Contents

[1 Configuration 2](#_Toc478221821)

[1.1 Web Config 2](#_Toc478221822)

[1.1.1 Enable custom sections 2](#_Toc478221823)

[1.1.2 Connection string to OleDB (Like SQL) 2](#_Toc478221824)

[1.1.3 Mongo DB 3](#_Toc478221825)

[1.1.4 ResolveHostList 3](#_Toc478221826)

[1.1.5 Custom Config Section 4](#_Toc478221827)

[1.1.6 System.webServer section 4](#_Toc478221828)

[1.2 Global.asax 5](#_Toc478221829)

[1.2.1 Init application 5](#_Toc478221830)

[1.2.2 Virtual Pages 5](#_Toc478221831)

[1.3 General Parameters 5](#_Toc478221832)

[2 Inherit UI, WebServices, Controller and Service 6](#_Toc478221833)

[2.1 .aspx web services 6](#_Toc478221834)

[2.2 BaseWebController and Virtual page in aspx 6](#_Toc478221835)

[2.3 UI Service 6](#_Toc478221836)

[2.4 BaseService 6](#_Toc478221837)

[3 BaseApp Methods 7](#_Toc478221838)

[3.1 BaseUserManager 7](#_Toc478221839)

[3.2 VirtualPageManager 7](#_Toc478221840)

[3.3 WSP – Warehouse Stored Procedures 7](#_Toc478221841)

[3.4 DB – Data Base (Ole) 7](#_Toc478221842)

[3.4.1 Generics methods 7](#_Toc478221843)

[3.4.2 GetEscalar 8](#_Toc478221844)

[3.4.3 GetRow/ColumnList 8](#_Toc478221845)

[3.4.4 StoredProcedure objects 8](#_Toc478221846)

[3.5 Session 9](#_Toc478221847)

[3.6 ContainerManager 9](#_Toc478221848)

[3.7 MongoDB – NoSQL Data Base 9](#_Toc478221849)

[3.8 BaseConfigManager 9](#_Toc478221850)

[3.9 BaseUser 10](#_Toc478221851)

[3.10 UI 10](#_Toc478221852)

[3.11 Resolve 10](#_Toc478221853)

[4 Lifestyles and Base Interfaces 11](#_Toc478221855)

[4.1 General Use 11](#_Toc478221856)

[4.2 Multiple lifestyles 11](#_Toc478221857)

[4.3 Windsor LifeStyles 11](#_Toc478221858)

[4.4 Base Custom LifeStyles 11](#_Toc478221859)

[5 Create Virtual Pages 12](#_Toc478221860)

[6 Helpers and Extensions 13](#_Toc478221861)

[7 Multiple Implementations 14](#_Toc478221862)

[8 Data Base Squema 15](#_Toc478221863)

# Configuration

## Web Config

### Enable custom sections

Add the costum configSection at the beginning of the web.config xml inside configuration section in order to enable all custom configuration.

<configSections>

<section name="BaseConnectionStrings" type=" MYB.BaseApplication.Security.Configuration.Sections.ConfigConnectionStringSection, MYB.BaseApplication.Security.Configuration"/>

<section name="BaseConnectionData" type=" MYB.BaseApplication.Security.Configuration.Sections.ConfigConnectionDataSection, MYB.BaseApplication.Security.Configuration"/>

<section name="BaseMongoConnectionStrings" type=" MYB.BaseApplication.Security.Configuration.Sections.ConfigMonoConnectionStringSection, MYB.BaseApplication.Security.Configuration"/>

<section name="BaseMongoConnectionData" type=" MYB.BaseApplication.Security.Configuration.Sections.ConfigMonoConnectionDataSection, MYB.BaseApplication.Security.Configuration"/>

<section name="ResolveHostList" type=" MYB.BaseApplication.Security.Configuration.Sections.ConfigResolveHostListSection, MYB.BaseApplication.Security.Configuration"/>

</configSections>

### Connection string to OleDB (Like SQL)

#### Use BaseConnectionStrings

Is like normal main connection string but you can specify host=”” and each request will save in his Host request connectionString. If you add a connection string without host it will marked as default

