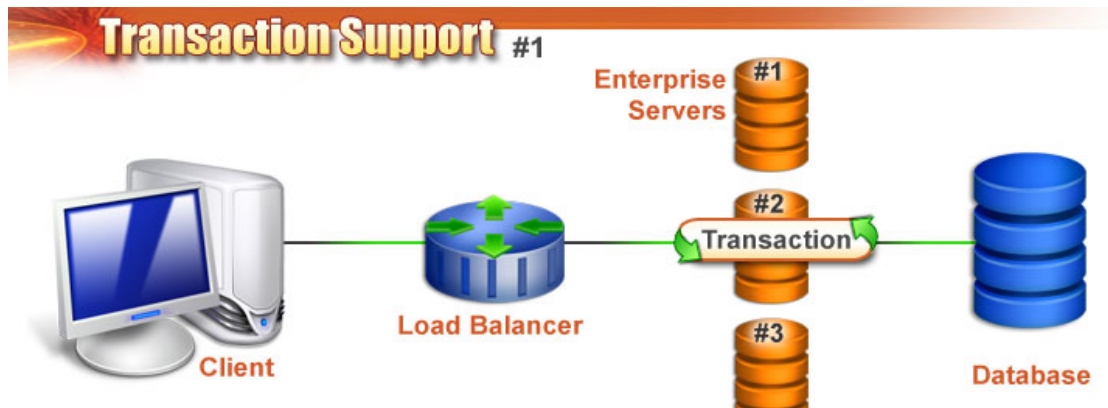
This functionality allows an application to be written for use with both the Enterprise Server and direct database connections. When the application is run from the local network, directly connecting to the database may be more efficient; however, when the application is run from a remote network or over the Internet, it can use the Enterprise Server to access the same database.

In fact, when testing the Enterprise Server, no new unit tests were developed; the same test used to test SQL Server connectivity were used to test the Enterprise Server connectivity by simply changing the data source.

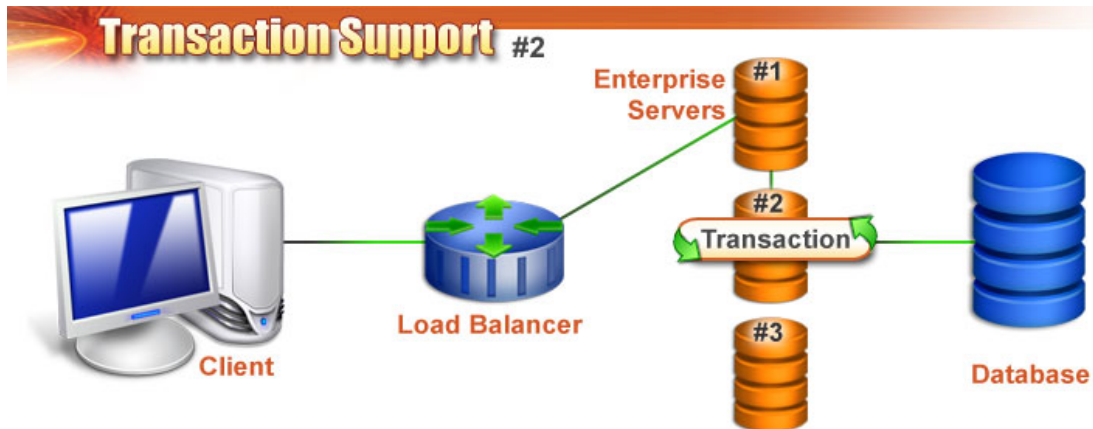
**Server farm support** - The Enterprise Server can be deployed in both single-server and multi-server environments. A multi-server environment increases the scalability of an application by distributing the load on the Enterprise Server across multiple computers. Any method of load balancing between the Enterprise Servers can be used such as Microsoft® Network Load Balancing (NLB) or a hardware round-robin request distribution system.

**Full transaction support** - Transactions are fully functional when using an Enterprise Server to provide access to the database, even when using several Enterprise Servers in a multi-server environment. Each of the Enterprise Servers in a farm is aware of the other servers. A transaction is maintained by a single Enterprise Server, and if an Enterprise Server receives a transaction request for a transaction on another Enterprise Server, it transparently forwards the request to the proper server for processing.

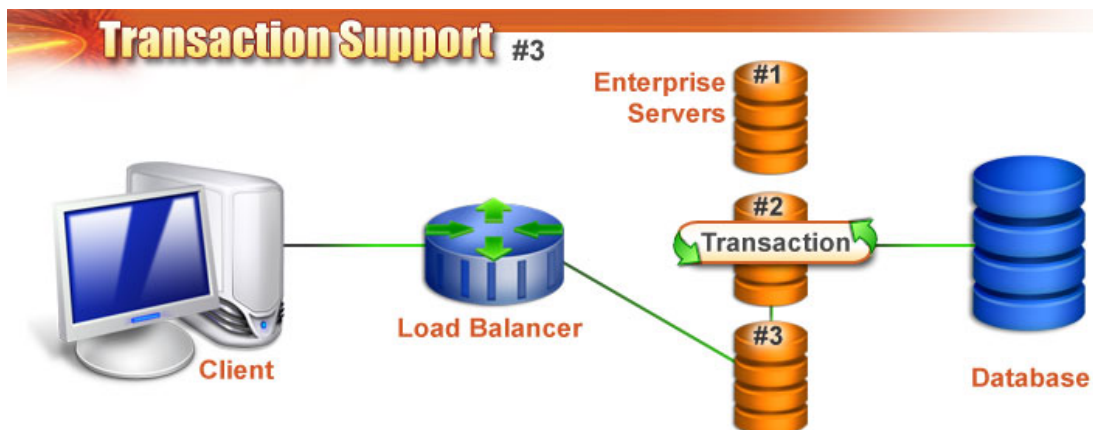
The client sends a Transaction Begin request to the Enterprise Servers which is sent to ES #2 by the load balancer.



The client sends a Save Record on Transaction request to the Enterprise Servers which is sent to ES #1 by the load balancer. ES #1 does not own the transaction, so the request is forwarded to ES #2 for the record to be saved on the transaction.



The client sends a Transaction Commit request to the Enterprise Servers which is sent to ES #3 by the load balancer. ES #3 does not own the transaction, so the request is forwarded to ES #2 to commit the transaction.



### 12.1.3 – "Drop-in" solution

#### 12.1.3.1 – Small client-side code change (1-6 lines)

Any existing StrataFrame application can enjoy the use of the Enterprise Server by simply changing the data source (1-6 lines of code) within the client application.

#### 12.1.3.2 – Install provided for Enterprise Server web site

A separate installation can be downloaded to install the Enterprise Server web site within IIS. The installation allows the Enterprise Server to be quickly and efficiently deployed and optimally configured on the server computer.

#### 12.1.3.3 – Existing applications can be easily adapted to use the ES

Since the Enterprise Server's client-side data source is interchangeable with all other StrataFrame data sources, an existing application written to connect directly to the database server can be adapted to use the Enterprise Server within minutes using the 1-6 lines of code required to change the data source.

### 12.1.4 – Design Considerations

The Enterprise Server does not use .NET serialization, .NET remoting, or web services. These 3 technologies are designed to be highly configurable, flexible, and interoperable; however, the flexibility gained by using one or more of these technologies comes with a significant performance penalty over using a custom solution. Developing the Enterprise Server would have been easier if we had used one or more of these technologies, but the increased development time was worth the performance gain in the end.

#### 12.1.4.1 – ES vs. .NET Serialization

By designing the Enterprise Server to work solely with a StrataFrame application, we were able to streamline the network communications. Where a serialized .NET object will contain a significant amount of meta-data used by the deserializer to reconstruct the object graph, the Enterprise Server's descriptor objects contain a 1-byte identifier that is used to instruct the remote endpoint how to reconstruct the object from the network stream. By not using .NET serialization, the required network traffic can be reduced by as much as 40%.

#### 12.1.4.2 – ES vs. .NET Remoting

The Enterprise Server uses a completely disconnected model for data access. After a client requests data from the server and that data is retrieved and returned, the connection between the client and server can be disconnected with no adverse effects. When using .NET remoting, the remoted object exists on the server application and an HTTP handle is used by the client to access the remote object. This requires a persistent connection between the server application and the client application since any and every access to a method or property of the remoted object requires a separate HTTP call from the client to the server.

Using .NET remoting over limited bandwidth or unstable network connections can cause an application to become unstable; if the connection drops, all remoted objects are lost. By not using .NET remoting,

the Enterprise Server gains a great deal of stability when working with clients that have limited bandwidth or unstable network connectivity since a persistent connection is not required.

#### 12.1.4.3 – ES vs. Web Services

Web services are designed to be highly interoperable so they can be consumed by numerous disparate systems. As with .NET serialization, every web service request and response object contains a great deal of meta-data, generally in the format of SOAP formatted XML tags. These tags can more than double the size of both the request and the response objects. With the Enterprise Server, this additional XML is not necessary since both endpoints know how to encode and decode the requests and responses.

#### 12.1.4.4 – HTTP vs. HTTPS (SSL)

The Enterprise Server is designed to work with the HTTP protocol. While the connection between the Enterprise Server and the client-side data source can be configured to use HTTPS, it runs more efficiently and just as securely on HTTP.

The Enterprise Server uses the symmetric 3DES encryption algorithm to encrypt all traffic between itself and the client-side data source. Symmetric encryption algorithms have much better encryption performance than asymmetric algorithms such as SSL, which is why symmetric algorithms such as 3DES and AES are used by VPNs. The Enterprise Server gains more throughput by using 3DES/HTTP over SSL/HTTPS.

Symmetric algorithms require a pre-shared key that must be configured manually on both sides of the connection. This increases security by creating a "password" system that is needed to access the Enterprise Server. Each data source handled by the Enterprise Server can have its own encryption key.

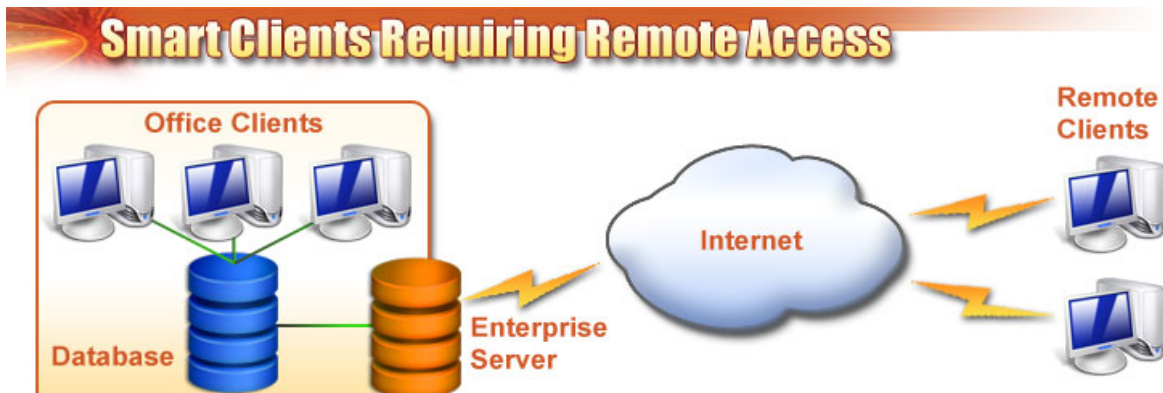
### 12.1.5 – Deployment Scenarios

#### 12.1.5.1 – Smart clients requiring remote access

In this age of mobile computing, quite often, systems will be designed to be used from both local and remote networks. Using the Enterprise Server, a StrataFrame application installed on a laptop can access data when it is connected to the local network and access the same data from a remote network through the Internet without requiring a recompile but rather a simple configuration change.

The Enterprise Server allows remote access to data using the HTTP protocol. It also provides excellent security for the data while increasing the performance for remotely connected clients. The Enterprise Server can be installed on a remotely accessible web server and exposed to the Internet rather than requiring that your database server be exposed to the Internet.

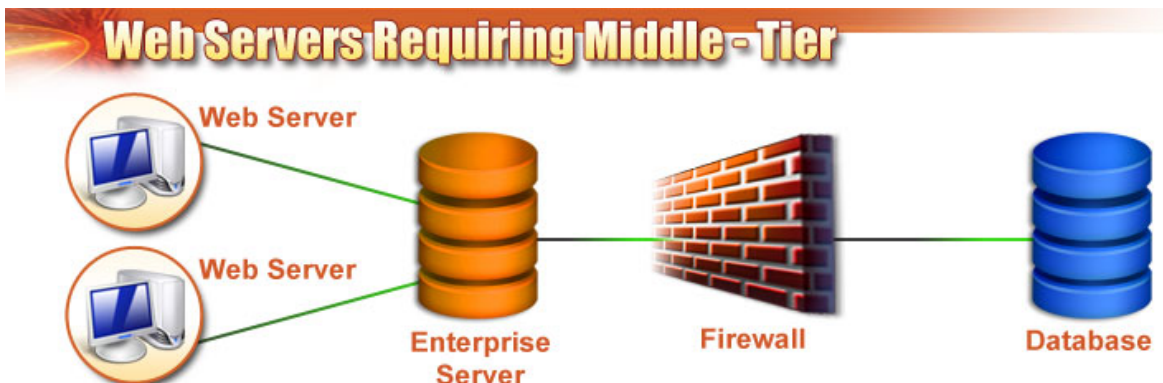
Clients within the office directly connect to the database server while clients outside of the office connect to the database over the Internet through the Enterprise Server.



#### 12.1.5.2 – Web servers requiring middle tier

The Enterprise Server can be used to provide a secure gateway to the database server for front-end web servers. All database access by the web servers would be routed through the Enterprise Server rather than directly to the database server. Access to the database server could then be restricted to only the Enterprise Server to increase security.

Both web servers connect to the database through the Enterprise Server. Only the Enterprise Server is allowed to communicate with the database through the firewall.

