

Business Logic Toolkit for .NET www.bltoolkit.net

Business Logic Toolkit is a set of components to simplify .NET application development customize for your applications. It is written in C# and compatible with .NET Framework.

Version 3.1

- [BLToolkit 3.1](#) Source code
- [BLToolkit 3.1 Dev](#) Full dev version

Current SVN working copy

- [BLToolkit](#) SVN checkout source code
- [BLToolkit Dev](#) SVN checkout full dev version

Current SVN snapshot

- [bltoolkit.zip](#) Source code
- [bltoolkit_dev.zip](#) Full dev version
- [bltoolkit_bin.zip](#) Binaries

ChangeLog.txt

```
v 3.2 (working)
```

```
Demo (see the dev version of the project)
```

- ```
- Linq.Demo.
- WebServices powered by BLToolkit.
- Partial.Trust demo.
```

```
Bug fixes
```

- ```
- Extension assemblies are ignored in TypeFactory.Get  
- Default parameter values were ignored by OverloadAs  
- [Parameter(null)] wasn't setting Nullable<> and Ed  
- DefaultTypeBuilder throwing exception if [Parameter
```

EditableValue<Nullable<>> generated properties when constructor accepting value type

- DateTimeOffset wasn't handled in DataReaderMapper (

Enhancement

- Linq provider for supported databases.
- PostgreSQL Data Provider added.
- Informix Data Provider added.
- DB2 Data Provider added.
- New SqlIgnore attribute.
- New SqlBuilder class.
- Partial trust support.
- Mapping to/from private classes (FW 3.x only).
- New ParamTypeName attribute for SqlServer2008 table
- EditableArrayList & BindingListImpl implements INot

v 3.1

Bug fixes

- MapValueAttribute ignored for nullable enums.
- MappingSchema.ConvertChangeType does not handle nu
- EditableArrayList indexer add/remove bug.
- ParameterAttribute doesn't work with Nullable<T> and when 'T' constructor is not trivial.
- BindingListImpl.AddNew does not fire ListChanged ev
- Some inner types are missing in ObjectMappers, some
- Setter for ObjectBinder.ItemType fires OnListChange
- Convert<To,From> does not handle `Nullable<To>' fro
- EditableXmlDocument fires PropertyChanged event bet

Enhancement

- CHM help file provided. See the Doc folder.
- New InstanceCache attribute.

- DataAccessor support for IEnumerable<T>.
- Multiple duck. See DuckTyping.Aggregate methods.
- New Overload aspect.
- Own configuration section.
- TypeFactory.GetType() is public now.
- Simplified GenerateAttributeAttribute usage.

v 3.0

Breaking changes

- No more support for .Net 1.1 Framework.

Bug fixes

- EmitHelper ldc_i4_ emits wrong opcode for numbers 1
- Some opcodes were not implemented.
- MappingSchema.MapResultSets fails with NullReference
- Nullable enums were not handled properly.
- BindingList.GetItemSortedPosition() returns wrong :
- Fixed AsyncAspect and other type builders compatib:
- DbMananger.DiscoverSpParameters fires both OpenConn

Enhancement

- DateTimeOffset type support.
- MemberAccessor.CloneValue to avoid boxing.
- Named arguments support for GenerateAttributeTypeBu
- Extended support for IsNull & Nullable types in Mer
- Support for enum values in EmitHelper.LoadWellKnowr
- Generic GetValue/SetValue methods for ValueHolders.
- Different projects to support FW 2.0 & FW 3.5.
- LinqMetadataProvider to support Linq To SQL attribu
- Relocated MetadataProvider's to BLToolkit.Reflectio
- New DbManager ctors to support manual/custom config
- Ability to configure SqlQueryAttribute dynamically.

- Assembly is signed.
- New ActualTypeAttribute to associate DataAccessor's
- New AutoImplementInterfaceAttribute to automatically
- New TypeAccessor.AssociateType method and TypeAccess
- to control interface/type implementation.
- DataException now has the 'Number' property.
- New ClearCache aspect.

v 2.0.7

Bug fixes

- Oracle/Odp does not handle colon (':') in ConvertTy
- Property change builder will use op_Inequality oper
- property values.
- DbManager.ExecuteList<T>(params object[]) add an ex
- EditableObject.Clone does reference cloning of its
- EditableObject.Clone does not track dirty state of
- Generated assembly name ends with '.dll'.
- As a result, System.Reflection.Assembly.Load() tries
- and fails.
- Oracle treat '\r' as an invalid character.
- DataAccessorBuilder calls DbProvider.Convert(NameTo
- parameters.
- SqlQueryBase.GetKeyFieldList was incorrectly marked
- TypeAccessorBuilder fails to build a getter/setter
- EditableXmlDocument fails to reject changes if their
- deletion.

Enhancement

- FdpDataProvider.QuoteIdentifiers option for FB data
- GenerateAttribute attribute to specify any custom at
- member.
- SQLite data provider.
- DeriveParameters implemented for MsAccess data prov

- XmlIncludeAbstractAttribute to simplify the xml ser type.
- Better support for internal types/members. The beha Configuration.EditableObjectUsesMemberwiseEquals pr
- VS 2008 support.

v 2.0.6

Demo

- MS PetShop 4.0 powered by BLToolkit (see the dev ve

Bug fixes

- DbManager.OutputParameter lacks overload with 'size
- CacheAspect hungs a desktop application on exit.
- ExecuteResultSet fails on many-to-one relations.
- TypeBuilder fails to override a method with generic

Enhancement

- ParamSize & ParamDbType attributes to tweak DataAcc ones.
- All 'emit.stloc' immediately followed by 'emit.ldloc' 'emit.dup' command.
Lots of redundant local variables were removed.
- UnitTests for Oracle, Firebird & Access (Jet)
- DataSetTableAttribute is handled by all DataSet/Dat DataAccessBuilder.
- ClearCache method for CacheAspect.
- Metadata provider for mapping.
- Ability to specify global assembly version & strong
- AsyncAspect to execute any method asynchronously.
- Added support for ConnectionStrings config settings
- Editable XmlDocument.
- Added support for IDictionary, IDictionary<,>, IList

accessor return types.

v 2.0.5

Breaking changes

- `DbManager.InitCommand` renamed to `OnInitCommand`.

Bug fixes

- `System.Reflection.Emit.AssemblyBuilder` assemblies a
- `RegexAttribute` - now properly returns `valid==true` (match.
- `EditableObject` serialization issue
- `StandardPropertyDescriptor.CheckNull` was always ret handler was returning null. Now configurable via Co
- `PropertyChangedBuilder` was crashing when property v
- `ObjectBinder.IsNull` was serialized to resources
- Oracle provider now handles arrays of nullable type
- `DataAccessor.PrepareParameters` was invoked only to
- `DataAccessor.PrepareParameters` lacks `DbManager` whic
- `DbManager.InitParameters` did not handle decimal typ
- `MappingSchema.ValueToEnum` fails with underlying typ
- `CounterAspect` catches the `OnCatch` event now to cour
- `DbManager.ExecuteForEach` does not call `InitParamete`

Enhancement

- Ability to specify `RegexOptions` for `RegexAttribute`. call to `IsValid`
- `Destination` attribute works with scalar types.
- `DataAccessorBuilder` generates getters/setters for a
- Ability to specify globally whether strings should b
- Ability to specify whether `PropertyChange` notificat underlying value is changed and not when setter is
- `XmlDocument` type support.

- `DbManager.SetParameters()` accepts arrays of `IDbData`
- `DataSetTableAttribute` for `DataAccessorBuilder` to ex
- Default `CacheAspect.IsWeak` value changed to false.
- Cleanup thread for `CacheAspect`.
- `MemberMapper` type can be specified directly in the
- Events for `DbManager.OnBeforeOperation`, `OnAfterOper`
`OnInitCommand`.

v 2.0.4

Bug fixes

- By default, the mapper included const fields into m
- If enum does not have explicit mapping, data accesss
does not convert it to underlying type.
- `MemberMapper` fails to convert an object to its base
- `BindingListImpl.RemoveSort` was not firing `ListChang`
- `TypeAccessor.CreateExtendedPropertyDescriptors` did
with 3rd+ level nested properties.
- Object holder ignored `NoInstanceAttribute`.
- `ValidatorContext` caches `NullValue`.
- `EditableObject.RejectChanges/RejectMemberChanges` we
for reverted fields.
- Fixed the generating `IsValueType && !IsPrimitive` re

Enhancement

- `Operator<T>` is extensible now.
- `ObjectBinder` can now control lifetime (Disposal) of
- DuckTyping.
- `ParamNullValue` attribute.
- Virtual method `DataProviderBase.PrepareCommand` to l
statement befory query.
- Virtual method `CacheAspect.GetCache`. Can be used to

v 2.0.3

Bug fixes

- Race condition in MappingSchema.GetObjectMapper.
- DbManager will not try to close the connection in t
- MinValue/MaxValue attributes now determine value ty of test value.
- Fixed obscure case when object is removed from Edit ItemPropertyChanged delegate is in process of being not yet invoked.

Enhancement

- HttpReader SOAP support.
- Convert<T,P> uses type cast operators when availabi
- Third party DataProvider registration via appsetting
- More specification adherent sorting behavior imple IBindingList/IBindingListView in BindingListImpl.
- EditableArrayList (potentially breaking changes) he IBindingList/IBindingListView and their base intert
- BindingListImpl is changed to be fully specificatio object lists/collection types.
- 2.0.1 version obsolete methods have been removed.

v 2.0.2

Bug fixes

- Abstract class builder override virtual final metho
- Append records to a log file.
- ObjectBinder's AllowNew, AllowEdit, AllowRemove pro

Enhancement

- New DataAccessor<T,A> class.

- `Direction.ReturnValueAttribute` takes the Member pair
- New property Object of the `InterceptCallInfo` class, reference.

Business Logic Toolkit for .NET www.bltoolkit.net

BLToolkit is licensed under [MIT License](#):

Copyright © 2010 www.bltoolkit.net

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Business Logic Toolkit for .NET www.bltoolkit.net

-
- Aspects
 - ComponentModel
 - Data
 - DataAccess
 - EditableObjects
 - Mapping
 - Reflection
 - TypeBuilder
 - Validation
-

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

All BLToolkit aspects are implemented as attributes and have to be applied to abstract or virtual members of an abstract class. You need to use **TypeAccessor<T>.CreateInstance()** method to create an instance of the class. BLToolkit class builder creates an actual class and applies aspects to abstract and virtual members by overriding the members and emitting (see **System.Reflection.Emit** namespace) additional code around them.

- [Async aspect](#)

Cache aspect

- [Cache aspect](#)
- [NoCache attribute](#)
- [ClearCache aspect](#)

- [Counter aspect](#)

- [Logging aspect](#)

- [Mixin aspect](#)

- [NotNull attribute](#)

- [Overload aspect](#)

Business Logic Toolkit for .NET www.bltoolkit.net

This aspect simplifies asynchronous operations. Note the AsyncAspect is not complete. If you apply AsyncAspect to an abstract method, they will be ignored.

AsyncAspect.cs

```
using System;
using System.Threading;

using NUnit.Framework;

using BLToolkit.Aspects;
using BLToolkit.Reflection;

namespace HowTo.Aspects
{
    public abstract class AsyncTestObject
    {
        // This is a member we will call asynchronously
        //
        public int Test(int intVal, string strVal)
        {
            Thread.Sleep(200);
            return intVal;
        }

        // Begin async method should take the same parameters as the original method
        // Two additional parameters can be provided: 'Begin' and 'End'
        // 'Begin' prefix is a part of the default name for the Begin method
        //
        [Async] public abstract IAsyncResult BeginTest(int intVal, string strVal);
        [Async] public abstract IAsyncResult BeginTest(int intVal, string strVal, object state);
        [Async] public abstract IAsyncResult BeginTest(int intVal, string strVal, object state, AsyncCallback callback);

        // End async method should take IAsyncResult
    }
}
```

```

// 'End' prefix is a part of the default name
//
[Async] public abstract int          EndTest

// Begin/End naming convention is not required
// if you provide the target method name as a prefix
//
[Async("Test")]
public abstract IAsyncResult AnyName(int intValue)

// Here we provide the parameter list along with the return type
//
[Async("Test", typeof(int), typeof(string))]
public abstract int          AnyName(IAsyncResult ar)
}

```

```

[TestFixture]
public class AsyncAspectTest
{
    [Test]
    public void AsyncTest()
    {
        AsyncTestObject o = TypeAccessor<AsyncTestObject>.Create();

        IAsyncResult ar = o.BeginTest(1, "10");
        Assert.AreEqual(1, o.EndTest(ar));
    }

    private static void Callback(IAsyncResult ar)
    {
        Console.WriteLine("Callback");

        AsyncTestObject o = (AsyncTestObject)ar.AsyncState;
        o.EndTest(ar);
    }

    [Test]

```

```

public void CallbackTest()
{
    AsyncTestObject o = TypeAccessor<AsyncTestObject>.Get(AsyncTestObject.class);
    o.BeginTest(2, null, Callback, o);
}

[Test]
public void AnyNameTest()
{
    AsyncTestObject o = TypeAccessor<AsyncTestObject>.Get(AsyncTestObject.class);
    IAsyncResult ar = o.AnyName(2, null, null);
    Assert.AreEqual(2, o.AnyName(ar));
}
}
}

```

BLToolkit type builder will generate the following for the class above:

```

[BLToolkitGenerated]
internal delegate int TestObject$Test$Delegate(int, string, string);

[BLToolkitGenerated]
public sealed class AsyncTestObject : HowTo.Aspects.AsyncTestObject
{
    public override IAsyncResult BeginTest(int intValue, string strValue, string strValue)
    {
        AsyncAspectBuilder.InternalAsyncResult r = new AsyncAspectBuilder.AsyncTestObject$Test$Delegate(
            this, intValue, strValue, strValue);
        r.Delegate = new TestObject$Test$Delegate(this, intValue, strValue, strValue);
        r.InnerResult = r.Delegate.BeginInvoke(intValue, strValue, strValue);
        return r;
    }

    public override IAsyncResult BeginTest(int intValue, string strValue, string strValue)

```

```

    {
        AsyncAspectBuilder.InternalAsyncResult r = ne

        r.Delegate      = new TestObject$Test$Delegat
        r.AsyncCallback = callback;
        r.InnerResult   = r.Delegate.BeginInvoke(int\

        return r;
    }

    public override IAsyncResult BeginTest(int intVal
    {
        AsyncAspectBuilder.InternalAsyncResult r = ne

        r.Delegate      = new TestObject$Test$Delegat
        r.AsyncCallback = callback;
        r.InnerResult   = r.Delegate.BeginInvoke(int\

        return r;
    }

    public override int EndTest(IAsyncResult asyncRes
    {
        AsyncAspectBuilder.InternalAsyncResult r = (A

        return ((TestObject$Test$Delegate)r.Delegate)
    }
}

```

Business Logic Toolkit for .NET www.bltoolkit.net

This aspect simplifies method overloading. Note the OverloadAspect is not compatible with the OverloadAspect, if you apply them along with the OverloadAspect to an abstract method, they will be ignored.

OverloadAspect.cs

```
using NUnit.Framework;

using BLToolkit.Aspects;
using BLToolkit.Reflection;

namespace HowTo.Aspects
{
    public abstract class OverloadTestObject
    {
        // This is a member we will overload.
        //
        public int Test(int intVal, string strVal)
        {
            return intVal;
        }

        // Overloaded methods simply calls a base method
        // and has a few parameters less or more.
        //
        [Overload] public abstract int Test(int intVal);
        [Overload] public abstract int Test(string strVal);
        [Overload] public abstract int Test(int intVal, string strVal);
    }

    [TestFixture]
    public class OverloadAspectTest
    {
        [Test]
        public void OverloadTest()
        {
        }
    }
}
```

```

        {
            OverloadTestObject o = TypeAccessor<OverloadTestObject>.Of(1, "str");
            Assert.AreEqual(1, o.Test(1));
            Assert.AreEqual(0, o.Test("str"));
        }
    }
}

```

BLToolkit type builder will generate the following for the class above:

```

[BLToolkitGenerated]
public sealed class OverloadTestObject : OverloadTestObject
{
    [BLToolkitGenerated]
    public override int Test(int intVal)
    {
        return this.Test(intVal, string.Empty);
    }

    [BLToolkitGenerated]
    public override int Test(string strVal)
    {
        return this.Test(0x0, strVal);
    }

    [BLToolkitGenerated]
    public override int Test(int intVal, string strVal)
    {
        return this.Test(intVal, strVal);
    }
}

```

Business Logic Toolkit for .NET www.bltoolkit.net

This aspect helps to cache method calls. The aspect uses input method parameters to create cache keys and string type of the method parameters are used to create a cache key. The `CacheAspect.IsCacheableParameterType` property is used to provide a delegate providing custom logic to determine if a parameter is cacheable.

CacheAspect.cs

```
using System;
using System.Reflection;

using NUnit.Framework;

using BLToolkit.Aspects;
using BLToolkit.Reflection;

namespace HowTo.Aspects
{
    public abstract class TestClass
    {
        public static int Value;

        // This is a method we will cache. Cached return value is used.
        // We will change the 'Value' field outside of this method.
        //
        [Cache(MaxCacheTime=500, IsWeak=false)]
        public virtual int CachedMethod(int p1, int p2)
        {
            return Value;
        }

        public static TestClass CreateInstance()
        {
            // Use TypeAccessor to create an instance
            //
            return TypeAccessor<TestClass>.CreateInstance();
        }
    }
}
```

```

    }
}

[TestFixture]
public class CacheAspectTest
{
    [Test]
    public void Test1()
    {
        TestClass tc = TestClass.CreateInstance()

        DateTime begin = DateTime.Now;

        // Initial setup for the test static variable
        //
        TestClass.Value = 777;

        while (tc.CachedMethod(2, 2) == 777)
        {
            // This change will not affect the TestClass
            //
            TestClass.Value++;
        }

        double totalMilliseconds = (DateTime.Now

        Assert.GreaterOrEqual(totalMilliseconds,

    }

    [Test]
    public void Test2()
    {
        TestClass tc = TestClass.CreateInstance()

        // Return value depends on parameter value
        //
        TestClass.Value = 1; Assert.AreEqual(1, tc

```

```

TestClass.Value = 2; Assert.AreEqual(1, t
TestClass.Value = 3; Assert.AreEqual(3, t

// However we can clear cache manually.
// For particular method:
//
CacheAspect.ClearCache(typeof(TestClass),
TestClass.Value = 4; Assert.AreEqual(4, t

// By MethodInfo:
//
MethodInfo methodInfo = tc.GetType().GetM
CacheAspect.ClearCache(methodInfo);
TestClass.Value = 5; Assert.AreEqual(5, t

// For the all cached methods.
//
CacheAspect.ClearCache();
TestClass.Value = 6; Assert.AreEqual(6, t
    }
}
}

```

If we decompile the actual emitted TestClass class, we may see something like the follow

```

[BLToolkitGenerated]
public sealed class TestClass : HowTo.Aspects.TestCla
{
    private static MethodInfo _methodInfo;
    private static IInterceptor _interceptor;

    public override int CachedMethod(int p1, int p2)
    {
        int returnValue = 0;

        if (_methodInfo == null)
        {

```

```

        _methodInfo = new CallMethodInfo((MethodInfo)
    }

    InterceptCallInfo info = new InterceptCallInfo()

    info.Object          = this;
    info.CallMethodInfo  = _methodInfo;
    info.ParameterValues[0] = p1;
    info.ParameterValues[1] = p2;
    info.ReturnValue     = returnValue;
    info.InterceptResult = InterceptResult.Complete;
    info.InterceptType   = InterceptType.Before;

    if (_interceptor == null)
    {
        _interceptor = new CacheAspect();
        _interceptor.Init(_methodInfo, "MaxCacheTTL");
    }

    // 'BeforeCall' step checks if the method is cached
    // If it is and the cache is not expired, the method is called
    // return value and output parameters with the cache
    // sets info.InterceptResult to InterceptResult.Complete
    // See the CacheAspect.BeforeCall method for more details
    //
    _interceptor.Intercept(info);

    returnValue = (int)info.ReturnValue;

    if (info.InterceptResult != InterceptResult.Complete)
    {
        // If the method call is not cached, target method is called
        //
        returnValue = base.CachedMethod(p1, p2);

        info.ReturnValue     = returnValue;
        info.InterceptResult = InterceptResult.Complete;
    }
}

```

```
        info.InterceptType    = InterceptType.After

        // 'AfterCall' step stores parameters and
        // See the CacheAspect.AfterCall method for
        //
        _interceptor.Intercept(info);

        returnValue = (int)info.ReturnValue;
    }

    return returnValue;
}
}
```

Business Logic Toolkit for .NET www.bltoolkit.net

The **ClearCache** attribute clears cached data for the provided method.

ClearCacheAspect.cs

```
using System;

using NUnit.Framework;

using BLToolkit.Aspects;
using BLToolkit.Reflection;

namespace HowTo.Aspects
{
    [TestFixture]
    public class ClearCacheAspect
    {
        public abstract class TestClass
        {
            public static int Value;

            // This is a method we will cache. Cached
            // We will change the 'Value' field outsi
            //
            [Cache(MaxCacheTime=500, IsWeak=false)]
            public virtual int CachedMethod(int p1, :
            {
                return Value;
            }

            // This method clears the CachedMethod ca
            //
            [ClearCache("CachedMethod")]
            public abstract void ClearCache();
        }
    }
}
```

```

// The CachedMethod is specified by name
// Also you can use declaring method type
//
[ClearCache("CachedMethod", typeof(int),
public abstract void ClearCache2();

// This method clears all caches for prov
//
[ClearCache(typeof(TestClass))]
public abstract void ClearAll();

// This method clears all caches for curi
//
[ClearCache]
public abstract void ClearAll2();

public static TestClass CreateInstance()
{
    // Use TypeAccessor to create an inst
    //
    return TypeAccessor<TestClass>.Create
}
}

[Test]
public void Test()
{
    TestClass tc = TypeAccessor<TestClass>.Cr

    TestClass.Value = 1;

    int value1 = tc.CachedMethod(1, 2);

    TestClass.Value = 2;

    int value2 = tc.CachedMethod(1, 2);

```

```

        // The cached values are equal.
        //
        Assert.AreEqual(value1, value2);

        tc.ClearCache();

        TestClass.Value = 3;

        // Previous and returned values are not equal.
        //
        Assert.AreNotEqual(value1, tc.CachedMethod());

        tc.ClearCache2();
    }
}
}

```

BLToolkit type builder will generate the following for the class above:

```

[BLToolkitGenerated]
public sealed class TestClass : ClearCacheAspect.TestClass
{
    private static MethodInfo _methodInfo1;
    private static MethodInfo _methodInfo2;
    private static Type _type3;
    private static Type _type4;

    public override int CachedMethod(int p1, int p2)
    {
        // Method implementation.
    }

    public override void ClearCache()
    {
        try
        {
            // Here should be main method implementation.
        }
    }
}

```

```

        // It is empty as this method does nothing
    }
    finally
    {
        if (_methodInfo1 == null)
        {
            _methodInfo1 =
                ClearCacheAspect.GetMethodInfo(th
        }

        CacheAspect.ClearCache(_methodInfo1);
    }
}

public override void ClearCache2()
{
    try
    {
    }
    finally
    {
        if (_methodInfo2 == null)
        {
            _methodInfo2 = ClearCacheAspect.GetMe
                this,
                null,
                "CachedMethod",
                new Type[] { typeof(int), typeof(
        }

        CacheAspect.ClearCache(_methodInfo2);
    }
}

public override void ClearAll()
{
    try

```

```

    {
    }
    finally
    {
        if (_type3 == null)
        {
            _type3 = ClearCacheAspect.GetType(th:
        }

        CacheAspect.ClearCache(_type3);
    }
}

public override void ClearAll2()
{
    try
    {
    }
    finally
    {
        if (_type4 == null)
        {
            _type4 = ClearCacheAspect.GetType(th:
        }

        CacheAspect.ClearCache(_type4);
    }
}
}
}

```

Business Logic Toolkit for .NET www.bltoolkit.net

Cache aspect can be applied to entire class. In this case all virtual and abstract member exclude particular members from caching.

NoCache.cs

```
using System;

using NUnit.Framework;

using BLToolkit.Aspects;
using BLToolkit.Reflection;

namespace HowTo.Aspects
{
    [Cache]
    public abstract class NoCacheTestClass
    {
        public static int Value;

        public virtual int CachedMethod(int p1, int p2)
        {
            return Value;
        }

        [NoCache]
        public virtual int NoCacheMethod(int p1, int p2)
        {
            return Value;
        }

        public static NoCacheTestClass CreateInstance()
        {
            // Use TypeAccessor to create an instance
            //
        }
    }
}
```

```

        return TypeAccessor<NoCacheTestClass>.Create();
    }
}

[TestFixture]
public class NoCacheAttributeTest
{
    [Test]
    public void Test()
    {
        NoCacheTestClass t = TypeAccessor<NoCacheTestClass>.Create();

        NoCacheTestClass.Value = 1; Assert.AreEqual(1, NoCacheTestClass.Value);
        NoCacheTestClass.Value = 2; Assert.AreEqual(2, NoCacheTestClass.Value);

        NoCacheTestClass.Value = 3; Assert.AreEqual(3, NoCacheTestClass.Value);
        NoCacheTestClass.Value = 4; Assert.AreEqual(4, NoCacheTestClass.Value);
    }
}
}

```

Business Logic Toolkit for .NET www.bltoolkit.net

This aspect helps to collect statistical information for the members it is applied to.

CounterAspect.cs

```
using System;
using System.Reflection;

using NUnit.Framework;

using BLToolkit.Aspects;
using BLToolkit.Reflection;

namespace HowTo.Aspects
{
    [TestFixture]
    public class CounterAspectTest
    {
        public abstract class TestClass
        {
            // This is a method we collect statistic
            // Actually the entire class or even a base class
            // can be decorated with the attribute.
            //
            [Counter]
            public virtual void TestMethod()
            {
            }
        }

        [Test]
        public void Test()
        {
            TestClass t = TypeAccessor<TestClass>.Create();
        }
    }
}
```

```

        for (int i = 0; i < 10; i++)
            t.TestMethod();

        MethodInfo methodInfo = typeof(TestClass).GetMethod("TestMethod");
        MethodCallCounter counter = CounterAspectBuilder.Build(counterAspect, methodInfo);

        Assert.AreEqual(10, counter.TotalCount);

        Console.WriteLine(@"
Method          : {0}.{1}
TotalCount      : {2}
ExceptionCount  : {3}
CachedCount     : {4}
CurrentCalls    : {5}
TotalTime       : {6}
MinTime         : {7}
MaxTime         : {8}
AverageTime     : {9}
");

        Console.WriteLine(
            counter.MethodInfo.DeclaringType.Name,
            counter.MethodInfo.Name,
            counter.TotalCount, // t
            counter.ExceptionCount, // c
            counter.CachedCount, // c
            counter.CurrentCalls.Count, // c
            counter.TotalTime, // t
            counter.MinTime, // r
            counter.MaxTime, // r
            counter.AverageTime); // a
    }
}
}

```

BLToolkit type builder will generate the following for the class above:

```

[BLToolkitGenerated]
public sealed class TestClass : CounterAspectTest.TestClass

```

```

{
    private static MethodInfo _methodInfo;
    private static IInterceptor _interceptor;

    public override void TestMethod()
    {
        if (_methodInfo == null)
        {
            _methodInfo = new MethodInfo((MethodI

        InterceptCallInfo info = new InterceptCallInt

        try
        {
            info.Object = this;
            info.CallMethodInfo = _methodInfo;
            info.InterceptResult = InterceptResult.Co
            info.InterceptType = InterceptType.Befo

            if (_interceptor == null)
            {
                _interceptor = new CounterAspect();
                _interceptor.Init(_methodInfo, null);
            }

            // 'BeforeCall' creates or gets a counter
            // registers the current call.
            // See the CounterAspect.BeforeCall metho
            //
            _interceptor.Intercept(info);

            if (info.InterceptResult != InterceptResu
            {
                // Target method call.
                //
                base.TestMethod();
            }
        }
    }
}

```

```

    }
}
catch (Exception exception)
{
    info.Exception          = exception;
    info.InterceptResult = InterceptResult.Co
    info.InterceptType    = InterceptType.OnCa

    // 'OnCatch' is required to count calls v
    //
    _interceptor.Intercept(info);

    if (info.InterceptResult != InterceptResu
    {
        throw;
    }
}
finally
{
    info.InterceptResult = InterceptResult.Co
    info.InterceptType    = InterceptType.OnFi

    // 'OnFinally' step adds statistic to the
    // See the CounterAspect.OnFinally method
    //
    _interceptor.Intercept(info);
}
}
}
}

```

The following picture shows the information collected for data accessors of the *BLToolki*

Counters - Windows Internet Explorer
 http://localhost:51456/Web/Admin/Counters.aspx

Counters

THE .NET PET SHOP

HOME > COUNTERS

ITEMS ORDERS COUNTERS

GC.CollectionCount: 2 Cache Cleanup Times: 20 Objects in Cache: 4
 GC.TotalMemory: 6M Total Cleanup Time: 00:00:00 Objects Expired: 0

Type	Method	Calls		Cache		Call Time			
		Count	Ex	In	From	Min	Average	Max	Total
OrderAccessor	GetAllOrderList	1	0	0	0	00:00:00.119	00:00:00.119	00:00:00.119	00:00:00.119
PetShopQuery<DB,ProductAccessor>	SelectAll	11	0	0	0	00:00:00	00:00:00.129	00:00:01.411	00:00:01.420
PetShopQuery<DB,ProductAccessor>	SelectByKey	8	0	0	0	00:00:00	00:00:00.001	00:00:00.007	00:00:00.010
ProductAccessor	GetAllItemList	3	0	0	0	00:00:00.002	00:00:00.247	00:00:00.736	00:00:00.741
ProductAccessor	GetCategory	2	2	8	0	00:00:00.001	00:00:00.067	00:00:00.134	00:00:00.135
ProductAccessor	GetItem	4	0	0	0	00:00:00.001	00:00:00.005	00:00:00.017	00:00:00.020
ProductAccessor	GetItemListByProductID	4	0	0	0	00:00:00	00:00:00.004	00:00:00.016	00:00:00.017
ProductAccessor	GetProductListByCategoryID	2	2	0	0	00:00:00	00:00:00.005	00:00:00.010	00:00:00.010
ProfileAccessor	CreateProfile	1	0	0	0	00:00:00.131	00:00:00.131	00:00:00.131	00:00:00.131
ProfileAccessor	GetCartItems	8	0	0	0	00:00:00	00:00:00.018	00:00:00.144	00:00:00.149
ProfileAccessor	GetUniqueIDAuth	4	0	0	0	00:00:00	00:00:00.001	00:00:00.004	00:00:00.004
ProfileAccessor	SetCartItems	4	0	0	0	00:00:00.002	00:00:00.039	00:00:00.117	00:00:00.159
ProfileAccessor	UpdateActivityAndUpdateDates	4	0	0	0	00:00:00.001	00:00:00.001	00:00:00.003	00:00:00.006

cache cleanup

BASED ON PET SHOP VERSION 4.0 - POWERED BY BLTOOLKIT FOR .NET

Internet | Protected Mode: Off 90%

© 2010 www.bltoolkit.net
 support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

This aspect allows logging some diagnostic information with minimum efforts. All you need to do is decorate only your base class. Diagnostic information will be logged for all virtual methods.