<BaseConnectionStrings>

<connectionString connectionString="Provider=SQLOLEDB;Data Source=bonait.com.ar;Initial Catalog=BaseApplication;User Id=userID;Password=asdegELgas324" host="localhost" providerName="System.Data.SqlClient"/>

<connectionString connectionString="Provider=SQLOLEDB;Data Source=bonait.com.ar;Initial Catalog=SecondApplication;User Id=username;Password= asdegELas324" host="bonait.com.ar" providerName="System.Data.SqlClient"/>

</BaseConnectionStrings>

#### Use BaseConnectionData

Like BaseconnectionString but the connectionString is divided in propery sections.

<BaseConnectionData>

<connectionData host="localhost" provider="SQLOLEDB" dataSource="bonait.com.ar" initialCatalog="BaseApplication" userID="webadmin" password="Homero32"/>

<connectionData provider="SQLOLEDB" dataSource="bonait.com.ar" initialCatalog="BaseApplication" userID="webadmin" password="Homero32"/>

</BaseConnectionData>

Note: If BaseConnectionString is defined this section will note be used.

#### Normal Connections string

User normal connectionStrings section without Host url mapping.

<connectionStrings>

<remove name="MainConnectionString"/>

<remove name="MongoConnectionString"/>

<add name="MainConnectionString" connectionString="Provider=SQLOLEDB;Data Source=bonait.com.ar;Initial Catalog=BaseApplication;User Id=webadmin;Password=Homero32" providerName="System.Data.SqlClient"/>

<add name="MongoConnectionString" connectionString="mongodb://[username:password@]host1[:port1][/[database][?options]]"/>

</connectionStrings>

### Mongo DB

Same as OleDB config but with the connection string of a mongo DB

<BaseMongoConnectionStrings>

<connectionString connectionString="mongodb://localhost:27017"/>

<connectionString host="bonait.com.ar" connectionString="mongodb://ausername:apassword@flame.mongohq.com:27065/dunedin" />

</BaseMongoConnectionStrings>

<BaseMongoConnectionData>

<connectionData host="localhost" dataSource="localhost" port="27017" />

<connectionData host="www.testApp.com.ar" dataSource="host1" port="27017" dataBase="[database]" username="[username]" password="password@"/>

<connectionData dataSource="bonait.com.ar" port="27017"/>

</BaseMongoConnectionData>

<connectionStrings>

<remove name="MainConnectionString"/>

<remove name="MongoConnectionString"/>

<add name="MainConnectionString" connectionString="Provider=SQLOLEDB;Data Source=bonait.com.ar;Initial Catalog=BaseApplication;User Id=webadmin;Password=Homero32" providerName="System.Data.SqlClient"/>

<add name="MongoConnectionString" connectionString="mongodb://[username:password@]host1[:port1][/[database][?options]]"/>

</connectionStrings>

### ResolveHostList

With the class that resolve by IPerHost interface, it will search in this table if you want to internal redirect the host.

In this example when you resolve something by localhost, TestApp.com or app.TestApp.com the system will think is the same host (localhost in this case), and will resolve the same object.

For test.TestApp.com and Bonait.com.ar will be resolve as Bonait.com.ar.

<ResolveHostList>

<domain ID="localhost" host="TestApp.com"/>

<domain ID="localhost" host="app.TestApp.com"/>

<domain ID="BonaIT.com.ar" host="test.TestApp.com"/>

</ResolveHostList>

### Custom Config Section

You can add your customs config section at webconfig. Note: Try to not abuse of it, the web.config is only for connection and server settings.