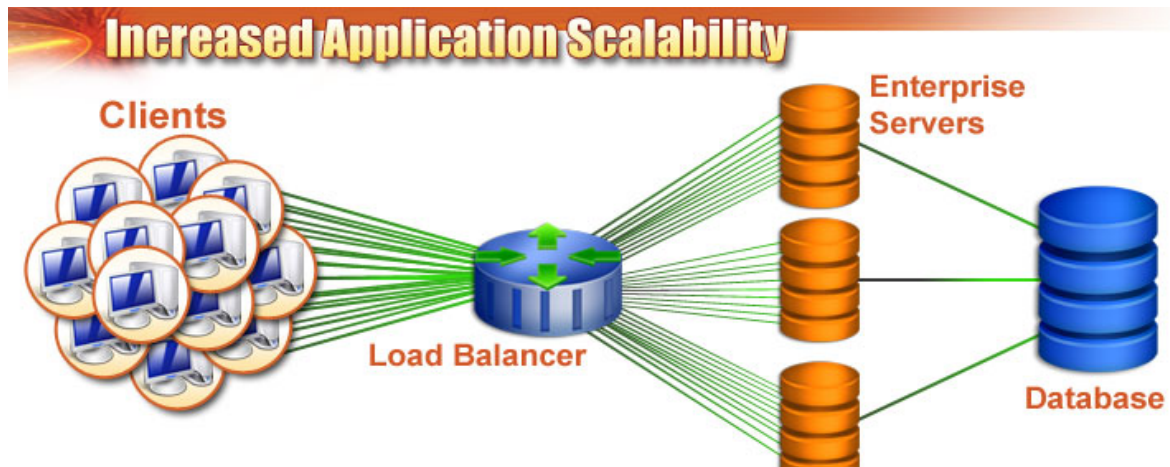


#### 12.1.5.3 – Increasing application scalability

The Enterprise Server can be deployed in a multi-server environment to increase the scalability of an application with a significant number of clients. The hardware resources needed by the database server would be greatly reduced as the number of database connections would be reduced to only the number needed by the Enterprise Servers.

A very large number of clients can use the same database server by allowing the Enterprise Server farm to reduce the load on the database server.





## 12.2 – Installing and configuring an Enterprise Server

### 12.2.1 – Running the install

The StrataFrame install includes an install for the Enterprise Server. Running the ES install on a target server will install the ES website under a new IIS web site or under a virtual directory within an existing IIS web site.

### 12.2.2 – Configuring the DataSources.config file

The DataSources.config file contains the information necessary to configure the DataSources that will be used by the ES. The ES uses the same MicroFour.StrataFrame.Data.Basics.DataSources collection as any other StrataFrame application, and that collection must be populated with the data sources that will be used by the ES.

Each data source is specified as a node within the <DataSources> root node. The following table lists the nodes and their uses:

Element	Description
<DataSources>	The root node of the document. Must contain 1 or more <DataSource> nodes.
<DataSource>	A node specifying a single data source.
DataSourceKey Attribute	Specifies the DataSourceKey for the data source. This key must be unique for each data source. This value is used by the EnterpriseDataSourceItem to specify the data source within the ES that should be used.
<ConnectionString>	Specifies the connection string for the data source. Remember when using Integrated Security that the IIS website is run under a specific user (by default ASPNET) and that user must be granted privileges on the database. Otherwise, explicitly specify the database user id and password for the connection string.
<TypeFullName>	The full name (including the namespace) of the type of DbDataSource item for this data source. The ES is not limited to the DbDataSourceItem classes included with StrataFrame but can be used with any class that inherits from

	MicroFour.StrataFrame.Data.DbDataSourceItem.
<TypeAssemblyName>	The name of the assembly containing the DbDataSourceItem class. This assembly should be located within the \bin folder of the IIS website. <b>Note:</b> This name is not the assembly's file name, but the assembly name and should not contain ".dll"
<TypeAssemblyVersion>	The version of the assembly containing the DbDataSourceItem class.
<TypeAssemblyCulture>	The culture of the assembly containing the DbDataSourceItem class (generally "neutral").
<TypeAssemblyPublicKey>	The public key token of the assembly containing the DbDataSourceItem class. <b>Note:</b> If the assembly is not strong-named (signed), then this value will be "null".
<IsEncrypted>	Must contain the value of either "True" or "False" to indicate whether traffic to and from the ES for this data source is encrypted using 3DES.
<IsCompressed>	Must contain the value of either "True" or "False" to indicate whether traffic to and from the ES for this data source is compressed using the Deflate algorithm.
<EncryptionKey>	If the <IsEncrypted> node is set to "True", this node specifies the 24-byte key for the 3DES encryption algorithm. The 24 byte values are specified in decimal format and separated by commas with no spaces.
<EncryptionVector>	If the <IsEncrypted> node is set to "True", this node specifies the 8-byte IV (initialization vector) for the 3DES encryption algorithm. The 8 byte values are specified in decimal format and separated by commas with no spaces.
<TransactionTimeout>	The <TransactionTimeout> node is used to specify the maximum time (in seconds) the data source will wait for a transaction to complete (either commit or roll back) before the server assumes that the client was disconnected and the transaction is hanging and should be dropped (rolled back).

### 12.2.3 – Configuring the web.config file

The web.config file contains the .NET configuration for the ASP.NET web application. The ES uses several application keys and values for configuration.

#### 12.2.3.1 – appSettings keys

Key	Value
StatusIpAddresses	A semi-colon separated list of IP addresses and/or IP address ranges that specify the hosts allowed to access the Status.aspx page. <i>(See Example Below)</i>
OtherEsServers	A semi-colon separated list of hostnames that specify the other Enterprise Servers that are part of this ES's server farm. This list is used to keep transactions centralized to the server that started them if clients can randomly connect to more than one server through load balancing. <i>(See Example Below)</i>



SmtpFromAddress	The e-mail address that will be used as the From: for all error e-mails sent from this ES.
SmtpToAddresses	A semi-colon separated list of e-mail addresses that specify the e-mail addresses to which all error e-mails will be sent.
SmtpServer	The SMTP server that can be used by the ES to send error e-mails.
SmtpPort	The TCP port number that the SMTP server uses.
SmtpAuth	A "True" or "False" specifying whether the ES must supply a username and password to the SMTP server to send e-mails.
SmtpUsername	Used when the SmtpAuth key is set to "True" to specify the username this ES will use to authenticate with the SMTP server.
SmtpPassword	Used when the SmtpAuth key is set to "True" to specify the password this ES will use to authenticate with the SMTP server.

### 12.2.3.2 – StatusIpAddress key examples

The value of the "value" attribute is a semi-colon separated list of IP addresses or IP address ranges that should be allowed access to the Status.aspx page. A single IP address is printed in dotted-decimal notation. An IP address range is specified as a network IP address in dotted-decimal notation, followed by a forward slash ("/") and a subnet mask in dotted-decimal notation.

Value	Result
127.0.0.1	
127.0.0.1;192.168.0.15	Only the localhost can access the Status.aspx page.
127.0.0.1;192.168.0.0/255.255.255.0	Only the localhost and the computer with the address 192.168.0.15 can access the Status.aspx page.
127.0.0.1;192.168.0.0/255.255.255.0; 192.168.16.16\255.255.255.240	The following computers can access the Status.aspx page: localhost (127.0.0.1) 192.168.0.0 – 192.168.0.255

### 12.2.3.3 – OtherEsServer key examples

The value of the "value" attribute is a semi-colon separated list of IP addresses and/or hostnames and port numbers that specify the other Enterprise Servers that are part of the server farm. This value is left blank if the ES is not part of a server farm.

Value	Result
192.168.0.1	Another ES exists and can be reached at 192.168.0.1
192.168.0.1:8080	Another ES exists and can be reached at 192.168.0.1 using TCP port 8080
192.168.0.1:8080;192.168.0.2:8080	Two other ES's exist: one can be reached at 192.168.0.1 using TCP port 8080 and the other can be reached at 192.168.0.2 using TCP port 8080.

### 12.2.4 – Installing the license file

To install a license file for the enterprise server, you must activate the enterprise server from the My Account area of the StrataFrame website, download the license file, and copy it to the bin\ folder of the ES web site.

### 12.2.5 – Configuring clients to use the enterprise server

The ES has a corresponding DbDataSourceItem that is used in place of other DbDataSourceItems to communicate with the ES. Configuring a client to use the ES requires that the DbDataSourceItem currently in use be replaced with an EnterpriseDataSourceItem from the MicroFour StrataFrame Enterprise Client.dll assembly.

#### Creating and configuring an EnterpriseDataSourceItem [C#]

```
using MicroFour.StrataFrame.Data.Enterprise;
...
private static void SetDataSources()
{
    //-- Create the data source
    EnterpriseDataSourceItem ds = new EnterpriseDataSourceItem("MyDataSourceKey",
        "es.domain.com", 80, "MyDataSourceKey",
        new SqlDataSourceItem("MyDataSourceKey"));

    //-- Configure the compression and encryption (optional)
    ds.IsCompressed = true;
    ds.IsEncrypted = true;
    ds.EncryptionKey = new byte[] {1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3,
4};
    ds.EncryptionVector = new byte[] {1, 2, 3, 4, 5, 6, 7, 8};

    //-- Add the EnterpriseDataSourceItem
    DataLayer.DataSources.Add(ds);
}
```

#### Creating and configuring an EnterpriseDataSourceItem [Visual Basic]

```
Imports MicroFour.StrataFrame.Data.Enterprise
...
Private Shared Sub SetDataSources()
    '-- Create the data source
    Dim ds As New EnterpriseDataSourceItem("MyDataSourceKey", _
        "es.domain.com", 80, "MyDataSourceKey", _
        New SqlDataSourceItem("MyDataSourceKey"))

    '-- Configure the compression and encryption (optional)
    ds.IsCompressed = True
    ds.IsEncrypted = True
    ds.EncryptionKey = New Byte() {1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4}
    ds.EncryptionVector = New Byte() {1, 2, 3, 4, 5, 6, 7, 8}

    '-- Add the EnterpriseDataSourceItem
    DataLayer.DataSources.Add(ds)
End Sub
```

## **13 – Web development**

### **13.1 – Differences between a StrataFrame WinForms application and a WebForms application**

#### **13.1.1 – AppMain.vb (Program.cs) vs. Global.asax**

The global.asax file performs the same function for an ASP.NET application as the AppMain.vb (Program.cs) does for a WinForms application. Rather than separating the logic for the application initialization between the SetDataSources() and InitApplication() methods, the global.asax file provides the Application\_Start() and Session\_Start() methods. The data sources are configured within the Application\_Start() method along with the rest of the application-level initialization.

#### **13.1.2 – Business objects within the project**

Business objects cannot be added directly to an ASP.NET web site, but must be contained within a class library that is referenced by the web site. The limitation on what component designers are allowed to run within the ASP.NET web site project limits the design-time functionality of business objects that are added directly to the App\_Code folder of the website.

Business objects can, however, be added directly to a Web Application Project. A Web Application Project more closely resembles a WinForms or Class Library project and allows the business objects to use the component designer. Additionally, the BOMapper can see business objects listed within a web project, but not a web site.

#### **13.1.3 – Business objects as bindable components**

Business objects cannot be dropped on web pages like they can be on WinForms forms. Instead, to use a business object for data binding, the business object must be declared and accessible as a public property on the ApplicationBasePage type inherited by the web page. Once the business object is declared, it becomes available for data-binding through the design-time type editors.

### **13.2 – Creating a WebForm project**

StrataFrame provides templates for creating StrataFrame ASP.NET applications, web pages, and ApplicationBasePages. Creating a new project or object is as simple as selecting the correct template.

### **13.3 – WebForms data binding**

Once a business object is declared as a public property within the ApplicationBasePage, it becomes available as a source for data binding for all web pages that inherit the ApplicationBasePage. Web controls bind to the name of the business object and the instance is retrieved at runtime. Data-binding on a web page can be one-way or two-way depending upon the setting specified on the control.

### **13.4 – Managing business object persistence**

Business objects declared through public properties on web pages are automatically persisted between postbacks of the page within session variables. This prevents the business object from needing to be recreated and repopulated each time the page posts back.

## 13.5 – StrataFrame web controls

### 13.5.1 – Bindable controls

The basic bindable web controls within StrataFrame include the TextBox, DropDownList, RadioButtonList, HiddenField, Button, and Label. The IWebBusinessBindable interface specifies the fields that are required for databinding to a web control and allows new web custom or 3<sup>rd</sup> party web controls to be bindable to StrataFrame business objects.

### 13.5.2 – List controls

Several list controls exist in the MicroFour.StrataFrame.UI.Web namespace that can be populated through design-time configuration similar to that of the WinForms list population. The BulletedList, CheckBoxLayout, DropDownList, ListBox, ListView, and RadioButtonList can all be populated through design-time configuration.

## **14 – Tips & tricks #3**

### **14.1 – Generics**

#### **14.1.1 – What are generics?**

Generics is the term used to describe a type or member that uses a placeholder to represent one or more types that will be stored or used by the generic. The placeholder's replacement is then defined when the generic type is used within code, and all uses of the placeholder are then automatically replaced by the replacement value. The most common use of generics within the .NET framework is the set of generic collection classes found within the System.Collections.Generic namespace.

#### **14.1.2 – Generic types**

A generic class is created by specifying type parameters in the class declaration. The type parameters of a generic type can be used throughout the type's definition.

When a generic type includes multiple partial declarations (partial classes), each partial declaration must use the same type parameter declarations.

##### **Generic class declaration [C#]**

```
//-- Defining a generic class
public class MyGenericClass<T>
{
    private T _MyValue;
    public T GetValue()
    {
        return this._MyValue;
    }
}
```

##### **Generic class declaration [Visual Basic]**

```
'-- Defining a generic class
Public Class MyGenericClass(Of T)

    Private _MyValue As T
    Public Function GetValue() As T
        Return Me._MyValue
    End Function

End Class
```

### 14.1.3 – Generic methods

A generic method is created by specifying the type parameters in the method declaration before the declaration of the method parameters. The type parameters of a generic method can only be used within the method.

#### Defining a generic method [C#]

```
public T ProcessInput<T>(T input)
{
    return input;
}
```

#### Defining a generic method [Visual Basic]

```
Public Function ProcessInput(Of T)(ByVal input As T) As T
    Return input
End Function
```

### 14.1.4 – Type constraints

Constraints can be applied to type parameters to restrict the types that can be used with the specific type parameter (i.e. the “struct” constraint on the `Nullable<>` generic type so that it can only be used with value types).

**class** - The “class” constraint specifies that the type used with that type parameter must be a class or reference type.

**struct** - The “struct” constraint specifies that the type used with the type parameter must be a structure or value type.

**new** - The “new” constraint specifies that the type used must have a public parameterless constructor so that the generic can create new instances of the type without passing parameters.

**Interfaces** – interfaces can be used as type constraints to specify that the type used as the type parameter must implement that interface.

**Types** – types can be used as type constraints to specify that the type used as the type parameter must be that type or inherit from that type.