LoggingAspect.cs

```
using System;
using System.Threading;

using NUnit.Framework;

using BLToolkit.Aspects;
using BLToolkit.Reflection;

namespace HowTo.Aspects
{
    [TestFixture]
    public class LoggingAspectTest
    {
        [Log]
        public abstract class TestClass
        {
            // Here we customize the logging settings
            // This call will be logged in spite of the [Log] attribute
            //
            [Log(MinCallTime=50)]
            public virtual void Test1(int i)
            {
                Thread.Sleep(100);
            }

            // This call is not going to be logged (because of the [Log] attribute)
            //
            public virtual void Test2(DateTime dt)
            {
```

```

        Thread.Sleep(100);
    }

    // By default exception calls are logged
    //
    public virtual void Test3(string s)
    {
        throw new ApplicationException("Test
    }
}

[Test]
public void Test()
{
    // By setting MinCallTime to some value,
    // which is shorter than the provided va
    //
    LoggingAspect.MinCallTime = 1000;

    TestClass t = TypeAccessor<TestClass>.Cre

    t.Test1(1);
    t.Test2(DateTime.Now);

    try
    {
        t.Test3("3");
    }
    catch
    {
    }
}
}
}
}
}

```

Here is the logging output.

4/20/2008 11:19:44 PM: HowTo.Aspects.LoggingAspectTes

4/20/2008 11:19:46 PM: HowTo.Aspects.LoggingAspectTes
with exception 'System.Applica

Business Logic Toolkit for .NET www.bltoolkit.net

A mixin is a... well, [Wiki](#) describes it much better.

MixinAspect.cs

```
using System;

using NUnit.Framework;

using BLToolkit.Aspects;
using BLToolkit.Reflection;

namespace HowTo.Aspects
{
    [TestFixture]
    public class MixinAspectTest
    {
        public interface ITestInterface1
        {
            int TestMethod(int value);
        }

        public class TestInterface1Impl : ITestInterface1
        {
            public int TestMethod(int value) { return 1; }
        }

        public interface ITestInterface2
        {
            int TestMethod1(int value);
            int TestMethod2(int value);
        }

        public class TestInterface2Impl : ITestInterface2
        {
```

```

        public int TestMethod1(int value) { return 10; }
        public int TestMethod2(int value) { return 30; }
    }

    [Mixin(typeof(ITestInterface1), "_testInterface1")]
    [Mixin(typeof(ITestInterface2), "TestInterface2")]
    public abstract class TestClass
    {
        public TestClass()
        {
            _testInterface1 = new TestInterface1();
        }

        protected object _testInterface1;

        private ITestInterface2 _testInterface2;
        public ITestInterface2 TestInterface2
        {
            get { return _testInterface2 ?? (_testInterface2 = new TestInterface2()); }
        }

        [MixinOverride(typeof(ITestInterface2))]
        protected int TestMethod1(int value) { return 20; }
    }

    [Test]
    public void Test()
    {
        TestClass tc = TypeAccessor<TestClass>.GetInstance();
        ITestInterface1 i1 = (ITestInterface1)tc;
        ITestInterface2 i2 = (ITestInterface2)tc;

        Assert.AreEqual(10, i1.TestMethod(10));
        Assert.AreEqual(15, i2.TestMethod1(20));
        Assert.AreEqual(30, i2.TestMethod2(30));
    }
}

```

```
}
```

BLToolkit type builder will generate the following for the class above:

```
[BLToolkitGenerated]
public sealed class TestClass : MixinAspectTest.TestClass
{
    int MixinAspectTest.ITestInterface1.TestMethod(int value)
    {
        if (base._testInterface1 == null)
            throw new InvalidOperationException("'ITestInterface1' is null");

        return base._testInterface1.TestMethod(value);
    }

    int MixinAspectTest.ITestInterface2.TestMethod1(int value)
    {
        // The MixinOverride attribute enforces direct override
        //
        return base.TestMethod1(value);
    }

    int MixinAspectTest.ITestInterface2.TestMethod2(int value)
    {
        if (base.TestInterface2 == null)
            throw new InvalidOperationException("'ITestInterface2' is null");

        return base.TestInterface2.TestMethod2(value);
    }
}
```

Business Logic Toolkit for .NET www.bltoolkit.net

This attribute allows checking a virtual or abstract method parameter if it is null at runtime by checking null value at the beginning of the method.

NotNull.cs

```
using System;

using NUnit.Framework;

using BLToolkit.Aspects;
using BLToolkit.Reflection;

namespace HowTo.Aspects
{
    [TestFixture]
    public class NotNullTest
    {
        public abstract class TestObject
        {
            public virtual void Foo1(string str1, [NotNull] string str2, string str3);
            public virtual void Foo2(string str1, [NotNull] string str2, string str3);
            public virtual void Foo3(string str1, [NotNull] string str2, string str3);

            public static TestObject CreateInstance()
            {
                return new TestObject();
            }
        }

        [Test, ExpectedException(typeof(ArgumentNullException))]
        public void Test1()
        {
            TestObject o = TestObject.CreateInstance();

            o.Foo1("str1", null, "str3");
        }
    }
}
```

```

[Test]
[ExpectedException(typeof(ArgumentNullException))]
public void Test2()
{
    TestObject o = TestObject.CreateInstance();

    o.Foo2("str1", null, "str3");
}

[Test]
[ExpectedException(typeof(ArgumentNullException))]
public void Test3()
{
    TestObject o = TestObject.CreateInstance();

    o.Foo3("str1", null, "str3");
}
}
}

```

BLToolkit type builder will generate the following for the class above:

```

[BLToolkitGenerated]
public sealed class TestObject : NotNullTest.TestObject
{
    public override void Foo1(string str1, string str2, string str3)
    {
        if (str2 == null) throw new ArgumentNullException("str2");

        base.Foo1(str1, str2, str3);
    }

    public override void Foo2(string str1, string str2, string str3)
    {
        if (str2 == null) throw new ArgumentNullException("str2");

        base.Foo2(str1, str2, str3);
    }
}

```

```
}  
  
public override void Foo3(string str1, string str2, string str3)  
{  
    if (str2 == null) throw new ArgumentNullException("str2");  
    base.Foo3(str1, str2, str3);  
}  
}
```

Business Logic Toolkit for .NET www.bltoolkit.net

- [ObjectBinder](#)

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

The **ObjectBinder** component has been designed to simplify binding controls on a form to objects for FW 1.x. Since Microsoft has released FW 2.0 we can use the **BindingSource** component to bind controls to objects. However the **ObjectBinder** still can be useful as it supports a few features which are not available for the **BindingSource**. Those features are:

- Support for field binding along with property binding.
- Support for inner class field and property binding such as *Order.Address.Line1*.
- Support for the *ObjectView* feature which is available by assigning an object view type to the **ObjectBinder.ObjectViewType** property. An object view is an object that implements the **IObjectView** interface. This interface includes only one property - *object Object { get; set; }*. An object view can implement additional properties based on the associated object. The **ObjectBinder** will combine all of these properties with main object properties and create a single **PropertyDescriptor** collection. This feature can be used to separate UI presentation logic from business model objects and to keep them clean. *ObjectView* should be a stateless, lightweight object as its single instance can be assigned to many associated objects.
- The **ObjectBinder** is optimized for high performance applications such real-time multithreaded message processing and distribution banking systems. So it does not use reflection to access objects. The standard way (which is used by the **BindingSource**) is to call the **TypeDescriptor.GetProperties** method to get a **PropertyDescriptor** collection. This method creates property descriptors that access object properties by reflection. The **ObjectBinder** has its own mechanism to avoid unnecessary reflection and boxing/unboxing operations.

The dev version of BLToolkit contains a demo project (Demo\WinForms) showing the use of the **ObjectBinder**.

Business Logic Toolkit for .NET www.bltoolkit.net

The **DbManager** class is a high-level, data provider independent wrapper for **ADO.NET**. It has been designed to simplify working with database and encapsulates most of **ADO.NET** objects such as **Connection**, **Transaction**, **Command**, and **Parameter** in one single object.

-
- [Compare DbManager with ADO.NET](#)
-

How to configure

- [Method 1](#) using app.config.
 - [Method 2](#) programmatically.
 - [Method 3](#) by passing DataProvider, Connection or Transaction object.
-

- [Data Providers](#)
-

- [SetCommand](#) Using SQL statements
 - [SetSpCommand](#) Using stored procedures
 - [Parameters](#)
 - [Prepare](#)
 - [Transactions](#)
 - [Close](#)
-

- [ExecuteDataSet](#)
 - [ExecuteDataTable](#)
 - [ExecuteDictionary](#)
 - [ExecuteForEach](#)
 - [ExecuteList](#)
 - [ExecuteNonQuery](#)
 - [ExecuteObject](#)
 - [ExecuteReader](#)
 - [ExecuteResultSet](#) Complex mapping
 - [ExecuteScalar](#)
 - [ExecuteScalarDictionary](#)
 - [ExecuteScalarList](#)
-
-

© 2010 www.bluetooth.net
support@bluetooth.net

Business Logic Toolkit for .NET www.bltoolkit.net

This example contains two demos to demonstrate the difference between **ADO.NET** and **Entity Framework**. The first demo is a simple data access method which takes one parameter and returns a list of objects. All routine mapping is done in the database.

AdoDemo.cs

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data.SqlClient;

using NUnit.Framework;

namespace HowTo.Data
{
    [TestFixture]
    public class AdoDemo
    {
        // Typified definition of the Gender database
        //
        public enum Gender
        {
            Female,
            Male,
            Unknown,
            Other
        }

        // Business object.
        //
        public class Person
        {
            public int ID { get; set; }
            public string FirstName { get; set; }
            public string MiddleName { get; set; }
        }
    }
}
```

```

        public string LastName    { get; set; }
        public Gender Gender      { get; set; }
    }

    // ADO.NET data access method.
    //
    public List<Person> GetList(Gender gender)
    {
        // Map the typed parameter value to it
        //
        string paramValue = "";

        switch (gender)
        {
            case Gender.Female: paramValue = "F";
            case Gender.Male:    paramValue = "M";
            case Gender.Unknown: paramValue = "U";
            case Gender.Other:   paramValue = "O";
        }

        // Read a database configuration string.
        //
        string cs = ConfigurationManager.ConnectionStrings["dbcs"].ConnectionString;

        // Create and open a database connection.
        //
        using (SqlConnection con = new SqlConnection(cs))
        {
            con.Open();

            // Create and initialize a Command object
            //
            using (SqlCommand cmd = con.CreateCommand())
            {
                cmd.CommandText = "SELECT * FROM Person WHERE Gender = @gender";
                cmd.Parameters.AddWithValue("@gender", gender);
            }
        }
    }
}

```

```

// Execute query.
//
using (SqlDataReader rd = cmd.Exe
{
    List<Person> list = new List<
    while (rd.Read())
    {
        Person person = new Perso

        // Map a data reader row
        //
        person.ID           = Conve
        person.FirstName    = Conve
        person.MiddleName  = Conve
        person.LastName     = Conve

        switch (rd["Gender"].ToSt
        {
            case "F": person.Genc
            case "M": person.Genc
            case "U": person.Genc
            case "O": person.Genc
        }

        list.Add(person);
    }

    return list;
}
}
}

[Test]
public void Test()
{

```

```

        List<Person> list = GetList(Gender.Male);
        Assert.Greater(list.Count, 0);
    }
}
}

```

The **DbManagerDemo** does the same work performed by **BLToolkit.DbManager**.

DbManagerDemo.cs

```

using System;
using System.Collections.Generic;

using NUnit.Framework;

using BLToolkit.Data;
using BLToolkit.Mapping;

namespace HowTo.Data
{
    [TestFixture]
    public class DbManagerDemo
    {
        // The MapValue attribute is used by BLToolkit
        //
        public enum Gender
        {
            [MapValue("F")] Female,
            [MapValue("M")] Male,
            [MapValue("U")] Unknown,
            [MapValue("O")] Other
        }

        // Business object. Here we use C# 3.0 automatic
        // however it can be public fields, regular class
        // The MapField attribute is used by BLToolkit
        // with a business object property if they have
        //
    }
}

```

```

public class Person
{
    [MapField("PersonID")]
    public int ID { get; set; }
    public string FirstName { get; set; }
    public string MiddleName { get; set; }
    public string LastName { get; set; }
    public Gender Gender { get; set; }
}

// BLToolkit data access method.
//
public List<Person> GetList(Gender gender)
{
    using (DbManager db = new DbManager("Demo
    {
        return db
            .SetCommand(
                "SELECT * FROM Person WHERE (
                db.Parameter("@gender", Map.F
            ).ExecuteList<Person>());
    }
}

[Test]
public void Test()
{
    List<Person> list = GetList(Gender.Male);
    Assert.Greater(list.Count, 0);
}
}
}

```

App.config

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>

```

```
<connectionStrings>
  <add
    name = "DemoConnection"
    connectionString = "Server=.;Database=BLT"
    providerName = "System.Data.SqlClient"
  </connectionStrings>
</configuration>
```

[Create.sql script](#)

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

Since Microsoft has released FW 2.0 and the `<connectionStrings>` section of configurat

OpenConfig1FW2.cs

```
using System;
using System.Data;

using NUnit.Framework;

using BLToolkit.Data;

namespace HowTo.Data
{
    [TestFixture]
    public class OpenConfig1FW2
    {
        [Test]
        public void FW2Configuration()
        {
            // <connectionString> section configurat:
            //
            using (DbManager db = new DbManager("Demo
            {
                Assert.AreEqual(ConnectionState.Open,
            }
        }
    }
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <connectionStrings>
```

```

        <add
            name                = "DemoConnection"
            connectionString    = "Server=.;Database=BLT"
            providerName       = "System.Data.SqlClient"
        </connectionStrings>
    </configuration>

```

In addition **BLToolkit** supports an alternative way which uses the `<appSettings>` section

BLToolkit recognizes configuration strings in the `<appSettings>` section by looking for

key value	Provider	Configuration
<code>key="ConnectionString"</code>	default provider	default configuration
<code>key="ConnectionString.Foo.Bar"</code>	'Foo' provider	'Bar' configuration
<code>key="ConnectionString.Foo."</code>	'Foo' provider	default configuration
<code>key="ConnectionString.Foo"</code>	default provider	'Foo' configuration
	or 'Foo' provider	default configuration
<code>key="ConnectionString..Foo"</code>	default provider	'Foo' configuration
<code>key="ConnectionString..Foo.Bar"</code>	default provider	'Foo.Bar' configuration

Default provider is **SqlDataProvider**. See also [Data Providers](#).

OpenConfig1.cs

```

using System;
using System.Data;

using NUnit.Framework;

using BLToolkit.Data;

namespace HowTo.Data
{
    [TestFixture]
    public class OpenConfig1
    {
        [Test]
        public void DefaultConfiguration()
        {

```

```

        // Default configuration and default data
        //
        using (DbManager db = new DbManager())
        {
            Assert.AreEqual(ConnectionString.Open,
        }
    }

[Test]
public void DevelopmentConfiguration()
{
    // Development configuration and default
    //
    using (DbManager db = new DbManager("Deve
    {
        Assert.AreEqual(ConnectionString.Open,
    }
}

[Test]
public void ProductionConfiguration()
{
    // Production configuration and default c
    //
    using (DbManager db = new DbManager("Proc
    {
        Assert.AreEqual(ConnectionString.Open,
    }
}

[Test]
public void OleDbDefaultConfiguration()
{
    // Default configuration and OleDb data p
    //
    using (DbManager db = new DbManager("OleDb
    {

```



```

</connectionStrings>
<appSettings>
  <!-- Default configuration -->
  <add key="ConnectionString"
  <!-- SQL Server Development configuration -->
  <add key="ConnectionString.Development"
  <!-- SQL Server Production configuration -->
  <add key="ConnectionString.Production"
  <!-- SQL Server configuration -->
  <add key="ConnectionString.Sql"
  <!-- OLEDB configuration -->
  <add key="ConnectionString.OleDb"
  <!-- OLEDB Development configuration -->
  <add key="ConnectionString.OleDb.Development"
  <!-- OLEDB Production configuration -->
  <add key="ConnectionString.OleDb.Production"
</appSettings>
</configuration>

```

The default configuration can be set by a configuration file:

App.config

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <!-- MySQL configuration -->
    <add
      key    = "ConnectionString.MySQL"
      value = "..."/>
    <add
      key    = "BLToolkit.DefaultConfiguration"
      value = "MySQL" />
  </appSettings>
</configuration>

```

Create.sql script

© 2010 www.bluetooth.net
support@bluetooth.net

Business Logic Toolkit for .NET www.bltoolkit.net

Connection string can be also assigned programmatically. The **DbManager.AddConnec** when your application starts.

OpenConfig2.cs

```
using System;
using System.Data;

using NUnit.Framework;

using BLToolkit.Data;

namespace HowTo.Data
{
    [TestFixture]
    public class OpenConfig2
    {
        const string sqlConnectionString =
            "Server=.;Database=BLToolkitData;Integrat
        const string oleDbConnectionString =
            "Provider=SQLOLEDB;Data Source=.;Integrat

        [Test]
        public void Test1()
        {
            string defaultConfiguration = DbManager.I
            DbManager.DefaultConfiguration = ""; //tc

            try
            {
                DbManager.AddConnectionString(
                    sqlConnectionString); // connec

                using (DbManager db = new DbManager(
```

```

        {
            Assert.AreEqual(ConnectionState.Open,
                connection.State);
        }
    }
    finally
    {
        DbManager.DefaultConfiguration = default;
    }
}

[Test]
public void Test2()
{
    DbManager.AddConnectionString(
        "NewConfig", // configurat
        sqlConnectionString); // connectio

    using (DbManager db = new DbManager("NewC
    {
        Assert.AreEqual(ConnectionState.Open,
    }
}

[Test]
public void Test3()
{
    DbManager.AddConnectionString(
        "OleDb", // provider r
        "NewConfig", // configurat
        OleDbConnectionString); // connectio

    using (DbManager db = new DbManager("Olel
    {
        Assert.AreEqual(ConnectionState.Open,
    }
}

```

```
}  
}
```

Create.sql script

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

OpenConfig3.cs

```
using System;
using System.Data;
using System.Data.SqlClient;

using NUnit.Framework;

using BLToolkit.Data;
using BLToolkit.Data.DataProvider;

namespace HowTo.Data
{
    [TestFixture]
    public class OpenConfig3
    {
        const string connectionString =
            "Server=.;Database=BLToolkitData;IntegratedSecurity=true";

        [Test]
        public void DbConnectionConfiguration()
        {
            using (SqlConnection con = new SqlConnection(connectionString))
            {
                con.Open();

                using (DbManager db = new DbManager(con))
                {
                    Assert.AreEqual(ConnectionState.Open, con.State);
                }
            }
        }

        [Test]
    }
}
```

```

public void DbTransactionConfiguration()
{
    using (SqlConnection con = new SqlConnection(
    {
        con.Open();

        SqlTransaction tran = con.BeginTransa

        using (DbManager db = new DbManager(t
        {
            Assert.AreEqual(ConnectionState.O
        }

        tran.Commit();
    }
}

[Test]
public void DataProviderConfiguration()
{
    SqlDataProvider dp = new SqlDataProvider(

    using (DbManager db = new DbManager(dp, c
    {
        Assert.AreEqual(ConnectionState.Open,
    }
}
}
}
}

```

Create.sql script

Business Logic Toolkit for .NET www.bltoolkit.net

The **BLToolkit.DbManager** component is a data provider independent wrapper for ADO classes called **DataProviders**.

The following table lists providers available with **BLToolkit**:

Class	Provider Name	Namespace	Vendor
AccessDataProvider	Access	System.Data.OleDb	Microsoft
OdbcDataProvider	Odbc	System.Data.Odbc	Microsoft
OleDbDataProvider	OleDb	System.Data.OleDb	Microsoft
OracleDataProvider	Oracle	System.Data.OracleClient	Microsoft
SqlDataProvider	Sql	System.Data.SqlClient	Microsoft
DB2DataProvider	DB2	IBM.Data.DB2	IBM
FdpDataProvider	Fdp	FirebirdSql.Data.FirebirdClient	Firebird
InformixDataProvider	Informix	IBM.Data.Informix	IBM
MySqlDataProvider	MySql	MySql.Data.MySqlClient	Sun Microsystems
OdpDataProvider	ODP	Oracle.DataAccess.Client	Oracle
SqlCeDataProvider	SqlCe	System.Data.SqlServerCe	Microsoft
SQLiteDataProvider	SQLite	System.Data.SQLite	SQLite.org
SybaseAdoDataProvider	SybaseAdo	System.Data.OleDb	Support for DataDirect S
SybaseDataProvider	Sybase	Sybase.Data.AseClient	Sybase

The first five providers are preregistered in the library and are ready to use. The remainder

You can add a listed above or your own data provider into your project and register it as :

AddDataProvider.cs

```
using System;

using NUnit.Framework;

using BLToolkit.Data;
using BLToolkit.Data.DataProvider;

namespace HowTo.Data.DataProvider
```

```

{
    [TestFixture]
    public class AddDataProvider
    {
        const string connectionString =
            "Provider=ASEOLEDB;Data Source=server;Cat

        [Test]
        public void Test()
        {
            // 3rd party data provider registration.
            //
            DbManager.AddDataProvider(new SybaseAdoDa

            // It can be configured by App.config.
            // We use this way for the demo purpose (
            //
            DbManager.AddConnectionString(
                "SybaseAdo",        // Provider name
                "Default",         // Configuration
                connectionString); // Connection str:

            using (DbManager db = new DbManager("Syba
            {
            }
        }
    }
}

```

Also a data provider can be registered by configuration file:

App.config

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "BLToolkit.DataProviders"

```

```
        value = "BLToolkit.Data.DataProvider.Oracle"
    </appSettings>
</configuration>
```

Also you can use BLToolkit config section.

```
<?xml version="1.0"?>
<configuration>
  <configSections>
    <section name="bltoolkit" type="BLToolkit.Configuration" />
  </configSections>
  <bltoolkit>
    <dataProviders>
      <add type="BLToolkit.Data.DataProvider.Db2" />
      <add type="BLToolkit.Data.DataProvider.Fdp" />
      <add type="BLToolkit.Data.DataProvider.Informix" />
      <add type="BLToolkit.Data.DataProvider.MySql" />
      <add type="BLToolkit.Data.DataProvider.Oracle" />
      <add type="BLToolkit.Data.DataProvider.PostgreSQL" />
      <add type="BLToolkit.Data.DataProvider.SQLite" />
      <add type="BLToolkit.Data.DataProvider.Sybase" />
    </dataProviders>
  </bltoolkit>
  <connectionStrings>
    <add name="Access" connectionString="Provider=Microsoft.ACE.OLEDB.12.0;Data Source=.;User ID=;Password=" />
    <add name="DB2" connectionString="Server=.;Database=;User ID=;Password=" />
    <add name="Fdp" connectionString="Data Source=.;User ID=;Password=" />
    <add name="Informix" connectionString="Host=.;Database=;User ID=;Password=" />
    <add name="MySQL" connectionString="Server=.;Database=;User ID=;Password=" />
    <add name="Oracle" connectionString="...;" />
    <add name="PostgreSQL" connectionString="Server=.;Database=;User ID=;Password=" />
    <add name="Sql2008" connectionString="Server=.;Database=;User ID=;Password=" />
    <add name="Sql2005" connectionString="Server=.;Database=;User ID=;Password=" />
    <add name="SqlCe" connectionString="Data Source=.;User ID=;Password=" />
    <add name="SQLite" connectionString="Data Source=.;User ID=;Password=" />
    <add name="Sybase" connectionString="Data Source=.;User ID=;Password=" />
  </connectionStrings>
</configuration>
```

```
        <add name="Northwind" connectionString="Data
    </connectionStrings>
</configuration>
```

Here *UnitTests.Linq* assembly name should be replaced by your own.

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

Typical scenario of using **DbManager** includes the following steps:

- Create a **DbManager** class instance.
- Set an SQL command or a stored procedure name.
- Call an **ExecuteXXX** method.

The **SetCommand** method is used to set an SQL statement and to provide command par

SetCommand.cs

```
using System;
using System.Collections.Generic;

using NUnit.Framework;

using BLToolkit.Data;
using BLToolkit.Mapping;

namespace HowTo.Data
{
    using DataAccess;

    [TestFixture]
    public class SetCommand
    {
        // Select a person list.
        //
        public IList<Person> GetPersonList()
        {
            using (DbManager db = new DbManager())
            {
                return db
                    .SetCommand("SELECT * FROM Person")
                    .ExecuteList<Person>();
            }
        }
    }
}
```

```

}

[Test]
public void Test1()
{
    IList<Person> list = GetPersonList();

    Assert.AreNotEqual(0, list.Count);
}

// Select a person.
//
public Person GetPersonByID(int id)
{
    using (DbManager db = new DbManager())
    {
        return db
            .SetCommand("SELECT * FROM Person
                db.Parameter("@id", id))
            .ExecuteObject<Person>();
    }
}

[Test]
public void Test2()
{
    Person person = GetPersonByID(1);

    Assert.IsNotNull(person);
}

// Insert, Update, and Delete a person.
//
public Person GetPersonByID(DbManager db, int
{
    return db
        .SetCommand("SELECT * FROM Person WHI

```

```

        db.Parameter("@id", id))
        .ExecuteObject<Person>();
    }

    public Person CreatePerson(DbManager db)
    {
        int id = db
            .SetCommand("
                INSERT INTO Person ( LastName, F
                VALUES                (@LastName, @F

                SELECT Cast(SCOPE_IDENTITY() as i
                db.Parameter("@LastName", "Frog'
                db.Parameter("@FirstName", "Crazy
                db.Parameter("@Gender",      Map.Er
            .ExecuteScalar<int>());

        return GetPersonByID(db, id);
    }

    public Person UpdatePerson(DbManager db, Perso
    {
        db
            .SetCommand("
                UPDATE
                Person
                SET
                LastName    = @LastName,
                FirstName   = @FirstName,
                Gender      = @Gender
                WHERE
                PersonID = @PersonID",
                db.CreateParameters(person))
            .ExecuteNonQuery();

        return GetPersonByID(db, person.ID);
    }

```

```
public Person DeletePerson(DbManager db, Person person)
{
    db
        .SetCommand("DELETE FROM Person WHERE ID=@id")
        .Parameter("@id", person.ID)
        .ExecuteNonQuery();

    return GetPersonByID(db, person.ID);
}
```

```
[Test]
public void Test3()
{
    using (DbManager db = new DbManager())
    {
        db.BeginTransaction();

        // Insert.
        //
        Person person = CreatePerson(db);

        Assert.IsNotNull(person);

        // Update.
        //
        Assert.AreEqual(Gender.Male, person.Gender);

        person.Gender = Gender.Female;

        person = UpdatePerson(db, person);

        Assert.AreEqual(Gender.Female, person.Gender);

        // Delete.
        //
        person = DeletePerson(db, person);
    }
}
```

```
        Assert.IsNotNull(person);  
        db.CommitTransaction();  
    }  
}  
}
```

Person.cs

```
using System;  
  
using BLToolkit.DataAccess;  
using BLToolkit.Mapping;  
  
namespace HowTo.DataAccess  
{  
    public class Person  
    {  
        [MapField("PersonID"), PrimaryKey, NonUpdateable]  
        public int ID;  
  
        public string LastName;  
        public string FirstName;  
        public string MiddleName;  
        public Gender Gender;  
    }  
}
```

Gender.cs

```
using System;  
  
using BLToolkit.Mapping;  
  
namespace HowTo.DataAccess  
{
```

```
public enum Gender
{
    [MapValue("F")] Female,
    [MapValue("M")] Male,
    [MapValue("U")] Unknown,
    [MapValue("O")] Other
}
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value = "Server=.;Database=BLToolkitData,
    </appSettings>
  </configuration>
```

Create.sql script

Business Logic Toolkit for .NET www.bltoolkit.net

Typical scenario of using **DbManager** includes the following steps:

- Create a **DbManager** class instance.
- Set an SQL command or a stored procedure name.
- Call an **ExecuteXXX** method.