<CustomConfigSection>

<testConfig ID="pepe"/>

<testConfig ID="juan" Name="Juan perez de la cruz"/>

</CustomConfigSection>

You need to do your oun config element class to tell the system wich propertys exists, your class must be children from ConfigurationElement

public class TestConfigElement : ConfigurationElement

{

[ConfigurationProperty("ID", IsRequired = true)]

public string ID

{

get { return (string)this["ID"]; }

set { this["ID"] = value; }

}

[ConfigurationProperty("Name")]

public string Name

{

get { return (string)this["Name"]; }

set { this["Name"] = value; }

}

}

And then when you want to call your config must do:

Enumerable<TestConfigElement> elements = BaseApp.BaseConfigManager<TestConfigElement>().GetSectionElementList("CustomConfigSection", "testConfig");

The variable element has all testConfig in these section.You can change CustomConfigSection and testConfig to you prefers names.

To enabled it add at the beginning in configSection this:

<section name="CustomConfigSection" type=" MYB.BaseApplication.Security.Configuration.Sections.ConfigSection`1[[TestApp.TestConfigElement, TestApp]], MYB.BaseApplication.Security.Configuration"/>

</configSections>

Where TestApp.TestConfigElement, TestApp must be change for the assembly name of your Configuration Element child class was defined. And change the CustomConfigSection to whetever you like.

### System.webServer section

We recommend you to add the verbs “GET,HEAD,POST,DEBUG,PUT,DELETE”

<system.webServer>

<handlers>

<remove name="WebDAV"/>

<remove name="PageHandlerFactory-Integrated-4.0"/>

<remove name="PageHandlerFactory-Integrated"/>

<add name="PageHandlerFactory-Integrated" path="\*.aspx" verb="GET,HEAD,POST,DEBUG,PUT,DELETE" type="System.Web.UI.PageHandlerFactory" resourceType="Unspecified" requireAccess="Script" preCondition="integratedMode"/>

<add name="PageHandlerFactory-Integrated-4.0" path="\*.aspx" verb="GET,HEAD,POST,DEBUG,PUT,DELETE" type="System.Web.UI.PageHandlerFactory" resourceType="Unspecified" requireAccess="Script" preCondition="integratedMode,runtimeVersionv4.0"/>

</handlers>

</system.webServer>

## Global.asax

### Init application

Add Windsor, CoreInterface and CoreApplication reference from the BaseApp to your Global.asax project or where your web service application start.

In there add: BaseApp.Initializate(new ContainerManager()); to init the entire base.

In global asax do:

protected void Application\_Start(object sender, EventArgs e)

{

BaseApp.Initializate(new ContainerManager());

}

### Virtual Pages

Add this line to the BeginRequest if you want to enabled the Virtual Page System.

protected void Application\_BeginRequest(object sender, EventArgs e)

{

if (BaseApp.VirtualPageManager.ExecutePage(Path.GetFileName(Request.PhysicalPath), Request, Response))

{

CompleteRequest();

}

}

Note: VirtualPages has no session in HttpContext. Check 2.2 to see correct virtual pages

## General Parameters

# Inherit UI, WebServices, Controller and Service

## .aspx web services

You can inherit each aspx web service from BaseWebService in order to have acces to JsonSerializer, getRequest and shortcut access to BaseApp properties.

public partial class Login : BaseWebService

{

protected void Page\_Load(object sender, EventArgs e) { }

}

## BaseWebController and Virtual page in aspx

Add a page and inherit from BaseWebController to call this new page and redirect his query string to another page. The page will try to execute the virtual page and then if not exists redirecto to the requested page.

public partial class WebServices : BaseWebController

{

}

And then you can just call by ajax:

'WebServices.aspx?/Module/WebServicePage.aspx'

## UI Service

You can inherit from class UIService if you are creating pages class that want to has shorctus to BaseApp and be automatic resolve as ISingleton by the method BaseApp.UI<T>(); with interceptor enabled.

I.E.

Your class: public class YourClass : UIService

Your call: BaseApp.UI<YourClass>();

## BaseService

Inherit from BaseService if you want to add shortcuts to BaseApp functionality.

public class YourClass : BaseService

# BaseApp Methods

## BaseUserManager

IBaseUserManager

Is the master class for user administrator, create new user or remove and disconnect one existing.