**Multiple constraints** – Multiple constraints can be combined to require that the type used meets all of the supplied constraints.

**Generic classes with type constraints [C#]**

```
public class MyGenericClass<TKey, TValue>
    where TKey : struct
    where TValue : class, new()
{
}

public class MyGenericClass2<T>
    where T : IEnumerable
{
}
```

**Generic classes with type constraints [Visual Basic]**

```
Public Class MyGenericClass(Of TKey As Structure, TValue As {Class, New})

End Class

Public Class MyGenericClass2(Of T As IEnumerable)

End Class
```

**14.1.5 – Generic base classes**

Several advanced uses exist for generics. One of the advanced applications is the use of generic type parameters to allow a base class to create factory methods or properties that match the sub class.

**Sub class/base class with factory method in base class [C#]**

```
public class SubClass : BaseClass<SubClass>
{
    public SubClass() { }

    protected override void PopulateNew(object param)
    {
        //-- Save off the info from the create parameters
    }
}

public abstract class BaseClass<T>
    where T : BaseClass<T>, new()
{
    public static T FactorMethod_CreateNew(object param)
    {
        T ret = new T();
        ret.PopulateNew(param);
        return ret;
    }

    protected abstract void PopulateNew(object param);
}
```

**Sub class/base class with factory method in base class [Visual Basic]**

```

Public Class SubClass
    Inherits BaseClass(Of SubClass)

    Public Sub New()
    End Sub

    Protected Overrides Sub PopulateNew(ByVal param As Object)
        '-- Save off the creation data
    End Sub

End Class

Public MustInherit Class BaseClass(Of T As {BaseClass(Of T), New})

    Public Shared Function FactoryMethod_CreateNew(ByVal param As Object) As T
        Dim ret As New T()
        ret.PopulateNew(param)
        Return ret
    End Function

    Protected MustOverride Sub PopulateNew(ByVal param As Object)

End Class

```



## 14.2 – Static classes

Static classes are used to hold global variables for an application. Any class can have static members, but defining a class as static requires that the class contain only static members. Visual Basic does not have an equivalent to the static class keyword in C#, so when creating a static class in VB, you should declare the class as `NotInheritable` and declare a private constructor to prevent instantiation of the class. One of the most common examples of a static class is the `System.Math` class.

### 14.2.1 – Static members

Declaring a static (shared) member allows the member to be accessed from anywhere within code by referencing it through the class (typing the class name, not an instance of the class). Methods, properties and fields can all be declared static.

Using a static field is the most common way to declare a singleton. A singleton is a design pattern that specifies that only one of a certain type should exist within the application. Pseudo-singletons are used by all classes that define an “Empty” field that stores an empty instance of the class (`System.String.Empty`, `System.EventArgs.Empty`, `System.IntPtr.Zero`, etc.) And empty static field allows anyone that needs an empty reference of that class to access one quickly and without needing to instantiate another one.

#### Static class for global variables and methods [C#]

```
public static class General
{
    private static string _preference1;
    public static string Preference1
    {
        get { return _preference1; }
        set { _preference1 = value; }
    }
}
```

#### Static class for global variables and methods [Visual Basic]

```
Public NotInheritable Class General

    Private Sub New()
    End Sub

    Private Shared _preference1 As String
    Public Shared Property Preference1() As String
        Get
            Return _preference1
        End Get
        Set(ByVal value As String)
            _preference1 = value
        End Set
    End Property

End Class
```

### 14.2.2 – Nested classes

When using static classes, it is common to want to “nest” classes. For instance, you may want to create a Preferences class that contains system-wide preferences. In the Preferences class, you want to define another class called General for general preferences. The static members of the general class are then referenced by typing Preferences.General.Property. Each nested class can be placed within a separate partial class of the containing class to allow each one to be in its own code file.

#### Nesting static classes [C#]

```
public static class Preferences
{
    public static class General
    {
        private static string _preference1;
        public static string Preference1
        {
            get { return _preference1; }
            set { _preference1 = value; }
        }
    }
}
```

#### Nesting static classes [Visual Basic]

```
Public MustInherit Class Preferences

    Private Sub New()
    End Sub

    Public NotInheritable Class General

        Private Sub New()
        End Sub

        Private Shared _preference1 As String
        Public Shared Property Preference1() As String
            Get
                Return _preference1
            End Get
            Set(ByVal value As String)
                _preference1 = value
            End Set
        End Property

    End Class

End Class
```

### 14.2.3 – Modifying the project to nest files within the Solution Explorer

When adding nested and/or partial classes you often want to add them in their own separate code file from the main code file. Just as often, you want to group those separate code files together like Visual

Studio does with the .designer.\* files of components. You can manually nest code files by modifying the .csproj (or .vbproj) file directly.

The project file is an XML file that contains a list of all of the files in the project:

1. Open the project within Visual Studio.
2. Open the project file in a standard text editor.
3. Locate the <Compile> node for the file you want to nest beneath the main code file
4. Change the XML node so that it contains a child node for <DependentUpon> that lists the name of the main code file.
5. Save the project file.
6. Return to Visual Studio and choose “Reload” when the dialog appears stating that the project file has been changed outside of Visual Studio.
7. The code file will now be nested beneath the main code file.

When modifying a project file that is under source control, you will need to open the project in VS and check out the project file so that it will no longer be read-only on disk. When VS reloads the project after the changes, you can check the file back in to check-in your changes.

## **15 – Messaging & Localization**

### **15.1 – Overview**

StrataFrame has the ability to easily localize an application. Every control or presentation made to the end-user has the ability to be localized. In addition to localization is the proprietary messaging interface which replaces the standard Windows message box. This messaging interface supports rich-text formatting which allows messages to be much more attractive and descriptive to the end-user.

### **15.2 – Messaging and Localization editor**

The Messaging and Localization editor can be launched from within Visual Studio on the StrataFrame menu. This editor is where both messages and text values are created. Additionally, this same editor is used to migrate existing keys over to other languages.

#### **15.2.1 – Creating a messaging project**

When creating a new messaging project the output path and project languages will be defined. These settings can be modified and changed at any time. Additionally, the project can either include or exclude the Common Repository items.

#### **15.2.2 – Creating message items**

A message item is the full rich text formatted message that will be displayed through the StrataFrame messaging interface. This includes which icons will be used, the buttons that will be displayed, the sound that will be played, and more.

#### **15.2.3 – Creating text values**

Text values represent any piece of information, excluding message items, which require localization. This includes broken rule messages, label or other control text, or any other information that may need to be displayed to an end-user.

Text values can also be created directly from a form designer within the Visual Studio IDE. This will be discussed on the next section.

#### **15.2.4 – Exporting the localization messages**

There are three places where the localized messages can be stored and retrieved within a StrataFrame application. The messages can be pulled from XML files on disk, from a SQL Server database, or from XML files embedded within the application. For most applications, embedding the XML files within the application is the fastest and safest method. For debugging, it is often recommended to leave the XML files on disk so that they do not have to be re-embedded each time you make a change to a message. For several internal applications, SQL Server can be the most useful place to store the messages because the messages can be modified in place without requiring the application to be redistributed.

### **15.3 – IDE localization integration**

All controls that have the StrataFrame ILocalizable interface implemented will have a control designer that allows text values to be assigned, searched, and created directly from the control. This saves a

phenomenal amount of energy on the part of the developer because each key does not have to be manually created through the Messaging and Localization editor.

## 15.4 – The MessageForm class

The actual StrataFrame class that is used to replace the standard Windows message box can be referenced through the MessageForm class using shared methods. Like the standard message box, it can be called directly and totally side-step localization altogether.

### **MessageForm Example [C#]**

```
MicroFour.StrataFrame.Messaging.MessageForm.ShowMessage("Hello World!");
```

### **MessageForm Example [Visual Basic]**

```
MicroFour.StrataFrame.Messaging.MessageForm.ShowMessage("Hello World!")
```

## 15.5 – Showing a message using a localization key

There are several ways to instruct a message to be displayed using a localization key. In this section we will discuss two of the most common methods, but there are many properties throughout the framework that allow a message key to be specified for particular events or actions.

### 15.4.1 – Directly calling ShowMessageByKey()

The Localization class has a method called ShowMessageByKey which accepts a localization key and pulls the message information from the database.

### **ShowMessageByKey() Example [C#]**

```
MicroFour.StrataFrame.Messaging.MessageForm.ShowMessageByKey("MyKey");
```

### **ShowMessageByKey() Example [Visual Basic]**

```
MicroFour.StrataFrame.Messaging.MessageForm.ShowMessageByKey("MyKey")
```

### 15.4.2 – ShowMessageByKey() method on StrataFrame forms

All StrataFrame forms have an exposed method called ShowMessageByKey() which has the exact same functionality as the direct method. By exposing this method through a form code is reduced and the MessageForm is made more accessible.

### **Form ShowMessageByKey() Example [C#]**

```
this.ShowMessageByKey("MyKey");
```

### **Form ShowMessageByKey() Example [Visual Basic]**

```
Me.ShowMessageByKey("MyKey")
```

## 15.6 – The Localization class

The Localization class is the StrataFrame class that contains all pertinent functionality needed by a developer. There are a number of properties, some of which are already setup within the AppMain.vb or Program.cs files. This is the class that will be used if a text value needs to be manually retrieved or to setup a property that determines from where the keys will be pulled.

### 15.5.1 – GetActiveLanguage()

This method returns the locale ID (language) currently active for a specified application key. By default, this method is only needed in one location, the Appmain.vb or Program.cs file within the InitApplication() method. If an active language cannot be found for an application and more than one language exists within the data, a window will automatically be presented to the end-user so that they may choose their language of choice.

### 15.5.2 – Message: DataSourceKey, KeyType, XMLPath, and LocaleID properties

Each of these properties are generally set within the InitApplication() method of the Appmain.vb or Program.cs file. These properties determine how the localization messages will be pulled from the database and what language will be used within the current application session. These properties can be ignored if you are embedding the XML files within the application.

### 15.5.3 – RetrieveMessage()

This method allows a message to be programmatically retrieved from the localization database. Once retrieved, the message can be used however the developer deems necessary. It is important to note that this method does not display the message to the end-user and only retrieves the message from the database.

### 15.5.4 – RetrieveTextValue()

There are times when a text value may need to be manually retrieved. When this need arises, this method will provide that functionality. This is the same method called automatically by any control that implements the ILocalizable interface.

#### RetrieveTextValue Example [C#]

```
MyLabel.Text = MicroFour.StrataFrame.UI.Localization.RetrieveTextValue("MyKey");
```

#### RetreiveTextValue Example [Visual Basic]

```
MyLabel.Text = MicroFour.StrataFrame.UI.Localization.RetrieveTextValue("MyKey")
```

### 15.5.5 – ShowAvailableLanguages()

This method presents the end-user with all of the available languages to choose from for a particular application. The language selected within this dialog becomes the active language for the end-user. Most applications that support multiple languages create a menu item that calls this method. Once selected, the languages will then be pulled using the new language values.

## 16 – Business layer advanced topics

### 16.1 – Adding custom field properties

You can define custom field properties on business objects that can be bound to controls in the same manner as the strong typed properties generated by the BOMapper. The simplest method of creating a new custom property is to locate a suitable property within the partial class and copy it; then paste it at the location within the main code file where you want the property to reside and modify it to create a new property. You will most likely need to add some namespace imports to the file for the property attributes.

After creating the custom property, you must specify a `FieldPropertyDescriptor` for the property that will be used during data-binding. The `GetCustomBindablePropertyDescriptor()` method must be overridden to specify the property descriptor(s). For most custom properties, you can use the `ReflectionPropertyDescriptor` class, but you will generally want to create a custom property descriptor for the new property by copying a property descriptor from the designer file and pasting it into the main code file and modifying it to match the new property.

#### GetCustomBindablePropertyDescriptor() implementations [C#]

```
using MicroFour.StrataFrame.Business;
...
protected override FieldPropertyDescriptor[] GetCustomBindablePropertyDescriptor()
{
    return new FieldPropertyDescriptor[] {
        new ReflectionPropertyDescriptor("cust_FullName", typeof(CustomersBO));
    }
}

protected override FieldPropertyDescriptor[] GetCustomBindablePropertyDescriptor()
{
    return new FieldPropertyDescriptor[] { new Field_cust_FullName_Descriptor() };
}
```

#### GetCustomBindablePropertyDescriptor() implementations [Visual Basic]

```
Imports MicroFour.StrataFrame.Business
...
Protected Overrides Function GetCustomBindablePropertyDescriptor() As FieldPropertyDescriptor()
    Return New FieldPropertyDescriptor() { _
        New ReflectionPropertyDescriptor("cust_FullName", GetType(CustomersBO))
    }
End Function

Protected Overrides Function GetCustomBindablePropertyDescriptor() As FieldPropertyDescriptor()
    Return New FieldPropertyDescriptor() { _
        New Field_cust_FullName_Descriptor()
    }
End Function
```

### 16.2 – Advanced BOMapper customizations

The Business Object Mapper provides the ability to customize the fields within a business object to perform some advanced tasks.

### 16.2.1 – Encrypting/decrypting fields

A field can be configured to encrypt data into and out of the field within the database by customizing the code within the field property and configuring the property to utilize the encryption wrappers within the MicroFour.StrataFrame.Security.Cryptography namespace.

The \_3DES object within the following samples is an instance of the TripleDesWrapper class within the StrataFrame framework.

#### Encrypted Business Object Field [C#]

```

/// <summary>
/// Credit Card Number
/// </summary>
/// <remarks></remarks>
[Description("Credit Card Number"),
DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden),
Browsable(False), BusinessFieldDisplayInEditor()]
public string cc_cardnumber
{
    get
    {
        //-- Establish Locals
        string lcValue;

        if (CurrentRow.RowState == DataRowState.Deleted)
        { lcValue = ((string)CurrentRow["cc_cardnumber", DataRowVersion.Original]) }
        else
        { lcValue = ((string)CurrentRow["cc_cardnumber"]) }

        if (lcValue.Length == 0)
        { return lcValue; }
        else
        { return _3DES.Decrypt(lcValue); }
    }
    set
    { CurrentRow["cc_cardnumber"] = _3DES.Encrypt(value); }
}

```



**Encrypted Business Object Field [Visual Basic]**

```

''' <summary>
''' Credit Card Number
''' </summary>
''' <remarks></remarks>
<Description("Credit Card Number"), _
DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden), _
Browsable(False), BusinessFieldDisplayInEditor())> _
Public Property cc_cardnumber() As System.String
    Get
        '-- Establish Locals
        Dim lcValue As String

        If (CurrentRow.RowState = DataRowState.Deleted) Then
            lcValue = CType(CurrentRow.Item("cc_cardnumber", _
                DataRowVersion.Original), System.String)
        Else
            lcValue = CType(CurrentRow.Item("cc_cardnumber"), System.String)
        End If

        If lcValue.Length = 0 Then
            Return lcValue
        Else
            Return _3DES.Decrypt(lcValue)
        End If
    End Get
    Set(ByVal value As System.String)
        CurrentRow.Item("cc_cardnumber") = _3DES.Encrypt(value)
    End Set
End Property

```

**16.2.2 – Serializing fields**

Fields can be customized to serialize objects into and out of the underlying database field. Serializing a field does not require you to enter custom code, but simply requires that the field be marked as serializable within the Business Object Mapper.