The **SetSpCommand** method is used to set a stored procedure name and to provide com

SetSpCommand.cs

```
using System;
using System.Collections.Generic;

using NUnit.Framework;

using BLToolkit.Data;
using BLToolkit.Mapping;

namespace HowTo.Data
{
    using DataAccess;

    [TestFixture]
    public class SetSpCommand
    {
        // Select a person list.
        //
        public IList<Person> GetPersonList()
        {
            using (DbManager db = new DbManager())
            {
                return db
                    .SetSpCommand("Person_SelectAll")
                    .ExecuteList<Person>();
            }
        }
    }
}
```

```

}

[Test]
public void Test1()
{
    IList<Person> list = GetPersonList();

    Assert.AreNotEqual(0, list.Count);
}

// Select a person.
//
public Person GetPersonByID1(int id)
{
    using (DbManager db = new DbManager())
    {
        // Pass a parameter using the Parameter
        //
        return db
            .SetSpCommand("Person_SelectByKey")
            .Parameter("@id", id)
            .ExecuteObject<Person>();
    }
}

public Person GetPersonByID2(int id)
{
    using (DbManager db = new DbManager())
    {
        // Pass a parameter using the params
        //
        return db
            .SetSpCommand("Person_SelectByKey")
            .ExecuteObject<Person>();
    }
}

```

```

[Test]
public void Test2()
{
    Person person = GetPersonByID1(1);
    Assert.IsNotNull(person);

    person = GetPersonByID2(1);
    Assert.IsNotNull(person);
}

// Insert, Update, and Delete a person.
//
public Person GetPersonByID(DbManager db, int id)
{
    return db
        .SetSqlCommand("Person_SelectByKey", db.CreateCommand())
        .ExecuteObject<Person>();
}

public Person CreatePerson(DbManager db)
{
    int id = db
        .SetSqlCommand("Person_Insert", db.CreateCommand())
        .db.Parameter("@LastName", "Frog")
        .db.Parameter("@MiddleName", null)
        .db.Parameter("@FirstName", "Crazy")
        .db.Parameter("@Gender", Map.Female)
        .ExecuteScalar<int>();

    return GetPersonByID(db, id);
}

public Person UpdatePerson(DbManager db, Person person)
{
    db
        .SetSqlCommand("Person_Update", db.CreateCommand())
        .ExecuteNonQuery();
}

```

```

        return GetPersonByID(db, person.ID);
    }

    public Person DeletePerson(DbManager db, Person person)
    {
        db
            .SetSpCommand("Person_Delete", person)
            .ExecuteNonQuery();

        return GetPersonByID(db, person.ID);
    }

    [Test]
    public void Test3()
    {
        using (DbManager db = new DbManager())
        {
            db.BeginTransaction();

            // Insert.
            //
            Person person = CreatePerson(db);

            Assert.IsNotNull(person);

            // Update.
            //
            Assert.AreEqual(Gender.Male, person.Gender);

            person.Gender = Gender.Female;

            person = UpdatePerson(db, person);

            Assert.AreEqual(Gender.Female, person.Gender);

            // Delete.

```

```
        //
        person = DeletePerson(db, person);

        Assert.IsNotNull(person);

        db.CommitTransaction();
    }
}
}
```

Person.cs

```
using System;

using BLToolkit.DataAccess;
using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public class Person
    {
        [MapField("PersonID"), PrimaryKey, NonUpdateable]
        public int ID;

        public string LastName;
        public string FirstName;
        public string MiddleName;
        public Gender Gender;
    }
}
```

Gender.cs

```
using System;

using BLToolkit.Mapping;
```

```
namespace HowTo.DataAccess
{
    public enum Gender
    {
        [MapValue("F")] Female,
        [MapValue("M")] Male,
        [MapValue("U")] Unknown,
        [MapValue("O")] Other
    }
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value  = "Server=.;Database=BLToolkitData;
    </appSettings>
  </configuration>
```

Create.sql script

Business Logic Toolkit for .NET www.bltoolkit.net

The **DbManager.Parameter** method allows assigning parameters to an SQL query, so after.

Parameter.cs

```
using System;
using System.Data;

using NUnit.Framework;

using BLToolkit.Data;

namespace HowTo.Data
{
    using DataAccess;

    [TestFixture]
    public class Parameter
    {
        [Test]
        public void AssignParameterTest()
        {
            using (DbManager db = new DbManager())
            {
                int n = db
                    .SetCommand("SELECT @par1 + @par2",
                        db.Parameter("@par1", 2),
                        db.Parameter("@par2", 2))
                    .ExecuteScalar<int>();

                Assert.AreEqual(4, n);
            }
        }
    }
}
```

```

[Test]
public void SetValueTest()
{
    using (DbManager db = new DbManager())
    {
        db.SetCommand("SELECT @par * 2",
            db.Parameter("@par", DbType.Int32);

        db.Parameter("@par").Value = 2;

        Assert.AreEqual(4, db.ExecuteScalar<int>());
    }
}

```

```

[Test]
public void ReturnValueTest()
{
    using (DbManager db = new DbManager())
    {
        /*
         * CREATE Function Scalar_ReturnParar
         * RETURNS int
         * AS
         * BEGIN
         *     RETURN 12345
         * END
         */
        db
            .SetSpCommand("Scalar_ReturnParar")
            .ExecuteNonQuery();

        int n = (int)db.Parameter("@RETURN_VALUE");

        Assert.AreEqual(12345, n);
    }
}

```

```

[Test]
public void ReturnValueTest2()
{
    using (DbManager db = new DbManager())
    {
        int n = db
            .SetSpCommand("Scalar_ReturnParam")
            .ExecuteScalar<int>(ScalarSource)

        Assert.AreEqual(12345, n);
    }
}

```

```

[Test]
public void OutputParameterTest()
{
    using (DbManager db = new DbManager())
    {
        /*
        * CREATE Procedure Scalar_OutputParameter
        *     @outputInt int = 0
        *     @outputString varchar(50) = ''
        * AS
        * BEGIN
        *     SET @outputInt = 12345
        *     SET @outputString = '54321'
        * END
        */

        db
            .SetSpCommand("Scalar_OutputParameter")
            .db.OutputParameter("@outputInt", 0)
            .db.OutputParameter("@outputString", "")
            .ExecuteNonQuery();

        Assert.AreEqual(12345, (int) db.
        Assert.AreEqual("54321", (string)db.

```

```

    }
}

[Test]
public void OutputParameterAsReturnValueTest()
{
    using (DbManager db = new DbManager())
    {
        string returnValue = db
            .SetSpCommand("Scalar_OutputParam")
            .ExecuteScalar<string>(ScalarSource);

        Assert.AreEqual("54321", returnValue);
    }
}

[Test]
public void CreateParametersTest()
{
    Person person = new Person();

    person.FirstName = "John";
    person.LastName = "Smith";
    person.Gender = Gender.Male;

    using (DbManager db = new DbManager())
    {
        db.BeginTransaction();

        // Prepare command.
        //
        int id = db
            .SetSpCommand("Person_Insert",
                db.CreateParameters(person))
            .ExecuteScalar<int>();

        // Check the result.
    }
}

```

```

        //
        person = db
            .SetCommand(
                "SELECT * FROM Person WHERE P
                db.Parameter("@id", id))
            .ExecuteObject<Person>();

        Assert.IsNotNull(person);

        // Cleanup.
        //
        db
            .SetCommand(
                "DELETE FROM Person WHERE Pei
                db.Parameter("@id", id))
            .ExecuteNonQuery();

        db.CommitTransaction();
    }
}
}
}
}

```

Person.cs

```

using System;

using BLToolkit.DataAccess;
using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public class Person
    {
        [MapField("PersonID"), PrimaryKey, NonUpdateable]
        public int ID;
    }
}

```

```
        public string LastName;
        public string FirstName;
        public string MiddleName;
        public Gender Gender;
    }
}
```

Gender.cs

```
using System;

using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public enum Gender
    {
        [MapValue("F")] Female,
        [MapValue("M")] Male,
        [MapValue("U")] Unknown,
        [MapValue("O")] Other
    }
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value = "Server=.;Database=BLToolkitData;
    </appSettings>
  </configuration>
```

[Create.sql script](#)

support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

The **Prepare** method can be useful if you need to execute the same command multiple ti

Prepare.cs

```
using System;
using System.Collections.Generic;

using NUnit.Framework;

using BLToolkit.Data;

namespace HowTo.Data
{
    using DataAccess;

    [TestFixture]
    public class Prepare
    {
        [Test]
        public void Test()
        {
            List<Person> list = new List<Person>
            {
                new Person { FirstName = "John", Last
                new Person { FirstName = "Jane", Last
            };

            using (DbManager db = new DbManager())
            {
                db.BeginTransaction();

                // Prepare command.
                //
                db
```



```
using System;

using BLToolkit.DataAccess;
using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public class Person
    {
        [MapField("PersonID"), PrimaryKey, NonUpdateable]
        public int ID;

        public string LastName;
        public string FirstName;
        public string MiddleName;
        public Gender Gender;
    }
}
```

Gender.cs

```
using System;

using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public enum Gender
    {
        [MapValue("F")] Female,
        [MapValue("M")] Male,
        [MapValue("U")] Unknown,
        [MapValue("O")] Other
    }
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value = "Server=.;Database=BLToolkitData;
    </appSettings>
  </configuration>
```

[Create.sql script](#)

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

Transaction.cs

```
using System;
using System.Data;

using NUnit.Framework;

using BLToolkit.Data;

namespace HowTo.Data
{
    [TestFixture]
    public class Transaction
    {
        [Test]
        public void Test1()
        {
            using (DbManager db = new DbManager())
            {
                db.BeginTransaction();

                // ...

                db.CommitTransaction();
            }
        }

        [Test]
        public void Test2()
        {
            using (DbManager db = new DbManager())
            {
                db.BeginTransaction(IsolationLevel.Re
```

```
        // ...  
        db.CommitTransaction();  
    }  
}  
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>  
<configuration>  
  <appSettings>  
    <add  
      key    = "ConnectionString"  
      value = "Server=.;Database=BLToolkitData;  
    </appSettings>  
</configuration>
```

[Create.sql script](#)

Business Logic Toolkit for .NET www.bltoolkit.net

Close.cs

```
using System;
using NUnit.Framework;
using BLToolkit.Data;

namespace HowTo.Data
{
    [TestFixture]
    public class Close
    {
        [Test]
        public void Test1()
        {
            DbManager db = new DbManager();

            try
            {
                // ...
            }
            finally
            {
                if (db != null)
                    db.Close();
            }
        }

        // Consider using the C# using statement inst
        //
        [Test]
        public void Test2()
        {
            using (DbManager db = new DbManager())
            {
```

```
        }  
    }  
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>  
<configuration>  
  <appSettings>  
    <add  
      key    = "ConnectionString"  
      value = "Server=.;Database=BLToolkitData;  
    </appSettings>  
</configuration>
```

Create.sql script

Business Logic Toolkit for .NET www.bltoolkit.net

ExecuteDataSet.cs

```
using System;
using System.Data;

using NUnit.Framework;

using BLToolkit.Data;

namespace HowTo.Data
{
    [TestFixture]
    public class ExecuteDataSet
    {
        [Test]
        public void Test()
        {
            using (DbManager db = new DbManager())
            {
                DataSet ds = db
                    .SetCommand("SELECT * FROM Person")
                    .ExecuteDataSet();

                Assert.AreNotEqual(0, ds.Tables[0].RowCount);
            }
        }
    }
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
```

```
        key    = "ConnectionString"
        value  = "Server=.;Database=BLToolkitData,
</appSettings>
</configuration>
```

[Create.sql script](#)

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

ExecuteDataTable.cs

```
using System;
using System.Data;

using NUnit.Framework;

using BLToolkit.Data;

namespace HowTo.Data
{
    [TestFixture]
    public class ExecuteDataTable
    {
        [Test]
        public void Test()
        {
            using (DbManager db = new DbManager())
            {
                DataTable dt = db
                    .SetCommand("SELECT * FROM Person")
                    .ExecuteDataTable();

                Assert.AreNotEqual(0, dt.Rows.Count);
            }
        }
    }
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
```

```
        key    = "ConnectionString"
        value  = "Server=.;Database=BLToolkitData,
</appSettings>
</configuration>
```

[Create.sql script](#)

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

ExecuteDictionary.cs

```
using System;
using System.Collections.Generic;

using NUnit.Framework;

using BLToolkit.Common;
using BLToolkit.Data;
using BLToolkit.Mapping;

namespace HowTo.Data
{
    using DataAccess;

    [TestFixture]
    public class ExecuteDictionary
    {
        // The dictionary key is built from an object
        //
        Dictionary<int, Person> GetPersonDictionary1()
        {
            using (DbManager db = new DbManager())
            {
                return db
                    .SetCommand("SELECT * FROM Person")
                    .ExecuteDictionary<int, Person>();
            }
        }

        [Test]
        public void Test1()
        {
            Dictionary<int, Person> dic = GetPersonD:
```

```

        Assert.AreNotEqual(0, dic.Count);
    }

    // The dictionary key is built from a records
    //
    Dictionary<int, Person> GetPersonDictionary2()
    {
        using (DbManager db = new DbManager())
        {
            return db
                .SetCommand("SELECT * FROM Person")
                .ExecuteDictionary<int, Person>();
        }
    }

    [Test]
    public void Test2()
    {
        Dictionary<int, Person> dic = GetPersonD:

        Assert.AreNotEqual(0, dic.Count);
    }

    // Complex dictionary key.
    //
    Dictionary<CompoundValue, Person> GetPersonD:
    {
        using (DbManager db = new DbManager())
        {
            return db
                .SetCommand("SELECT * FROM Person")
                .ExecuteDictionary<Person>(new Ma
        }
    }

    [Test]

```

```
        public void Test3()
        {
            Dictionary<CompoundValue, Person> dic = (
                Assert.AreNotEqual(0, dic.Count);
            }
        }
    }
```

Person.cs

```
using System;

using BLToolkit.DataAccess;
using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public class Person
    {
        [MapField("PersonID"), PrimaryKey, NonUpdateable]
        public int ID;

        public string LastName;
        public string FirstName;
        public string MiddleName;
        public Gender Gender;
    }
}
```

Gender.cs

```
using System;

using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
```

```
public enum Gender
{
    [MapValue("F")] Female,
    [MapValue("M")] Male,
    [MapValue("U")] Unknown,
    [MapValue("O")] Other
}
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value  = "Server=.;Database=BLToolkitData;
    </appSettings>
  </configuration>
```

[Create.sql script](#)

Business Logic Toolkit for .NET www.bltoolkit.net

The **ExecuteForEach** method executes an SQL statement for every item in the provider affected.

ExecuteForEach.cs

```
using System;
using System.Collections.Generic;

using NUnit.Framework;

using BLToolkit.Data;

namespace HowTo.Data
{
    using DataAccess;

    [TestFixture]
    public class ExecuteForEach
    {
        [Test]
        public void Test()
        {
            List<Person> list = new List<Person>
            {
                new Person { FirstName = "John", Last
                new Person { FirstName = "Jane", Last
            };

            using (DbManager db = new DbManager())
            {
                db.BeginTransaction();

                // Execute.
                //
            }
        }
    }
}
```



```

public class Person
{
    [MapField("PersonID"), PrimaryKey, NonUpdateable]
    public int ID;

    public string LastName;
    public string FirstName;
    public string MiddleName;
    public Gender Gender;
}
}

```

Gender.cs

```

using System;

using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public enum Gender
    {
        [MapValue("F")] Female,
        [MapValue("M")] Male,
        [MapValue("U")] Unknown,
        [MapValue("O")] Other
    }
}

```

App.config

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key = "ConnectionString"
      value = "Server=.;Database=BLToolkitData;
    </appSettings>

```

```
</configuration>
```

[Create.sql script](#)

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

ExecuteList.cs

```
using System;
using System.Collections;
using System.Collections.Generic;

using NUnit.Framework;

using BLToolkit.Data;
using BLToolkit.Mapping;
using BLToolkit.Reflection;

namespace HowTo.Data
{
    [TestFixture]
    public class ExecuteList
    {
        [MapValue(Gender.Female, "F")]
        [MapValue(Gender.Male, "M")]
        [MapValue(Gender.Unknown, "U")]
        [MapValue(Gender.Other, "O")]
        public enum Gender
        {
            Female,
            Male,
            Unknown,
            Other
        }

        [MapField("PersonID", "ID")]
        public class Person
        {
            public int ID;
        }
    }
}
```

```

        public string LastName;
        public string FirstName;
        public string MiddleName;
        public Gender Gender;
    }

    IList<Person> GetPersonListSqlText()
    {
        using (DbManager db = new DbManager())
        {
            return db
                .SetCommand("SELECT * FROM Person")
                .ExecuteList<Person>();
        }
    }

    [Test]
    public void SqlText()
    {
        IList<Person> list = GetPersonListSqlText();

        foreach (Person p in list)
            TypeAccessor.WriteDebug(p);
    }

    IList<Person> GetPersonListSproc()
    {
        using (DbManager db = new DbManager())
        {
            return db
                .SetSpCommand("Person_SelectAll")
                .ExecuteList<Person>();
        }
    }

    [Test]
    public void Sproc()

```

```

    {
        IList<Person> list = GetPersonListSproc();

        foreach (Person p in list)
            TypeAccessor.WriteDebug(p);
    }

    void GetCustomPersonList(IList list)
    {
        using (DbManager db = new DbManager())
        {
            db
                .SetSpCommand("Person_SelectAll")
                .ExecuteList(list, typeof(Person));
        }
    }

    [Test]
    public void CustomList()
    {
        ArrayList list = new ArrayList(10);

        GetCustomPersonList(list);

        foreach (Person p in list)
            TypeAccessor.WriteDebug(p);
    }
}

```

App.config

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"

```

```
        value = "Server=.;Database=BLToolkitData;  
    </appSettings>  
</configuration>
```

[Create.sql script](#)

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

The **ExecuteNonQuery** executes an SQL statement and returns the number of rows affected.

ExecuteNonQuery.cs

```
using System;
using NUnit.Framework;
using BLToolkit.Data;

namespace HowTo.Data
{
    using DataAccess;

    [TestFixture]
    public class ExecuteNonQuery
    {
        [Test]
        public void Test()
        {
            Person person = new Person();

            person.FirstName = "John";
            person.LastName = "Smith";
            person.Gender = Gender.Male;

            using (DbManager db = new DbManager())
            {
                db.BeginTransaction();

                // Execute.
                //
                db
                    .SetSpCommand("Person_Insert", dt
                    .ExecuteNonQuery());
            }
        }
    }
}
```



```
        public string LastName;
        public string FirstName;
        public string MiddleName;
        public Gender Gender;
    }
}
```

Gender.cs

```
using System;

using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public enum Gender
    {
        [MapValue("F")] Female,
        [MapValue("M")] Male,
        [MapValue("U")] Unknown,
        [MapValue("O")] Other
    }
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value  = "Server=.;Database=BLToolkitData,
    </appSettings>
  </configuration>
```

Create.sql script

© 2010 www.bluetooth.net
support@bluetooth.net

Business Logic Toolkit for .NET www.bltoolkit.net

ExecuteObject.cs

```
using System;

using NUnit.Framework;

using BLToolkit.Data;
using BLToolkit.Mapping;
using BLToolkit.Reflection;

namespace HowTo.Data
{
    [TestFixture]
    public class ExecuteObject
    {
        public enum Gender
        {
            [MapValue("F")] Female,
            [MapValue("M")] Male,
            [MapValue("U")] Unknown,
            [MapValue("O")] Other
        }

        public abstract class Person
        {
            [MapField("PersonID")]
            public abstract int ID { get; }

            public abstract string LastName { get; }
            public abstract string FirstName { get; }
            public abstract string MiddleName { get; }
            public abstract Gender Gender { get; }
        }
    }
}
```

```
Person GetPersonSqlText(int id)
{
    using (DbManager db = new DbManager())
    {
        return db
            .SetCommand("SELECT * FROM Person
                db.Parameter("@id", id))
            .ExecuteObject<Person>();
    }
}
```

```
[Test]
public void SqlText()
{
    Person person = GetPersonSqlText(1);

    TypeAccessor.WriteConsole(person);
}
```

```
Person GetPersonSproc1(int id)
{
    using (DbManager db = new DbManager())
    {
        return db
            .SetSpCommand("Person_SelectByKey
                db.Parameter("@id", id))
            .ExecuteObject<Person>();
    }
}
```

```
[Test]
public void Sproc1()
{
    Person person = GetPersonSproc1(1);

    TypeAccessor.WriteConsole(person);
}
```

```

    Person GetPersonSproc2(int id)
    {
        using (DbManager db = new DbManager())
        {
            return db
                .SetSpCommand("Person_SelectByKey")
                .ExecuteObject<Person>();
        }
    }

    [Test]
    public void Sproc2()
    {
        Person person = GetPersonSproc2(1);

        TypeAccessor.WriteConsole(person);
    }
}

```

App.config

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value  = "Server=.;Database=BLToolkitData;
    </appSettings>
  </configuration>

```

Create.sql script

Business Logic Toolkit for .NET www.bltoolkit.net

ExecuteReader.cs

```
using System;
using System.Data;

using NUnit.Framework;

using BLToolkit.Data;

namespace HowTo.Data
{
    [TestFixture]
    public class ExecuteReader
    {
        [Test]
        public void Test()
        {
            using (DbManager db = new DbManager())
            {
                db.SetCommand("SELECT * FROM Person");

                using (IDataReader rd = db.ExecuteReader())
                {
                    while (rd.Read())
                    {
                        // ...
                    }
                }
            }
        }
    }
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value = "Server=.;Database=BLToolkitData,
    </appSettings>
  </configuration>
```

[Create.sql script](#)

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

ComplexMapping.cs

```
using System;
using System.Collections.Generic;

using NUnit.Framework;

using BLToolkit.Mapping;
using BLToolkit.Data;
using BLToolkit.DataAccess;
using BLToolkit.Reflection.Extension;

namespace HowTo.Data
{
    [TestFixture]
    public class ComplexMapping
    {
        const string TestQuery = @"
            -- Parent Data
            SELECT          1 as ParentID
            UNION SELECT 2 as ParentID

            -- Child Data
            SELECT          4 ChildID, 1 as ParentID
            UNION SELECT 5 ChildID, 2 as ParentID
            UNION SELECT 6 ChildID, 2 as ParentID
            UNION SELECT 7 ChildID, 1 as ParentID

            -- Grandchild Data
            SELECT          1 GrandchildID, 4 as ChildID
            UNION SELECT 2 GrandchildID, 4 as ChildID
            UNION SELECT 3 GrandchildID, 5 as ChildID
            UNION SELECT 4 GrandchildID, 5 as ChildID
            UNION SELECT 5 GrandchildID, 6 as ChildID
```

```
UNION SELECT 6 GrandchildID, 6 as ChildID
UNION SELECT 7 GrandchildID, 7 as ChildID
UNION SELECT 8 GrandchildID, 7 as ChildID
```

```
";
```

```
public class Parent
{
    [MapField("ParentID"), PrimaryKey]
    public int ID;

    [Relation(typeof(Child))]
    public List<Child> Children = new List<Child>();
}

[MapField("ParentID", "Parent.ID")]
public class Child
{
    [MapField("ChildID"), PrimaryKey]
    public int ID;

    [Relation]
    public Parent Parent = new Parent();

    [Relation(typeof(Grandchild))]
    public List<Grandchild> Grandchildren = new List<Grandchild>();
}

[MapField("ChildID", "Child.ID")]
public class Grandchild
{
    [MapField("GrandchildID"), PrimaryKey]
    public int ID;

    [Relation]
    public Child Child = new Child();
}
```

```

[Test]
public void Test()
{
    List<Parent> parents = new List<Parent>();
    MapResultSet[] sets = new MapResultSet[3];

    sets[0] = new MapResultSet(typeof(Parent));
    sets[1] = new MapResultSet(typeof(Child));
    sets[2] = new MapResultSet(typeof(Grandchild));

    sets[0].AddRelation(sets[1], "ParentID", "ChildID");
    sets[1].AddRelation(sets[0], "ParentID", "ChildID");

    sets[1].AddRelation(sets[2], "ChildID", "GrandchildID");
    sets[2].AddRelation(sets[1], "ChildID", "GrandchildID");

    using (DbManager db = new DbManager())
    {
        db
            .SetCommand(TestQuery)
            .ExecuteResultSet(sets);
    }

    Assert.IsEmpty(parents);

    foreach (Parent parent in parents)
    {
        Assert.IsNotNull(parent);
        Assert.IsEmpty(parent.Children);

        foreach (Child child in parent.Children)
        {
            Assert.AreEqual(parent, child.Parent);
            Assert.IsEmpty(child.Grandchildren);

            foreach (Grandchild grandchild in child.Grandchildren)
            {

```

```

        Assert.AreEqual(child, grandchild);
        Assert.AreEqual(parent, grandparent);
    }
}
}
}

```

```

[Test]
public void Test2()
{
    List<Parent> parents = new List<Parent>();
    MapResultSet[] sets = new MapResultSet[3];

    sets[0] = new MapResultSet(typeof(Parent));
    sets[1] = new MapResultSet(typeof(Child));
    sets[2] = new MapResultSet(typeof(Grandchild));

    using (DbManager db = new DbManager())
    {
        db
            .SetCommand(TestQuery)
            .ExecuteResultSet(sets);
    }

    Assert.IsEmpty(parents);

    foreach (Parent parent in parents)
    {
        Assert.IsNotNull(parent);
        Assert.IsEmpty(parent.Children);

        foreach (Child child in parent.Children)
        {
            Assert.AreEqual(parent, child.Parent);
            Assert.IsEmpty(child.Grandchildren);

            foreach (Grandchild grandchild in child.Grandchildren)
            {
                Assert.AreEqual(child, grandchild.Parent);
            }
        }
    }
}

```

```

        {
            Assert.AreEqual(child, grandchild);
            Assert.AreEqual(parent, grandparent);
        }
    }
}

public class ParentEx
{
    public int ID;
    public List<ChildEx> Children = new List<>();
}

public class ChildEx
{
    public int ID;
    public ParentEx Parent = null;
    public List<GrandchildEx> Grandchildren = new List<>();
}

public class GrandchildEx
{
    public int ID;
    public ChildEx Child = new ChildEx();
}

static readonly MappingSchema _mappingSchema
{
    Extensions = TypeExtension.GetExtensions();
};

[Test]
public void Test3()
{
    List<ParentEx> parents = new List<ParentEx>();
    MapResultSet[] sets = new MapResultSet[3];
}

```

```
sets[0] = new MapResultSet(typeof(ParentEx));
sets[1] = new MapResultSet(typeof(ChildEx));
sets[2] = new MapResultSet(typeof(GrandchildEx));
```

```
using (DbManager db = new DbManager())
{
    db.MappingSchema = _mappingSchema;

    db
        .SetCommand(TestQuery)
        .ExecuteResultSet(sets);
}
```

```
Assert.IsEmpty(parents);
```

```
foreach (ParentEx parent in parents)
{
    Assert.IsNotNull(parent);
    Assert.IsEmpty(parent.Children);

    foreach (ChildEx child in parent.Children)
    {
        Assert.AreEqual(parent, child.Parent);
        Assert.IsEmpty(child.Grandchildren);

        foreach (GrandchildEx grandchild in child.Grandchildren)
        {
            Assert.AreEqual(child, grandchild.Parent);
            Assert.AreEqual(parent, grandchild.Grandparent);
        }
    }
}
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value  = "Server=.;Database=BLToolkitData;
    </add>
  </appSettings>
</configuration>
```

[Create.sql script](#)

Business Logic Toolkit for .NET www.bltoolkit.net

ExecuteScalar.cs

```
using System;
using NUnit.Framework;
using BLToolkit.Data;

namespace HowTo.Data
{
    [TestFixture]
    public class ExecuteScalar
    {
        string GetFirstName(int id)
        {
            using (DbManager db = new DbManager())
            {
                return db
                    .SetCommand("SELECT FirstName FROM Customers WHERE ID=@id", id)
                    .Parameter("@id", id)
                    .ExecuteScalar<string>();
            }
        }

        [Test]
        public void ReaderTest()
        {
            string firstName = GetFirstName(1);

            Assert.IsNotNull(firstName);
        }

        [Test]
        public void ReturnValueTest()
        {
            using (DbManager db = new DbManager())
```

```

    {
        /*
         * CREATE Function Scalar_ReturnParam
         * RETURNS int
         * AS
         * BEGIN
         *     RETURN 12345
         * END
         */
        int n = db
            .SetSpCommand("Scalar_ReturnParam")
            .ExecuteScalar<int>(ScalarSource);

        Assert.AreEqual(12345, n);
    }
}

```

```

[Test]
public void OutputParameterAsReturnValueTest()
{
    using (DbManager db = new DbManager())
    {
        /*
         * CREATE Procedure Scalar_OutputParameter
         * @outputInt int = 0
         * @outputString varchar(50) = ''
         * AS
         * BEGIN
         *     SET @outputInt = 12345
         *     SET @outputString = '54321'
         * END
         */
        string returnValue = db
            .SetSpCommand("Scalar_OutputParameter")
            .ExecuteScalar<string>(ScalarSource);

        Assert.AreEqual("54321", returnValue);
    }
}

```

```
}  
  }  
} }
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>  
<configuration>  
  <appSettings>  
    <add  
      key    = "ConnectionString"  
      value = "Server=.;Database=BLToolkitData;  
    </appSettings>  
</configuration>
```

[Create.sql script](#)

Business Logic Toolkit for .NET www.bltoolkit.net

The **ExecuteScalarDictionary** method executes the query, and returns the dictionary. The *keyField* parameter and values are loaded from a column specified by the *valueField*. Of

ExecuteScalarDictionary.cs

```
using System;
using System.Collections.Generic;

using NUnit.Framework;

using BLToolkit.Data;

namespace HowTo.Data
{
    using DataAccess;