## VirtualPageManager

IVirtualPagesManager

Giving a namepage (i.e. test.aspx), a HttpMethod and a function that recive parameters HttpRequest and HttpResponse.

bool RegisterPage(string pageName, HttpMethod httpMethod, Func<HttpRequest,HttpResponse,bool> func);

You can later execute this page with te method ExecutePage from here.

bool ExecutePage(string PageName, HttpRequest request, HttpResponse response);

## WSP – Warehouse Stored Procedures

IWSP

Normally is no need to use it. It has the acces to all stored procedures that the BaseApp use to work, is divided in two groups, Session and User.

## DB – Data Base (Ole)

### Generics methods

You can use all generic method to execute the sql command with params and return it like you want.

T Add<T>(string sTable, OleDbParameter[] dbParams, OleDbTransaction Trx = null);

T Delete<T>(string sTable, OleDbParameter[] dbParams, OleDbTransaction Trx = null);

T Update<T>(string sTable, OleDbParameter[] dbParams, OleDbTransaction Trx = null);

T Get<T>(string sAction, OleDbParameter[] dbParams, OleDbTransaction Trx = null);

T Emun<T>(string sTable, OleDbTransaction Trx = null);

T GetOne<T>(string sTable, OleDbParameter[] dbParams, OleDbTransaction Trx = null);

T GetAll<T>(string sTable, OleDbTransaction Trx = null);

T Execute<T>(string sSqlCmd, OleDbParameter[] dbParams, OleDbTransaction Trx = null);

T Execute<T>(string sSqlCmd, OleDbTransaction Trx = null);

T Execute<T>(IStoredProcedure<T> SP);

The procedures Add, Delete, etc.. only execute the sp called like this “AddTable” and “DeleteTable”. Is only a string conversion.

Use Execute to normal execute a stored procedure object, a stored procedure name as string or just a direct query.

The type <T> is the type that the result will be casted.

DataSet ds = Execute<DataSet>(“CreateUser”, dbParams);

object ds = Execute<object>(“CreateUser”, dbParams);

### GetEscalar

Type GetEscalar<Type>

Return the first row and first column casting as the type that you specify.

i.e.

if(GetEscalar<bool>(“IsValid”,dbParams)) { return “Valid user” }

### GetRow/ColumnList

List<Type> GetColumnList<Type>

List<Type> GetRowList<Type>

Get a DataTable or execute a procedure and return:

For GetColumnList: All columns of the first row as a list of the type specify.

For GetRowList: All rows value of the first column as a list of the type specify.

List<int> listOfIntGP = GetRowList<int>(“GetAllGeneralParameterValuesOrderer”);

### StoredProcedure objects

IStoredProcedure<T> StoredProcedure<T>

The purpose of create a StoredProcedure object is o predefine each stored procedure to be execute, how his parameters are given and how and what return.

i.e.

public IStoredProcedure<int> SetGeneralParameter(int implementationID,

string parameterName, string type, string value, OleDbTransaction Trx = null)

{

List<OleDbParameter> dbParams = new List<OleDbParameter>

{

DB.CreateParameter("@ImplementationID", implementationID, OleDbType.Integer),

DB.CreateParameter("@GeneralParameterID", parameterName, OleDbType.VarChar),

DB.CreateParameter("@Type", type, OleDbType.VarChar),

DB.CreateParameter("@Value", value, OleDbType.VarChar)

};

return DB.StoredProcedure<int>("SetGeneralParameter", dbParams, Trx);

}

Then you can execute this SP using .Execute(), .Execute(IDB) or just DB.Execute(IStoredProcedure):

BaseApp.WSP.SetGeneralParameter(ImplementationID, parameterName, typeof(T).ToString(), value.ToString()).Execute(BaseApp.DB);

## Session

IAppSession

You can check if IsLogged, login a user, logout it or validate a username. It contain the session string and the user object access that is logged.

## ContainerManager

IContainerManager

Return the container manager to use for:

T Resolve<T>

T[] ResolveAll<T>

void Release(object obj)

Or use the method Extended to return a IWindsorContainer. This will has all Windsor properties and methods.