The underlying data type for a serialized field is assumed to be a variable binary data type (VarBinary(), VarBinary(MAX), Image, Blob, etc.) that maps to the System.Byte[] data type within .NET. A binary formatter is used to serialize and deserialize the data into and out of the field.

Each time the field property is accessed, a new object of the return type is deserialized and returned (creating clones of the object); so, each time the property is accessed, a different object reference will be returned.

**16.2.3 – NULL value support**

Fields can be configured to handle NULL database values through the Business Object Mapper by customizing the field and configuring it to return an alternate value when a NULL value is encountered or to return a Nullable<> instance as the property value.

The “Return Alternate on NULL / Set NULL on Alternate” value was added to the null value support options within the field customizations of the BOMapper. The basic Return Alternate on NULL option will return a user specified alternate value when a DBNull.Value is encountered within Business Object’s field. The Set NULL on Alternate tests the value of the setter to see if it matches the alternate value; if it does, then a DBNull.Value is placed into the DataTable rather than the alternate value itself.

#### 16.2.4 – Using the customization wizard

The Customization Wizard has been provided to allow you to customize several fields at the same time within the Business Object Mapper. To customize several fields at once, open the Customization Wizard, select the fields you want to edit, select the customizations you want to apply, and click Apply.

#### 16.2.5 – Field-level events

Field-level events within business objects can be turned on to add Accessing, Changing and Changed events to the fields within the Business Object Mapper. Each of the 3 event types can be turned on individually and can be configured to raise a single event for all of the fields or to declare and raise a unique event for each field.

### 16.3 – Many-to-many relationships

Business objects only directly support a single parent relationship. However, for link tables that are used to configure a many-to-many relationship between two tables, you can use one of the following methods:

#### 16.3.1 – Manual relationship maintenance

Manual relationship maintenance requires that one of the relationships between the link table and one of the accompanying tables be defined. The business object will maintain the foreign key between the two objects. The additional foreign key must then be set manually each time a new link table record is created.

#### 16.3.2 – BusinessLayerLinkManager

The BusinessLayerLinkManger is a component that is used to maintain the relationships between the 3 tables that are part of a many-to-many relationship. When properly configured, automatically set the foreign key properties of the link table, as well as allow you to filter or populate any two tables within the relationship from the data contained within the 3<sup>rd</sup> table.

### 16.4 – Using a BusinessBindingSource

The BusinessBindingSource provides an interface to allow a single business object to function as a list of business objects for data binding to a grid or control that accepts an IBindingListView as its data source.

A BusinessBindingSource is a component that can dropped on a form. The BBS is then configured to attach to a business object and provide the proper list formatting when used as a data source. The BBS does not bind directly through to the CurrentDataTable, but rather utilizes the property descriptors of the business object to allow binding to the strong-typed field properties.

## 16.5 – Shared DataTables

When a ChildFormDialog or BusinessObjectTranslations are not feasible within a scenario, business objects can share their data tables. When business objects share a DataTable, all business objects with the same DataSourceKey use the same object reference for their CurrentDataTable. When one of the business objects repopulates the internal DataTable, the CurrentDataTableRefilled event is raised on all business objects to inform them that their CurrentDataTable has changed.

A business object can share its internal DataTable by using the same SharedDataTableKey as the other business object, or through the ShareCurrentDataTable() method. The ShareCurentDataTable() method uses a new GUID as the SharedDataTableKey to ensure that the key is unique.

## 16.6 – Appending, copying, and merging DataTables

Advanced DataTable manipulation methods exist that allow you to merge data between two business objects, copy data from one business object into another business object, or append the records from a database query to the current records within the business object.

### 16.6.1 – AppendDataTable()

The AppendDataTable() method accepts the same parameters as the FillDataTable() method, but unlike the FillDataTable() method, AppendDataTable() does not clear the CurrentDataTable before populating the business object with the results. Rather, it appends the results to the CurrentDataTable. A parameter within the method call specifies how to resolve conflicts between records with the same primary key (either leave the old records within the business object or replace them).

### 16.6.2 – CopyDataTable()

The CopyDataTable() method is used to copy the CurrentDataTable from one business object to another. The CopyDataTable() method does not share the internal DataTable between the two business objects, but rather creates a complete copy of the data. The BusinessCloneDataType parameter specifies how the data will be copied. NOTE: When the CopyDataTable() method is used with an Append clone data type, conflicting records are not resolved (the business object will not test to see if the record already exists within it before copying the new record).

### 16.6.3 – MergeDataTable()

The MergeDataTable() method is very similar to the CopyDataTable() method, except that it will check to ensure that duplicate rows do not exist after the copy of the data. A parameter specifies whether the old records within the business object will be kept or overwritten with the new records.

## 16.7 – Business object serialization

Business objects implement the ISerializable interface and can be serialized using any serialization formatter (SOAP, Binary, etc.). While .NET remoting and web services will automatically serialize objects when necessary, you can explicitly serialize a business object through the methods on the BusinessLayer class.

### **16.7.1 – SerializeToByteArray()**

Returns a new System.Byte[] containing the business object serialized through a BinaryFormatter.

### **16.7.2 – SerializeToStream()**

Accepts a System.IO.Stream object the serialized content of the business object it written to. This method also uses a BinaryFormatter object.

### **16.7.3 – DeserializeBusinessObject()**

This method is a shared/static method that accepts either a System.Byte[] or a System.IO.Stream containing the serialized content of a business object to deserialize. It returns a new instance of the business object contained within the data.

## 17 – Data access advanced topics

### 17.1 – Transaction processing

Transaction processing is provided through the transaction implementation in the DbDataSourceItem classes. Wrapper methods exist on the BusinessLayer class that allow you to start one or more transactions. After a transaction is started, a business object can be saved on the transaction by using the overloads of the Save() method.

An unlimited number of transactions can be started, but each separate transaction on a data source must have a unique key. Omitting the transaction key will cause the DbDataSourceItem class to use the default transaction key of "DEF\_TRANS\_KEY".

#### Transaction Processing [C#]

```
public void SaveOnTransaction()
{
    //-- Start the transaction
    MicroFour.StrataFrame.Business.BusinessLayer.TransactionBegin("", "MyTranKey",
        IsolationLevel.ReadCommitted);

    //-- Start the try block so that we can roolback
    // the transaction if an error is encountered
    try
    {
        //-- Save the business object on the transaction
        this.businessObject11.Save(true, "MyTranKey");

        //-- Commit the transaction
        MicroFour.StrataFrame.Business.BusinessLayer.TransactionCommit("", "MyTranKey");
    }
    catch
    {
        //-- Rollback the transaction
        MicroFour.StrataFrame.Business.BusinessLayer.TransactionRollback("", "MyTranKey");
    }
}
```

**Transaction Processing [Visual Basic]**

```

Public Sub SaveOnTransaction()
    '-- Start the transaction
    MicroFour.StrataFrame.Business.BusinessLayer.TransactionBegin("", "MyTranKey", _
        IsolationLevel.ReadCommitted)

    '-- Start the try block so that we can roolback
    ' the transaction if an error is encountered
    Try
        '-- Save the business object on the transaction
        Me.BusinessObject11.Save(True, "MyTranKey")

        '-- Commit the transaction
        MicroFour.StrataFrame.Business.BusinessLayer.TransactionCommit("", "MyTranKey")
    Catch
        '-- Rollback the transaction
        MicroFour.StrataFrame.Business.BusinessLayer.TransactionRollback("", "MyTranKey")
    End Try
End Sub

```

**17.2 – Handling concurrency exceptions**

Handling a concurrency exception depends upon how the business object is configured to bubble the concurrency exceptions. The `CollisionNotificationType` property determines how a concurrency exception will be given to the developer. The possible options are: 1) `RaiseEvent` – the business object will raise the `ConcurrencyException` event containing the information about the collision. 2) `ThrowException` – the business object will throw a `ConcurrencyExcpetion` object containing the information about the concurrency exception.

Other properties that govern the handling of concurrency exceptions are: 1) `GetServerValuesOnConcurrencyException` – when this property is true and the `CollisionNotificationType` property is set to `ThrowException`, the business object will retrieve the current values of the record from the server and provide them in the exception data. If the `CollisionNotificationType` is `RaiseEvent`, the business object will always retrieve the row from the server in order to compile a `RowCollision` collection containing the fields that collide on SQL Server. 2) `ErrorSavingMode` – This property pertains to the handling of concurrency exceptions because if the `CollisionNotificationType` is set to `ThrowException` and the `ErrorSavingMode` is set to `FailOnError`, all of the records after the record that caused the concurrency exception will not be saved.

A `CollisionNotificationType` of `FailOnError` forces the developer to manually call `Save()` on the business object after the error has been resolved in order to save the offending record. The `ConcurrencyException` event is raised during the save process to allow the developer/end-user to handle the concurrency exception and resave the record while still in the same save process.

The `StandardForm` class can automatically handle the `ConcurrencyException` event of business objects to present a Data Collision Dialog to the end-user and allow them to pick which values to save to the server.

**Handling the ConcurrencyException Event [C#]**

```
private void businessObject11_ConcurrencyException(ConcurrencyExceptionEventArgs e)
{
    //-- Cycle through the record and select the server values
    foreach (FieldCollision field in e.Collision.FieldCollisions)
    { field.AcceptServerData = true; }

    //-- Force the record to re-save and commit the selected data
    e.ResaveRecord = true;
}
```

**Handling the ConcurrencyException Event [Visual Basic]**

```
Private Sub BusinessObject11_ConcurrencyException(ConcurrencyExceptionEventArgs e) _
    Handles BusinessObject11.ConcurrencyException

    '-- Cycle through the record and select the server values
    For Each field As FieldCollision In e.Collision.FieldCollisions
        field.AcceptServerData = True
    Next

    '-- Force the record to re-save and commit the selected data
    e.ResaveRecord = True
End Sub
```

## 17.3 – Debugging a DataSource

The DbDataSourceItem class contains the functionality to examine every command executed through the class and create a debug file containing the collected information. The SetDebugOn() method allows you to configure a data source to collect debug information while the SetDebugOff() method turns off debugging on the data source. The IsDebugModeOn property returns a value that indicates whether the data source is debugging.

The debug output file is in HTML format and contains the following information on every command executed while collecting debug information:

- The number of the command executed
- The timestamp of the executed command
- The CommandType and CommandText
- The object type of the command
- The connection string used to execute the command
- True/False to indicate whether the command was executed on a transaction
- Parameter list containing all parameters contained within the command, including their name, value, size, DbType, Direction, SourceColumn, and SourceVersion

## 17.4 – The ConnectionManager class

The ConnectionManager class provides access to the connection string management within the StrataFrame application framework.

### 17.4.1 – Changing the DataSources collection at runtime

The DataSources collection can be changed at runtime, either by manually setting the DataSources, or by calling the ShowAvailableConnections() method on the ConnectionManager class. Calling the ShowAvailableConnections() method shows the Available Connections dialog for the application. Within the Available Connections Dialog, the end-user can edit, add, or delete connections and select the one that will be the currently active connection within the application.

When the DataSources collection changes, you do NOT need to restart the application. All references to the data sources within the business objects will be updated to the new data source without needing to recreate any objects.

### 17.4.2 – Understanding the Shared Settings File

A Shared Settings File is a configuration file stored on a shared location on the network that can store the configuration information used by the ConnectionManager for the application. The file provides a centralized location for the connection data to be stored and shared between multiple workstations that utilize the same connection string. Each workstation can override the settings specified within the Shared Settings File in case the workstation needs to utilize a different connection string than the rest of the workstations.

You can create a new Shared Settings File by using the ConnectionManager.ShowSharedSettingsWizard() method to show the wizard used to create and edit shared settings files. Once the file is created, the standard Connection String Wizard can be used to point the connection information to the shared settings file.



## **18 – Presentation layer advanced topics**

### **18.1 – InitializationPriority**

Many objects within StrataFrame, including business objects and most controls, have the ability to order their initialization. This can be used to avoid problems when one object relies on another for initialization. Objects with a lower InitializationPriority are initialized first. This does not control the instantiation order of the objects, but controls the order in which the ParentFormLoading events and InitializeObject() methods are raised/called on the respective objects.

### **18.2 – Advanced list population**

When populating lists using the BusinessObject method, an instance of the business object type specified within the PopulationDataSource settings gets created. At times, a business object of the same type has already been populated with the desired data, so to resolve a server trip and populate using the existing business object, a list can be populated using the CopyDataFrom() method. All business objects automatically inherit this method which allows another business objects data to be copied. Additionally, there is an overload which allows a standard ADO.NET data table to be copied into a business object.

### **18.3 – Exclusive StrataFrame controls**

StrataFrame provides a number of controls past the standard .NET control collection. Some of these controls, such as the BrowseDialog, quickly and easily extend the ability of an application. Other controls provide extended functionality that allows the developer to be more creative when designing and building forms. Some of these include the ThemedGroupBox, ThemedDetailWindow, and ThemedContainer.

The overall look and feel of the Themed\* controls can be controlled globally through application-level theme supported provided by the DefaultApplicationTheme component. The DefaultApplicationTheme component should be placed on the main form of the application and globally controls all Themed\* controls that are configured to respect the application-level themes.

#### **18.3.1 – BrowseDialog**

The browse dialog allows developers to build search forms dynamically simply by setting properties. Even better is that the searches are dynamically created by the control and do not require any pre-built methods on the business object. This object will be discussed in more detail later in the session.

#### **18.3.2 – ChildDialog**

The child dialog controls provides the ability to share data sessions between multiple forms. If a child form needs to be displayed while accessing the already populated business objects on the parent form, then this control is the answer. A collection is provided that allows the developer the have the business objects of one form automatically translated (events and all references, even in code) to the business object on the other.