    [TestFixture]
    public class ExecuteScalarDictionary
    {
        Dictionary<int, string> GetNameDictionary()
        {
            using (DbManager db = new DbManager())
            {
                return db
                    .SetCommand("SELECT * FROM Person")
                    .ExecuteScalarDictionary<int, string>();
            }
        }

        [Test]
        public void Test()
        {
            Dictionary<int, string> dic = GetNameDic
```

```
        Assert.AreNotEqual(0, dic.Count);
        Assert.IsNotNull(dic[1]);
    }
}
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value = "Server=.;Database=BLToolkitData;
    </appSettings>
  </configuration>
```

[Create.sql script](#)

Business Logic Toolkit for .NET www.bltoolkit.net

The **ExecuteScalarList** method executes the query and returns a list of values of the specified column for every row in the resultset returned by the query. Other columns are ignored.

ExecuteScalarList.cs

```
using System;
using System.Collections.Generic;

using NUnit.Framework;

using BLToolkit.Data;

namespace HowTo.Data
{
    using DataAccess;

    [TestFixture]
    public class ExecuteScalarList
    {
        List<string> GetNameList1()
        {
            using (DbManager db = new DbManager())
            {
                return db
                    .SetCommand("SELECT FirstName FROM Customers")
                    .ExecuteScalarList<string>();
            }
        }

        [Test]
        public void Test1()
        {
            List<string> list = GetNameList1();
        }
    }
}
```

```

        Assert.AreNotEqual(0, list.Count);
        Assert.IsNotNull(list[0]);
    }

    List<string> GetNameList2()
    {
        using (DbManager db = new DbManager())
        {
            return db
                .SetCommand("SELECT * FROM Person")
                .ExecuteScalarList<string>("First");
        }
    }

    [Test]
    public void Test2()
    {
        List<string> list = GetNameList2();

        Assert.AreNotEqual(0, list.Count);
        Assert.IsNotNull(list[0]);
    }
}
}

```

App.config

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value  = "Server=.;Database=BLToolkitData;
    </appSettings>
  </configuration>

```

Create.sql script

© 2010 www.bluetooth.net
support@bluetooth.net

Business Logic Toolkit for .NET www.bltoolkit.net

Create.sql

```
--CREATE DATABASE BLToolkitData ON PRIMARY
--(NAME=N'BLToolkitTest',      FILENAME=N'C:\Data\MSSD
--LOG ON
--(NAME=N'BLToolkitTest_log', FILENAME=N'C:\Data\MSSD
--GO

IF EXISTS (SELECT * FROM sys.objects WHERE object_id
BEGIN DROP TABLE Doctor END

IF EXISTS (SELECT * FROM sys.objects WHERE object_id
BEGIN DROP TABLE Patient END

-- Person Table

IF EXISTS (SELECT * FROM sys.objects WHERE object_id
BEGIN DROP TABLE Person END

CREATE TABLE Person
(
    PersonID    int          NOT NULL IDENTITY(1,1) CO
    FirstName   nvarchar(50) NOT NULL,
    LastName    nvarchar(50) NOT NULL,
    MiddleName  nvarchar(50)  NULL,
    Gender      char(1)      NOT NULL CONSTRAINT CK_Pe
)
ON [PRIMARY]
GO

INSERT INTO Person (FirstName, LastName, Gender) VALU
GO
INSERT INTO Person (FirstName, LastName, Gender) VALU
GO
```

```
-- Doctor Table Extension
```

```
CREATE TABLE Doctor
```

```
(  
    PersonID int NOT NULL  
        CONSTRAINT PK_Doctor PRIMARY KEY CLUSTERED  
        CONSTRAINT FK_Doctor_Person FOREIGN KEY  
            REFERENCES Person ([PersonID])  
            ON UPDATE CASCADE  
            ON DELETE CASCADE,  
    Taxonomy nvarchar(50) NOT NULL  
)  
ON [PRIMARY]  
GO
```

```
INSERT INTO Doctor (PersonID, Taxonomy) VALUES (1, 'F'  
GO
```

```
-- Patient Table Extension
```

```
CREATE TABLE Patient
```

```
(  
    PersonID int NOT NULL  
        CONSTRAINT PK_Patient PRIMARY KEY CLUSTERED  
        CONSTRAINT FK_Patient_Person FOREIGN KEY  
            REFERENCES Person ([PersonID])  
            ON UPDATE CASCADE  
            ON DELETE CASCADE,  
    Diagnosis nvarchar(256) NOT NULL  
)  
ON [PRIMARY]  
GO
```

```
INSERT INTO Patient (PersonID, Diagnosis) VALUES (2,  
GO
```

```
-- Person_SelectByKey
```

```
IF EXISTS (SELECT * FROM sys.objects WHERE type = 'P'  
BEGIN DROP Procedure Person_SelectByKey  
END  
GO
```

```
CREATE Procedure Person_SelectByKey  
    @id int  
AS
```

```
SELECT * FROM Person WHERE PersonID = @id
```

```
GO
```

```
GRANT EXEC ON Person_SelectByKey TO PUBLIC  
GO
```

```
-- Person_SelectAll
```

```
IF EXISTS (SELECT * FROM sys.objects WHERE type = 'P'  
BEGIN DROP Procedure Person_SelectAll END  
GO
```

```
CREATE Procedure Person_SelectAll  
AS
```

```
SELECT * FROM Person
```

```
GO
```

```
GRANT EXEC ON Person_SelectAll TO PUBLIC  
GO
```

```
-- Person_SelectByName
```

```
IF EXISTS (SELECT * FROM sys.objects WHERE type = 'P
```

```
BEGIN DROP Procedure Person_SelectByName END
GO

CREATE Procedure Person_SelectByName
    @firstName nvarchar(50),
    @lastName  nvarchar(50)
AS

SELECT
    *
FROM
    Person
WHERE
    FirstName = @firstName AND LastName = @lastName

GO

GRANT EXEC ON Person_SelectByName TO PUBLIC
GO

-- Person_SelectListByName

IF EXISTS (SELECT * FROM sys.objects WHERE type = 'P')
BEGIN DROP Procedure Person_SelectListByName
END
GO

CREATE Procedure Person_SelectListByName
    @firstName nvarchar(50),
    @lastName  nvarchar(50)
AS

SELECT
    *
FROM
    Person
WHERE
```

```

        FirstName like @firstName AND LastName like @last
GO

GRANT EXEC ON Person_SelectByName TO PUBLIC
GO

-- Person_Insert

IF EXISTS (SELECT * FROM sys.objects WHERE type = 'P
BEGIN DROP Procedure Person_Insert END
GO

CREATE Procedure Person_Insert
    @FirstName nvarchar(50),
    @LastName nvarchar(50),
    @MiddleName nvarchar(50),
    @Gender char(1)
AS

INSERT INTO Person
    ( LastName,  FirstName,  MiddleName,  Gender)
VALUES
    (@LastName, @FirstName, @MiddleName, @Gender)

SELECT Cast(SCOPE_IDENTITY() as int) PersonID

GO

GRANT EXEC ON Person_Insert TO PUBLIC
GO

-- Person_Insert_OutputParameter

IF EXISTS (SELECT * FROM sys.objects WHERE type = 'P
BEGIN DROP Procedure Person_Insert_OutputParameter EI
GO

```

```

CREATE Procedure Person_Insert_OutputParameter
    @FirstName nvarchar(50),
    @LastName nvarchar(50),
    @MiddleName nvarchar(50),
    @Gender char(1),
    @PersonID int output
AS

INSERT INTO Person
    ( LastName, FirstName, MiddleName, Gender)
VALUES
    (@LastName, @FirstName, @MiddleName, @Gender)

SET @PersonID = Cast(SCOPE_IDENTITY() as int)

GO

GRANT EXEC ON Person_Insert_OutputParameter TO PUBLIC
GO

-- Person_Update

IF EXISTS (SELECT * FROM sys.objects WHERE type = 'P'
BEGIN DROP Procedure Person_Update END
GO

CREATE Procedure Person_Update
    @PersonID int,
    @FirstName nvarchar(50),
    @LastName nvarchar(50),
    @MiddleName nvarchar(50),
    @Gender char(1)
AS

UPDATE
    Person

```

```
SET
    LastName    = @LastName,
    FirstName    = @FirstName,
    MiddleName   = @MiddleName,
    Gender       = @Gender
WHERE
    PersonID = @PersonID

GO

GRANT EXEC ON Person_Update TO PUBLIC
GO

-- Person_Delete

IF EXISTS (SELECT * FROM sys.objects WHERE type = 'P'
BEGIN DROP Procedure Person_Delete END
GO

CREATE Procedure Person_Delete
    @PersonID int
AS

DELETE FROM Person WHERE PersonID = @PersonID

GO

GRANT EXEC ON Person_Delete TO PUBLIC
GO

-- Patient_SelectAll

IF EXISTS (SELECT * FROM sys.objects WHERE type = 'P'
BEGIN DROP Procedure Patient_SelectAll END
GO

CREATE Procedure Patient_SelectAll
```

```

AS

SELECT
    Person.*, Patient.Diagnosis
FROM
    Patient, Person
WHERE
    Patient.PersonID = Person.PersonID

GO

GRANT EXEC ON Patient_SelectAll TO PUBLIC
GO

-- Patient_SelectByName

IF EXISTS (SELECT * FROM sys.objects WHERE type = 'P'
BEGIN DROP Procedure Patient_SelectByName END
GO

CREATE Procedure Patient_SelectByName
    @firstName nvarchar(50),
    @lastName  nvarchar(50)
AS

SELECT
    Person.*, Patient.Diagnosis
FROM
    Patient, Person
WHERE
    Patient.PersonID = Person.PersonID
    AND FirstName = @firstName AND LastName = @lastNa

GO

GRANT EXEC ON Person_SelectByName TO PUBLIC
GO

```

```
-- BinaryData Table
```

```
IF EXISTS (SELECT * FROM sys.objects WHERE object_id  
BEGIN DROP TABLE BinaryData END
```

```
CREATE TABLE BinaryData
```

```
(  
    BinaryDataID int NOT NULL IDENTITY(1,  
    Stamp timestamp NOT NULL,  
    Data varbinary(1024) NOT NULL)
```

```
ON [PRIMARY]
```

```
GO
```

```
-- OutRefTest
```

```
IF EXISTS (SELECT * FROM sys.objects WHERE type = 'P'  
BEGIN DROP Procedure OutRefTest END
```

```
GO
```

```
CREATE Procedure OutRefTest
```

```
    @ID int,  
    @outputID int output,  
    @inputOutputID int output,  
    @str varchar(50),  
    @outputStr varchar(50) output,  
    @inputOutputStr varchar(50) output
```

```
AS
```

```
SET @outputID = @ID
```

```
SET @inputOutputID = @ID + @inputOutputID
```

```
SET @outputStr = @str
```

```
SET @inputOutputStr = @str + @inputOutputStr
```

```
GO
```

```
-- OutRefEnumTest
```

```
IF EXISTS (SELECT * FROM sys.objects WHERE type = 'P'  
BEGIN DROP Procedure OutRefEnumTest END  
GO
```

```
CREATE Procedure OutRefEnumTest  
    @str          varchar(50),  
    @outputStr    varchar(50) output,  
    @inputOutputStr varchar(50) output
```

```
AS
```

```
SET @outputStr      = @str  
SET @inputOutputStr = @str + @inputOutputStr
```

```
GO
```

```
-- ExecuteScalarTest
```

```
IF EXISTS (SELECT * FROM sys.objects WHERE type = 'P'  
BEGIN DROP Procedure Scalar_DataReader END  
GO
```

```
CREATE Procedure Scalar_DataReader
```

```
AS
```

```
SELECT Cast(12345 as int) AS intField, Cast('54321' as
```

```
GO
```

```
IF EXISTS (SELECT * FROM sys.objects WHERE type = 'P'  
BEGIN DROP Procedure Scalar_OutputParameter END  
GO
```

```
CREATE Procedure Scalar_OutputParameter
```

```
    @outputInt    int = 0 output,  
    @outputString varchar(50) = '' output
```

```
AS
```

```
BEGIN
```

```

    SET @outputInt      = 12345
    SET @outputString = '54321'
END

GO

IF EXISTS (SELECT * FROM sys.objects WHERE type in ('F', 'P'))
BEGIN DROP Function Scalar_ReturnParameter END
GO

CREATE Function Scalar_ReturnParameter()
RETURNS int
AS
BEGIN
    RETURN 12345
END

GO

IF EXISTS (SELECT * FROM sys.objects WHERE type = 'P')
BEGIN DROP Procedure Scalar_ReturnParameterWithObjectID END
GO

CREATE Procedure Scalar_ReturnParameterWithObjectID
    @id int
AS
BEGIN
    SELECT * FROM Person WHERE PersonID = @id
    RETURN @id
END

GO

-- Data Types test

IF EXISTS (SELECT * FROM sys.objects WHERE object_id = OBJECT_ID(N'DataTypeTest'))
BEGIN DROP TABLE DataTypeTest END

```

```
GO
```

```
CREATE TABLE DataTypeTest
```

```
(  
    DataTypeID      int          NOT NULL IDENTITY(1,  
    Binary_         binary(50)   NULL,  
    Boolean_        bit         NULL,  
    Byte_           tinyint     NULL,  
    Bytes_          varbinary(50) NULL,  
    Char_           char(1)     NULL,  
    DateTime_       datetime    NULL,  
    Decimal_        decimal(20,2) NULL,  
    Double_         float        NULL,  
    Guid_           uniqueidentifier NULL,  
    Int16_          smallint    NULL,  
    Int32_          int         NULL,  
    Int64_          bigint      NULL,  
    Money_         money        NULL,  
    SByte_         tinyint     NULL,  
    Single_         real         NULL,  
    Stream_        varbinary(50) NULL,  
    String_        nvarchar(50) NULL,  
    UInt16_        smallint    NULL,  
    UInt32_        int         NULL,  
    UInt64_        bigint      NULL,  
    Xml_           xml         NULL
```

```
) ON [PRIMARY]
```

```
GO
```

```
INSERT INTO DataTypeTest
```

```
(Binary_, Boolean_, Byte_, Bytes_, Char_, Da  
    Double_, Guid_, Int16_, Int32_, Int64_,  
    Single_, Stream_, String_, UInt16_, UInt32_,  
VALUES  
    (  
        NULL, NULL, NULL, NULL, NULL,  
        NULL, NULL, NULL, NULL, NULL,  
        NULL, NULL, NULL, NULL, NULL,
```

```
GO
```

```
INSERT INTO DataTypeTest
```

```
(Binary_, Boolean_, Byte_, Bytes_, Char_, Da  
Double_, Guid_, Int16_, Int32_, Int64_,  
Single_, Stream_, String_, UInt16_, UInt32_,  
Xml_)
```

```
VALUES
```

```
(NewID(), 1, 255, NewID(), 'B', Ge  
1234.567, NewID(), 32767, 32768, 1000000,  
1234.123, NewID(), 'string', 32767, 32768, 20  
'<root><element strattr="strvalue" intattr="12345
```

```
GO
```

```
-- SKIP Sql2005 BEGIN
```

```
--
```

```
-- Arrays
```

```
--
```

```
IF EXISTS (SELECT * FROM sys.objects WHERE type = 'P'  
BEGIN DROP PROCEDURE ArrayTest END
```

```
GO
```

```
--IF EXISTS (SELECT * FROM sys.objects WHERE type =
```

```
--BEGIN
```

```
    DROP TYPE IntArray
```

```
--END
```

```
GO
```

```
CREATE TYPE IntArray AS TABLE
```

```
(
```

```
    Num int NULL
```

```
)
```

```
GO
```

```
CREATE PROCEDURE ArrayTest
```

```
    @InputIntArray IntArray READONLY
```

```
AS
```

```

BEGIN
    SELECT Num * 2 FROM @InputIntArray;
END
GO
-- SKIP Sql2005 END

DROP TABLE Parent
GO
DROP TABLE Child
GO
DROP TABLE GrandChild
GO

CREATE TABLE Parent      (ParentID int, Value1 int)
GO
CREATE TABLE Child       (ParentID int, ChildID int)
GO
CREATE TABLE GrandChild  (ParentID int, ChildID int,
GO

IF EXISTS (SELECT * FROM sys.objects WHERE object_id
BEGIN DROP TABLE LinqDataTypes END
GO

CREATE TABLE LinqDataTypes
(
    ID                int,
    MoneyValue        decimal(10,4),
    DateTimeValue     datetime,
    BoolValue         bit,
    GuidValue         uniqueidentifier,
    BinaryValue       varbinary(5000)
)
GO

```

support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

Data providers supported.

Data Provider	Provider Name	Namespace	Vendor
AccessDataProvider	Access	System.Data.OleDb	Microsoft
DB2DataProvider	DB2	IBM.Data.DB2	IBM
InformixDataProvider	Informix	IBM.Data.Informix	IBM
FdpDataProvider	Fdp	FirebirdSql.Data.FirebirdClient	Firebird
MySqlDataProvider	MySql	MySql.Data.MySqlClient	Sun Microsystems
OdpDataProvider	ODP	Oracle.DataAccess.Client	Oracle
PostgreSQLProvider	PostgreSQL	Npgsql	PostgreSQL.org
SqlCeDataProvider	SqlCe	System.Data.SqlServerCe	Microsoft
SQLiteDataProvider	SQLite	System.Data.SQLite	SQLite.org
SqlDataProvider	Sql	System.Data.SqlClient	Microsoft
SybaseDataProvider	Sybase	Sybase.Data.AseClient	Sybase

Operators.

	Linq	DB2	Informix	Oracle	Firebird
Bitwise AND	a & b	BitAnd(a, b)			Bin_And(a,
Bitwise OR	a b	BitOr(a, b)		(a + b) - BitAnd(a, b)	Bin_Or(a, b
Bitwise XOR	a ^ b	BitXor(a, b)		(a + b) - BitAnd(a, b) * 2	Bin_Xor(a,
Coalesce	a ?? b	Coalesce(a, b)	Nvl(a, b)		Coalesce(a,
	a ?? b ?? c	Coalesce(a, b, c)	Nvl(a, b, c)		Coalesce(a,
Concatenation	a + b + c	a b c			
Conditional	a ? b : c	CASE WHEN a THEN b ELSE c END			
	a ? b : c ? d : e	CASE WHEN a THEN b WHEN c THEN d ELSE e END			
Modulo	a % b	Mod(a, b)			

Core string functions.

	Linq	DB2	Informix	Oracle
Case	Sql.Lower(s)	Lower(s)		
	Sql.Upper(s)	Upper(s)		
CharIndex	Sql.CharIndex(a, s)	Locate(a, s)	X	InStr(s, a)

	Sql.CharIndex(a, s, b)	Locate(a, s, b)	X	InStr(s, a, b)
Length	Sql.Length(s)	Length(s)		
LIKE	!s.Contains(a)	s NOT LIKE a	NOT s LIKE a	s NOT LIKE
	s.Contains("a~b")	s LIKE '%a~%b%' ESCAPE '~'		
	s.Contains("abc")	s LIKE '%abc%'		
	s.Contains(str)	s LIKE @str ESCAPE '~'		
	s.EndsWith("abc")	s LIKE '%abc'		
	s.StartsWith("abc")	s LIKE 'abc%'		
	Sql.Like(s, "%abc%")	s LIKE '%abc%'		
	Sql.Like(s, "a~%b", '~')	s LIKE "a~%b" ESCAPE '~'		
	SqlMethods.Like(s, "%abc%")	s LIKE '%abc%'		
SqlMethods.Like(s, "a~%b", '~')	s LIKE "a~%b" ESCAPE '~'			
Pad	Sql.PadLeft(s, a, b)	VarChar(Repeat(b, a-Length(s)), 1000) s		LPad(s, a, b)
	Sql.PadRight(s, a, b)	s VarChar(Repeat(b, a-Length(s)), 1000)		RPad(s, a, b)
Replace	Sql.Replace(s, a, b)	Replace(s, a, b)		
Reverse	Sql.Reverse(s)		X	Reverse(s)
Space	Sql.Space(a)	VarChar(Repeat(' ', a), 1000)		RPad(' ', a, '')
Stuff	Sql.Stuff(s, a, b, c)	Substring(s, 1, a-1) + c + Substring(s, a+b, Length(s)-a-1)		
Substring	Sql.Left(s, a)	Left(s, a)		Substr(s, 1, a)
	Sql.Right(s, a)	Right(s, a)		Substr(s, Length(s)-a+1, a)
	Sql.Substring(s, a, b)	Substr(s, a, b)		
Trim	Sql.Trim(s)	Trim(s)		
	Sql.TrimLeft(s)	LTrim(s)		
	Sql.TrimRight(s)	RTrim(s)		

Extended Linq string functions.

To support methods listed below provider must implement all used functions.

	Linq	All data
Case	s.ToLower ()	Sql.Lower(s)

	s.ToUpper ()	Sql.Upper(s)
Compare	s.CompareTo (a)	s > a ? 1 : s == 0 ? 0 : -1
IndexOf	s.IndexOf (a)	Sql.Length(a) == 0 ? 0 : (Sql.CharIndex(a, s) ?? 0) - 1
	s.IndexOf (a, b)	Sql.Length(a) == 0 && Sql.Length(s) > b ? b : (Sql.CharIndex(a
	s.IndexOf (a, b, c)	Sql.Length(a) == 0 && Sql.Length(s) > b ? b : (Sql.CharIndex(a
Insert	s.Insert (a, b)	Sql.Length(s) == a ? s + b : Sql.Stuff(s, a + 1, 0, b)
LastIndexOf	s.LastIndexOf(a)	Sql.Length(a) == 0 ? Sql.Length(s) - 1 : (Sql.CharIndex(a, s) ?? 0) == 0 ? -1 : Sql.Length(s) - (Sql.CharIndex(Sql.Reverse(a), Sql.Reverse(s)))
	s.LastIndexOf(a, b)	Sql.Length(a) == 0 ? b : (Sql.CharIndex(a, s, b + 1) ?? 0) == 0 ? -1 : Sql.Length(s) - (Sql.CharIndex(Sql.Reverse(a), Sql.Reverse(Sq
	s.LastIndexOf(a, b, c)	Sql.Length(a) == 0 ? b : (Sql.CharIndex(a, Sql.Left(s, b + c), b + 1) ?? 0) == 0 ? -1 : b + c - (Sql.CharIndex(Sql.Reverse(a), Sql.Reverse(Sql.Substri
Length	s.Length	Sql.Length(s)
Pad	s.PadLeft (a)	Sql.PadLeft(s, a, ' ')
	s.PadLeft (a, b)	Sql.PadLeft(s, a, b)
	s.PadRight (a)	Sql.PadRight(s, a, ' ')
	s.PadRight (a, b)	Sql.PadRight(s, a, b)
Remove	s.Remove (a)	Sql.Left(s, a)
	s.Remove (a, b)	Sql.Stuff(s, a + 1, b, "")
Replace	s.Replace (a, b)	Sql.Replace(s, a, b)
Substring	s.Substring (a)	Sql.Substring(s, a + 1, Sql.Length(s) - a)
	s.Substring (a, b)	Sql.Substring(s, a + 1, b)
Trim	s.Trim ()	Sql.Trim(s)
	s.TrimEnd ()	Sql.TrimRight(s)
	s.TrimStart ()	Sql.TrimLeft(s)

Core datetime functions.

	Linq	DB2	Informix	Oracle	Firebird
Current Timestamp	Sql.CurrentTimestamp	CURRENT_TIMESTAMP	CURRENT	CURRENT_	
	Sql.GetDate()	CURRENT_TIMESTAMP	CURRENT	CURRENT_	

Extended Linq datetime functions.

	Linq	All data providers

| Current Timestamp | DateTime.Now | Sql.CurrentTimestamp |

SQL transformations.

	Linq	DB2	I
SELECT x	db.Select(() => x)	<code>SELECT x FROM SYSIBM.SYSDUMMY1 FETCH FIRST 1 ROW ONLY</code>	<code>SELEC FROM :</code>
TAKE x	query.Take(x)	<code>... FETCH FIRST x ROWS ONLY</code>	<code>SELEC</code>

Business Logic Toolkit for .NET www.bltoolkit.net

• [Introduction](#)

Abstract accessors.

- [Open and Configure](#) How to open and configure data accessors.
- [Transactions](#)

-
- [AbstractAccessor](#) Abstract accessor example.
 - [ExecuteList](#) DataAccessor + List.
 - [ExecuteDictionary](#) DataAccessor + Dictionary.
 - [ExecuteObject](#) DataAccessor + business objects.
 - [ExecuteScalar](#) DataAccessor + scalar values.
-

Abstract accessor attributes.

- [ActionName](#)
 - [CommandBehavior](#)
 - [Direction](#)
 - [Index](#)
 - [ParamName](#)
 - [ScalarFieldName](#)
 - [SqlQuery](#)
 - [ActionSprocName](#)
 - [DataSetTable](#)
 - [DiscoverParameters](#)
 - [ObjectType](#)
 - [ParamNullValue](#)
 - [ScalarSource](#)
 - [Custom SqlQuery 1](#)
 - [ActualType](#)
 - [Destination](#)
 - [Format](#)
 - [ParamDbType](#)
 - [ParamSize](#)
 - [SprocName](#)
 - [Custom SqlQuery 2](#)
-

SqlQuery & SprocQuery

- [Open and Configure](#) How to open and configure SqlQuery & SprocQuery.
-

Basic CRUDL operations and SQL text generation.

- [SelectByKeySql](#) Reads a record by Primary Key.
 - [SelectAllSql](#) Reads all records from specified table.
 - [InsertSql](#) Inserts a new record.
 - [UpdateSql](#) Updates a record.
 - [DeleteSql](#) Deletes a record either by Primary Key or by provided object.
-

Basic CRUDL operations and stored procedures.

- [SelectByKey](#) Reads a record by Primary Key.
- [SelectAll](#) Reads all records from specified table.

- [Insert](#) Inserts a new record.
 - [Update](#) Updates a record.
 - [Delete](#) Deletes a record either by Primary Key or by provided object.
-

SQL text generation control attributes.

- [TableName](#)
 - [PrimaryKey](#)
 - [NonUpdatable](#)
 - [XmlExtension](#)
-

Business Logic Toolkit for .NET www.bltoolkit.net

Introduction

Before we start talking about the data accessors, let us create a few examples of typical data access methods. The following table contains three stored procedures and three data access methods implemented in C#.

Stored procedure	C# Method
------------------	-----------

The first stored procedure takes filter and page parameters and returns recordset from the Person table.

<pre>CREATE Procedure GetPersonListByName(@firstName varchar(50), @lastName varchar(50), @pageNumber int, @pageSize int) AS -- stored procedure implementation --</pre>	<pre>public List< string f string l int p int p > { // method implementation }</pre>
--	--

Second example will return single **Person** record by **id**.

<pre>CREATE Procedure GetPersonByID(@id int) AS -- stored procedure implementation --</pre>	<pre>public Person { // method implementation }</pre>
---	---

The last example will delete a record from the database by **id**.

<pre>CREATE Procedure DeletePersonByID(@id int) AS -- stored procedure implementation --</pre>	<pre>public void { // method implementation }</pre>
--	---

Now let's see what we can say if we compare the stored procedure and C# method signatures.

1. Stored procedure and method names match up.
2. Sequential order, method parameter types and names correspond to stored procedure.
3. Methods' return values can give us an idea what **Execute** method we should utilize data from recordset if needed.

As demonstrated above method definition contains all the information we need to implement the method signatures, we completed the most intelligent part of data accessor development job. Honestly, I got bored of being just a coding machine writing the same data a understanding that this process can be automated.

This introduction shows how to avoid the implementation step of data access development the method declaration.

Abstract classes

Unfortunately, mainstream .NET languages still do not have a compile-time transform languages do. All we have today is pre-compile- and run-time code generation.

This introduction concentrates on run-time code generation and its support by [Business I](#)

Let us step back and bring the methods from the previous examples together in one class like the following:

```
using System;
using System.Collections.Generic;

public class PersonAccessor
{
    public List<Person> GetPersonListByName(
        string firstName, string lastName, int pageNum)

    public Person GetPersonByID (int id);
    public void DeletePersonByID(int id);
}
```

The bad news about this sample is that we cannot use such syntax as the compiler expects

The good news is we can use abstract classes and methods that give us quite similar, con

```
using System;
```

```

using System.Collections.Generic;

public abstract class PersonAccessor
{
    public abstract List<Person> GetPersonListByName(
        string firstName, string lastName, int pageNum)

    public abstract Person GetPersonByID (int id);
    public abstract void DeletePersonByID(int id);
}

```

This code is 100% valid and our next step is to make it workable.

Abstract DataAccessor

Business Logic Toolkit provides the **DataAccessor** class, which is used as a base class. Adding **DataAccessor** to our previous example, it will look like the following:

```

using System;
using System.Collections.Generic;

public abstract class PersonAccessor : DataAccessor
{
    public abstract List<Person> GetPersonListByName(
        string firstName, string lastName, int pageNum)

    public abstract Person GetPersonByID (int id);
    public abstract void DeletePersonByID(int id);
}

```

That's it! Now this class is complete and fully functional. The code below shows how to

```

using System;
using System.Collections.Generic;

using BLToolkit.Reflection;

```

```

namespace DataAccess
{
    class Program
    {
        static void Main(string[] args)
        {
            PersonAccessor pa = TypeAccessor<PersonAc

            List<Person> list = pa.GetPersonListByNam

            foreach (Person p in list)
                Console.WriteLine("{0} {1}", p.FirstName,

        }
    }
}

```

The only magic here is the **TypeAccessor.CreateInstance** method. First of all this method creates the **PersonAccessor** class and then generates abstract method bodies depending on each method. If we manually, we could get something like the following:

```

using System;
using System.Collections.Generic;

using BLToolkit.Data;

namespace Example.BLToolkitExtension
{
    public sealed class PersonAccessor : Example.PersonAccessor
    {
        public override List<Person> GetPersonListByN
            string firstName,
            string lastName,
            int    pageNumber,
            int    pageSize)
        {
            using (DbManager db = GetDbManager())
            {

```

```

        return db
            .SetSpCommand("GetPersonListByNar
                db.Parameter("@firstName", 1
                db.Parameter("@lastName", 1
                db.Parameter("@pageNumber", 1
                db.Parameter("@pageSize", 1
            .ExecuteList<Person>();
    }
}

public override Person GetPersonByID(int id)
{
    using (DbManager db = GetDbManager())
    {
        return db
            .SetSpCommand("GetPersonByID", db
            .ExecuteObject<Person>();
    }
}

public override void DeletePersonByID(int id)
{
    using (DbManager db = GetDbManager())
    {
        db
            .SetSpCommand("DeletePersonByID",
            .ExecuteNonQuery();
    }
}
}
}
}

```

(The `DbManager` class is another BLToolkit class used for 'low-level' database access).