## MongoDB – NoSQL Data Base

IMongoDataService MongoDB()

Return the mongo data base interface and then use Collection<T>(string collectionName) to get a collection of the specify document type.

IMongoDataService<TDocument> MongoDB<TDocument>();

Use the other form to call MongoDB if you allways will use the same type of document. Yo can shortcut this like this:

public IMongoDataService<BsonDocument> MDB {

get { return BaseApp.MongoDB<BsonDocument>():

}

## BaseConfigManager

IBaseConfigurationManager BaseConfigManager()

Get acces to the configuration.

string PhysicalAppDir { get; }

string VirtualAppDir { get; }

string UrlHost { get; }

string LogDir { get; }

IConfigConnectionSection Connection { get; }

And in extended method you can get IGeneralParameters

You can get general parameters for the Implementation and host, or just get the default that is the same for the implementation 0. T is the returned value to be casted.

T Get<T>(string parameterName);

T GetDefault<T>(string parameterName);

Or just set a new GP.

void Set<T>(string parameterName, T value);

void SetDefault<T>(string parameterName, T value);

IBaseConfigurationManager<TConfigElement> BaseConfigManager<TConfigElement>() where TConfigElement : ConfigurationElement

Is used to configure a ConfigurationElement. See point ***1.1.5 Custom Config Section***

## BaseUser

IBaseUser BaseUser(long userID, int implementationID)

IBaseUser BaseUser(long userID)

IBaseUser BaseUser()

Resolve an specific baseUser. If you are not using multiple implementations you can just use userID.

BaseUser() will resolve the actual logged user of the session.

## UI

Check point ***2.3 UI Service***

## Resolve

Is only a shortcut to ContainerManager in order to access more easy to resolve interfaces.

public static Interface Resolve<Interface>(IDictionary dictionary)

{

return ContainerManager.Resolve<Interface>(dictionary);

}

# Lifestyles and Base Interfaces

## General Use

## Multiple lifestyles

## Windsor LifeStyles

## Base Custom LifeStyles

# Create Virtual Pages

# Helpers and Extensions

JsonTools

Give tools to administrate Json array and json objects.

GuidGenerator

Generate new guids.

TypeExtension

Extend parsed type, object, SqlDbType and OleDbType.

CurrencyExt

Extend decimal to convert is as currency and get symbols

TypeExt

Extend to get ParentTypes of Type class.

ObjectExt

Extend to give the change to all object type to return a string that represent a properties list.

Dictionary Extended

Extend Dictonary class in order to have dictonary with more keys.

Dictionary<TKey1, TKey2, TValue>

Dictionary<TKey1, TKey2, TKey3, TValue>

Dictionary<TKey1, TKey2, TKey3, TKey4, TValue>

DataTableExt

Extend to parse from columns and rows from datatable to a list or JsonObjects.

List<Type> RowsToList<Type>

List<Tuple<TypeA, TypeB>> RowsToList<TypeA, TypeB>

List<Tuple<TypeA, TypeB, TypeC>> RowsToList<TypeA, TypeB, TypeC>

Change DataTable rows to a list of specific casted types.

DataSetExt

Same of DataTable but for DataSet and other functions.

Dictionary<TKey, TValue> RowToDic<TKey,TValue>

Cast 2 specific row as a new dictonary.

T ToEscalar<T>(this DataSet ds, string fieldName)

Cast dataset firs table specific row to a single object of the specify type.

# Multiple Implementations

For get multiple implementations declare a class that inherit from interface IImplementation

public class LifestyleImplementation : IImplementation

{

public int GetImplementation()

{

//Here your code to specify how to resolve each time the implementation.

Return 0;

}

}

Then when you want to resolve your custom class by implementation just inherit his interface from IImplementation

public class ImplementationHome : IImpHome

{

public ImplementationHome(int implementationID)

{

}

}

And then your IB

public interface IImpHome: IPerImplementation

{

}

Remember the class that want to resolve by Implementation must have the constructor implementatioID

# Data Base Squema