### 18.3.3 – ColorizedTextBox

This control is ultimately a radically enhanced rich text control. By defining keywords, rules, and colors through properties on the control, these keywords will be automatically colorized to match those definitions at run-time when a defined keyword appears within the text. This is akin to the text editor within a programming language that colorizes the keywords.

### 18.3.4 – FontComboBox and FontSizeComboBox

These two controls are just as their names describe them. They were created for use within the rich text toolstrip control for font selections. The FontComboBox presents all of the fonts installed on the machine. When dropped down the name of the font is drawn in its own style. Additionally, the FontSizeComboBox shows the supported font sizes for a selected font.

### 18.3.5 – GradientFormHeader

This is one of the most commonly used StrataFrame controls. By default, the maintenance form template already has this control within it. The GradientFormHeader provides a very aesthetically pleasing appearance to any form or user control and can be configured to follow the Windows themes or controlled by the developer. In addition to the colorization of the control, a title, detail line, and image can all be set to appear within the control. This control has the ability to follow the Windows Themes.

### 18.3.6 – MaintenanceFormToolstrip

This toolstrip is specially designed to be dropped on a form and “magically” work with any business objects that are associated with that form, including the adding, editing, deleting, and navigation of records. This is one of the most commonly used controls within many applications since it has the ability to aide the developer in record manipulation.

### 18.3.7 – RadioButtonGroup

By default, .NET does not provide the ability to take multiple radio buttons and bind them to a single value. StrataFrame has created a radio group control that allows a group of radio buttons to be bound to a single data value.

### 18.3.8 – RegistryRepository

The registry repository control makes short work of interacting with the registry. By simply providing a “root key” the control will use this as a base to read, create, and update registry values in very few lines of code.

### 18.3.9 – RichTextToolstrip

The rich text toolstrip control provides all of the necessary functionality required by a rich text control by simply dropping in on a form. One rich text toolstrip can control an infinite number rich text controls. Functionality such as bold, font selection, image imports, and paragraph manipulation exists within this control.

### 18.3.10 – ThemedContainer

The themed container control behaves very much like the Explorer navigation controls used within Windows. This is a great tool for creating menus that need the ability to grow and shrink vertically. But the control is not limited to a menu type of functionality. Since the control is a container, it can house any type of control which then gains the ability to grow and shrink vertically based on the expanded state of the control. This control even has the ability to control the state of the containing form by using the minimization and close features. This control has the ability to follow the Windows Themes.

### 18.3.11 – ThemedDetailWindow

This control is for display purposes only and allows an easy way for read-only data to be presented to an end-user. The detail window has an items collection that can be referenced by key to dynamically set any value within the control. This control has the ability to follow the Windows Themes.

### 18.3.12 – ThemedGroupBox

This is a very commonly used control as it replaces the standard group box provided by .NET. Though it can be made to look identical to the standard group box, it has much better docking support as well not to mention the most obvious visual enhancements. The heading can be placed in a tab style appearance or as a blocked header. Additionally, the body of the control supports gradient backgrounds. This control has the ability to follow the Windows Themes.

### 18.3.13 – WaitWindow

This control evolved from a useful feature within Visual FoxPro. The StrataFrame wait window provides much more functionality and supports internal threading. Additionally, the control can be forced to show within the active screen (determined by the location of the mouse if multiple monitors are used) or within a parent container such as a form. The purpose of this control is allow a title and message to be dynamically displayed to an end-user without requiring the developer to create a custom display form to state the progress of any type of information. This wait window dialog dynamically resizes itself based on the size of the message and follows the Windows Themes.

## 18.4 – Application themes

Application Themes provide a facility for an application to have a more aesthetic appearance which can be changed from a single point and then reflected throughout the entire environment. StrataFrame takes a different approach to Application Themes than many other themed or skinned component collections. There are no bitmaps or skin files that are used to provide the themed effects. Only pre-existing properties that exist for each of the themed controls are used when rendering the effects and themed appearances.

### 18.4.1 – ApplicationTheme class

The ApplicationTheme shared (static) class is the entry point when dealing with StrataFrame Application Themes. There are a number of methods and properties that can be used to adjust the themes or implement themed functionality into custom controls.

**18.4.1.1 – Theme property**

The Theme property determines which StrataFrame application theme will be loaded or applied to the current environment. This only applies to StrataFrame themes and custom themes cannot be loaded through this property.

**18.4.1.2 – ThemeProperties property**

The ThemeProperties exposes all of the themed properties which determine the effects, colors, and settings that are associated with the current theme. All of these properties are read only and cannot be changed outside of the theme class.

**18.4.1.3 – GlassOpacityPercent property**

The GlassOpacityPercent property determines how strong any glass reflection will be rendered. This is set at a global level rather than at the control level to provide a consistent appearance.

**18.4.1.4 – RefractedLightOpacityPercent property**

Similar to the glass opacity, the RefractedLightOpacityPercent determines how strong any refracted light effect will be rendered. This too is only available at the global level.

**18.4.1.5 – LoadCustomTheme method**

The LoadCustomTheme method will be discussed further in later sections; however, this method allows a custom application theme to be loaded into the application environment. This allows developers to create or modify their own themes and then implement the theme into the application environment.

**18.4.1.6 – RenderGlassOverlay method**

The RenderGlassOverlay method accepts a number of different parameters which will return a rendered glass effect image. This image can then be drawn on top of an existing or custom control which will simulate a glass effect.

**18.4.1.7 – RenderRefractedOverlay method**

The RenderRefractedOverlay method accepts a number of different parameters which will return a rendered refracted light effect image. This image can then be drawn on top of an existing or custom control which will simulate a refracted light effect.

**18.4.1.8 – ApplicationThemeChanged event**

This event is raised any time the application theme is changed. This is a great event to handle when notification of any changes have been made to the themed environment so that any custom logic can be added or controls can be redrawn.

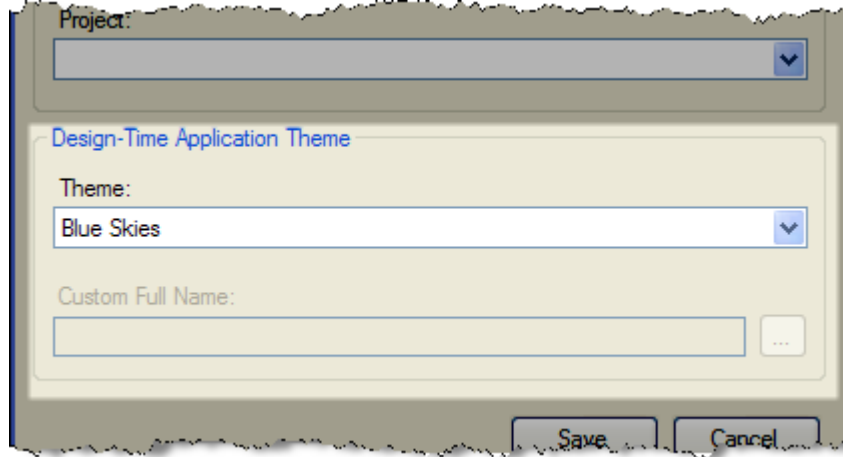
**18.4.2 – DefaultApplicationTheme component**

The Default Application Theme control simply exposes the shared ApplicationTheme class properties so that they can be modified at a component level rather than in code. It is recommended that a

DefaultApplicationTheme control be placed on the initial form that gets loaded, such as a login form. This control should only be dropped on a single form as any subsequent control may conflict or override the previous.

### 18.4.3 – Setting up the solution preferences

The Solution Preferences now support the ability to assign a default theme for a particular solution. This aides in the development process since the themes are not loaded until set through a DefaultApplicationTheme control or set at run-time. To setup the design-time preferences, click the StrataFrame menu and then click Solution Preferences.



### 18.4.4 – Supported controls

Not all of the StrataFrame controls support themes. However, most relevant controls as it relates to the run UI have support. The supported controls are as follows:

- ThemedForm
- BrowseDialog
- GradientFormHeader
- MaintenanceFormToolStrip
- RichTextToolStrip
- ThemedContainer
- ThemedContextMenuStrip
- ThemedDetailWindow
- ThemedGroupBox
- ThemedLinkMenu
- ThemedMenuStrip
- ThemedPanel
- ThemedSplitContainer
- ThemedStatusStrip
- ThemedToolStrip
- WaitWindow
- WizardControl

### 18.4.5 – ThemeSupport properties

The ThemeSupport property will exist on any control that supports StrataFrame themes. This determines the level of theme support that will be applied to a control when rendering. This property only has an affect when the ApplicationTheme.Theme property is set to a value other than “NoTheme.”

### 18.4.6 – Creating custom themes

Creating custom theme is just a matter of inheriting the ApplicationThemeProperties class and overriding each of the properties. Many times it is best to copy an existing theme over and then begin to make changes one control at a time until the theme is complete. Once the theme has been created it can be loaded into the application environment using the LoadCustomTheme method. The LoadCustomTheme method expects a type rather than a reference and should be called as such.

#### LoadCustomTheme Example [C#]

```
ApplicationTheme.LoadCustomTheme(typeof(MyCustomTheme));
```

#### LoadCustomTheme Example [Visual Basic]

```
ApplicationTheme.LoadCustomTheme(GetType(MyCustomTheme))
```

## 18.5 – New themed controls

### 18.5.1 – ThemedContextMenuStrip

The ThemedContextMenuStrip control is an inherited context menu strip that provides StrataFrame theme support. The same functionality that is provided for the standard .NET context menu strip exists for this control as well. The only difference is that the ThemedContextMenuStrip is rendered according to the StrataFrame themes.

### 18.5.2 – ThemedMenuStrip

The ThemedMenuStrip control is an inherited menu strip that provides StrataFrame theme support. The same functionality that is provided for the standard .NET menu strip exists for this control as well. The only difference is that the ThemedMenuStrip is rendered according to the StrataFrame themes.

### 18.5.3 – ThemedToolStrip

The ThemedToolStrip control is an inherited toolstrip that provides StrataFrame theme support. The same functionality that exists for the .NET toolstrip exists for this control as well. One thing worth noting about this control is the support for the Alternate Theme Colors. Many times when a toolstrip is dropped onto a form there may be a main toolstrip and subsequent toolstrips. In this example, the ThemedToolStrip can render differently for the primary and secondary toolstrips.

### 18.5.4 – ThemedPanel

The ThemedPanel provides a panel control that follows the StrataFrame application themes. The same functionality that exists for the .NET panel exists for this control as well.

### 18.5.5 – ThemedSplitContainer

The ThemedSplitContainer is a splitter control which follows the StrataFrame application themes. Like the standard .NET split container, the ThemedSplitContainer provides a splitter panel environment. All of the same functionality that exists for the standard split container exists for this control as well. However, there are some additional rendering options, such as tick marks, for the separator bar.

## 18.6 – BrowseDialog

The browse dialog control is separated within this course since it is such a commonly used control with so much functionality. This section will basically go over the creation of a search form created by the browse dialog control. Throughout this section, some of the more pertinent properties will be pointed out and discussed in more detail.

### 18.6.1 – BusinessObjectType property

This is the first property that needs to be defined when setting up a new browse dialog control. This property defines the type of business object that will be used by the dialog. All other properties require this property to be set first.

### 18.6.2 – SearchFields collection

The search fields are the fields that will be presented to the end-user and allow entry to narrow down the criteria to retrieve the desired records from the database. If no fields are defined this will result in an error since there is nothing for the end-user to enter search criteria. Each search field has a number of settings and includes the ability to populate data using an enumeration, support advanced search features, have initial criteria, and even be excluded from being visible within the search area.

### 18.6.3 – AdvancedOptions property

This property determines if advanced options will be supported as a whole within the search. Advanced options allow fields to be more specific when queried by adding “Begins With” or “Contains” type of attributes.

### 18.6.4 – BrowseResultsLayout property

This property determines how the results will be displayed to the end-user. Setting the BrowseResultsLayout is very similar to the PopulationDataSourceSettings of a ListView with the additional ability to define the columns dynamically.

### 18.6.5 – FormLayout property

Setting the form layout determines the size of the form as well as where each type of section will be displayed within the dialog. The side panel must be turned on if a browse information panel is going to be used since this would make a totally of three sections: Information Panel, Search Fields, and Browse Results. An information panel is a control that allows a developer to place any type of additional fields or logic that may need to be included with the browse.

### 18.6.6 – Using an InformationPanel

Since the search dialog is dynamically created by the BrowseDialog control the InformationPanel class adds the ability to create custom functionality and fields to a search dialog. In fact, each time a new row is selected within the browse results an event is raised in the information panel, much like the RowPopulating event of a ListView, which allows the control access to the business object already on the active row. This can then be used, if necessary to populate any additional data in the panel.

### 18.6.7 – ForceCaseInsensitive property

When True, the queries executed will not test on case sensitivity. This is especially important when talking to a case sensitive SQL Server or other databases such as Visual FoxPro. By default, VFP is case sensitive and this must be set to False in order to have a case insensitive query.

### 18.6.8 – OrderBy property

The order by is not used to order the results of the query, though it can have an affect, but exists for the purpose of supporting older databases which require an ORDER BY clause in order to create a top most query. For example, Visual FoxPro requires an ORDER BY clause when using the TOP command.

### 18.6.9 – AllowHideResults property

This property determines if the results can be hidden during run-time. When set to False, the Hide Results toolbar button will not appear on the toolbar.

### 18.6.10 – AllowSearchFieldsButton property

This property determines whether or not the Search Fields button will appear on the toolbar at run-time. This feature allows the end-user to customize the search fields by order and inclusion. This is a great way to include a large variety of search fields with many being hidden and then allow the end-user the ability to pick and choose which fields they most prefer to search on.

### 18.6.11 – BrowseResultsImageList property

The BrowseResultsImageList property is the image list that is associated with the search results list. The images housed in this image list can be associated with list items through the RowPopulating event of the BrowseDialog.

### 18.6.12 – Shortcut keys

The shortcut keys associated with the BrowseDialog can be adjusted through the AcceptKey, CancelKey, and SearchKey properties. This allows the developer to customize the search interaction to their liking.

### 18.6.13 – Localization values

Any of the text displayed within the BrowseDialog can be overwritten by the developer through the localization text properties. All of these properties are already translated in seven languages and will appear at design-time in the operating language of the developer but will follow the language localization options chosen by the end-user at run-time.



#### 18.6.14 – RowPopulating event

This event is very similar to the ListView RowPopulating event. Additional row settings, such as an image, can be assigned when the results are being loading.