Every part of the method declaration is important. Method's return value specifies one of

Return Type	Execute Method
<code>IDataReader</code> interface	<code>ExecuteReader</code>

Subclass of <i>DataSet</i>	ExecuteDataSet
Subclass of <i>DataTable</i>	ExecuteDataTable
Class implementing the <i>IList</i> interface	ExecuteList or ExecuteScalarList
Class implementing the <i>IDictionary</i> interface	ExecuteDictionary or ExecuteScalarDictionary
<i>void</i>	ExecuteNonQuery
<i>string</i> , <i>byte[]</i> or value type	ExecuteScalar
In any other case	ExecuteObject

The method name explicitly defines the action name, which is converted to the stored name of the method parameters are mapped to the command parameters. Exceptions from

- a parameter of *DbManager* type. In this case generator uses provided *DbManager* to
- parameters decorated with attribute *FormatAttribute*, *DestinationAttribute*.

Generating process control

The **PersonAccessor** class above is a very simple example and, of course, it seems to flexibility and more control over the generated code. BLToolkit contains a bunch of attributes in addition to **DataAccessor** virtual members.

Method CreateDbManager

```
protected virtual DbManager CreateDbManager()
{
    return new DbManager();
}
```

By default, this method creates a new instance of **DbManager** that uses default database configuration by overriding this method. For example:

```
public abstract class OracleDataAccessor : DataAccessor
{
    protected override DbManager CreateDbManager()
    {
        return new DbManager("Oracle", "Production");
    }
}
```

This code will use the *Oracle* data provider and *Production* configuration.

Method `GetDefaultSpName`

```
protected virtual string GetDefaultSpName(string typeName)
{
    return typeName == null?
        actionName:
        string.Format("{0}_{1}", typeName, actionName)
}
```

As I mentioned, the method name explicitly defines the so-called action name. The following is the default implementation of the `GetDefaultSpName` method. The default implementation uses the following naming convention:

- If type name is provided, the method constructs the stored proc name by concatenating the type name and the action name. For example, if the type name is "Person" and the action name is "GetAll", the resulting stored procedure name will be "Person_GetAll".
- If the type name is NOT provided, the stored procedure name will equal the action name.

You can easily change this behavior. For example, for the naming convention "p_Person_GetAll", you can change the implementation to the following:

```
public abstract class MyBaseDataAccessor<T,A> : DataAccessor<T,A>
    where A : DataAccessor<T,A>
{
    protected override string GetDefaultSpName(string typeName)
    {
        return string.Format("p_{0}_{1}", typeName, actionName)
    }
}
```

Method `GetTableName`

```
protected virtual string GetTableName(Type type)
{
    // ...
    return type.Name;
}
```

By default, the table name is the associated object type name (*Person* in our examples) when you use a data accessor. By providing a generic parameter:

```
public abstract class PersonAccessor : DataAccessor<P>
{
}
```

And by the **ObjectType** attribute:

```
[ObjectType(typeof(Person))]
public abstract class PersonAccessor : DataAccessor
{
}
```

If you want to have different table and type names in your application, you may override

```
public abstract class OracleDataAccessor<T,A> : DataAccessor<T,A>
    where A : DataAccessor<T,A>
{
    protected override string GetTableName(Type type)
    {
        return base.GetTableName(type).ToUpper();
    }
}
```

TableNameAttribute

Also, you can change the table name for a particular object type by decorating this object:

```
[TableName("PERSON")]
public class Person
{
    public int ID;
    public string FirstName;
    public string LastName;
}
```

ActionNameAttribute

This attribute allows changing the action name.

```

public abstract class PersonAccessor : DataAccessor<Person>
{
    [ActionName("GetByID")]
    protected abstract IDataReader GetByIDInternal(DbManager db)

    public Person GetByID(int id)
    {
        using (DbManager db = GetDbManager())
        using (IDataReader rd = GetByIDInternal(db, id))
        {
            Person p = new Person();

            // do something complicated.

            return p;
        }
    }
}

```

ActionSprocNameAttribute

This attribute associates the action name with a stored procedure name:

```

[ActionSprocName("Insert", "sp_Person_Insert")]
public abstract class PersonAccessor : DataAccessor<Person>
{
    public abstract void Insert(Person p);
}

```

This attribute can be useful when you need to reassign a stored procedure name for a method.

SprocNameAttribute

The regular way to assign a different default sproc name for a method is the **SprocNameAttribute**.

```

public abstract class PersonAccessor : DataAccessor<Person>
{

```

```
[SprocName("sp_Person_Insert")]
public abstract void Insert(Person p);
}
```

DestinationAttribute

By default, the *DataAccessor* generator uses method's return value to determine which *E* current operation. The **DestinationAttribute** indicates that target object is a parameter d

```
public abstract class PersonAccessor : DataAccessor<P
{
    public abstract void GetAll([Destination] List<P
}
```

DirectionAttributes

DataAccessor generator can map provided business object to stored procedure parameter process more precisely.

```
public abstract class PersonAccessor : DataAccessor<P
{
    public abstract void Insert(
        [Direction.Output("ID"), Direction.Ignore("L
}
```

In addition, BLToolkit provides two more direction attributes: **Direction.ReturnValueAttribute**.

DiscoverParametersAttribute

By default, BLToolkit expects method parameter names to match stored procedure parameters is not important in this case. This attribute enforces BLToolkit to retrieve j assign method parameters in the order they go. Parameter names are ignored.

FormatAttribute

This attribute indicates that the specified parameter is used to construct the stored proced

```
public abstract class PersonAccessor : DataAccessor<P
```

```

{
    [SqlQuery("SELECT {0} FROM {1} WHERE {2}")]
    public abstract List<string> GetStrings(
        [Format(0)] string fieldName,
        [Format(1)] string tableName,
        [Format(2)] string whereClause);
}

```

IndexAttribute

If you want your method to return a dictionary, you will have to specify fields to build it. Here is how you can do that:

```

public abstract class PersonAccessor : DataAccessor<Person>
{
    [SqlQuery("SELECT * FROM Person")]
    [Index("ID")]
    public abstract Dictionary<int, Person> GetAll();

    [SqlQuery("SELECT * FROM Person")]
    [Index("@PersonID", "LastName")]
    public abstract Dictionary<CompoundValue, Person> GetAllByLastName();
}

```

Note: if your key has more than one field, the type of this key should be **CompoundValue**.

If the field name starts from '@' symbol, BLToolkit reads the field value from data source explicitly.

ParamNameAttribute

By default, the method parameter name should match the stored procedure parameter name explicitly.

```

public abstract class PersonAccessor : DataAccessor<Person>
{
    public abstract Person SelectByName(
        [ParamName("FirstName")] string name1,
        [ParamName("@LastName")] string name2);
}

```

```
}
```

ScalarFieldNameAttribute

If your method returns a dictionary of scalar values, you will have to specify the name of the list. The **ScalarFieldName** attribute allows you to do that:

```
public abstract class PersonAccessor : DataAccessor<F
{
    [SqlQuery("SELECT * FROM Person")]
    [Index("@PersonID")]
    [ScalarFieldName("FirstName")]
    public abstract Dictionary<int, string>

    [SqlQuery("SELECT * FROM Person")]
    [Index("PersonID", "LastName")]
    [ScalarFieldName("FirstName")]
    public abstract Dictionary<CompoundValue, string>
}
```

ScalarSourceType

If a method returns a scalar value, this attribute can be used to specify how database records are retrieved. It takes a parameter of the **ScalarSourceType** type:

ScalarSourceType	Description
DataReader	Calls the DbManager.ExecuteReader method, and then calls IDataReader.GetV
OutputParameter	Calls the DbManager.ExecuteNonQuery method, and then reads value from the I
ReturnValue	Calls the DbManager.ExecuteNonQuery method, and then reads return value fro
AffectedRows	Calls the DbManager.ExecuteNonQuery method, and then returns its return valu

SqlQueryAttribute

This attribute allows specifying SQL statement.

```
public abstract class PersonAccessor : DataAccessor<F
{
    [SqlQuery("SELECT * FROM Person WHERE PersonID =
```

```
public abstract Person GetByID(int @id);  
}
```

Conclusion

I hope this brief tutorial demonstrates one of the simplest, quickest and most low-ma layer. In addition, you will get one more benefit, which is incredible object mapping per later.

Business Logic Toolkit for .NET www.bltoolkit.net

The following example demonstrates how to create and use an abstract data accessor class. The class and its methods are generated at run-time depending on each method declaration. Every part of the Method's return value specifies one of the Execute methods in the following way:

Return Type	Execute Method
<i>IDataReader</i> interface	ExecuteReader
Subclass of <i>DataSet</i>	ExecuteDataSet
Subclass of <i>DataTable</i>	ExecuteDataTable
Class implementing the <i>IList</i> interface	ExecuteList or ExecuteScalarList
Class implementing the <i>IDictionary</i> interface	ExecuteDictionary or ExecuteScalarDictionary
<i>void</i>	ExecuteNonQuery
<i>string</i> , <i>byte[]</i> or value type	ExecuteScalar
In any other case	ExecuteObject

Method name explicitly defines action name which is converted to stored procedure name. Type, sequential order, and name of the method parameters is mapped to the command rule are:

- a parameter of *DbManager* type. In this case generator uses provided *DbManager* to
- parameters decorated with attribute *FormatAttribute*.

AbstractAccessor.cs

```
using System;
using System.Collections.Generic;

using NUnit.Framework;

using BLToolkit.Data;
using BLToolkit.DataAccess;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class AbstractAccessor
    {
        public abstract class PersonAccessor : DataAc
```

```

{
    public abstract Person SelectByName(Person p)
    public abstract Person SelectByName(string name)

    public abstract int Insert (Person p)

    [SqlQuery("SELECT Top {0} * FROM Person (
    [Index("ID")]
    public abstract Dictionary<int,Person> SelectAll()

    private SprocQuery<Person> _query;
    public SprocQuery<Person> Query
    {
        get
        {
            if (_query == null)
                _query = new SprocQuery<Person>();
            return _query;
        }
    }
}

[Test]
public void Test()
{
    using (DbManager db = new DbManager())
    {
        PersonAccessor pa = DataAccessor.Create(db);

        pa.BeginTransaction();

        // Insert and get id.
        //
        Person person = new Person();

        person.FirstName = "Crazy";
        person.LastName = "Frog";
    }
}

```



```

        [MapField("PersonID"), PrimaryKey, NonUpdateable]
        public int ID;

        public string LastName;
        public string FirstName;
        public string MiddleName;
        public Gender Gender;
    }
}

```

Gender.cs

```

using System;

using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public enum Gender
    {
        [MapValue("F")] Female,
        [MapValue("M")] Male,
        [MapValue("U")] Unknown,
        [MapValue("O")] Other
    }
}

```

App.config

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value  = "Server=.;Database=BLToolkitData;
    </appSettings>
  </configuration>

```

Create.sql script

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

ExecuteDictionary.cs

```
using System;
using System.Collections.Generic;

using NUnit.Framework;

using BLToolkit.Common;
using BLToolkit.DataAccess;
using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class ExecuteDictionary1
    {
        public class Person
        {
            [MapField("PersonID"), PrimaryKey]
            public int ID;

            public string LastName;
            public string FirstName;
            public string MiddleName;
        }

        public abstract class PersonAccessor : DataAccess
        {
            // This method uses Person class primary
            //
            [ActionName("SelectAll")]
            public abstract Dictionary<int, Person> Get
            // Define index field explicitly. "ID" is

```

```

//
[ActionName("SelectAll")]
[Index("ID")]
public abstract Dictionary<int, Person> GetPersonDictionary1()

// Define index field explicitly. "@PersonID"
// Note that the '@' symbol enforces the index to be taken
// from recordset (not from object).
//
[ActionName("SelectAll")]
[Index("@PersonID")]
public abstract Dictionary<int, Person> GetPersonDictionary2()

// This method reads a dictionary containing the first name
//
[ActionName("SelectAll")]
[SqlQuery("SELECT PersonID, FirstName FROM Person")]
[Index("PersonID")]
[ScalarFieldName("FirstName")]
public abstract Dictionary<int, string> GetPersonDictionary3()
}

[Test]
public void Test()
{
    PersonAccessor pa = DataAccessor.CreateInstance();

    // ExecuteDictionary.
    //
    Dictionary<int, Person> dic;

    dic = pa.GetPersonDictionary1();
    dic = pa.GetPersonDictionary2();
    dic = pa.GetPersonDictionary3();

    foreach (int id in dic.Keys)
        Console.WriteLine("{0}: {1} {2}", id, dic[id].FirstName, dic[id].LastName);
}

```

```

        // ExecuteScalarDictionary.
        //
        Dictionary<int, string> sdic = pa.GetPersonDictionary();
        foreach (int id in dic.Keys)
            Console.WriteLine("{0}: {1}", id, sdic[id]);
    }
}

```

```

[TestFixture]
public class ExecuteDictionary2
{
    // This example demonstrates how to use a multi-key dictionary.
    //
    public class Person
    {
        [PrimaryKey(1), MapField("PersonID")]
        public int ID;
        [PrimaryKey(2)]
        public string LastName;

        public string FirstName;
        public string MiddleName;
    }

    public abstract class PersonAccessor : DataAccess
    {
        // This method uses Person class primary key.
        // Note that the key type of the dictionary is int.
        // It is required if the index consists of multiple keys.
        //
        [ActionName("SelectAll")]
        public abstract Dictionary<CompoundValue, Person> GetAll();

        // This method reads a dictionary containing Person objects.
        //
        [SqlQuery("SELECT PersonID, LastName, FirstName, MiddleName FROM Person")]
    }
}

```

```

        [Index("PersonID", "LastName")]
        [ScalarFieldName("FirstName")]
        public abstract Dictionary<CompoundValue,
    }

[Test]
public void Test()
{
    PersonAccessor pa = DataAccessor.CreateIn

    // ExecuteDictionary.
    //
    Dictionary<CompoundValue, Person> dic = pa

    foreach (CompoundValue idx in dic.Keys)
        Console.WriteLine("{0}: {1} {2}", dic

    // ExecuteScalarDictionary.
    //
    Dictionary<CompoundValue, string> sdic = p

    string firstName = sdic[new CompoundValue

    Assert.AreEqual("Tester", firstName);
}
}
}

```

App.config

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value  = "Server=.;Database=BLToolkitData;
    </appSettings>

```

```
</configuration>
```

[Create.sql script](#)

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

ExecuteList.cs

```
using System;
using System.Collections;
using System.Collections.Generic;

using NUnit.Framework;

using BLToolkit.DataAccess;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class ExecuteList
    {
        public abstract class PersonAccessor : DataAccess
        {
            // This method reads a list of Person objects
            //
            [ActionName("SelectAll")]
            public abstract List<Person> GetPersonList();

            // Here we help the method to get object
            // ObjectTypeAttribute can be applied to
            // In this case there is no need to specify
            // Another way to specify object type is
            // of the DataAccessor<T> class.
            //
            [SqlQuery("SELECT * FROM Person")]
            [ObjectType(typeof(Person))]
            public abstract ArrayList GetPersonList2();

            // This method reads a list of scalar values
            //
        }
    }
}
```

```

        [SqlQuery("SELECT PersonID FROM Person")]
        public abstract List<int> GetPersonIDList()
    }

    [Test]
    public void Test()
    {
        PersonAccessor pa = DataAccessor.CreateInstance();

        // ExecuteList.
        //
        IList list;

        list = pa.GetPersonList1();
        list = pa.GetPersonList2();

        foreach (Person p in list)
            Console.WriteLine("{0}: {1} {2}", p.ID, p.Name, p.Address);

        // ExecuteScalarList.
        //
        List<int> slist = pa.GetPersonIDList();

        foreach (int id in slist)
            Console.WriteLine("{0}", id);
    }
}

```

Person.cs

```

using System;

using BLToolkit.DataAccess;
using BLToolkit.Mapping;

namespace HowTo.DataAccess

```

```

{
    public class Person
    {
        [MapField("PersonID"), PrimaryKey, NonUpdateable]
        public int ID;

        public string LastName;
        public string FirstName;
        public string MiddleName;
        public Gender Gender;
    }
}

```

Gender.cs

```

using System;

using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public enum Gender
    {
        [MapValue("F")] Female,
        [MapValue("M")] Male,
        [MapValue("U")] Unknown,
        [MapValue("O")] Other
    }
}

```

App.config

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value = "Server=.;Database=BLToolkitData;

```

```
</appSettings>  
</configuration>
```

[Create.sql script](#)

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

ExecuteObject.cs

```
using System;

using NUnit.Framework;

using BLToolkit.DataAccess;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class ExecuteObject
    {
        public abstract class PersonAccessor : DataAc
        {
            // Here we explicitly specify a stored pr
            //
            [SprocName("Person_SelectByKey")]
            public abstract Person GetByID(int @id);

            // SQL query text.
            //
            [SqlQuery("SELECT * FROM Person WHERE Per
            public abstract Person GetPersonByID(int

            // Specify action name.
            // Stored procedure name is generated bas
            // defined by DataAccessor.GetDefaultSpNa
            //
            [ActionName("SelectByName")]
            public abstract Person GetPersonByName(st

            // By default method name defines an act:
            // which is converted to a stored procedu
```

```

        // Default conversion rule is ObjectName_
        // This method calls the Person_SelectByI
        //
        public abstract Person SelectByName(strin
    }

[Test]
public void Test()
{
    PersonAccessor pa = DataAccessor.CreateIr

    // ExecuteObject.
    //
    Assert.IsNotNull(pa.GetByID           (1));
    Assert.IsNotNull(pa.GetPersonByID     (2));
    Assert.IsNotNull(pa.GetPersonByName("Test
    Assert.IsNotNull(pa.SelectByName     ("Test

}
}
}

```

Person.cs

```

using System;

using BLToolkit.DataAccess;
using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public class Person
    {
        [MapField("PersonID"), PrimaryKey, NonUpdatable]
        public int ID;

        public string LastName;
        public string FirstName;
    }
}

```

```
        public string MiddleName;
        public Gender Gender;
    }
}
```

Gender.cs

```
using System;

using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public enum Gender
    {
        [MapValue("F")] Female,
        [MapValue("M")] Male,
        [MapValue("U")] Unknown,
        [MapValue("O")] Other
    }
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value = "Server=.;Database=BLToolkitData,
    </appSettings>
  </configuration>
```

Create.sql script

Business Logic Toolkit for .NET www.bltoolkit.net

ExecuteScalar.cs

```
using System;

using NUnit.Framework;

using BLToolkit.DataAccess;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class ExecuteScalar
    {
        public abstract class PersonAccessor : DataAccess
        {
            [SqlQuery("SELECT Count(*) FROM Person")]
            public abstract int GetCount();

            // The Person_Insert sproc returns an id
            //
            [SprocName("Person_Insert")]
            public abstract int Insert(Person person);
        }

        [Test]
        public void Test()
        {
            PersonAccessor pa = DataAccessor.CreateIr

            // ExecuteScalar.
            //
            Assert.IsTrue(pa.GetCount() > 0);

            // Insert and get id.
        }
    }
}
```

```

        //
        Person person = new Person();

        person.FirstName = "Crazy";
        person.LastName  = "Frog";
        person.Gender    = Gender.Unknown;

        int id = pa.Insert(person);

        Assert.IsFalse(id == 0);

        new SprocQuery<Person>().DeleteByKey(id);
    }
}

```

Person.cs

```

using System;

using BLToolkit.DataAccess;
using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public class Person
    {
        [MapField("PersonID"), PrimaryKey, NonUpdateable]
        public int ID;

        public string LastName;
        public string FirstName;
        public string MiddleName;
        public Gender Gender;
    }
}

```

Gender.cs

```
using System;

using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public enum Gender
    {
        [MapValue("F")] Female,
        [MapValue("M")] Male,
        [MapValue("U")] Unknown,
        [MapValue("O")] Other
    }
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value  = "Server=.;Database=BLToolkitData,
    </appSettings>
  </configuration>
```

[Create.sql script](#)

Business Logic Toolkit for .NET www.bltoolkit.net

OpenConfig.cs

```
using System;

using NUnit.Framework;

using BLToolkit.Data;
using BLToolkit.DataAccess;
using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class OpenConfig
    {
        public class Person
        {
            [MapField("PersonID"), PrimaryKey, NonUpdatable]
            public int ID;

            public string LastName;
            public string FirstName;
            public string MiddleName;
        }

        public abstract class TestAccessor : DataAccessor
        {
            public abstract Person SelectByKey(int id);
            public abstract Person SelectByKey(DbManager dbManager);
        }

        // DbManager is created by DataAccessor.
        //
        [Test]
    }
}
```

```
public void Test1()
{
    TestAccessor ta = DataAccessor.CreateInst

    Person person = ta.SelectByKey(1);

    Assert.IsNotNull(person);
}
```

```
// DataAccessor takes DbManager as a parameter
//
```

```
[Test]
public void Test2()
{
    using (DbManager db = new DbManager())
    {
        TestAccessor ta = DataAccessor.Create

        Person person = ta.SelectByKey(1);

        Assert.IsNotNull(person);
    }
}
```

```
// DataAccessor method takes DbManager as a parameter
//
```

```
[Test]
public void Test3()
{
    using (DbManager db = new DbManager())
    {
        TestAccessor ta = DataAccessor.Create

        Person person = ta.SelectByKey(db, 1);

        Assert.IsNotNull(person);
    }
}
```

```
}  
  }  
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>  
<configuration>  
  <appSettings>  
    <add  
      key    = "ConnectionString"  
      value = "Server=.;Database=BLToolkitData;  
    </appSettings>  
</configuration>
```

Create.sql script

Business Logic Toolkit for .NET www.bltoolkit.net

Transaction.cs

```
using System;

using NUnit.Framework;

using BLToolkit.Data;
using BLToolkit.DataAccess;
using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class Transaction
    {
        public abstract class TestAccessor : DataAccessor
        {
            public abstract int Insert(Person person);
            public abstract void Delete(int @PersonID);

            public abstract Person SelectByKey(int id);
            public abstract Person SelectByKey(DbManager db);
        }

        // DataAccessor takes DbManager as a parameter
        //
        [Test]
        public void Test1()
        {
            using (DbManager db = new DbManager())
            {
                TestAccessor ta = DataAccessor.Create<TestAccessor>(db);

                ta.BeginTransaction();
            }
        }
    }
}
```

```

        int id = ta.Insert(new Person { First
        Assert.AreEqual(0, id);

        Person person = ta.SelectByKey(id);
        Assert.IsNotNull(person);

        ta.Delete(id);

        ta.CommitTransaction();
    }
}

// DataAccessor method takes DbManager as a p
//
[Test]
public void Test2()
{
    using (DbManager db = new DbManager())
    {
        db.BeginTransaction();

        TestAccessor ta = DataAccessor.Create

        int id = ta.Insert(new Person { First
        Assert.AreEqual(0, id);

        Person person = ta.SelectByKey(db, id
        Assert.IsNotNull(person);

        ta.Delete(id);

        db.CommitTransaction();
    }
}
}
}
}
}

```

Person.cs

```
using System;

using BLToolkit.DataAccess;
using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public class Person
    {
        [MapField("PersonID"), PrimaryKey, NonUpdateable]
        public int ID;

        public string LastName;
        public string FirstName;
        public string MiddleName;
        public Gender Gender;
    }
}
```

Gender.cs

```
using System;

using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public enum Gender
    {
        [MapValue("F")] Female,
        [MapValue("M")] Male,
        [MapValue("U")] Unknown,
        [MapValue("O")] Other
    }
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value = "Server=.;Database=BLToolkitData;
    </add>
  </appSettings>
</configuration>
```

[Create.sql script](#)

Business Logic Toolkit for .NET www.bltoolkit.net

ActionName is a logical name of an operation. By default a method name is an **ActionNameAttribute** allows overriding this behavior. Actual stored procedure name depending on naming convention which is defined in the [GetDefaultSpName](#) method.

ActionName.cs

```
using System;
using NUnit.Framework;
using BLToolkit.DataAccess;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class ActionName
    {
        public abstract class PersonAccessor : DataAc
        {
            // Default action name is 'SelectByKey'.
            // Stored procedure name is 'Person_Select
            //
            public abstract Person SelectByKey(int @:

            // Explicit action name is 'SelectByName'
            // Stored procedure name is 'Person_Select
            //
            [ActionName("SelectByName")]
            public abstract Person AnyName (string

        }

        [Test]
        public void Test()
        {
            PersonAccessor pa = PersonAccessor.Create
```

```
        Person person1 = pa.SelectByKey(1);

        Assert.IsNotNull(person1);

        Person person2 = pa.AnyName(person1.FirstName);

        Assert.AreEqual(person1.ID, person2.ID);
    }
}
}
```

Person.cs

```
using System;

using BLToolkit.DataAccess;
using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public class Person
    {
        [MapField("PersonID"), PrimaryKey, NonUpdateable]
        public int ID;

        public string LastName;
        public string FirstName;
        public string MiddleName;
        public Gender Gender;
    }
}
```

Gender.cs

```
using System;

using BLToolkit.Mapping;
```

```
namespace HowTo.DataAccess
{
    public enum Gender
    {
        [MapValue("F")] Female,
        [MapValue("M")] Male,
        [MapValue("U")] Unknown,
        [MapValue("O")] Other
    }
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value = "Server=.;Database=BLToolkitData;"
    </appSettings>
  </configuration>
```

Create.sql script

Business Logic Toolkit for .NET www.bltoolkit.net

The **ActionSprocName** attribute is applied to a business object (NOT to data accessor) and specifies the stored procedure name.

ActionSprocName.cs

```
using System;
using NUnit.Framework;
using BLToolkit.DataAccess;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class ActionSprocName
    {
        [ActionSprocName("GetByName", "Person_Select")]
        public class Person
        {
            public int    PersonID;
            public string LastName;
            public string FirstName;
            public string MiddleName;
        }

        public abstract class PersonAccessor : DataAccessor
        {
            // Default action name is 'SelectByKey'.
            // Stored procedure name is 'Person_SelectByKey'.
            //
            public abstract Person SelectByKey(int @key);

            // Default action name is 'GetByName'.
            // Stored procedure name is 'Person_SelectByName'.
            // defined by the ActionSprocName attribute.
            //
        }
    }
}
```

```

        public abstract Person GetByName(string name)
    }

    [Test]
    public void Test()
    {
        PersonAccessor pa = DataAccessor.CreateInstance(
            "DataAccessor");

        Person person1 = pa.SelectByKey(1);

        Assert.IsNotNull(person1);

        Person person2 = pa.GetByName(person1.FirstName);

        Assert.AreEqual(person1.PersonID, person2.PersonID);
    }
}

```

App.config

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value = "Server=.;Database=BLToolkitData;
    </appSettings>
  </configuration>

```

Create.sql script

Business Logic Toolkit for .NET www.bltoolkit.net

The **ActualType** attribute associates an actual type with the type returned by an abstract

ActualType.cs

```
using System.Collections.Generic;

using NUnit.Framework;

using BLToolkit.DataAccess;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class ActualType
    {
        public interface IName
        {
            string Name { get; }
        }

        public class NameBase : IName
        {
            private string _name;
            public string Name { get { return _name; } }
        }

        public class Name1 : NameBase {}
        public class Name2 : NameBase {}

        [ActualType(typeof(IName), typeof(Name1))]
        public abstract class TestAccessor : DataAcces
        {
            [SqlQuery("SELECT 'John' as Name")]
            public abstract IName GetName1();
        }
    }
}
```

```

        [SqlQuery("SELECT 'John' as Name"), Object]
        public abstract IName GetName2();

        [SqlQuery("SELECT 'John' as Name")]
        public abstract IList<IName> GetName1List();

        [SqlQuery("SELECT 'John' as Name"), Object]
        public abstract IList<IName> GetName2List();

        [SqlQuery("SELECT 1 as ID, 'John' as Name")]
        public abstract IDictionary<int, IName> GetName1Dictionary();

        [SqlQuery("SELECT 1 as ID, 'John' as Name"), Object]
        public abstract IDictionary<int, IName> GetName2Dictionary();
    }

    [Test]
    public void Test()
    {
        TestAccessor ta = DataAccessor.CreateInstance();

        Assert.IsTrue(ta.GetName1());
        Assert.IsTrue(ta.GetName2());
        Assert.IsTrue(ta.GetName1List()[0]);
        Assert.IsTrue(ta.GetName2List()[0]);
        Assert.IsTrue(ta.GetName1Dictionary()[1]);
        Assert.IsTrue(ta.GetName2Dictionary()[1]);
    }
}

```

App.config

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>

```

```
    <add
      key    = "ConnectionString"
      value  = "Server=.;Database=BLToolkitData;
    </appSettings>
</configuration>
```

[Create.sql script](#)

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

The **CommandBehavior** attribute provides a description of the results of the query and i

CommandBehavior.cs

```
using System.Data;

using NUnit.Framework;

using BLToolkit.Data;
using BLToolkit.DataAccess;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class CommandBehaviorDemo
    {
        public abstract class TestAccessor : DataAcces
        {
            [SprocName("Person_SelectAll"), CommandBe
            public abstract IDataReader SelectAllIDat
        }

        [Test]
        public void Test()
        {
            TestAccessor ta = DataAccessor.CreateInst

            using (DbManager db = ta.GetDbManager())
            using (IDataReader dr = ta.SelectAllIData
            {
                DataTable table = dr.GetSchemaTable()

                Assert.AreEqual("PersonID", table.Re
                Assert.AreEqual(typeof(int), table.Re
            }
        }
    }
}
```

```
}  
  }  
} }
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>  
<configuration>  
  <appSettings>  
    <add  
      key    = "ConnectionString"  
      value = "Server=.;Database=BLToolkitData;  
    </appSettings>  
</configuration>
```

[Create.sql script](#)

Business Logic Toolkit for .NET www.bltoolkit.net

The **DataSetTable** attribute specifies table to be populated by the method in destination

DataSetTable.cs

```
using System;
using System.Data;

using NUnit.Framework;

using BLToolkit.DataAccess;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class DataSetTable
    {
        public abstract class TestAccessor : DataAcces
        {
            [SprocName("Person_SelectAll"), DataSetTa]
            public abstract void SelectFirstTable (|

            [SprocName("Person_SelectAll"), DataSetTa]
            public abstract void SelectSecondTable (|

            [SprocName("Person_SelectAll"), DataSetTa]
            public abstract void SelectFirstTable2 (|

            [SprocName("Person_SelectAll"), DataSetTa]
            public abstract void SelectSecondTable2(|

        }

        [Test]
        public void Test()
        {
```

```

        TestAccessor ta = DataAccessor.CreateInst

        DataSet ds = new DataSet();

        ta.SelectFirstTable (ds);
        ta.SelectSecondTable (ds);
        ta.SelectFirstTable2 (ds);
        ta.SelectSecondTable2(ds);

        Assert.IsTrue (ds.Tables.Contains("First'
        Assert.IsTrue (ds.Tables.Contains("Second
        Assert.IsFalse(ds.Tables.Contains("Table'
    }
}
}

```

App.config

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value  = "Server=.;Database=BLToolkitData;
    </appSettings>
  </configuration>

```

Create.sql script

Business Logic Toolkit for .NET www.bltoolkit.net

Return value of an abstract method defines a result that this method returns. The **Destination** method parameters as the result.

