



DesignedNet Framework Architecture Document

The DesignedNet framework provides an enterprise-level architecture for simplified development of web based applications using the Microsoft .NET framework. The architecture consists of three application layers (Data Access, Business Logic and User Interface) and is consistent with the Microsoft Solution Framework development methodology. Each application layer is built upon a single base class which provides a consolidated point of implementation for common functionality leveraged by all objects in the application layer.

Previous experiences have extensively demonstrated how often an application's requirements are directly derived from the database model. Many of an application's forms used to manage data in the database require only simple validation to assure the integrity of the data before passing it along to the database. In order to leverage this realization, the Design Studio component uses the database schema of any SQL Server or Access database to automatically generate a web based interface to maintain the simple data structures of the database. Stored procedures and data access code are created to facilitate the data querying and updating processes. Business objects for each database table act as cursors to navigate the data retrieved from the data access calls. User interface forms provide data editing and validation for each business object.

This means that the DesignedNet framework is able to generate a working proof-of-concept application from a proper database design in a matter of hours, rather than weeks or months. Projects built by Designed Simplicity using the DesignedNet framework are often fully functional before a competing solution provider would be able to deliver a demonstration. This translates into a better application in less time and for less money than others can provide.

Data Access Layer Functional Requirements

The data access layer of the DesignedNet framework is engineered to provide database access to Microsoft SQL Server. The `DesignedNet.Framework.Dal` namespace has three entities and consists of one base class and two interfaces.

The base class, `DalSqlDb`, implements the code needed to instantiate and query a database connection. The class also supports default functions to provide stored procedure command object building for common verbs: `Insert`, `Select`, `Update`, `Delete`, `List` and `ListByColumn`. The database connection string should be provided in the `App.config` or `Web.config` file of the development project. The two interfaces, `IDalData` and `IDalSql`, are fully implemented in `DalSqlDb`.

Web.config

```
/// -----  
<?xml version="1.0" encoding="utf-8" ?>  
<configuration>  
  <appSettings>  
    <add key="SqlConnectionString" value="server=; uid=; database=;  
password=";/>  
  </appSettings>  
</configuration>  
/// -----
```

The IDalData interface provides the baseline data access verbs using standard .NET Data namespace objects which are not data source specific. This provides a common base interface for all data access entity objects regardless of data store requirements. The IDalSql interface provides a common SQL specific command building for the data access verbs. These command building functions should be overridden by the consuming entity class for specific optimization.

Any class implementing the IDalData interface is fully consumable by the business logic layer state and entity classes. The IDalSql interface is only referenced within the Dal namespace to provide data store specific command generation and execution. Future interfaces and base classes will be developed for the .NET OleDb namespace compliant databases such as Access, MySql and Oracle.

DalSqlDb.cs

```
/// -----  
public class DesignedNet.Framework.Dal.DalSqlDb : IDalSql, IDalData  
{  
  public DalSqlDb(string tableName) { _tableName = tableName; }  
  
  int Insert(DataRow row);  
  int Select(object pk, DataTable table);  
  int Update(DataRow row);  
  int Update(DataTable table);  
  int Delete(object pk);  
  int Delete(DataRow row);  
  int List(DataTable table);  
  int ListByColumn(string column, object data, DataTable table);  
  
  SqlCommand GetListCmd();  
  SqlCommand GetListByColumnCmd(string index, object fk);  
  SqlCommand GetInsertCmd();  
  SqlCommand GetSelectCmd(object pk);  
  SqlCommand GetUpdateCmd();  
  SqlCommand GetDeleteCmd();  
  SqlCommand GetDeleteCmd(object pk);  
}  
/// -----
```



Business Logic Layer Functional Requirements

The business logic layer of the DesignedNet framework is engineered to provide enumeration and typed property access for each defined entity. The DesignedNet.Framework.Biz namespace has four entities and consists of two base classes and two interfaces. The first base class, BizState, implements the code needed to maintain the state of one or more entities of the same type. Currently support types include only the non-specific .NET Data namespace objects: DataSet, DataTable, DataView, DataRow. The state class provides an enumeration engine and index project access for data source binding: IEnumerable, IEnumerator. Since the state class needs to communicate with a specific data access class, the CreateDal function is virtual and must be overridden in the consuming classes. This function creates a data access class implemented using the IDalData interface for common data access verbs. The second base class, BizEntity, is inherited from BizState. This child object adds the common functionality needed to mirror the common data access verbs. Each interface, IBizState and IBizEntity, provides the interfaces definitions for each base class requirements. Any business object which fully implements the IBizEntity and IBizState interfaces may be used with the DesignedNet Framework.