#### 18.6.15 – ComboListPopulating event

This event appeared when the functionality to load a search combo box via a business object appeared. This event functions almost identically to any other combo box or list within StrataFrame. The only difference is the FieldName argument that is passed into the event. Since more than one combo can be loaded an identifier had to accompany the event arguments so that the proper combo box parameters could be provided.

#### 18.6.16 – BrowseDialogClosed and BrowseDialogOpening events

The two events provide an entry point for developers to take certain actions when the browse dialog is opening or has been closed.

#### 18.6.17 – ResetFormSavedDimensions()

Since the BrowseDialog can be adjusted and retained at run-time if the RegistryKey property is provided, there may be a need for the end-user to reset the dialog back to the original dimensions. A good place to call this method is through an InformationPanel. The information panel provides a reference to the BrowseDialog control and can then call then ResetFormSavedDimensions() method. Once called, the form will immediately return to the original dimensions and size. This also includes the splitter bar locations.

#### 18.6.18 – Creating a reusable BrowseDialog control

There are two options when creating a reusable BrowseDialog control. The BrowseDialog can be directly inherited and maintained through code or dropped on a StrataFrame user control and have the business objects translated once dropped on a destination for use.

#### 18.6.19 – Programmatically executing searches

Each field within the SearchFields collection has an InitialValue property. This property can be set a design-time or run-time programmatically. Even fields that are hidden and not displayed to the end-user can be set and will act as part of the query equation if set. Once the values are set the BrowseDialog can automatically execute the search without user intervention if the ShowDialog method is called with a True, which will execute the search using the InitialValue properties.

### 18.7 – UserControl and BOTranslations

A StrataFrame user control contains a property called BOTranslations which allows controls to be created and have internal binding, but when dropped on a parent form or container, those internal business objects can be translated to use the parent references business objects. This works very similar to the ChildDialog control.

## 18.8 – Adding native data binding and localization support

StrataFrame provides two interfaces that allow any type of control to use native data binding and support the localization functionality provided by the framework. Review the code snippet provided in section 6.1 on how to implement an interface to a control. In this section we will actually create a custom control and add both localization support as well as data-binding support.

### 18.8.1 – IBusinessBindable interface

This interface is responsible for all of the properties and methods that are used to provide native StrataFrame data binding. Data binding is very similar from control to control and can generally just be copied into each new control. After making a simple change or two the control will have complete native data-binding.

### 18.8.1 – ILocalizable interface

This interface is responsible for all of the properties and methods that are used to provide native StrataFrame localization functionality.

## 18.9 – WizardControl

The wizard control is an extremely useful tool that allows wizard forms to be created very quickly and effectively. In addition to quick wizard form creation, there are many features that allow the wizard control to be very attractive to the end-user. Wizard pages can even be classed and reused again and again in different wizard environments.

### 18.9.1 – Creating a wizard form

There are two ways to create a wizard form. First is to simply drop the wizard control on an existing form. The second is to use the SF Wizard Form template which will create a form with the control already on the form.

### 18.9.2 – WizardControl page types

There are basically three different page types: Standard, Setup, and Welcome. The welcome page is the first page that is presented to the end-user while a standard page is where most of the content on subsequent pages will reside. A setup page is a much more attractive page that emulates a setup type of environment.

### 18.9.3 – Using a ProgressIndicator control

Another StrataFrame exclusive control is a progress indicator. Though this control can be used alone and outside of a Wizard environment, the wizard control has built-in functionality to work hand-in-hand with this control.

### 18.9.4 – Handling page events

The developer is given the ultimately control of each page and the wizard as a whole. Page events can be handled to prevent forward navigation or to set things up when the page is activated. The wizard

control has more events that can be handled to further extend the control the developer has over the page progression.

## 18.10 – Creating a report using business objects

This particular topic could be covered as an entire course itself. In this section we are going to use a standard .NET report and a business object as the data source. We can then drag the desired fields onto the report and in no time have a usable report that can be implemented into the application. With this being said, most development shops create many reporting standards and may use a third party tool such as Crystal Reports or Active Reports. Any .NET reporting environment that supports ADO.NET inherently supports the StrataFrame business objects. This is just a simple explanation and example to help get developers to thinking about implement business objects into a report environment.

## 18.11 – InfoBox class

The information box class provides the same functionality seen in Microsoft Office 2003 with the notification style windows that pop up. The InfoBox class has extended functionality to allow icons, special effects, and other features when used. This class is generally called manually through code. However, there are certain places within the framework that this class is dynamically used, such as the automatic broken rules alert on a form.

### InfoBox Example [C#]

```
using MicroFour.StrataFrame.Messaging;
...
InfoBox.AlertBox("Hello World", "This is a simple message.");
```

### InfoBox Example [Visual Basic]

```
Imports MicroFour.StrataFrame.Messaging
...
InfoBox.AlertBox("Hello World", "This is a simple message.")
```

## 18.12 – ThemedLinkMenu control

The ThemedLinkMenu control provides similar functionality as the Microsoft Outlook navigation controls or the Office 2007 options menus. Additionally, this control can be used in conjunction with a PanelManager control to produce a tab control type of effect. Another option would be to use the ThemedLinkMenu in conjunction with a ThemedContainer to produce an Explorer style control.

### 18.12.1 – Understanding the control

Since the ThemedLinkMenu can perform a number of different functions, it is important to know which properties really stand out in order to make the control transform itself. Below are a few of the properties worth noting and need a bit of explanation to fully take advantage of the control.

#### 18.12.1.1 – ItemRenderStyle

This property has a significant effect on how the control will be rendered. There are two options: Standard Links and Office 2007 Style Buttons. Each of the descriptions are somewhat self-explanatory but in short provide two different styles of rendering. This option does not affect the functionality.

**18.12.1.2 - LinkedPanelManager**

The PanelManager associated with ThemedLinkMenu when creating a tab control type of functionality.

**18.12.1.3 – LinkItemBehavior**

One of the primary properties that determine how the control will behave is the LinkItemBehavior property. There are two settings: Click and Stick and Standard Links. The Click and Stick approach indicates that when a root item is clicked, it will remain “selected” until another root item is selected. The Standard Links functionality will treat each root element as a link and will not “stick” when clicked.

It is important to note that sub items will always behave as a standard link and does not support the “stick” behavior. This behavior is specific to the root elements.

**18.12.1.4 – ItemBackNormalBackground property**

This property is specific to the appearance and rendering and determines whether or not a background will be rendered when the item is in a normal state (or not selected or hovered). This property only has an affect when the ItemRenderStyle property is set to Office2007StyleButtons.

**18.12.1.5 – ItemClicked event**

The ItemClicked event is raised each time an element is clicked. A reference to the item which was click is provided within the event arguments and can be used to determine what action to take.

**18.12.1.6 – SetActiveItem()**

Allows the developer to forcibly activate an item programmatically and does not require an intervention from the end-user.

**18.12.2 – The Item collection**

The ItemCollection property is the collection that contains all of the root and sub item elements that will be rendered. It is important to note that only one level of sub-items is supported. Even though the type editor will allow sub-items to be added for a sub-item at design-time, the control will not behave properly. So in all there can only be two levels, the root and sub-items underneath the root.

**18.12.2.1 – Key**

The key is a unique identifier for an item. This key can be used for identifying the item when setting a property on the item or handling the ItemClicked event.

**18.12.2.2 – GroupKey property**

The GroupKey property provides a method of linking similar items together. For example, there may be an option to Print underneath more than one root element yet they perform the same action. In this case, the GroupKey property can be tested on in the ItemClicked event rather than the Key.

### 18.12.2.3 – Visible and Enabled properties

Each item supports enabled and visibility statuses. When a feature may not exist, it may be hidden from the end-user, for example.

### 18.12.2.4 – PanelManagerPageName property

The name of the PanelManager page that will be activated when clicked. In order for this property to have any impact, the LinkedPanelManager property must be set on the control.

## 18.13 – ThreadManager control

The ThreadManager control provides an easy way to launch any number of threads by executing a handler method via a delegate. The developer is then notified of the status of the threads through events.

### 18.13.1 – AddProcess()

This is the entry point that launches a handler method on a thread.

### 18.13.2 – ThreadStarted event

Raised when a new thread has been started.

### 18.13.3 – ThreadCompleted event

Raised when a thread has completed execution.

### 18.13.4 – AllThreadsCompleted

Raised once all of the threads have completed execution.

## 19 – Tips & tricks #4

### 19.1 – Reflection

#### 19.1.1 - Overview

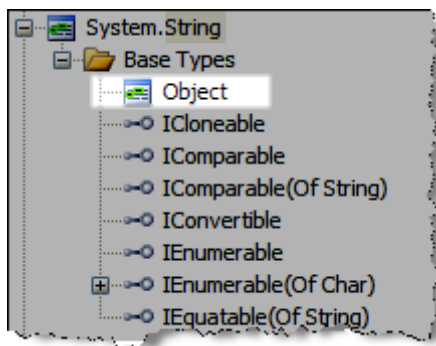
Reflection is the ability to obtain information about a class at run-time and even modify its own structure using abstraction. Reflection is not unique to .NET but rather higher level languages that are strong-typed. So developers coming from a weak-typed language generally have a more difficult time with the concept of reflection since macros and data types could be manipulated at run-time without consequence or without the use of abstraction.

The scope of this section is not to have a 100% understanding of how to use reflection within .NET, as this could consume an entire class on its own. But rather to provide a basic understanding of what reflection is and how it can be used in a real-world environment as well as provide a starting place for introducing reflection into an application.

#### 19.1.2 – System.Reflection Namespace and the GetType() method

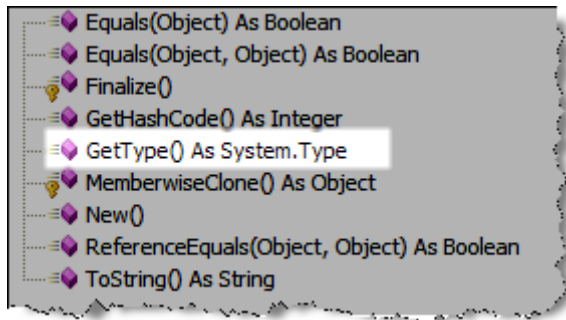
The System.Reflection namespace houses the majority of all of the tools needed in order to take advantage of reflection classes. However, the GetType() method is available on every class within .NET since this is a base method on the Object type. Every class is a “type” of some sort, for example a data or object type of String or TextBox respectively. Though a String is a data type in .NET, it is also an object. If you use the object browser you will see that even a string is derived from an Object.

#### *String Inheritance*



Notice that a string's base type is an object in the image above. The Object in .NET has several primary methods that include `GetType()` and `ToString()`. This is why Intellisense always shows these methods for any class regardless of its type.

## Object Type Methods



### 19.1.3 – Invoking a method

Invoking is a fancy way of saying “call” or “execute.” There may be times when a method needs to be executed at run-time but that method name may not be directly accessible at design-time. In weak-typed languages this would just be a matter of creating a macro for executing it. However, this violates the very foundation of strong-typed languages, including .NET. In this scenario an invocation of a method will be necessary.

#### Method Invocation Example [C#]

```
MyCustomClass loClass = new MyCustomClass();

loClass.GetType().GetMethod("MyMethodName").Invoke(loClass, null);
```

#### Method Invocation Example [Visual Basic]

```
Dim loClass As New MyCustomClass()

loClass.GetType().GetMethod("MyMethodName").Invoke(loClass, Nothing)
```

### 19.1.4 – Reflecting over properties

As with the invocation of a method there are times when properties need to be set or retrieved through reflection. Additionally all of the properties for a type can be enumerated and dealt with however according to the needs of the circumstance. The PropertyInfo class in the System.Reflection namespace is the type that is returned by the GetProperty() method which provides all of the information pertaining to that property.

#### Property Enumeration Example [C#]

```
MyCustomClass loClass = new MyCustomClass();

//-- Cycle through all of the properties and print the name and
// type of the property to the console.
foreach (PropertyInfo loProp in loClass.GetType().GetProperties())
{
    Console.WriteLine(loProp.Name + ": " + loProp.PropertyType.ToString());
}
```

**Property Enumeration Example [Visual Basic]**

```

Dim loClass As New MyCustomClass()
Dim loProp As PropertyInfo

'-- Cycle through all of the properties and print the name and
'   type of the property to the console.
For Each loProp In loClass.GetType().GetProperties()
    Console.WriteLine(loProp.Name & ": " & loProp.PropertyType.ToString())
Next

```

A value for a property can also be set through reflection once a reference to the PropertyInfo has been obtained. The PropertyInfo class has a method called SetValue which allows the value to be set through reflection. Conversely the GetValue() method can be used to retrieve the current value of the property.

**Property Enumeration Example [C#]**

```

MyCustomClass loClass = new MyCustomClass();

loClass.GetType().GetProperty("MyProp").SetValue(loClass, MyNewValue, null);

```

**Property SetValue Example [Visual Basic]**

```

Dim loClass As New MyCustomClass()

loClass.GetType().GetProperty("MyProp").SetValue(loClass, MyNewValue, Nothing)

```

**19.1.5 – Instantiating classes through the Activator class**

Now that we have used some basic reflection to invoke methods and reflect over some properties, how do you create an instance of an object using reflection? This is probably one of the more common uses of reflection in a normal application environment. .NET has a class called Activator which has the ability to create instances of objects using a passed type.