Destination.cs

```
using System;
using System.Collections.Generic;

using NUnit.Framework;

using BLToolkit.DataAccess;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class Destination
    {
        public abstract class PersonAccessor : DataAc
        {
            public abstract void SelectAll([Destinat
        }

        [Test]
        public void Test()
        {
            PersonAccessor pa = DataAccessor.CreateIr

            List<Person> list = new List<Person>();

            pa.SelectAll(list);

            Assert.AreNotEqual(0, list.Count);
        }
    }
}
```

```
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>  
<configuration>  
  <appSettings>  
    <add  
      key    = "ConnectionString"  
      value = "Server=.;Database=BLToolkitData;  
    </appSettings>  
</configuration>
```

Create.sql script

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

The **Direction** attributes allow controlling mapping from/to a business object. The follow

Attribute	Description
Output	Provided members are output parameters of the query.
InputOutput	Provided members are input/output parameters of the query.
Ignore	Provided members are excluded from mapping.
ReturnValue	Provided member is mapped to return value.

Direction.cs

```
using System;
using NUnit.Framework;
using BLToolkit.DataAccess;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class DirectionTest
    {
        public abstract class PersonAccessor : DataAccess
        {
            [SprocName("Person_Insert_OutputParameter")]
            public abstract void Insert_OutputParameter(
                string PersonID, string FirstName, string LastName);

            [SprocName("Scalar_ReturnParameter")]
            public abstract void Insert_ReturnParameter(
                [Direction.ReturnValue("@PersonID"),
                 Direction.Ignore("PersonID", "FirstName", "LastName")]
                string PersonID, string FirstName, string LastName);

            [SprocName("Scalar_ReturnParameter")]
            public abstract void Insert_ReturnParameter(
                [Direction.ReturnValue("ID"),
                 Direction.Ignore("PersonID", "FirstName", "LastName")]
                string PersonID, string FirstName, string LastName);
        }
    }
}
```

```

PersonAccessor Accessor
{
    get { return DataAccessor.CreateInstance<
}

[Test]
public void TestOutputParameter()
{
    Person p = new Person { FirstName = "Craz
    Accessor.Insert_OutputParameter(p);
    Assert.IsTrue(p.ID > 0);
    new SprocQuery().Delete(p);
}

[Test]
public void TestReturnParameter()
{
    Person p = new Person();
    Accessor.Insert_ReturnParameter(p);
    Assert.AreEqual(12345, p.ID);
}

[Test]
public void TestReturnParameter2()
{
    Person p = new Person();
    Accessor.Insert_ReturnParameter2(p);
    Assert.AreEqual(12345, p.ID);
}

```

```
}  
}
```

Person.cs

```
using System;  
  
using BLToolkit.DataAccess;  
using BLToolkit.Mapping;  
  
namespace HowTo.DataAccess  
{  
    public class Person  
    {  
        [MapField("PersonID"), PrimaryKey, NonUpdateable]  
        public int ID;  
  
        public string LastName;  
        public string FirstName;  
        public string MiddleName;  
        public Gender Gender;  
    }  
}
```

Gender.cs

```
using System;  
  
using BLToolkit.Mapping;  
  
namespace HowTo.DataAccess  
{  
    public enum Gender  
    {  
        [MapValue("F")] Female,  
        [MapValue("M")] Male,  
        [MapValue("U")] Unknown,  
        [MapValue("O")] Other  
    }  
}
```

```
}  
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>  
<configuration>  
  <appSettings>  
    <add  
      key    = "ConnectionString"  
      value = "Server=.;Database=BLToolkitData;  
    </appSettings>  
</configuration>
```

[Create.sql script](#)

Business Logic Toolkit for .NET www.bltoolkit.net

By default, BLToolkit expects method parameter names to match stored procedure parameters is not important in this case. The **DiscoverParameters** attribute enforces E from the sproc and to assign method parameters in the order they go. Parameter names are

DiscoverParameters.cs

```
using NUnit.Framework;
using BLToolkit.DataAccess;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class DiscoverParameters
    {
        public abstract class PersonAccessor : DataAcce
        {
            [DiscoverParameters]
            public abstract Person SelectByName(strin
        }

        [Test]
        public void Test()
        {
            PersonAccessor pa = DataAccessor.CreateIn
            Person p = pa.SelectByName("Test

            Assert.AreEqual(2, p.ID);
        }
    }
}
```

Person.cs

```
using System;
```

```

using BLToolkit.DataAccess;
using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public class Person
    {
        [MapField("PersonID"), PrimaryKey, NonUpdateable]
        public int ID;

        public string LastName;
        public string FirstName;
        public string MiddleName;
        public Gender Gender;
    }
}

```

Gender.cs

```

using System;

using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public enum Gender
    {
        [MapValue("F")] Female,
        [MapValue("M")] Male,
        [MapValue("U")] Unknown,
        [MapValue("O")] Other
    }
}

```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value  = "Server=.;Database=BLToolkitData;
    </add>
  </appSettings>
</configuration>
```

[Create.sql script](#)

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

The **Format** indicates that the specified parameter is used to construct the stored procedure.

Format.cs

```
using System;
using System.Collections.Generic;

using NUnit.Framework;

using BLToolkit.DataAccess;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class Format
    {
        public abstract class PersonAccessor : DataAccessor
        {
            [SqlQuery("SELECT TOP {0} * FROM Person")]
            public abstract List<Person> GetPersonList(int count);
        }

        [Test]
        public void Test()
        {
            PersonAccessor pa = DataAccessor.Create<PersonAccessor>();
            List<Person> list = pa.GetPersonList(2);

            Assert.That(list, Is.Not.Null);
            Assert.That(list.Count, Is.LessThanOrEqualTo(2));
        }
    }
}
```

Person.cs

```
using System;

using BLToolkit.DataAccess;
using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public class Person
    {
        [MapField("PersonID"), PrimaryKey, NonUpdateable]
        public int ID;

        public string LastName;
        public string FirstName;
        public string MiddleName;
        public Gender Gender;
    }
}
```

Gender.cs

```
using System;

using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public enum Gender
    {
        [MapValue("F")] Female,
        [MapValue("M")] Male,
        [MapValue("U")] Unknown,
        [MapValue("O")] Other
    }
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value = "Server=.;Database=BLToolkitData;
    </appSettings>
  </configuration>
```

[Create.sql script](#)

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

ExecuteDictionary.cs

```
using System;
using System.Collections.Generic;

using NUnit.Framework;

using BLToolkit.Common;
using BLToolkit.DataAccess;
using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class ExecuteDictionary1
    {
        public class Person
        {
            [MapField("PersonID"), PrimaryKey]
            public int ID;

            public string LastName;
            public string FirstName;
            public string MiddleName;
        }

        public abstract class PersonAccessor : DataAccess
        {
            // This method uses Person class primary
            //
            [ActionName("SelectAll")]
            public abstract Dictionary<int, Person> Get
            // Define index field explicitly. "ID" is

```

```

//
[ActionName("SelectAll")]
[Index("ID")]
public abstract Dictionary<int, Person> GetPersonDictionary1()

// Define index field explicitly. "@PersonID"
// Note that the '@' symbol enforces the index to be retrieved
// from recordset (not from object).
//
[ActionName("SelectAll")]
[Index("@PersonID")]
public abstract Dictionary<int, Person> GetPersonDictionary2()

// This method reads a dictionary containing Person objects
//
[ActionName("SelectAll")]
[SqlCommand("SELECT PersonID, FirstName FROM Person")]
[Index("PersonID")]
[ScalarFieldName("FirstName")]
public abstract Dictionary<int, string> GetPersonDictionary3()
}

[Test]
public void Test()
{
    PersonAccessor pa = DataAccessor.CreateInstance();

    // ExecuteDictionary.
    //
    Dictionary<int, Person> dic;

    dic = pa.GetPersonDictionary1();
    dic = pa.GetPersonDictionary2();
    dic = pa.GetPersonDictionary3();

    foreach (int id in dic.Keys)
        Console.WriteLine("{0}: {1} {2}", id, dic[id].FirstName, dic[id].LastName);
}

```

```

        // ExecuteScalarDictionary.
        //
        Dictionary<int, string> sdic = pa.GetPersonDictionary();

        foreach (int id in dic.Keys)
            Console.WriteLine("{0}: {1}", id, sdic[id]);
    }
}

```

```

[TestFixture]
public class ExecuteDictionary2
{
    // This example demonstrates how to use a multi-key dictionary.
    //
    public class Person
    {
        [PrimaryKey(1), MapField("PersonID")]
        public int ID;
        [PrimaryKey(2)]
        public string LastName;

        public string FirstName;
        public string MiddleName;
    }

    public abstract class PersonAccessor : DataAccess
    {
        // This method uses Person class primary key.
        // Note that the key type of the dictionary is string.
        // It is required if the index consists of multiple keys.
        //
        [ActionName("SelectAll")]
        public abstract Dictionary<CompoundValue, Person> GetAll();

        // This method reads a dictionary containing Person objects.
        //
        [SqlQuery("SELECT PersonID, LastName, FirstName, MiddleName FROM Person")]
    }
}

```

```

        [Index("PersonID", "LastName")]
        [ScalarFieldName("FirstName")]
        public abstract Dictionary<CompoundValue,
    }

[Test]
public void Test()
{
    PersonAccessor pa = DataAccessor.CreateIn

    // ExecuteDictionary.
    //
    Dictionary<CompoundValue,Person> dic = pa

    foreach (CompoundValue idx in dic.Keys)
        Console.WriteLine("{0}: {1} {2}", dic

    // ExecuteScalarDictionary.
    //
    Dictionary<CompoundValue,string> sdic = p

    string firstName = sdic[new CompoundValue

    Assert.AreEqual("Tester", firstName);
}
}
}
}

```

App.config

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value  = "Server=.;Database=BLToolkitData;
    </appSettings>

```

```
</configuration>
```

[Create.sql script](#)

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

The **ObjectType** attribute associates an actual type with the type returned by an abstract

ActualType.cs

```
using System.Collections.Generic;

using NUnit.Framework;

using BLToolkit.DataAccess;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class ActualType
    {
        public interface IName
        {
            string Name { get; }
        }

        public class NameBase : IName
        {
            private string _name;
            public string Name { get { return _name; } }
        }

        public class Name1 : NameBase {}
        public class Name2 : NameBase {}

        [ActualType(typeof(IName), typeof(Name1))]
        public abstract class TestAccessor : DataAcces
        {
            [SqlQuery("SELECT 'John' as Name")]
            public abstract IName GetName1();
        }
    }
}
```

```

        [SqlQuery("SELECT 'John' as Name"), Object]
        public abstract IName GetName2();

        [SqlQuery("SELECT 'John' as Name")]
        public abstract IList<IName> GetName1List();

        [SqlQuery("SELECT 'John' as Name"), Object]
        public abstract IList<IName> GetName2List();

        [SqlQuery("SELECT 1 as ID, 'John' as Name")]
        public abstract IDictionary<int, IName> GetName1Dictionary();

        [SqlQuery("SELECT 1 as ID, 'John' as Name"), Object]
        public abstract IDictionary<int, IName> GetName2Dictionary();
    }

    [Test]
    public void Test()
    {
        TestAccessor ta = DataAccessor.CreateInstance();

        Assert.IsTrue(ta.GetName1());
        Assert.IsTrue(ta.GetName2());
        Assert.IsTrue(ta.GetName1List()[0]);
        Assert.IsTrue(ta.GetName2List()[0]);
        Assert.IsTrue(ta.GetName1Dictionary()[1]);
        Assert.IsTrue(ta.GetName2Dictionary()[1]);
    }
}

```

App.config

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>

```

```
    <add
      key    = "ConnectionString"
      value  = "Server=.;Database=BLToolkitData;
    </appSettings>
</configuration>
```

[Create.sql script](#)

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

The **ParamDbType** can be used to specify parameter DbType.

Param.cs

```
using System;
using System.Data;

using NUnit.Framework;

using BLToolkit.Data;
using BLToolkit.DataAccess;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class Param
    {
        public abstract class TestAccessor : DataAccessor
        {
            [SqlQuery("SELECT {0} = {1} FROM Person V")]
            public abstract void SelectJohn(
                [ParamSize(50), ParamDbType(DbType.String)]
                [Format] string paramName,
                [Format] string fieldName);
        }

        [Test]
        public void AccessorTest()
        {
            using (DbManager db = new DbManager())
            {
                TestAccessor ta = DataAccessor.Create(
                    db,
                    string actualName;
```

```
        ta.SelectJohn(out actualName, "@name'  
        Assert.AreEqual("John", actualName);  
    }  
}  
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>  
<configuration>  
  <appSettings>  
    <add  
      key    = "ConnectionString"  
      value = "Server=.;Database=BLToolkitData;  
    </appSettings>  
</configuration>
```

Create.sql script

Business Logic Toolkit for .NET www.bltoolkit.net

By default, the method parameter name should match the stored procedure parameter name or the stored procedure parameter name explicitly.

ParamName.cs

```
using System;
using NUnit.Framework;
using BLToolkit.DataAccess;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class ParamName
    {
        public abstract class TestAccessor : DataAccessor
        {
            public abstract Person SelectByName(
                [ParamName("FirstName")] string name,
                [ParamName("@LastName")] string name2);
        }

        [Test]
        public void Test()
        {
            TestAccessor ta = DataAccessor.CreateInstance<TestAccessor>();
            Person p = ta.SelectByName("Tester", "Test");

            Assert.AreEqual(2, p.ID);
        }
    }
}
```

Person.cs

```
using System;

using BLToolkit.DataAccess;
using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public class Person
    {
        [MapField("PersonID"), PrimaryKey, NonUpdateable]
        public int ID;

        public string LastName;
        public string FirstName;
        public string MiddleName;
        public Gender Gender;
    }
}
```

Gender.cs

```
using System;

using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public enum Gender
    {
        [MapValue("F")] Female,
        [MapValue("M")] Male,
        [MapValue("U")] Unknown,
        [MapValue("O")] Other
    }
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value = "Server=.;Database=BLToolkitData,
    </appSettings>
  </configuration>
```

[Create.sql script](#)

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

The **ParamNullValue** specifies the parameter value replaced with NULL.

ParamNullValue.cs

```
using System;
using NUnit.Framework;
using BLToolkit.DataAccess;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class ParamNullValue
    {
        public abstract class TestAccessor : DataAccessor
        {
            public abstract Person SelectByKey([ParamNullValue] int id);
        }

        [Test]
        public void Test()
        {
            TestAccessor ta = DataAccessor.CreateInstance<TestAccessor>();

            // Parameter id == 1 will be replaced with null
            //
            Person p1 = ta.SelectByKey(1);
            Assert.IsNull(p1);

            // Parameter id == 2 will be send as is
            //
            Person p2 = ta.SelectByKey(2);
            Assert.IsNotNull(p2);
        }
    }
}
```

```
}
```

Person.cs

```
using System;

using BLToolkit.DataAccess;
using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public class Person
    {
        [MapField("PersonID"), PrimaryKey, NonUpdateable]
        public int ID;

        public string LastName;
        public string FirstName;
        public string MiddleName;
        public Gender Gender;
    }
}
```

Gender.cs

```
using System;

using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public enum Gender
    {
        [MapValue("F")] Female,
        [MapValue("M")] Male,
        [MapValue("U")] Unknown,
        [MapValue("O")] Other
    }
}
```

```
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>  
<configuration>  
  <appSettings>  
    <add  
      key    = "ConnectionString"  
      value = "Server=.;Database=BLToolkitData;  
    </appSettings>  
</configuration>
```

Create.sql script

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

The **ParamSize** can be used to specify db parameter size.

Param.cs

```
using System;
using System.Data;

using NUnit.Framework;

using BLToolkit.Data;
using BLToolkit.DataAccess;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class Param
    {
        public abstract class TestAccessor : DataAccessor
        {
            [SqlQuery("SELECT {0} = {1} FROM Person V")]
            public abstract void SelectJohn(
                [ParamSize(50), ParamDbType(DbType.String)]
                [Format] string paramName,
                [Format] string fieldName);
        }

        [Test]
        public void AccessorTest()
        {
            using (DbManager db = new DbManager())
            {
                TestAccessor ta = DataAccessor.Create(
                    db,
                    "SELECT {0} = {1} FROM Person V",
                    "John",
                    "Name",
                    "ID",
                    ParamDbType.String);

                string actualName;
```

```
        ta.SelectJohn(out actualName, "@name'  
        Assert.AreEqual("John", actualName);  
    }  
}  
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>  
<configuration>  
  <appSettings>  
    <add  
      key    = "ConnectionString"  
      value = "Server=.;Database=BLToolkitData;  
    </appSettings>  
</configuration>
```

Create.sql script

Business Logic Toolkit for .NET www.bltoolkit.net

If your method returns a dictionary of scalar values, you will have to specify the name of the scalar list. The **ScalarFieldName** attribute allows you to do that.

ScalarFieldName.cs

```
using System;
using System.Collections.Generic;

using NUnit.Framework;

using BLToolkit.DataAccess;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class ScalarFieldName
    {
        public abstract class TestAccessor : DataAccessor
        {
            [SqlQuery("SELECT PersonID, FirstName FROM Person")
            [Index("PersonID")]
            [ScalarFieldName("FirstName")]]
            public abstract Dictionary<int, string> GetPersons();
        }

        [Test]
        public void Test()
        {
            TestAccessor pa = DataAccessor.CreateInstance<TestAccessor>();

            IDictionary<int, string> dic = pa.GetPersons();

            Assert.AreEqual("John", dic[1]);
        }
    }
}
```

```
}  
}
```

Person.cs

```
using System;  
  
using BLToolkit.DataAccess;  
using BLToolkit.Mapping;  
  
namespace HowTo.DataAccess  
{  
    public class Person  
    {  
        [MapField("PersonID"), PrimaryKey, NonUpdateable]  
        public int ID;  
  
        public string LastName;  
        public string FirstName;  
        public string MiddleName;  
        public Gender Gender;  
    }  
}
```

Gender.cs

```
using System;  
  
using BLToolkit.Mapping;  
  
namespace HowTo.DataAccess  
{  
    public enum Gender  
    {  
        [MapValue("F")] Female,  
        [MapValue("M")] Male,  
        [MapValue("U")] Unknown,  
        [MapValue("O")] Other  
    }  
}
```

```
}  
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>  
<configuration>  
  <appSettings>  
    <add  
      key    = "ConnectionString"  
      value = "Server=.;Database=BLToolkitData;  
    </appSettings>  
</configuration>
```

[Create.sql script](#)

Business Logic Toolkit for .NET www.bltoolkit.net

If a method returns a scalar value, this attribute can be used to specify how database attribute take a parameter of the **ScalarSourceType** type:

ScalarSourceType	Description
DataReader	Calls the DbManager.ExecuteReader method, and then calls IDataReader.GetV
OutputParameter	Calls the DbManager.ExecuteNonQuery method, and then reads value from the I
ReturnValue	Calls the DbManager.ExecuteNonQuery method, and then reads return value fro
AffectedRows	Calls the DbManager.ExecuteNonQuery method, and then returns its return valu

ScalarSource.cs

```
using System;

using NUnit.Framework;

using BLToolkit.Data;
using BLToolkit.DataAccess;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class ScalarSource
    {
        public abstract class TestAccessor : DataAcces
        {
            [ScalarSource(ScalarSourceType.DataReader
            public abstract int Scalar_DataReader();

            [ActionName("Scalar_DataReader")]
            [ScalarSource(ScalarSourceType.DataReader
            public abstract string Scalar_DataReader?

            [ActionName("Scalar_DataReader")]
            [ScalarSource(ScalarSourceType.DataReader
```

```

public abstract string Scalar_DataReader()

[ScalarSource(ScalarSourceType.OutputParameter)]
public abstract int Scalar_OutputParameter()

[ActionName("Scalar_OutputParameter")]
[ScalarSource(ScalarSourceType.OutputParameter)]
public abstract string Scalar_OutputParameter()

[ActionName("Scalar_OutputParameter")]
[ScalarSource(ScalarSourceType.OutputParameter)]
public abstract string Scalar_OutputParameter()

[ScalarSource(ScalarSourceType.ReturnValue)]
public abstract int Scalar_ReturnParameter()

[ActionName("Scalar_DataReader")]
[ScalarSource(ScalarSourceType.AffectedRows)]
public abstract int Scalar_AffectedRows()

public static TestAccessor CreateInstance()
{
    return (TestAccessor)CreateInstance(typeof(TestAccessor));
}

[Test]
public void DataReaderTest()
{
    TestAccessor ta = TestAccessor.CreateInstance();

    int expectedValue = 12345;
    int actualValue = ta.Scalar_DataReader();

    Assert.AreEqual(expectedValue, actualValue);
}

```

```
[Test]
public void DataReader2Test()
{
    TestAccessor ta = TestAccessor.CreateInstance();

    string expectedValue = "54321";
    string actualValue    = ta.Scalar_DataReader;

    Assert.AreEqual(expectedValue, actualValue);
}
```

```
[Test]
public void DataReader3Test()
{
    TestAccessor ta = TestAccessor.CreateInstance();

    string expectedValue = "54321";
    string actualValue    = ta.Scalar_DataReader;

    Assert.AreEqual(expectedValue, actualValue);
}
```

```
[Test]
public void OutputParameterTest()
{
    TestAccessor ta = TestAccessor.CreateInstance();

    int expectedValue = 12345;
    int actualValue    = ta.Scalar_OutputParameter;

    Assert.AreEqual(expectedValue, actualValue);
}
```

```
[Test]
public void OutputParameter2Test()
{
    TestAccessor ta = TestAccessor.CreateInstance();
```

```

        string expectedValue = "54321";
        string actualValue    = ta.Scalar_OutputPa

        Assert.AreEqual(expectedValue, actualValu
    }

[Test]
public void OutputParameter3Test()
{
    TestAccessor ta = TestAccessor.CreateInst

    string expectedValue = "54321";
    string actualValue    = ta.Scalar_OutputPa

    Assert.AreEqual(expectedValue, actualValu
}

[Test]
public void ReturnParameterTest()
{
    TestAccessor ta = TestAccessor.CreateInst

    int expectedValue = 12345;
    int actualValue    = ta.Scalar_ReturnParam

    Assert.AreEqual(expectedValue, actualValu
}

[Test]
public void AffectedRowsTest()
{
    TestAccessor ta = TestAccessor.CreateInst

    int expectedValue = -1;
    int actualValue    = ta.Scalar_AffectedRow

```

```
        Assert.AreEqual(expectedValue, actualValue);
    }
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value  = "Server=.;Database=BLToolkitData;
    </appSettings>
  </configuration>
```

[Create.sql script](#)

Business Logic Toolkit for .NET www.bltoolkit.net

The **SprocName** attribute allows assigning a stored procedure name explicitly.

SprocName.cs

```
using System;
using System.Collections.Generic;

using NUnit.Framework;

using BLToolkit.DataAccess;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class SprocName
    {
        public abstract class TestAccessor : DataAccessor
        {
            [SprocName("Person_SelectAll")]
            public abstract List<Person> GetPersonList();
        }

        [Test]
        public void Test()
        {
            TestAccessor ta = DataAccessor.CreateInstance<TestAccessor>();

            List<Person> list = ta.GetPersonList();

            Assert.AreNotEqual(0, list.Count);
        }
    }
}
```

Person.cs

```
using System;

using BLToolkit.DataAccess;
using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public class Person
    {
        [MapField("PersonID"), PrimaryKey, NonUpdateable]
        public int ID;

        public string LastName;
        public string FirstName;
        public string MiddleName;
        public Gender Gender;
    }
}
```

Gender.cs

```
using System;

using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public enum Gender
    {
        [MapValue("F")] Female,
        [MapValue("M")] Male,
        [MapValue("U")] Unknown,
        [MapValue("O")] Other
    }
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value = "Server=.;Database=BLToolkitData;
    </appSettings>
  </configuration>
```

[Create.sql script](#)

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

The **SqlQuery** attribute allows specifying SQL statement.

SqlQuery.cs

```
using System;
using System.Collections.Generic;

using NUnit.Framework;

using BLToolkit.DataAccess;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class SqlQuery
    {
        public abstract class TestAccessor : DataAccessor
        {
            [SqlQuery(@"
                SELECT
                    *
                FROM
                    Person
                WHERE
                    FirstName like @firstName AND
                    LastName like @lastName")]
            public abstract List<Person> GetPersonList();
        }

        [Test]
        public void Test()
        {
            TestAccessor ta = DataAccessor.CreateInstance<TestAccessor>();
        }
    }
}
```

```
        List<Person> list = ta.GetPersonListByNar  
  
        Assert.AreNotEqual(0, list.Count);  
    }  
}  
}
```

Person.cs

```
using System;  
  
using BLToolkit.DataAccess;  
using BLToolkit.Mapping;  
  
namespace HowTo.DataAccess  
{  
    public class Person  
    {  
        [MapField("PersonID"), PrimaryKey, NonUpdateable]  
        public int ID;  
  
        public string LastName;  
        public string FirstName;  
        public string MiddleName;  
        public Gender Gender;  
    }  
}
```

Gender.cs

```
using System;  
  
using BLToolkit.Mapping;  
  
namespace HowTo.DataAccess  
{  
    public enum Gender  
    {
```

```
        [MapValue("F")] Female,  
        [MapValue("M")] Male,  
        [MapValue("U")] Unknown,  
        [MapValue("O")] Other  
    }  
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>  
<configuration>  
  <appSettings>  
    <add  
      key    = "ConnectionString"  
      value = "Server=.;Database=BLToolkitData;  
    </appSettings>  
</configuration>
```

[Create.sql script](#)

Business Logic Toolkit for .NET www.bltoolkit.net

This example demonstrates how to create a custom attribute which allows specifying mo

CustomSqlQuery1.cs

```
using System;
using System.Collections.Generic;

using NUnit.Framework;

using BLToolkit.Data;
using BLToolkit.DataAccess;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class CustomSqlQuery1
    {
        public class TestQueryAttribute : SqlQueryAttribute
        {
            public TestQueryAttribute()
            {
                IsDynamic = true;
            }

            public string OracleText { get; set; }
            public string FbText     { get; set; }
            public string SQLiteText { get; set; }

            public override string GetSqlText(DataAccess
            {
                switch (dbManager.DataProvider.Name)
                {
                    case "Sql"      :
                    case "Access": return SqlText;
                }
            }
        }
    }
}
```

```

        case "Oracle": return OracleText
        case "Fdp"    : return FbText
        case "SQLite": return SQLiteText
    }

    throw new ApplicationException(string)
}
}

public abstract class PersonAccessor : DataAc
{
    [TestQuery(
        SqlText      = "SELECT * FROM Person Wh
        OracleText = "SELECT * FROM Person Wh
    public abstract List<Person> SelectByLast

    [TestQuery(
        SqlText      = "SELECT * FROM Person Wh
        OracleText = "SELECT * FROM Person Wh
    public abstract List<Person> SelectBy([F

    [TestQuery(
        SqlText      = "SELECT TOP {0} * FROM P
        OracleText = "SELECT * FROM Person Wh
        FbText      = "SELECT FIRST {0} * FROM
        SQLiteText = "SELECT * FROM Person Wh
    public abstract List<Person> SelectByLast

    [TestQuery(
        SqlText      = "SELECT @id as PersonID'
        OracleText = "SELECT :id PersonID FR
        FbText      = "SELECT CAST(@id AS INTE
    public abstract List<Person> SelectID(int

}

[Test]
public void Test1()

```

```
{
    PersonAccessor da = DataAccessor.CreateIr
    List<Person> list = da.SelectByLastName('
    Assert.AreNotEqual(0, list.Count);
}
```