BizState.cs

```
/// -----  
abstract public class DesignedNet.Framework.Biz.BizState : IBizState  
{  
    public BizState(string entityName)  
    public BizState(string entityName, DataSet state)  
  
    int Count { get; }  
    DataRow Row { get; }  
    DataTable Table { get; }  
    DataView DefaultView { get; }  
    bool CreateDal();  
    bool CreateState();  
    bool HasState();  
    bool Sort(string orderBy);  
    bool Filter(string fk, int id);  
    bool Filter(string col, string val);  
    bool HasChildren(string relatedTable);  
}  
  
abstract public class DesignedNet.Framework.Biz.BizEntity : IBizEntity  
{  
    public BizEntity(string entityName)  
    public BizEntity(string entityName, DataSet state)  
  
    bool New();  
    bool Save();  
    bool List();  
    bool Find(int id);  
    bool Load(int id);  
    bool Delete();  
    bool Update();  
}  
/// -----
```



DesignedNet Backend Management Application Library

The DesignedNet.Management library provides the minimal set of objects required to create a database driven application for ASP.NET or WinForms. The single business and data access entity is BizUser. This user entity serves as both an open example of the DesignedNet Framework implementation and as the start of a namespace dedicated to providing a common set of functionality. This entity class includes functions for user login, registration, profile management and password retrieval. The BizUser entity is persisted in a SQL Server 2000 database table. The following schema defines the table structure and is used to generate the entity object's structure and interface.

	Column Name	Data Type	Length	Allow Nulls
?	UserID	int	4	
	UserTypeID	int	4	
	Subscribed	bit	1	
	Username	varchar	25	
	Password	varchar	25	
	FirstName	varchar	50	
	LastName	varchar	50	
	EmailAddress	varchar	100	
	CompanyName	varchar	100	✓
	Address1	varchar	100	✓
	Address2	varchar	50	✓
	City	varchar	50	✓
	Province	varchar	50	✓
	PostalCost	varchar	50	✓
	Country	varchar	50	✓
	Phone	varchar	25	✓
	Mobile	varchar	25	✓
	Home	varchar	25	✓
	Fax	varchar	25	✓
	IM	varchar	25	✓
	LastLogin	datetime	8	✓
	Created	datetime	8	
	Updated	datetime	8	✓

```
CREATE TABLE [dbo].[User] (
  [UserID] [int] IDENTITY (1, 1) NOT NULL ,
  [UserTypeID] [int] NOT NULL ,
  [Subscribed] [bit] NOT NULL ,
  [Username] [varchar] (25) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
  [Password] [varchar] (25) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
  [FirstName] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
  [LastName] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
  [EmailAddress] [varchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
  [CompanyName] [varchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
  [Address1] [varchar] (100) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
  [Address2] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
  [City] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
  [Province] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
  [PostalCost] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
  [Country] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
  [Phone] [varchar] (25) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
  [Mobile] [varchar] (25) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
  [Home] [varchar] (25) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
  [Fax] [varchar] (25) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
  [IM] [varchar] (25) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
  [LastLogin] [datetime] NULL ,
  [Created] [datetime] NOT NULL ,
  [Updated] [datetime] NULL
) ON [PRIMARY]
GO
```

BizUser.cs

```
/// -----
public class BizUser : BizEntity
{
    public BizUser() : base("User") {}
    public BizUser(DataSet state) : base("User", state) {}
    public bool Login(string username, string password)
    public bool FindUsers(string emailAddress, string username)

    public object UserID, UserTypeID, Subscribed, Username, Password,
    FirstName, LastName, EmailAddress, CompanyName, Address1, Address2,
    City, Province, PostalCost, Country, Phone, Mobile, Home, Fax, IM,
    LastLogin, Created, Updated

    public bool CompanyNameIsNull, Address1IsNull, Address2IsNull,
    CityIsNull, ProvinceIsNull, PostalCostIsNull, CountryIsNull,
    PhoneIsNull, MobileIsNull, HomeIsNull, FaxIsNull, IMIsNull,
    LastLoginIsNull, UpdatedIsNull
}
/// -----
```