**Property Enumeration Example [C#]**

```

System.Windows.Forms.Form loForm = null;

loForm = (System.Windows.Forms.Form)(Activator.CreateInstance(typeof(MyCustomForm)));

```

**Activator.CreateInstance Example [VB.NET]**

```

Dim loForm As System.Windows.Forms.Form

loForm = CType(Activator.CreateInstance(GetType(MyCustomForm)), System.Windows.Forms.Form)

```



## 19.2 – Reflector

### 19.2.1 – Overview

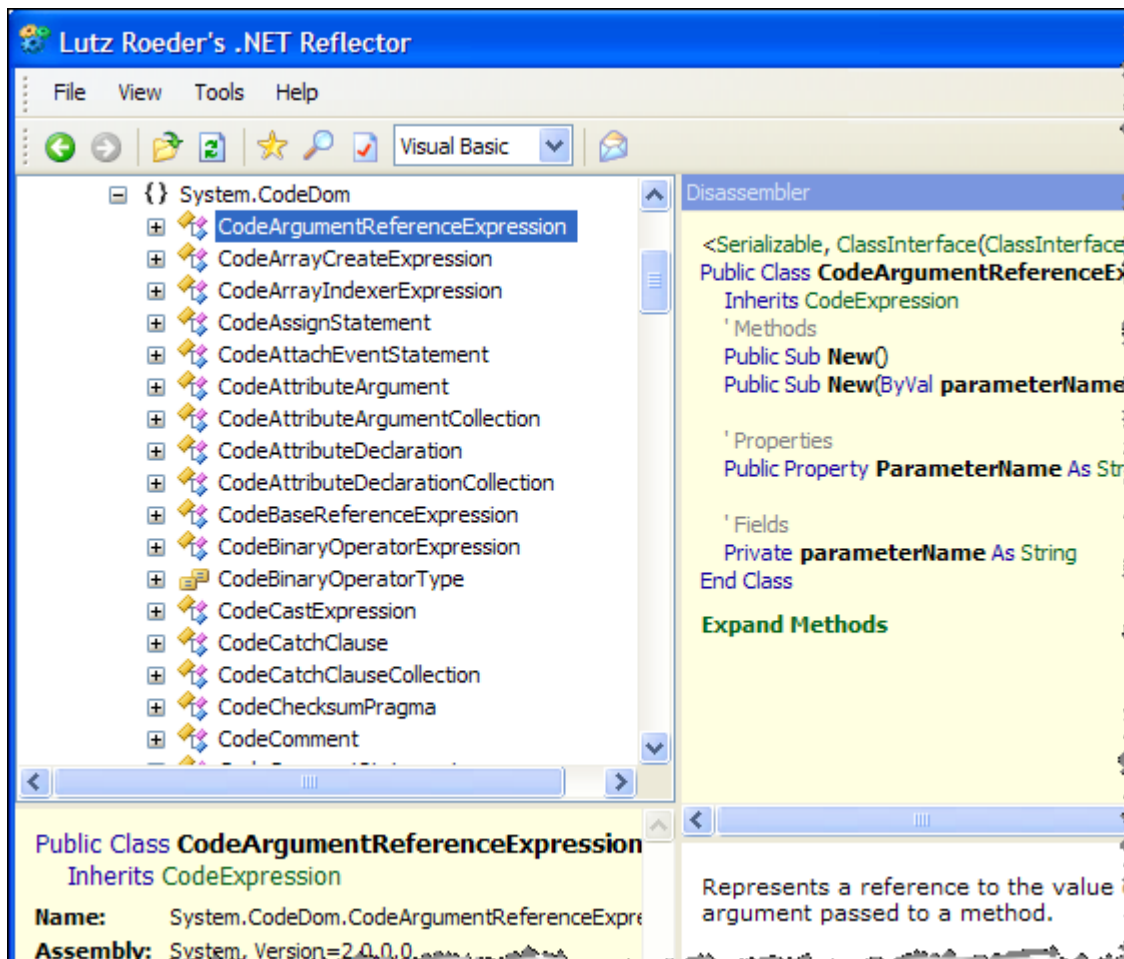
Reflector is a very useful and comprehensive tool that should be in every developer's arsenal. This tool written by Lutz Roeder allows any non-obfuscated .NET assembly to be opened and in essence decompiled allowing a peek at the code behind a class. This is extremely useful at times when trying to understand a 3<sup>rd</sup> parties classes and you do not have the source code. Though as mentioned above, if the assembly is obfuscated then the assembly will not reveal any of the class structures or code.

Reflector is a free download and can be found at the following location:

<http://www.aisto.com/roeder/dotnet/> .

### 19.2.2 – Setting up and running Reflector

Reflector is not a complicated tool to use but provides a great way to view class structures, references, and disassembly code. One really nice feature is the support for different development languages. For example, the disassembled code can be viewed in IL (Intermediate Language), VB.NET, C#, Delphi, Microsoft C+, and Chrome. Though some of the more obscure languages such as Delphi and Chrome may not be 100% accurate in the translation, they can be very helpful. The managed languages disassembled code is very accurate and can really aide in “uncovering the mysteries” of some classes!



## 20 – Tools classes

### 20.1 – MicroFour.StrataFrame.Tools

#### 20.1.1 – Common.BuildDataTableFromEnum()

The BuildDataTableFromEnum() methods are used to create a DataTable or populate a list control from the values in an enumeration. These methods are used by the framework list controls through the PopulationEnumName and PopulationType properties. When you want to manually create a DataTable of an enum's values and the corresponding display values, then use BuildDataTableFromEnum(). These methods respect the [EnumDisplayValueAttribute()] to specify the display value for an enum value.

#### 20.1.2 – Random methods in Common & Helpers

The MicroFour.StrataFrame.Tools.Common class contains several more static (shared) methods that can be used for several things from formatting data to verifying credit card numbers. The Helpers class contains a Between implementation for testing whether a value exists between two other values of the same type.

#### 20.1.3 – Exceptions

The Exceptions class contains methods for working with exceptions. The Exceptions class currently contains one method to generate an exception details string from a specified exception. The details are the same details that are displayed within the red application error window.

#### 20.1.4 – ScreenCapture

The ScreenCapture class contains several methods for capturing the screen or a specific window and saving the captured screen to either a System.Drawing.Image or to a file. The ScreenCapture class uses several API calls that can be found in the user32.dll and gdi.dll libraries.

##### **Capturing the desktop and windows using ScreenCapture [C#]**

```
private void CaptureToImage()
{
    Image screen = ScreenCapture.CaptureScreen();
    Image window = ScreenCapture.CaptureWindow(this.Handle); }

private void CaptureToFile()
{
    ScreenCapture.CaptureScreenToFile(@"C:\screen.bmp", ImageFormat.Bmp);
    ScreenCapture.CaptureWindowToFile(this.Handle, @"C:\window.bmp", ImageFormat.Bmp); }
```

##### **Capturing the desktop and windows using ScreenCapture [Visual Basic]**

```
Private Sub CaptureToImage()
    Dim screen As Image = ScreenCapture.CaptureScreen()
    Dim window As Image = ScreenCapture.CaptureWindow(Me.Handle)
End Sub

Private Sub CaptureToFile()
    ScreenCapture.CaptureScreenToFile("C:\screen.bmp", ImageFormat.Bmp)
    ScreenCapture.CaptureWindowToFile(Me.Handle, "C:\window.bmp", ImageFormat.Bmp)
End Sub
```

## 20.2 – PackageFiles

### 20.2.1 - Overview

Generally all developers have the need for file compression tools at one point or another during their application development cycle. This generally means that another 3<sup>rd</sup> party tool must be purchased in order to place files and folders into a single compressed file. StrataFrame has a proprietary compression tool that uses GZIP technology and is already in the framework. In fact, the Package-It! tool that is installed with StrataFrame is the UI reader that supports the proprietary compressed files. Additionally, this is the same technology used to create the meta-data package files for the Database Deployment Toolkit. The benefit here is that there are no royalties and may be distributed freely with any StrataFrame based application. Additionally, no third party tools or licenses are necessary in order to create compressed package files.

### 20.2.2 – Creating a Package File

The PackageFile class is located in the MicroFour.StrataFrame.IO.Compression namespace. This single class is really the only class required to create, open, read, and extract files. To create a new package file a new instance of the PackageFile class and a new file created. Once this is done, files can be freely added to the package file.

#### Creating a Package File Example [C#]

```
//-- Create a new package file and overwrite any
//    existing package with the same name.
PackageFile loPackage = new PackageFile("c:\\MyNewPackage.pkg", false, true);

//-- Add a file to the package
loPackage.AddFile("C:\\MyNewFile.txt");

//-- Close the package file
loPackage.Close();
```

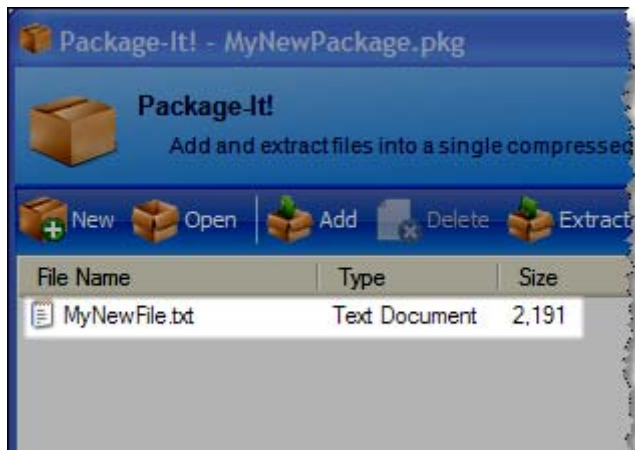
#### Creating a Package File Example [VB.NET]

```
'-- Create a new package file and overwrite any
'    existing package with the same name.
Dim loPackage As New PackageFile("c:\\MyNewPackage.pkg", False, True)

'-- Add a file to the package
loPackage.AddFile("C:\\MyNewFile.txt")

'-- Close the package file
loPackage.Close()
```

Any compressed file created through the PackageFile class can be viewed and modified through PackageIt! UI interface. Using the sample code from above the following result would appear within the PackageIt! Editor if viewed.



### 20.2.3 – Extracting files from a PackageFile

The extraction options are similar to WinZip and WinRAR as it relates to the hierarchy of the extraction. To extract a single file or all files from a package file the file must first be opened and the password authenticated if one exists.

#### 20.2.3.1 – Extracting a single file

It is important to note that all of the extraction methods accept an index rather than a file name. The purpose for this is that a more than one file with the same name may exist within the package. So the `GetIndexesByFilename()` method must be called in order to retrieve an array of the indexes with files of that name.

##### Extracting a Single File Example [C#]

```
//-- Open the package file
PackageFile loPackage = new PackageFile("c:\\MyNewPackage.pkg", false, false);

//-- See if the file exists. If so, extract it.
if (loPackage.GetIndexesByFilename("MyNewFile.txt").Length > 0)
{
    loPackage.ExtractFile(loPackage.GetIndexesByFilename(
        "MyNewFile.txt")[0], PackageExtractionType.SingleFolder, "c:\\temp");
}

//-- Close the package file
loPackage.Close();
```

**Extracting a Single File Example [VB.NET]**

```
'-- Open the package file
Dim loPackage As New PackageFile("c:\MyNewPackage.pkg", False, False)

'-- See if the file exists. If so, extract it.
If loPackage.GetIndexesByFilename("MyNewFile.txt").Length > 0 Then
    loPackage.ExtractFile(loPackage.GetIndexesByFilename("MyNewFile.txt")(0), _
        PackageExtractionType.SingleFolder, "c:\temp")
End If

'-- Close the package file
loPackage.Close()
```

**20.2.3.2 – Extracting all files**

Extracting all of the files is actually simpler than extracting a single file. A single method called `ExtractAllFiles` accepts all of the necessary parameters to extract the files in a single line of code. The below example extracts all of the files within the package file maintaining the integrity of the original folder structure but to the specified path.

**Extracting All Files Example [C#]**

```
//-- Open the package file
PackageFile loPackage = new PackageFile("c:\\MyNewPackage.pkg", false, false);

//-- Extract all of the files
loPackage.ExtractAllFiles(PackageExtractionType.Relative,
    "c:\\temp\\extract", true);

//-- Close the package file
loPackage.Close();
```

**Extracting All Files Example [VB.NET]**

```
'-- Open the package file
Dim loPackage As New PackageFile("c:\MyNewPackage.pkg", False, False)

'-- Extract all of the files
loPackage.ExtractAllFiles(PackageExtractionType.Relative, _
    "c:\temp\extract", True)

'-- Close the package file
loPackage.Close()
```

**20.2.3.3 – Package extraction types**

There are three different folder hierarchy extraction options that can be used. When extracting files it may be necessary to extract the files different ways with respect to the folder extraction.

**Absolute** – Maintains the full folder hierarchy from where the file originally resided. The original folder hierarchy, minus the drive, is appended to the specified extraction folder. For example, if the original

file path was c:\temp\sample\myfile.txt and the extraction folder specified was c:\newfolder the resulting location would be c:\newfolder\temp\sample\myfile.txt.

**Relative** – Maintains the folder hierarchy relative from the originated point. For example, if all of the files and sub-folders in the folder c:\temp were added then any files that resided in Temp folder would be at the root level while any sub folders would be added relative to the root. For example, let's assume the originating file path was c:\temp\sample\myfile.txt and the entire c:\temp folder and sub folders were included when adding the files to the package file. Then if the extraction folder specified was c:\newfolder when extracting the file the resulting location would be c:\newfolder\sample\myfile.txt. However, if only the c:\temp\sample\myfile.txt file was specified when adding the file to the package file, then the relative path would be itself. Using this example, if the extraction folder specified was c:\newfolder when extracting the file the resulting location would be c:\newfolder\myfile.txt.

**Single Folder** – This option will extract all files, even if they originated from different folder locations, in a single folder. No sub-folders will be created or acknowledged when using this option. This is a very handy extraction method when dealing with one or two files. However, if there are files with the same name in the package file and they are all extracted to the same location, one will overwrite the other.

#### 20.2.4 – Available features

The PackageFile class has a number of features such as security and an API interface. Like most other compression tools and formats, a package file can be password protected. Since the package format is proprietary to StrataFrame and part of the framework, the API is integrated in the PackageFile class. In fact, the Packagelt! Editor was written using PackageFile class.

### 20.3 – XmlBasics

The XMLBasics class is a great tool when dealing with XML data structures that may have the need for encryption or dynamic structure changes. The XMLBasics class is the XML equivalent of the Database Deployment Toolkit for SQL Server in many respects. One of the most difficult issues associated with using XML tables for anything is changing the structure of an existing file. When using the XMLBasics class this is all done automatically and even keeps a log so that any structure changes can be noted.

#### 20.3.1 – Creating or opening an XML file with XMLBasics

To begin, let's create a new XML file using the XMLBasics class. The XMLBasics class has a number of overloads when opening an XML file, but the method name is OpenXmlFile(). When opening XML files through the XmlBasics class the XML file will always be stored as a DataSet rather than a DataTable. The reason for this is that there is a small version information table that is stored along with the actual XML structure which is being specified for automatic and dynamic structure changes. This is also the internal log file that can be used to determine if the XML file has been upgraded. One other key component is the structure provided to this method. A collection of DataColumn objects supplied as a parameter to the OpenXmlFile() method provides the meta-data structure as to how the XML file is to be defined.

#### 20.3.2 – All potential parameters for OpenXmlFile()

**CurrentVersion** – This is the version of the XML file which determines if a potential conversion or upgrade will be performed. One way to automate this version with requiring the developer to manually change the value in code is by using the version of the assembly for this value.