```
[Test]
public void Test2()
{
    PersonAccessor da = DataAccessor.CreateIr
    List<Person> list = da.SelectBy("FirstNar
    Assert.AreNotEqual(0, list.Count);
}
```

```
[Test]
public void Test3()
{
    PersonAccessor da = DataAccessor.CreateIr
    List<Person> list = da.SelectByLastName('
    Assert.AreNotEqual(0, list.Count);
}
```

```
[Test]
public void Test4()
{
    PersonAccessor da = DataAccessor.CreateIr
    List<Person> list = da.SelectID(42);
    Assert.AreEqual(42, list[0].ID);
}
```

```
}  
}
```

Person.cs

```
using System;  
  
using BLToolkit.DataAccess;  
using BLToolkit.Mapping;  
  
namespace HowTo.DataAccess  
{  
    public class Person  
    {  
        [MapField("PersonID"), PrimaryKey, NonUpdateable]  
        public int ID;  
  
        public string LastName;  
        public string FirstName;  
        public string MiddleName;  
        public Gender Gender;  
    }  
}
```

Gender.cs

```
using System;  
  
using BLToolkit.Mapping;  
  
namespace HowTo.DataAccess  
{  
    public enum Gender  
    {  
        [MapValue("F")] Female,  
        [MapValue("M")] Male,  
        [MapValue("U")] Unknown,  
        [MapValue("O")] Other  
    }  
}
```

```
}  
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>  
<configuration>  
  <appSettings>  
    <add  
      key    = "ConnectionString"  
      value = "Server=.;Database=BLToolkitData;  
    </appSettings>  
</configuration>
```

[Create.sql script](#)

Business Logic Toolkit for .NET www.bltoolkit.net

This example demonstrates how to specify SQL query for different data providers by using

CustomSqlQuery2.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Reflection;
using System.Xml;

using NUnit.Framework;

using BLToolkit.Data;
using BLToolkit.DataAccess;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class CustomSqlQuery2
    {
        public abstract class TestAccessorBase<T> : I
            where T : TestAccessorBase<T>
        {
            const int Sql      = 0;
            const int Access  = 1;
            const int Oracle  = 2;
            const int Fdp     = 3;
            const int SQLite  = 4;

            Dictionary<int, string> _sql = new Dictio

            private string GetSql(string providerName
            {
                Stream stream = Assembly.GetCallingAs
```

```

        "HowTo.DataAccess.Sql." + providerName);
        XmlDocument doc = new XmlDocument();
        doc.Load(stream);
        XmlNode node = doc.SelectSingleNode(
            string.Format("//*[local-name='{0}']", providerName));
        return node != null? node.InnerText:
    }

    protected override string PrepareSqlQuery
    {
        int provider = Sql;
        string providerName = db.DataProviderName;

        switch (providerName)
        {
            case "Sql" : provider = Sql;
            case "Access": provider = Access;
            case "Oracle": provider = Oracle;
            case "Fdp" : provider = Fdp;
            case "SQLite": provider = SQLite;
            default:
                throw new ApplicationException(
                    string.Format("Unknown data provider: {0}",
                        providerName));
        }

        string text;
        int key = provider * 10000000 + providerName.GetHashCode();

        if (!_sql.TryGetValue(key, out text))
            return text;

        _sql[key] = text = GetSql(providerName);
    }
}

```

```

        return text;
    }

    public static T CreateInstance()
    {
        return DataAccessor.CreateInstance<T>();
    }
}

public abstract class PersonAccessor : TestAccessor
{
    [SqlQuery(ID = 1)]
    public abstract List<Person> SelectByLastName(string lastName);

    [SqlQuery(ID = 2)]
    public abstract List<Person> SelectBy([From] string firstName);

    [SqlQuery(ID = 3)]
    public abstract List<Person> SelectByLastInitial(string lastInitial);

    [SqlQuery(ID = 4)]
    public abstract List<Person> SelectID(int id);
}

[Test]
public void Test1()
{
    PersonAccessor da = PersonAccessor.CreateInstance();

    List<Person> list = da.SelectByLastName("Smith");

    Assert.AreNotEqual(0, list.Count);
}

[Test]
public void Test2()
{

```

```

        PersonAccessor da = PersonAccessor.Create
        List<Person> list = da.SelectBy("FirstName")
        Assert.AreNotEqual(0, list.Count);
    }

    [Test]
    public void Test3()
    {
        PersonAccessor da = PersonAccessor.Create
        List<Person> list = da.SelectByLastName("
        Assert.AreNotEqual(0, list.Count);
    }

    [Test]
    public void Test4()
    {
        PersonAccessor da = PersonAccessor.Create
        List<Person> list = da.SelectID(42);
        Assert.AreEqual(42, list[0].ID);
    }
}
}
}

```

Sql.xml

```

<?xml version="1.0" encoding="utf-8" ?>
<sql>
    <query id="1">SELECT * FROM Person WHERE LastName
    <query id="2">SELECT * FROM Person WHERE {0} = @\
    <query id="3">SELECT TOP {0} * FROM Person WHERE
    <query id="4">SELECT @id as PersonID</query>

```

```
</sql>
```

Access.xml

```
<?xml version="1.0" encoding="utf-8" ?>  
<sql>  
</sql>
```

Oracle.xml

```
<?xml version="1.0" encoding="utf-8" ?>  
<sql>  
  <query id="1">SELECT * FROM Person WHERE LastName  
  <query id="2">SELECT * FROM Person WHERE {0} = :\  
  <query id="3">SELECT * FROM Person WHERE LastName  
  <query id="4">SELECT :id PersonID FROM Dual</quer  
</sql>
```

Fdp.xml

```
<?xml version="1.0" encoding="utf-8" ?>  
<sql>  
  <query id="3">SELECT FIRST {0} * FROM Person WHEF  
  <query id="4">SELECT CAST(@id AS INTEGER) PersonI  
</sql>
```

SQLite.xml

```
<?xml version="1.0" encoding="utf-8" ?>  
<sql>  
  <query id="3">SELECT * FROM Person WHERE LastName  
</sql>
```

Person.cs

```
using System;  
  
using BLToolkit.DataAccess;  
using BLToolkit.Mapping;
```

```

namespace HowTo.DataAccess
{
    public class Person
    {
        [MapField("PersonID"), PrimaryKey, NonUpdateable]
        public int ID;

        public string LastName;
        public string FirstName;
        public string MiddleName;
        public Gender Gender;
    }
}

```

Gender.cs

```

using System;

using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public enum Gender
    {
        [MapValue("F")] Female,
        [MapValue("M")] Male,
        [MapValue("U")] Unknown,
        [MapValue("O")] Other
    }
}

```

App.config

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add

```

```
        key    = "ConnectionString"
        value  = "Server=.;Database=BLToolkitData;
    </appSettings>
</configuration>
```

[Create.sql script](#)

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

The **SqlQuery** and **SprocQuery** classes can be used to perform simple CRUDL operations: **SelectByKey**, **SelectAll**, **Insert**, **Update**, and **Delete**.

OpenConfigQuery.cs

```
using System;

using NUnit.Framework;

using BLToolkit.Data;
using BLToolkit.DataAccess;
using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class OpenConfigQuery
    {
        public class Person
        {
            [MapField("PersonID"), PrimaryKey, NonUpdatable]
            public int ID;

            public string LastName;
            public string FirstName;
            public string MiddleName;
        }

        // DbManager is created by SqlQuery.
        //
        [Test]
        public void Test1()
        {
            SqlQuery<Person> query = new SqlQuery<Person>
```

```

        Person person = query.SelectByKey(1);

        Assert.IsNotNull(person);
    }

    // SqlQuery takes DbManager as a parameter.
    //
    [Test]
    public void Test2()
    {
        using (DbManager db = new DbManager())
        {
            SqlQuery<Person> query = new SqlQuery

            Person person = query.SelectByKey(1);

            Assert.IsNotNull(person);
        }
    }

    // SqlQuery method takes DbManager as a parar
    //
    [Test]
    public void Test3()
    {
        using (DbManager db = new DbManager())
        {
            SqlQuery<Person> query = new SqlQuery

            Person person = query.SelectByKey(db,

            Assert.IsNotNull(person);
        }
    }
}

```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value = "Server=.;Database=BLToolkitData;"
    </add>
  </appSettings>
</configuration>
```

[Create.sql script](#)

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

SelectByKeySql.cs

```
using System;

using NUnit.Framework;

using BLToolkit.Data;
using BLToolkit.DataAccess;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class SelectByKeySql
    {
        [Test]
        public void Test1()
        {
            SqlQuery<Person> query = new SqlQuery<Pei

            Person person = query.SelectByKey(1);

            Assert.IsNotNull(person);
        }

        [Test]
        public void Test2()
        {
            using (DbManager db = new DbManager())
            {
                SqlQuery<Person> query = new SqlQuery

                Person person = query.SelectByKey(db,

                Assert.IsNotNull(person);
            }
        }
    }
}
```

```
}  
    }  
}

}
```

DataAccessor.SelectByKetSql method generates and executes the following SQL statement

```
SELECT  
    [MiddleName],  
    [PersonID],  
    [LastName],  
    [FirstName]  
FROM  
    [Person]  
WHERE  
    [PersonID] = @PersonID
```

Person.cs

```
using System;  
  
using BLToolkit.DataAccess;  
using BLToolkit.Mapping;  
  
namespace HowTo.DataAccess  
{  
    public class Person  
    {  
        [MapField("PersonID"), PrimaryKey, NonUpdateable]  
        public int ID;  
  
        public string LastName;  
        public string FirstName;  
        public string MiddleName;  
        public Gender Gender;  
    }  
}
```

Gender.cs

```
using System;

using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public enum Gender
    {
        [MapValue("F")] Female,
        [MapValue("M")] Male,
        [MapValue("U")] Unknown,
        [MapValue("O")] Other
    }
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value  = "Server=.;Database=BLToolkitData;
    </appSettings>
  </configuration>
```

Create.sql script

Business Logic Toolkit for .NET www.bltoolkit.net

SelectAllSql.cs

```
using System;
using System.Collections.Generic;

using NUnit.Framework;

using BLToolkit.Data;
using BLToolkit.DataAccess;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class SelectAllSql
    {
        [Test]
        public void Test1()
        {
            SqlQuery<Person> query = new SqlQuery<Person>

            List<Person> list = query.SelectAll();
        }

        [Test]
        public void Test2()
        {
            using (DbManager db = new DbManager())
            {
                SqlQuery<Person> query = new SqlQuery

                List<Person> list = query.SelectAll()
            }
        }
    }
}
```

```
}
```

DataAccessor.SelectAllSql method generates and executes the following SQL statement:

```
SELECT
    [MiddleName],
    [PersonID],
    [LastName],
    [FirstName]
FROM
    [Person]
```

Person.cs

```
using System;

using BLToolkit.DataAccess;
using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public class Person
    {
        [MapField("PersonID"), PrimaryKey, NonUpdateable]
        public int ID;

        public string LastName;
        public string FirstName;
        public string MiddleName;
        public Gender Gender;
    }
}
```

Gender.cs

```
using System;

using BLToolkit.Mapping;
```

```
namespace HowTo.DataAccess
{
    public enum Gender
    {
        [MapValue("F")] Female,
        [MapValue("M")] Male,
        [MapValue("U")] Unknown,
        [MapValue("O")] Other
    }
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value  = "Server=.;Database=BLToolkitData;
    </appSettings>
  </configuration>
```

Create.sql script

Business Logic Toolkit for .NET www.bltoolkit.net

InsertSql.cs

```
using System;

using NUnit.Framework;

using BLToolkit.Data;
using BLToolkit.DataAccess;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class InsertSql
    {
        [Test]
        public void Test1()
        {
            SqlQuery<Person> query = new SqlQuery<Pei

            Person person = new Person();

            person.FirstName = "Crazy";
            person.LastName = "Frog";
            person.Gender = Gender.Unknown;

            query.Insert(person);
        }

        [Test]
        public void Test2()
        {
            using (DbManager db = new DbManager())
            {
                SqlQuery<Person> query = new SqlQuery

```

```

        Person person = new Person();

        person.FirstName = "Crazy";
        person.LastName  = "Frog";
        person.Gender    = Gender.Other;

        query.Insert(db, person);
    }
}
}
}

```

DataAccessor.InsertSql method generates and executes the following SQL statement:

```

INSERT INTO [Person] (
    [MiddleName],
    [Gender],
    [LastName],
    [FirstName]
) VALUES (
    @MiddleName,
    @Gender,
    @LastName,
    @FirstName
)

```

Person.cs

```

using System;

using BLToolkit.DataAccess;
using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public class Person
    {

```

```

        [MapField("PersonID"), PrimaryKey, NonUpdateable]
        public int ID;

        public string LastName;
        public string FirstName;
        public string MiddleName;
        public Gender Gender;
    }
}

```

Gender.cs

```

using System;

using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public enum Gender
    {
        [MapValue("F")] Female,
        [MapValue("M")] Male,
        [MapValue("U")] Unknown,
        [MapValue("O")] Other
    }
}

```

App.config

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value  = "Server=.;Database=BLToolkitData;
    </appSettings>
  </configuration>

```

Create.sql script

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

UpdateSql.cs

```
using System;

using NUnit.Framework;

using BLToolkit.Data;
using BLToolkit.DataAccess;
using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class UpdateSql
    {
        int _id;

        [TestFixtureSetUp]
        public void Insert()
        {
            using (DbManager db = new DbManager())
            {
                _id = db
                    .SetCommand(@"
                        INSERT INTO Person (
                            FirstName, LastName, (
                        ) VALUES (
                            @FirstName, @LastName, @(
                        )
                        SELECT Cast(SCOPE_IDENTITY()
                            db.Parameter("@FirstName", "(
                            db.Parameter("@LastName", "F
                            db.Parameter("@Gender", "Ma
                    ).ExecuteScalar<int>());
```

```

    }
}

[Test]
public void Test1()
{
    SqlQuery<Person> query = new SqlQuery<Pei

    Person person = query.SelectByKey(_id);

    person.Gender = Gender.Other;

    query.Update(person);
}

[Test]
public void Test2()
{
    using (DbManager db = new DbManager())
    {
        SqlQuery<Person> query = new SqlQuery

        Person person = query.SelectByKey(db,

        person.Gender = Gender.Other;

        query.Update(db, person);
    }
}

[TestFixtureTearDown]
public void Delete()
{
    using (DbManager db = new DbManager())
    {
        db

        .SetCommand("DELETE FROM Person v

```

```

        db.Parameter("@id", _id))
        .ExecuteNonQuery();
    }
}
}
}
}

```

DataAccessor.UpdateSql method generates and executes the following SQL statement:

```

UPDATE
    [Person]
SET
    [MiddleName] = @MiddleName,
    [Gender]     = @Gender,
    [LastName]   = @LastName,
    [FirstName]  = @FirstName
WHERE
    [PersonID]  = @PersonID

```

Person.cs

```

using System;

using BLToolkit.DataAccess;
using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public class Person
    {
        [MapField("PersonID"), PrimaryKey, NonUpdateable]
        public int ID;

        public string LastName;
        public string FirstName;
        public string MiddleName;
        public Gender Gender;
    }
}

```

```
}
```

Gender.cs

```
using System;

using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public enum Gender
    {
        [MapValue("F")] Female,
        [MapValue("M")] Male,
        [MapValue("U")] Unknown,
        [MapValue("O")] Other
    }
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value  = "Server=.;Database=BLToolkitData;
    </appSettings>
  </configuration>
```

[Create.sql script](#)

Business Logic Toolkit for .NET www.bltoolkit.net

DeleteSql.cs

```
using System;

using NUnit.Framework;

using BLToolkit.Data;
using BLToolkit.DataAccess;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class DeleteSql
    {
        private int Insert()
        {
            using (DbManager db = new DbManager())
            {
                return db
                    .SetCommand(@"
                        INSERT INTO Person (
                            FirstName, LastName, (
                        ) VALUES (
                            @FirstName, @LastName, @(
                        )
                        SELECT Cast(SCOPE_IDENTITY()
                            db.Parameter("@FirstName", "(
                            db.Parameter("@LastName", "F
                            db.Parameter("@Gender", "M
                    ).ExecuteScalar<int>());
            }
        }

        [Test]
```

```

public void Test1()
{
    int id = Insert();

    SqlQuery<Person> query = new SqlQuery<Pe

    query.DeleteByKey(id);
}

[Test]
public void Test2()
{
    int id = Insert();

    using (DbManager db = new DbManager())
    {
        SqlQuery<Person> query = new SqlQuery

        Person person = query.SelectByKey(db,

        query.Delete(db, person);
    }
}
}
}
}

```

Both *DataAccessor.DeleteByKeySql* and *DataAccessor.DeleteSql* methods generate and execute the following SQL statement:

```

DELETE FROM
    [Person]
WHERE
    [PersonID] = @PersonID

```

Person.cs

```

using System;

```

```
using BLToolkit.DataAccess;
using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public class Person
    {
        [MapField("PersonID"), PrimaryKey, NonUpdateable]
        public int ID;

        public string LastName;
        public string FirstName;
        public string MiddleName;
        public Gender Gender;
    }
}
```

Gender.cs

```
using System;

using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public enum Gender
    {
        [MapValue("F")] Female,
        [MapValue("M")] Male,
        [MapValue("U")] Unknown,
        [MapValue("O")] Other
    }
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
```

```
<appSettings>
  <add
    key    = "ConnectionString"
    value  = "Server=.;Database=BLToolkitData;
  </appSettings>
</configuration>
```

[Create.sql script](#)

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

SelectByKey.cs

```
using System;

using NUnit.Framework;

using BLToolkit.Data;
using BLToolkit.DataAccess;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class SelectByKey
    {
        [Test]
        public void Test1()
        {
            SprocQuery<Person> query = new SprocQuery

            Person person = query.SelectByKey(1);

            Assert.IsNotNull(person);
        }

        [Test]
        public void Test2()
        {
            using (DbManager db = new DbManager())
            {
                SprocQuery<Person> query = new Sproc

                Person person = query.SelectByKey(db,

                Assert.IsNotNull(person);
            }
        }
    }
}
```

```
}  
    }  
  }  
}
```

DataAccessor.SelectByKey method generates and executes the following SQL statement:

```
exec Person_SelectByKey @id=1
```

Person.cs

```
using System;  
  
using BLToolkit.DataAccess;  
using BLToolkit.Mapping;  
  
namespace HowTo.DataAccess  
{  
    public class Person  
    {  
        [MapField("PersonID"), PrimaryKey, NonUpdateable]  
        public int ID;  
  
        public string LastName;  
        public string FirstName;  
        public string MiddleName;  
        public Gender Gender;  
    }  
}
```

Gender.cs

```
using System;  
  
using BLToolkit.Mapping;  
  
namespace HowTo.DataAccess  
{
```

```
public enum Gender
{
    [MapValue("F")] Female,
    [MapValue("M")] Male,
    [MapValue("U")] Unknown,
    [MapValue("O")] Other
}
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value = "Server=.;Database=BLToolkitData,
    </appSettings>
  </configuration>
```

[Create.sql script](#)

Business Logic Toolkit for .NET www.bltoolkit.net

SelectAll.cs

```
using System;
using System.Collections.Generic;

using NUnit.Framework;

using BLToolkit.Data;
using BLToolkit.DataAccess;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class SelectAll
    {
        [Test]
        public void Test1()
        {
            SprocQuery<Person> query = new SprocQuery

            List<Person> list = query.SelectAll();
        }

        [Test]
        public void Test2()
        {
            using (DbManager db = new DbManager())
            {
                SprocQuery<Person> query = new Sproc

                List<Person> list = query.SelectAll(c
            }
        }
    }
}
```

```
}
```

DataAccessor.SelectAll method generates and executes the following SQL statement:

```
exec Person_SelectAll
```

Person.cs

```
using System;

using BLToolkit.DataAccess;
using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public class Person
    {
        [MapField("PersonID"), PrimaryKey, NonUpdateable]
        public int ID;

        public string LastName;
        public string FirstName;
        public string MiddleName;
        public Gender Gender;
    }
}
```

Gender.cs

```
using System;

using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public enum Gender
    {
        [MapValue("F")] Female,
    }
}
```

```
        [MapValue("M")] Male,  
        [MapValue("U")] Unknown,  
        [MapValue("O")] Other  
    }  
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>  
<configuration>  
  <appSettings>  
    <add  
      key    = "ConnectionString"  
      value = "Server=.;Database=BLToolkitData;  
    </appSettings>  
</configuration>
```

[Create.sql script](#)

Business Logic Toolkit for .NET www.bltoolkit.net

Insert.cs

```
using System;

using NUnit.Framework;

using BLToolkit.Data;
using BLToolkit.DataAccess;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class Insert
    {
        [Test]
        public void Test1()
        {
            SprocQuery<Person> query = new SprocQuery

            Person person = new Person();

            person.FirstName = "Crazy";
            person.LastName = "Frog";
            person.Gender = Gender.Unknown;

            query.Insert(person);
        }

        [Test]
        public void Test2()
        {
            using (DbManager db = new DbManager())
            {
                SprocQuery<Person> query = new Sproc
            }
        }
    }
}
```

```

        Person person = new Person();

        person.FirstName = "Crazy";
        person.LastName  = "Frog";
        person.Gender    = Gender.Other;

        query.Insert(db, person);
    }
}
}
}

```

DataAccessor.Insert method generates and executes the following SQL statement:

```
exec Person_Insert @FirstName='Crazy', @LastName='Frog'
```

Person.cs

```

using System;

using BLToolkit.DataAccess;
using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public class Person
    {
        [MapField("PersonID"), PrimaryKey, NonUpdateable]
        public int ID;

        public string LastName;
        public string FirstName;
        public string MiddleName;
        public Gender Gender;
    }
}

```

Gender.cs

```
using System;

using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public enum Gender
    {
        [MapValue("F")] Female,
        [MapValue("M")] Male,
        [MapValue("U")] Unknown,
        [MapValue("O")] Other
    }
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value = "Server=.;Database=BLToolkitData;
    </appSettings>
  </configuration>
```

Create.sql script

Business Logic Toolkit for .NET www.bltoolkit.net

Update.cs

```
using System;

using NUnit.Framework;

using BLToolkit.Data;
using BLToolkit.DataAccess;
using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class Update
    {
        int _id;

        [TestFixtureSetUp]
        public void Insert()
        {
            using (DbManager db = new DbManager())
            {
                _id = db
                    .SetCommand(@"
                        INSERT INTO Person (
                            FirstName, LastName, (
                        ) VALUES (
                            @FirstName, @LastName, @(
                        )
                        SELECT Cast(SCOPE_IDENTITY()
                            db.Parameter("@FirstName", "(
                            db.Parameter("@LastName", "F
                            db.Parameter("@Gender", "M
                    ).ExecuteScalar<int>());
            }
        }
    }
}
```

```

    }
}

[Test]
public void Test1()
{
    SprocQuery<Person> query = new SprocQuery

    Person person = query.SelectByKey(_id);

    person.Gender = Gender.Other;

    query.Update(person);
}

[Test]
public void Test2()
{
    using (DbManager db = new DbManager())
    {
        SprocQuery<Person> query = new SprocQuery

        Person person = query.SelectByKey(db,

        person.Gender = Gender.Other;

        query.Update(db, person);
    }
}

[TestFixtureTearDown]
public void Delete()
{
    using (DbManager db = new DbManager())
    {
        db

        .SetCommand("DELETE FROM Person v

```

```

        db.Parameter("@id", _id))
        .ExecuteNonQuery();
    }
}
}
}
}

```

DataAccessor.Update method generates and executes the following SQL statement:

```
exec Person_Update @PersonID=2, @FirstName='Crazy', @
```

Person.cs

```

using System;

using BLToolkit.DataAccess;
using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public class Person
    {
        [MapField("PersonID"), PrimaryKey, NonUpdateable]
        public int ID;

        public string LastName;
        public string FirstName;
        public string MiddleName;
        public Gender Gender;
    }
}

```

Gender.cs

```

using System;

using BLToolkit.Mapping;

```

```
namespace HowTo.DataAccess
{
    public enum Gender
    {
        [MapValue("F")] Female,
        [MapValue("M")] Male,
        [MapValue("U")] Unknown,
        [MapValue("O")] Other
    }
}
```

App.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"
      value = "Server=.;Database=BLToolkitData;
    </appSettings>
  </configuration>
```

Create.sql script

Business Logic Toolkit for .NET www.bltoolkit.net

Delete.cs

```
using System;

using NUnit.Framework;

using BLToolkit.Data;
using BLToolkit.DataAccess;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class Delete
    {
        private int Insert()
        {
            using (DbManager db = new DbManager())
            {
                return db
                    .SetCommand(@"
                        INSERT INTO Person (
                            FirstName, LastName, (
                        ) VALUES (
                            @FirstName, @LastName, @(
                        )
                        SELECT Cast(SCOPE_IDENTITY()
                        db.Parameter("@FirstName", "(
                        db.Parameter("@LastName", "F
                        db.Parameter("@Gender", "M
                    ).ExecuteScalar<int>();
            }
        }

        [Test]
```

```

public void Test1()
{
    int id = Insert();

    SprocQuery<Person> query = new SprocQuery

    query.DeleteByKey(id);
}

[Test]
public void Test3()
{
    int id = Insert();

    using (DbManager db = new DbManager())
    {
        SqlQuery<Person> query = new SqlQuery

        Person person = query.SelectByKey(db,

        query.Delete(db, person);
    }
}
}
}

```

Both *DataAccessor.DeleteByKey* and *DataAccessor.Delete* methods generate and execute SQL statement:

```
exec Person_Delete @PersonID=2
```

Person.cs

```

using System;

using BLToolkit.DataAccess;
using BLToolkit.Mapping;

```

```

namespace HowTo.DataAccess
{
    public class Person
    {
        [MapField("PersonID"), PrimaryKey, NonUpdateable]
        public int ID;

        public string LastName;
        public string FirstName;
        public string MiddleName;
        public Gender Gender;
    }
}

```

Gender.cs

```

using System;

using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    public enum Gender
    {
        [MapValue("F")] Female,
        [MapValue("M")] Male,
        [MapValue("U")] Unknown,
        [MapValue("O")] Other
    }
}

```

App.config

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key = "ConnectionString"

```

```
        value = "Server=.;Database=BLToolkitData;  
    </appSettings>  
</configuration>
```

[Create.sql script](#)

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

The **NonUpdatable** attribute indicates the field that should not be updated by *UPDATE* SQL statements.

NonUpdatable.cs

```
using System;

using NUnit.Framework;

using BLToolkit.DataAccess;
using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class NonUpdatable
    {
        public enum Gender
        {
            [MapValue("F")] Female,
            [MapValue("M")] Male,
            [MapValue("U")] Unknown,
            [MapValue("O")] Other
        }

        public class Person
        {
            [MapField("PersonID"), PrimaryKey, NonUpdatable]
            public int ID;

            public string LastName;
            public string FirstName;
            public string MiddleName;
            public Gender Gender;
        }
    }
}
```

```

    }

    [Test]
    public void Test()
    {
        SqlQuery<Person> query = new SqlQuery<Per

        Person person = new Person();

        person.FirstName = "Crazy";
        person.LastName  = "Frog";
        person.Gender    = Gender.Other;

        query.Insert(person);
    }
}

```

DataAccessor.Insert method generates and executes the following SQL statement:

```

INSERT INTO [Person] (
    [MiddleName],
    [Gender],
    [LastName],
    [FirstName]
) VALUES (
    @MiddleName,
    @Gender,
    @LastName,
    @FirstName
)

```

App.config

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add

```

```
        key    = "ConnectionString"
        value  = "Server=.;Database=BLToolkitData,
</appSettings>
</configuration>
```

[Create.sql script](#)

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

The **PrimaryKey** attribute indicates the field that is a part of a primary key.

PrimaryKey.cs

```
using System;

using NUnit.Framework;

using BLToolkit.DataAccess;
using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class PrimaryKey
    {
        public class Person
        {
            [MapField("PersonID"), PrimaryKey, NonUpd
            public int ID;

            public string LastName;
            public string FirstName;
            public string MiddleName;
        }

        [Test]
        public void Test()
        {
            SqlQuery<Person> da = new SqlQuery<Person>

            Person person = da.SelectByKey(1);

            Assert.IsNotNull(person);
        }
    }
}
```

```
}  
    }  
}
```

DataAccessor.SelectByKeySql method generates and executes the following SQL statement

```
SELECT  
    [MiddleName],  
    [PersonID],  
    [LastName],  
    [FirstName]  
FROM  
    [Person]  
WHERE  
    [PersonID] = @PersonID
```

MultiplePrimaryKey.cs

```
using System;  
  
using NUnit.Framework;  
  
using BLToolkit.DataAccess;  
using BLToolkit.Mapping;  
  
namespace HowTo.DataAccess  
{  
    [TestFixture]  
    public class MultiplePrimaryKey  
    {  
        [TableName("Person")]  
        public class Person  
        {  
            [MapField("PersonID"), NonUpdatable]  
            public int ID;  
  
            // These fields are not real primary key  
            // They are made primary key for demonstrati
```

```

        //
        [PrimaryKey(1)] public string FirstName;
        [PrimaryKey(2)] public string LastName;

        public string MiddleName;
    }

    [Test]
    public void Test()
    {
        SqlQuery<Person> query = new SqlQuery<Per

        Person person = query.SelectByKey("Teste

        Assert.IsNotNull(person);
    }
}

```

In this case *DataAccessor.SelectByKeySql* method generates and executes the following :

```

SELECT
    [MiddleName],
    [PersonID],
    [LastName],
    [FirstName]
FROM
    [Person]
WHERE
    [FirstName] = @FirstName AND
    [LastName] = @LastName

```

App.config

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <appSettings>
        <add

```

```
        key    = "ConnectionString"
        value  = "Server=.;Database=BLToolkitData;
    </appSettings>
</configuration>
```

[Create.sql script](#)

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

The **TableName** attribute specifies a name of the table represented by the decorated object.

TableName.cs

```
using System;

using NUnit.Framework;

using BLToolkit.DataAccess;
using BLToolkit.Mapping;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class TableName
    {
        [TableName("Person")]
        public class MyPersonObject
        {
            [MapField("PersonID"), PrimaryKey, NonUpdateable]
            public int ID;

            public string LastName;
            public string FirstName;
            public string MiddleName;
        }

        [Test]
        public void Test1()
        {
            SqlQuery<MyPersonObject> query = new SqlQuery();

            MyPersonObject person = query.SelectByKey
```

```

        Assert.IsNotNull(person);
    }

    [Test]
    public void Test2()
    {
        SprocQuery<MyPersonObject> query = new Sp
        MyPersonObject person = query.SelectByKey
        Assert.IsNotNull(person);
    }
}

```

DataAccessor.SelectByKeySql and *DataAccessor.SelectByKey* methods generate and execute the following SQL statement:

```

-- SelectByKeySql
SELECT
    [MiddleName],
    [PersonID],
    [LastName],
    [FirstName]
FROM
    [Person]
WHERE
    [PersonID] = @PersonID

-- SelectByKey
exec Person_SelectByKey @id=1

```

App.config

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add

```

```
        key    = "ConnectionString"
        value  = "Server=.;Database=BLToolkitData,
</appSettings>
</configuration>
```

[Create.sql script](#)

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

BLToolkit supports different ways to provide metadata information for business objects. Another one is XmlExtension. XmlExtension is a simple xml file where you attributes.