**TableName** – The name of the table that will be housed within the XML file (i.e. Customers).

**PathAndFileName** – The path and file name of the XML file.

**ColumnStructure** – The DataColumn collection that defines the structure of the XML table in which the XML file represents. This structure is the meta-data definition of the XML file.

**DecryptFile** – When True, the OpenXmlFile() will attempt to decrypt the file assuming that it has been encrypted through the WriteXmlFile() method.

**PrimaryKey** – Accepts a DataColumn array which allows a primary key to be defined for the data table. The purpose of the array is for compound primary key support.

**ValidationKey** – The validation key allows an additional level of security to be applied to the XML file. For example, if the XML is to be specific to a certain computer, a MAC address may be used as the validation key so that when copied to another computer it would not be able to authenticate.

#### **OpenXmlFile() example [C#]**

```
//-- Open and XML File using the XMLBasics class
private void OpenXMLTable()
{
    //-- Establish Locals
    DataSet loDS = null;
    DataTable loTable = null;

    //-- Open the XML file
    loDS = XmlBasics.OpenXmlFile("1.0", "Sample", "c:\\temp",
        GetSampleXmlStructure());

    //-- Get only the Sample table
    loTable = loDS.Tables["Sample"];
}

//-- Returns the structure of the XML file in DataColumn format
private List<DataColumn> GetSampleXmlStructure()
{
    //-- Establish Locals
    List<DataColumn> loReturn = new List<DataColumn>();

    //-- Create the columns and provide a default value
    loReturn.Add(XmlBasics.CreateXmlDataColumn("FirstName", typeof(string), ""));
    loReturn.Add(XmlBasics.CreateXmlDataColumn("LastName", typeof(string), ""));

    //-- Return Results
    return loReturn;
}
```

**OpenXmlFile() example [Visual Basic]**

```

'-- Open and XML File using the XMLBasics class
Private Sub OpenXMLTable()
    '-- Establish Locals
    Dim loDS As DataSet
    Dim loTable As DataTable

    '-- Open the XML file
    loDS = XmlBasics.OpenXmlFile("1.0", "Sample", "c:\temp", _
        GetSampleXmlStructure())

    '-- Get only the Sample table
    loTable = loDS.Tables("Sample")
End Sub

'-- Returns the structure of the XML file in DataColumn format
Private Function GetSampleXmlStructure() As List(Of DataColumn)
    '-- Establish Locals
    Dim loReturn As New List(Of DataColumn)

    '-- Create the columns and provide a default value
    loReturn.Add(XmlBasics.CreateXmlDataColumn("FirstName", GetType(String), ""))
    loReturn.Add(XmlBasics.CreateXmlDataColumn("LastName", GetType(String), ""))

    '-- Return Results
    Return loReturn
End Function

```

**20.3.3 – Updating the Structure of a Pre-Existing XML File**

There is little that needs to be done when updating an existing XML file. However, the existing XML file MUST have been created through the XMLBasics class. Second, the CurrentVersion parameter must be different than the pre-existing version stored within the XML file. Otherwise, there is nothing else that needs to be done. Simply change the version and the XML file will open and update with the new structure.

**20.3.4 – Writing an XML File With XMLBasics**

The WriteXmlFile really does nothing more than take the DataSet and store it to disk in XML format. However, there is some additional functionality such as validation keys, encryption, and formatting that takes place through this method. In order for an existing XML file to be opened through the XMLBasics class, it must have been saved through the WriteXmlFile method.



**Write XML File Example [C#]**

```

/-- Open and XML File using the XMLBasics class
private void OpenXMLTable()
{
    //-- Establish Locals
    DataSet loDS = null;
    DataTable loTable = null;

    //-- Open the XML file
    loDS = XmlBasics.OpenXmlFile("1.0", "Sample", "c:\\temp", GetSampleXmlStructure());

    //-- Get only the Sample table
    loTable = loDS.Tables["Sample"];

    //-- Write the XML file to disk
    XmlBasics.WriteXmlFile(loDS, "c:\\temp");
}

```

**Write XML File Example [VB.NET]**

```

'-- Open and XML File using the XMLBasics class
Private Sub OpenXMLTable()
    '-- Establish Locals
    Dim loDS As DataSet
    Dim loTable As DataTable

    '-- Open the XML file
    loDS = XmlBasics.OpenXmlFile("1.0", "Sample", "c:\\temp", GetSampleXmlStructure())

    '-- Get only the Sample table
    loTable = loDS.Tables("Sample")

    '-- Write the XML file to disk
    XmlBasics.WriteXmlFile(loDS, "c:\\temp")
End Sub

```

**20.3.5 – Encryption and XMLBasics**

Many times encryption can be very important when saving data to an XML file. Many times these types of files will contain password information or something of a nature that should not be exposed to all eyes. This is why the WriteXmlFile() and OpenXmlFile() methods support encryption. When encryption is used, the 3DES encryption algorithm is used. To apply a validation key to the XML file the ChangeValidationKey() method must be called.

**Change validation key example [C#]**

```

XmlBasics.ChangeValidationKey(loDS, "MyNewKey");

```

**Change Validation Key Example [VB.NET]**

```

XmlBasics.ChangeValidationKey(loDS, "MyNewKey")

```

**Tip:** The IsEncrypted method can be used to determine if the XML file is encrypted or not. This can be very helpful when determining how to open existing XML files.

### 20.3.6 – CreateXmlDataColumn and CreateXmlDataColumnAutoInc

These two methods provide an easy way to create an ADO.NET data column in a single line of code. The difference between the two is that the CreateXmlDataColumn accepts a default or initial value whereas the CreateXmlDataColumnAutoInc does not. The CreateXmlDataColumnAutoInc method will create and auto-incrementing data column, for example a primary key field.

### 20.3.7 – CLS Compliant Methods

CLS (Common Language Specifications) is the ability for any disparate or non-related application language to have common support. A good example of where a CLS type of environment may come into play is when creating COM classes. Since any language outside of .NET will not support generic collections CLS compliant methods have been provided so that these features can be exposed and used in a COM model environment.

## **21 – Tips & Tricks #5**

### **21.1 – .NET data binding**

#### **21.1.1 – Control.DataBindings collection and Binding objects**

The `System.Windows.Forms.Control` class contains a collection of Binding objects that determine the properties on the control that are bound. For each property on the control that is bound to a data source, the control will have a Binding object in the collection.

#### **21.1.2 – Update modes**

Data-binding in .NET WinForms is 2-way; the data will be pushed from the control to the data source when the control's property changes and the data will be pulled from the data source into the control when the data source changes. There are 2 properties that determine whether and how the data is moved between the data source and the control:

- `DataSourceUpdateMode` – contains a value of the `System.Windows.Forms.DataSourceUpdateMode` enumeration and determine how the data is copied from the control to the data source.
  - `Never` – The data is never copied from the control to the data source.
  - `OnPropertyChanged` – The data is copied from the control to the data source when a change notification is raised by the control.
  - `OnValidation` – The data is copied from the control to the data source when the `Validating` event fires (and is not canceled) on the control.
- `ControlSourceUpdateMode` – contains a value of the `ControlUpdateMode` enumeration that specifies how the data is copied from the data source into the control.
  - `Never` – The data is never copied from the data source into the control.
  - `OnPropertyChanged` – The data is copied from the data source to the control when a changed notification is raised in the data source.

#### **21.1.3 – BindingContexts, CurrencyManagers, PropertyManagers, and PropertyDescriptors**

Several objects work behind the scenes to control the binding between a control and the data source.

**BindingContext** – The `BindingContext` is a management object that manages all data-bindings for a form. The `Form.BindingContext` property is used to retrieve the `BindingContext` object for that form. The `BindingContext` can retrieve the `BindingManagerBase` object for any data source on the form.

**CurrencyManager** – A `CurrencyManager` object is a `BindingManagerBase` object that manages the bindings for a list data source. The `CurrencyManager` maintains the `Position` property that determines which object from the list is currently bound to the controls.

**PropertyManager** – A `PropertyManager` is a `BindingManagerBase` object that manages the bindings to a data source that is a single object with properties (not a list of objects).

**PropertyDescriptor** – A `PropertyDescriptor` is an object that manages the meta-data associated with and access to a single property on a data source.

### 21.1.4 – Requirements for .NET data binding

To implement 2-way .NET data binding through Binding objects and the Control.DataBindings collection, there are several requirements on both the data source and the control:

- Properties (not fields) – Data binding will not work with fields. The bound columns on the data source must be properties. (Exceptions to this rule are WithEvents objects in VB since a property is created behind the scenes and any object that uses custom property descriptors, such as a DataView.)
- Changed events – Both the control and the data source must support a changed event for the property that is bound.
- PropertyChanged event – The control and the data source can implement an event that matches the name of the property (a Text property would have a TextChanged event) and is of the EventHandler type.
- INotifyPropertyChanged.PropertyChanged event – The control and data source can implement the INotifyPropertyChanged interface and raise the PropertyChanged event when the value of the property changes.
- IBindingList.ListChanged – A list data source can implement the IBindingList.ListChanged event when an object within the list changes.

Raising a changed event on the data source causes all of the controls bound to that data source to refresh.

### 21.1.5 – Important interfaces

When creating a data source for inherent .NET data binding, it must implement one or more of the following interfaces:

#### 21.1.5.1 – IComponent

If you want to be able to drop your data source from the toolbox onto your form, it will need to implement the IComponent interface either directly or by inheriting System.ComponentModel.Component or System.ComponentModel.MarshalByValueComponent.

#### 21.1.5.2 – IList

Implementing the IList interface causes the object to be recognized as a data source by the .NET binding type editors.

#### 21.1.5.3 – IListSource

Implementing the IListSource causes the object to be recognized as a data source by the .NET binding type editors. An IListSource object is not a list itself, but uses the GetList() method to retrieve an IList object that is the list needed for binding. For example, the DataTable class does not implement IList, but implements IListSource and returns a DataView for the table from the GetList() method.

#### 21.1.5.4 – ITypedList

Implementing the ITypedList allows you to control what fields on the data source show up as bindable properties to the type editors. If the ITypedList interface is not implemented, the object returned from

the default property (this[]) of the IList is reflected and all properties not marked with [Bindable(false)] are determined to be bindable fields.

#### 21.1.5.5 – IBindingList

The IBindingList interface extends the IList interface to add sorting and change notification support through the ListChanged event. The IBindingList is the most commonly implemented interface for creating a bindable data source (the IBindingListView interface is usually overkill).

#### 21.1.5.6 – IBindingListView

The IBindingListView interface extends the IBindingList interface to add support for filtering and advanced sorting (sorting on more than one column).

#### 21.1.5.7 – INotifyPropertyChanged

The INotifyPropertyChanged interface can be implemented on the objects within the list (it's not needed by the actual list, but the items within the list) in place of the [PropertyName]Changed events that would otherwise be necessary.

## 21.2 – Signing, basing, & obfuscation

<http://msdn.microsoft.com/msdnmag/issues/06/07/CLRInsideOut/>

<http://msdn.microsoft.com/msdnmag/issues/06/02/CLRInsideOut/>

### 21.2.1 – SNK pairs, signed assemblies and the GAC

Assembly signing is a security feature implemented by .NET that allows the user of an assembly to ensure that the assembly has not been tampered with since it was compiled by the assembly's writer. A strong-named assembly is the same thing as a signed assembly.

Signing an assembly requires the use of an RSA public/private key pair. The assembly is hashed using the private key and the public key is then stored within the assembly. The user of an assembly can then use the public key to ensure that the hash of the assembly matches the correct value and it has not been tampered with.

Assemblies must be signed before they can be added to the GAC. The GAC verifies the signing of an assembly before it is allowed to be added to the GAC. Assemblies loaded from the GAC do not need to be re-verified which removes the load-time penalty of re-verifying the assembly's strong-name.

### 21.2.2 – Obfuscation

Obfuscation refers to the modifying of the MSIL of an assembly so that it cannot be easily reverse-engineered. Tools such as Reflector and other ISL Disassemblers can be used to decompile an assembly into readable source code that can be produced in a variety of .NET languages. Method names and non-local variable names are preserved within the MSIL of a compiled assembly, allowing decompiled source code to be easily readable.

Obfuscators use several techniques to modify the MSIL of an assembly to prevent the assembly from being easily decompiled:

- Renaming references – private and internal references (methods, properties, variables) can be easily renamed unreadable characters to disguise the use of the reference.
- Encrypting strings – string literals within the assembly are replaced with encrypted strings. This option can slow the performance of the assembly because the strings must be decrypted when they are used.
- Loop modification – loops within the code can be replaced with similar but more difficult to read loops. This option can slow the performance of the assembly because it may remove any optimization you have added to the loop.

### 21.2.3 – Delay signing & resigning assemblies

There are times when a signed assembly is required, but you do not want to automatically sign the assembly through Visual Studio:

- If an assembly is signed and then obfuscated, it will break the signing.
- You may not trust your developers with the .pfx or .snk file.

An assembly can be delay signed using only the public key of the RSA key pair. This allows the assembly to be executed on development machines and even obfuscated without breaking the ability for the assembly to be used on the development machine. After obfuscation, you can then use the key pair to re-sign the assembly. When delay signing, the re-signing process is generally accomplished on a build computer or build server where the key-pair can be retrieved from a protected location that is not available to non-trusted individuals.

### 21.2.4 – Assembly base addresses

The .NET CLR uses a type loading method called lazy loading. If an assembly is referenced by an executing assembly, it is not loaded until actually needed by the executing assembly; it is loaded when a type from the referenced assembly is needed or a method located within the referenced assembly is needed.

All assemblies are compiled with a preferred base address that indicates the starting location in the applications committed memory where it wants to be loaded. Each compiled method is then assigned an offset value to indicate where it should be loaded into memory relative to the base address. If the assembly attempts to load into its preferred memory space and that space is already occupied by another assembly, the loading assembly must be “rebased.” Rebasings an assembly means that the application determines a new location for the assembly and then re-calculates and re-assigns memory locations to each type and method that must be loaded into memory.

Once an assembly is loaded, its runtime performance is the same whether it was rebased or not, but rebasing will greatly increase the load-up time of an assembly and can possibly cause a perceptible pause in the application as the assembly is loaded for the first time.

The base address for an assembly is defined within the project properties. Determining the best possible base address for an assembly can be accomplished with programs like Process Explorer which can indicate when an assembly loading collision occurs.