XmlExtension.cs

```
using NUnit.Framework;

using BLToolkit.DataAccess;
using BLToolkit.Mapping;
using BLToolkit.Reflection.Extension;

namespace HowTo.DataAccess
{
    [TestFixture]
    public class XmlExtension
    {
        public class MyPersonObject
        {
            [MapField("PersonID")]
            public int ID;

            public string LastName;
            public string FirstName;
            public string MiddleName;
        }

        [Test]
        public void Test()
        {
            SqlQuery<MyPersonObject> query = new SqlC
query.Extensions = TypeExtension.GetExter

```

```

        MyPersonObject person = query.SelectByKey

        Assert.IsNotNull(person);
    }
}

```

XmlExtension.xml

```

<?xml version="1.0" encoding="utf-8" ?>
<Types
  xmlns      = "urn:schemas-bltoolkit-net:typeext"
  xmlns:da   = "urn:schemas-bltoolkit-net:dataaccess"

  <Type Name="MyPersonObject" da:TableName="Person"
    <Member Name="ID" da:PrimaryKey="1" da:NonUpd
  </Type>
</Types>

```

DataAccessor.SelectByKetSql method generates and executes the following SQL statement

```

SELECT
    [MiddleName],
    [PersonID],
    [LastName],
    [FirstName]
FROM
    [Person]
WHERE
    [PersonID] = @PersonID

```

App.config

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add
      key    = "ConnectionString"

```

```
        value = "Server=.;Database=BLToolkitData;  
    </appSettings>  
</configuration>
```

[Create.sql script](#)

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

The EditableObjects namespace contains classes that can help you to make your object model editable. That means your objects will support the following list of standard and BLToolkit interfaces:

for business objects

namespace	interface
System.ComponentModel	IEditableObject
System.ComponentModel	INotifyPropertyChanged
System.ComponentModel	ICustomTypeDescriptor
System	ICloneable
BLToolkit.ComponentModel	INotifyObjectEdit
BLToolkit.EditableObjects	IEditable
BLToolkit.Validation	IValidatable

for collections

namespace	interface
System.ComponentModel	ITypedList
System.ComponentModel	IBindingList
System.ComponentModel	IBindingListView
System.ComponentModel	ICancelAddNew
BLToolkit.ComponentModel	ISortable
BLToolkit.EditableObjects	IEditable

-
- [EditableObject](#)
 - [Accept and Reject changes](#)
 - [IsDirty flag](#)
-

Business Logic Toolkit for .NET www.bltoolkit.net

EditableObject is an object that provides functionality to commit and rollback changes. The **AcceptChanges** method of the object, which will set the **Current** field values to be the last called. The **IsDirty** property gets a value that indicates whether the object has changed.

If we wrote an editable object manually, we could get the following code just for two edi

```
public class TestObject : INotifyPropertyChanged
{
    // The FirstName editable property.
    //
    private string _originalFirstName;
    private string _currentFirstName;

    public override string FirstName
    {
        get { return _currentFirstName; }
        set
        {
            _currentFirstName = value;
            OnPropertyChanged("FirstName");
        }
    }

    bool IsFirstNameDirty
    {
        get { return _currentFirstName != _originalF:
    }

    void AcceptFirstNameChange()
    {
        if (IsFirstNameDirty)
        {
            _originalFirstName = _currentFirstName;
            OnPropertyChanged("FirstName");
        }
    }
}
```

```

    }
}

void RejectFirstNameChange()
{
    if (IsFirstNameDirty)
    {
        _currentFirstName = _originalFirstName;
        OnPropertyChanged("FirstName");
    }
}

// The LastName editable property.
//
private string _originalLastName;
private string _currentLastName;

public override string LastName
{
    get { return _currentLastName; }
    set
    {
        _currentLastName = value;
        OnPropertyChanged("LastName");
    }
}

bool IsLastNameDirty
{
    get { return _currentLastName != _originalLas
}

void AcceptLastNameChange()
{
    if (IsLastNameDirty)
    {
        _originalLastName = _currentLastName;

```

```

        OnPropertyChanged("LastName");
    }
}

void RejectLastNameChange()
{
    if (IsLastNameDirty)
    {
        _currentLastName = _originalLastName;
        OnPropertyChanged("LastName");
    }
}

// Common members.
//
public bool IsDirty
{
    get
    {
        return IsFirstNameChange || IsLastNameChange;
    }
}

public void AcceptChanges()
{
    AcceptFirstNameChange();
    AcceptLastNameChange();
}

public void RejectChanges()
{
    RejectFirstNameChange();
    RejectLastNameChange();
}

public virtual event PropertyChangedEventHandler

```

```

protected virtual void OnPropertyChanged(string propertyName)
{
    if (PropertyChanged != null)
        PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
}
}

```

BLToolkit allows implementing the same functionality by inheriting your object from the

EditableObjectTest.cs

```

using System;
using NUnit.Framework;
using BLToolkit.EditableObjects;

namespace HowTo.EditableObjects
{
    [TestFixture]
    public class EditableObjectTest
    {
        public abstract class TestObject : EditableObject
        {
            // Any abstract property becomes editable
            //
            public abstract string FirstName { get; set; }
            public abstract string LastName { get; set; }

            // This field is not editable.
            //
            public string FullName
            {
                get { return string.Format("{0} {1}",
                    FirstName, LastName); }
            }
        }

        [Test]
        public void Test()
        {

```

```

        TestObject obj = TestObject.CreateInstance

        obj.FirstName = "Tester";
        obj.LastName = "Testerson";

        Assert.IsTrue(obj.IsDirty);

        obj.AcceptChanges();

        Assert.IsFalse(obj.IsDirty);
    }
}
}

```

BLToolkit type builder will generate the following for the class above:

```

[BLToolkitGenerated]
public sealed class TestObject : EditableObjectTest.1
{
    // Note that the internal representation of the p
    // The EditableValue class provides a mechanism t
    //
    private EditableValue<string> _firstName;
    private EditableValue<string> _lastName;

    // PropertyInfo is used for internal purposes.
    //
    private static PropertyInfo _firstName_propertyIn
        TypeHelper.GetPropertyInfo(typeof(EditableOb
    private static PropertyInfo _lastName_propertyInf
        TypeHelper.GetPropertyInfo(typeof(EditableOb

    // Constructors.
    //
    public TestObject()
    {
        this._firstName = new EditableValue<string>('

```

```

        this._lastName = new EditableValue<string>('
    }

    public TestObject(InitContext ctx)
    {
        this._firstName = new EditableValue<string>('
        this._lastName = new EditableValue<string>('
    }

    // Abstract property implementation.
    //
    public override string FirstName
    {
        get
        {
            return _firstName.Value;
        }

        set
        {
            _firstName.Value = value;

            // The PropertyChanged event support.
            //
            ((IPropertyChanged)this).OnPropertyChange
        }
    }

    public override string LastName
    {
        get
        {
            return _lastName.Value;
        }

        set
        {

```

```

        _lastName.Value = value;
        ((IPropertyChanged)this).OnPropertyChange
    }
}

// The IEditable interface implementation.
//
bool IEditable.IsDirty
{
    get { return _firstName.IsDirty || _lastName.
}

void IEditable.AcceptChanges()
{
    this._firstName.AcceptChanges();
    this._lastName. AcceptChanges();
}

void IEditable.RejectChanges()
{
    this._firstName.RejectChanges();
    this._lastName. RejectChanges();
}

// The IMemberwiseEditable interface implementati
//
bool IMemberwiseEditable.AcceptMemberChanges(Prop
{
    return
        _firstName.AcceptMemberChanges(_firstName
        _lastName. AcceptMemberChanges(_lastName_
}

void IMemberwiseEditable.GetDirtyMembers(Property
{
    _firstName.GetDirtyMembers(_firstName_propert
    _lastName. GetDirtyMembers(_lastName_property

```

```

}

bool IMemberwiseEditable.IsDirtyMember(PropertyInfo prop)
{
    return
        _firstName.IsDirtyMember(_firstName_property) ||
        _lastName.IsDirtyMember(_lastName_property);
}

bool IMemberwiseEditable.RejectMemberChanges(PropertyInfo prop)
{
    return
        _firstName.RejectMemberChanges(_firstName_property) ||
        _lastName.RejectMemberChanges(_lastName_property);
}

// The IPrintDebugState interface implementation
//
void IPrintDebugState.PrintDebugState(PropertyInfo prop)
{
    _firstName.PrintDebugState(_firstName_property);
    _lastName.PrintDebugState(_lastName_property);
}
}

```

Business Logic Toolkit for .NET www.bltoolkit.net

After verifying the accuracy of changes made to the object, you can accept the changes by calling the **AcceptChanges** method of the object, which will set the **Current** field values to be the original values. The **RejectChanges** method rejects all changes made to the object since **AcceptChanges** last called.

AcceptRejectChanges.cs

```
using System;
using NUnit.Framework;
using BLToolkit.EditableObjects;

namespace HowTo.EditableObjects
{
    [TestFixture]
    public class AcceptRejectChanges
    {
        public abstract class TestObject : EditableObject
        {
            public abstract string FirstName { get; set; }
            public abstract string LastName { get; set; }
        }

        [Test]
        public void Test()
        {
            // Create an instance.
            //
            TestObject obj = TestObject.CreateInstance(
                "Tester", "Testerson");

            // Accept changes.
            //
            obj.FirstName = "Tester";
            obj.LastName = "Testerson";
        }
    }
}
```

```
Assert.IsTrue(obj.IsDirty);

obj.AcceptChanges();

Assert.AreEqual("Tester", obj.FirstName);
Assert.IsFalse(obj.IsDirty);

// Reject changes.
//
obj.FirstName = "Developer";

Assert.IsTrue(obj.IsDirty);

obj.RejectChanges();

Assert.AreEqual("Tester", obj.FirstName);
Assert.IsFalse(obj.IsDirty);
    }
}
}
```

Business Logic Toolkit for .NET www.bltoolkit.net

The **IsDirty** property gets a value that indicates whether the object has changed.

IsDirty.cs

```
using System;
using NUnit.Framework;
using BLToolkit.EditableObjects;

namespace HowTo.EditableObjects
{
    [TestFixture]
    public class IsDirty
    {
        public abstract class TestObject : EditableObject
        {
            public abstract string FirstName { get; set; }
            public abstract string LastName { get; set; }
        }

        [Test]
        public void Test()
        {
            TestObject obj = TestObject.CreateInstance();

            Assert.IsFalse(obj.IsDirty);

            obj.FirstName = "Tester";
            obj.LastName = "Testerson";

            Assert.IsTrue(obj.IsDirty);

            obj.AcceptChanges();

            Assert.IsFalse(obj.IsDirty);
        }
    }
}
```

```
}  
  }  
}
```

© 2010 www.bluetooth.net
support@bluetooth.net

Business Logic Toolkit for .NET www.bltoolkit.net

-
- [ExecuteObject](#) How to map DB source to object.
 - [ObjectToObject](#) How to map object to object.
-

Enumerator mapping.

- [EnumToValue](#) How to map enumerator to value.
 - [ValueToEnum](#) How to map value to enumerator.
-

Mapping attributes.

- [MapFieldAttribute](#)
 - [MapValueAttribute](#)
-

Custom mapping.

- [MapToJson](#)
-

Business Logic Toolkit for .NET www.bltoolkit.net

EnumToValue.cs

```
using System;

using NUnit.Framework;

using BLToolkit.Mapping;

namespace HowTo.Mapping
{
    [TestFixture]
    public class EnumToValue
    {
        public enum Gender1
        {
            [MapValue("F")] Female,
            [MapValue("M")] Male,
            [MapValue("U")] Unknown,
            [MapValue("O")] Other
        }

        [Test]
        public void Test1()
        {
            object value = Map.EnumToValue(Gender1.M

            Assert.AreEqual("M", value);
        }

        public enum Gender2
        {
            [MapValue(1)] Female,
            [MapValue(2)] Male,
            [MapValue(3)] Unknown,
        }
    }
}
```

```

        [MapValue(4)] Other
    }

[Test]
public void Test2()
{
    object value = Map.EnumToValue(Gender2.M

    Assert.AreEqual(2, value);
}

public enum Gender3
{
    Female = 1,
    Male = 2,
    Unknown = 3,
    Other = 4
}

[Test]
public void Test3()
{
    object value = (int)Map.EnumToValue(Gend

    Assert.AreEqual(2, value);
}
}
}

```

Business Logic Toolkit for .NET www.bltoolkit.net

Note that the mapper tries to convert the source values to their destination types.

ObjectToObject.cs

```
using System;

using NUnit.Framework;

using BLToolkit.Mapping;

namespace HowTo.Mapping
{
    [TestFixture]
    public class ObjectToObject
    {
        public class SourceObject
        {
            public bool    Value1    = true;
            public string  Value2    = "10";
            public string  StrValue  = "test";
        }

        public class DestObject
        {
            [MapField("Value1")] public bool    BoolVa
            [MapField("Value2")] public int     IntVa

            // If the source and destination field/pr
            // there is no need for using the MapFie
            //

            public string StrVa

        }

        [Test]
```

```

public void Test1()
{
    SourceObject source = new SourceObject();
    DestObject dest = Map.ObjectToObject<

    Assert.AreEqual(true, dest.BoolValue);
    Assert.AreEqual(10, dest.IntValue);
    Assert.AreEqual("test", dest.StrValue);
}
}
}

```

To map data to inner objects the MapField attribute can be used in the following way:

MapFieldAttribute.cs

```

using System;

using NUnit.Framework;

using BLToolkit.Mapping;

namespace HowTo.Mapping
{
    [TestFixture]
    public class MapField
    {
        public class SourceObject1
        {
            public string Street = "1 Main";
            public string City = "Bigtown";
            public string State = "XX";
            public string Zip = "00000";
        }

        public class Address
        {
            public string Street;

```

```

        public string City;
        public string State;
        public string Zip;
    }

    [MapField("Street", "Address.Street")]
    [MapField("City", "Address.City")]
    [MapField("State", "Address.State")]
    [MapField("Zip", "Address.Zip")]
    public class Order1
    {
        public Address Address = new Address();
    }

    [Test]
    public void MapFieldTest1()
    {
        SourceObject1 source = new SourceObject1(
            Order1          order  = Map.ObjectToObject

        Assert.AreEqual("1 Main",    order.Address
        Assert.AreEqual("Bigtown",   order.Address
        Assert.AreEqual("XX",        order.Address
        Assert.AreEqual("00000",     order.Address

    }

    public class SourceObject2
    {
        public string BillingStreet = "1 Main";
        public string BillingCity   = "Bigtown";
        public string BillingState  = "XX";
        public string BillingZip    = "00000";

        public string ShippingStreet = "2 Main";
        public string ShippingCity   = "Bigtown";
        public string ShippingState  = "XX";
        public string ShippingZip    = "00000";
    }

```

```

    }

    public class Order2
    {
        [MapField(Format="Billing{0}")]
        public Address BillingAddress = new Address

        [MapField(Format="Shipping{0}")]
        public Address ShippingAddress = new Address
    }

    [Test]
    public void MapFieldTest2()
    {
        SourceObject2 source = new SourceObject2(
            Order2          order = Map.ObjectToObject

        Assert.AreEqual("1 Main",    order.BillingAddress.Main),
        Assert.AreEqual("Bigtown",   order.BillingAddress.City),
        Assert.AreEqual("XX",        order.BillingAddress.State),
        Assert.AreEqual("00000",     order.BillingAddress.Zip),

        Assert.AreEqual("2 Main",    order.ShippingAddress.Main),
        Assert.AreEqual("Bigtown",   order.ShippingAddress.City),
        Assert.AreEqual("XX",        order.ShippingAddress.State),
        Assert.AreEqual("00000",     order.ShippingAddress.Zip)
    }
}

```

Business Logic Toolkit for .NET www.bltoolkit.net

MapToJson.cs

```
using System;
using System.Globalization;
using System.Text;
using System.Xml;

using NUnit.Framework;

using BLToolkit.Mapping;
using BLToolkit.Reflection;

namespace HowTo.Mapping
{
    public class JsonMapper : MapDataDestinationBase,
    {
        private static readonly long    InitialJavaScript;

        private string[]                _fieldNames;
        private readonly StringBuilder _sb;
        private MappingSchema           _mappingSchema;
        private bool                    _scalar;
        private bool                    _first;
        private bool                    _firstElement;
        private int                     _indent;

        public JsonMapper() : this(new StringBuilder())
        {
        }

        public JsonMapper(StringBuilder sb) : this(sb)
        {
        }
    }
}
```

```

public JsonMapper(StringBuilder sb, int indent)
{
    _sb      = sb;
    _indent = indent;
}

public override Type GetFieldType(int index)
{
    // Same as typeof(object)
    //
    return null;
}

public override int GetOrdinal(string name)
{
    return Array.IndexOf(_fieldNames, name);
}

public override void SetValue(object o, int index, object value)
{
    SetValue(o, _fieldNames[index], value);
}

public override void SetValue(object o, string name, object value)
{
    if (!_scalar)
    {
        // Do not json null values until it's a scalar
        //
        if (value == null || (value is XmlNode))
            return;

        if (_first)
            _first = false;
        else
            _sb
                .Append(',')
    }
}

```

```

        .AppendLine()
        ;

    for (int i = 0; i < _indent; ++i)
        _sb.Append(' ');

    _sb
        .Append('"')
        .Append(name)
        .Append("\":")
        ;
}

if (value == null)
    _sb.Append("null");
else
{
    switch (Type.GetTypeCode(value.GetType()))
    {
        case TypeCode.Empty:
        case TypeCode.DBNull:
            _sb.Append("null");
            break;
        case TypeCode.Boolean:
            _sb.Append((bool)value? "true" : "false");
            break;
        case TypeCode.Char:
            _sb
                .Append('\''')
                .Append((char)value)
                .Append('\''')
                ;
            break;
        case TypeCode.SByte:
        case TypeCode.Int16:
        case TypeCode.Int32:
        case TypeCode.Int64:

```

```

case TypeCode.Byte:
case TypeCode.UInt16:
case TypeCode.UInt32:
case TypeCode.UInt64:
case TypeCode.Single:
case TypeCode.Double:
case TypeCode.Decimal:
    _sb.Append(((IFormattable)value)
break;
case TypeCode.DateTime:
    _sb
        .Append("new Date(")
        .Append(((DateTime)value)
        .Append(")");
break;
case TypeCode.String:
    _sb
        .Append("'")
        .Append(encode((string)value)
        .Append("'")
        ;
break;
default:
    if (value is XmlNode)
    {
        if (IsEmptyNode((XmlNode)
            _sb.Append("null");
        else
            WriteXmlJson((XmlNode)
    }
    else
    {
        JsonSerializer inner = new Js

        if (value.GetType().IsArr
            _mappingSchema.MapSou
            _mappingSchema.Ge

```

```

else
    _mappingSchema.MapSou
        _mappingSchema.Ge
    }
    break;
}
}
}

private static string encode(string value)
{
    return value.Replace("\r\n", "\\r")
        .Replace("\n\r", "\\r")
        .Replace("\n", "\\r")
        .Replace("\r", "\\r")
        .Replace("\"", "\\\"");
}

private void WriteXmlToJson(XmlNode node)
{
    XmlNode textNode = GetTextNode(node);
    if (textNode != null)
    {
        _sb
            .Append("\"")
            .Append(encode(textNode.Value))
            .Append("\"");
    }
    else
    {
        bool first = true;

        _sb.Append('{');

        if (node.Attributes != null)

```

```

{
    foreach (XmlAttribute attr in node.Attributes)
    {
        if (first)
            first = false;
        else
            _sb.Append(', ');

        _sb
            .Append("\"@" + attr.Name)
            .Append("\":\")")
            .Append(encode(attr.Value))
            .Append("\"");
    }
}

foreach (XmlNode child in node.Children)
{
    if (IsWhitespace(child) || IsEmpty(child))
        continue;

    if (first)
        first = false;
    else
        _sb.Append(', ');

    if (child is XmlText)
    {
        _sb
            .Append("#text\"")
            .Append(encode(child.Value))
            .Append("\"");
    }
    else if (child is XmlElement)
    {
        _sb

```

```

        .Append('"')
        .Append(child.Name)
        .Append("\":")
        ;
        WriteXmlJson(child);
    }
    else
        System.Diagnostics.Debug.Fail
    }
    _sb.Append('}');
}
}

private static bool IsWhitespace(XmlNode node)
{
    switch (node.NodeType)
    {
        case XmlNodeType.Comment:
        case XmlNodeType.Whitespace:
        case XmlNodeType.SignificantWhitespace:
            return true;
    }
    return false;
}

private static bool IsEmptyNode(XmlNode node)
{
    if (node.Attributes != null && node.Attributes.Count > 0)
        return false;

    if (node.HasChildNodes)
        foreach (XmlNode childNode in node.ChildNodes)
        {
            if (IsWhitespace(childNode) || IsEmptyNode(childNode))
                continue;

            // Not a whitespace, nor inner empty node

```

```

        //
        return false;
    }

    return node.Value == null;
}

private static XmlNode GetTextNode(XmlNode node)
{
    if (node.Attributes != null && node.Attributes.Count > 0)
        return null;

    XmlNode textNode = null;

    foreach (XmlNode childNode in node.ChildNodes)
    {
        // Ignore all whitespace.
        //
        if (IsWhitespace(childNode))
            continue;

        if (childNode is XmlText)
        {
            // More than one text node.
            //
            if (textNode != null)
                return null;

            // First text node.
            //
            textNode = childNode;
        }
        else
        {
            // Not a text node - break;
            //
            return null;
        }
    }
}

```

```

        return textNode;
    }

    #region ISupportMapping Members

    void ISupportMapping.BeginMapping(InitContext
    {
        _first          = true;
        _mappingSchema = initContext.MappingScher
        _fieldNames     = new string[initContext.L

        for (int i = 0; i < _fieldNames.Length; -
            _fieldNames[i] = initContext.DataSou

        _scalar = _fieldNames.Length == 1 && str:

        if (_scalar)
            return;

        if (_fieldNames.Length <= 1)
        {
            // Reset the indent since output is a
            //
            _indent = 0;
            _sb.Append('{');
        }
        else
        {
            if (_indent > 0)
                _sb.AppendLine();

            for (int i = 0; i < _indent; ++i)
                _sb.Append(' ');

            _sb
                .Append('{')

```

```

        .AppendLine()
        ;
    }
}

void ISupportMapping.EndMapping(InitContext :
{
    if (_scalar)
        return;

    if (_fieldNames.Length > 1)
        _sb.AppendLine();

    for (int i = 0; i < _indent; ++i)
        _sb.Append(' ');
    _sb.Append('}');
}

#endregion

#region IMapDataDestinationList Members

void IMapDataDestinationList.InitMapping(Init
{
    _firstElement = true;
    _sb.Append('[');
}

IMapDataDestination IMapDataDestinationList.(
{
    return this;
}

object IMapDataDestinationList.GetNextObject(
{
    if (_firstElement)
        _firstElement = false;
}

```

```

        else
            _sb.Append(', ');

        return this;
    }

    void IMapDataDestinationList.EndMapping(Init(
    {
        _sb.Append(']');
    }

    #endregion

    public override string ToString()
    {
        return _sb.ToString();
    }
}

[TestFixture]
public class MapToJson
{
    public class Inner
    {
        public string Name = "inner \"object \n r
    }

    public class Inner2
    {
        public string Name;
        public int Value;
    }

    public class SourceObject
    {
        public string Foo = "Foo";
        public double Bar = 1.23;
    }
}

```

```

    public DateTime Baz = DateTime.Today;
    [MapIgnore(false)]
    public Inner Inner = new Inner();
    [MapIgnore(false)]
    public Inner2 Inner2 = new Inner2();
    public string[] StrArray = {"One", "Two",
}

[Test]
public void Test()
{
    JsonSerializer jm = new JsonSerializer(new Str

    Map.MapSourceToDestination(Map.GetObjectM
    Console.WriteLine(jm.ToString());

    // Expected output:
    //
    // {
    //   "Foo": "Foo",
    //   "Bar": 1.23,
    //   "Baz": new Date(1182384000000000000),
    //   "Inner": { "Name": "inner \object \r na
    //   "Inner2":
    //     {
    //       "Name": null,
    //       "Value": 0
    //     },
    //   "StrArray": ["One", "Two", "Three"]
    // }

}
}
}

```

Business Logic Toolkit for .NET www.bltoolkit.net

MapValueAttribute1.cs

```
using System;

using NUnit.Framework;

using BLToolkit.Mapping;

namespace HowTo.Mapping
{
    [TestFixture]
    public class MapValue1
    {
        public class SourceObject
        {
            public string Value = "Y";
        }

        public class TestObject1
        {
            // The attribute is applied to a field/pr
            //
            [MapValue(true, "Y")]
            [MapValue(false, "N")]
            public bool Value;
        }

        [Test]
        public void Test1()
        {
            SourceObject so = new SourceObject();
            TestObject1 to = Map.ObjectToObject<Test

            Assert.AreEqual(true, to.Value);
        }
    }
}
```

```

}

// The attribute is applied to a class.
//
[MapValue(true, "Y")]
[MapValue(false, "N")]
public class TestObject2
{
    public bool Value;
}

[Test]
public void Test2()
{
    SourceObject so = new SourceObject();
    TestObject2 to = Map.ObjectToObject<TestObject2>(so);

    Assert.AreEqual(true, to.Value);
}

// The attribute is applied to a base class.
//
[MapValue(typeof(bool), true, "Y")]
[MapValue(typeof(bool), false, "N")]
public class ObjectBase
{
}

public class TestObject3 : ObjectBase
{
    public bool Value;
}

[Test]
public void Test3()
{
    SourceObject so = new SourceObject();

```

```

        TestObject3 to = Map.ObjectToObject<Test
        Assert.AreEqual(true, to.Value);
    }
}

```

MapValueAttribute2.cs

```

using System;

using NUnit.Framework;

using BLToolkit.Mapping;

namespace HowTo.Mapping
{
    [TestFixture]
    public class MapValue2
    {
        public enum Gender1
        {
            [MapValue("F")] Female,
            [MapValue("M")] Male,
            [MapValue("U")] Unknown,
            [MapValue("O")] Other
        }

        [Test]
        public void Test1()
        {
            object value = Map.EnumToValue(Gender1.Ma

            Assert.AreEqual("M", value);
        }

        [MapValue(Gender2.Female, 1)]
    }
}

```

```
[MapValue(Gender2.Male, 2)]
[MapValue(Gender2.Unknown, 3)]
[MapValue(Gender2.Other, 4)]
public enum Gender2
{
    Female,
    Male,
    Unknown,
    Other
}

[Test]
public void Test2()
{
    Gender2 g = Map.ToEnum<Gender2>(2);

    Assert.AreEqual(Gender2.Male, g);
}
}
```

Business Logic Toolkit for .NET www.bltoolkit.net

Note that the mapper tries to convert the source values to their destination types.

ObjectToObject.cs

```
using System;

using NUnit.Framework;

using BLToolkit.Mapping;

namespace HowTo.Mapping
{
    [TestFixture]
    public class ObjectToObject
    {
        public class SourceObject
        {
            public bool    Value1    = true;
            public string  Value2    = "10";
            public string  StrValue  = "test";
        }

        public class DestObject
        {
            [MapField("Value1")] public bool    BoolVa
            [MapField("Value2")] public int     IntVa

            // If the source and destination field/pr
            // there is no need for using the MapFie
            //

            public string StrVa

        }

        [Test]
```

```
public void Test1()
{
    SourceObject source = new SourceObject();
    DestObject dest = Map.ObjectToObject<

    Assert.AreEqual(true, dest.BoolValue);
    Assert.AreEqual(10, dest.IntValue);
    Assert.AreEqual("test", dest.StrValue);
}
}
```

Business Logic Toolkit for .NET www.bltoolkit.net

ValueToEnum.cs

```
using System;

using NUnit.Framework;

using BLToolkit.Mapping;

namespace HowTo.Mapping
{
    [TestFixture]
    public class ValueToEnum
    {
        public enum Gender1
        {
            [MapValue("F")] Female,
            [MapValue("M")] Male,
            [MapValue("U")] Unknown,
            [MapValue("O")] Other
        }

        [Test]
        public void Test1()
        {
            Gender1 g = Map.ToEnum<Gender1>("M");

            Assert.AreEqual(Gender1.Male, g);
        }

        public enum Gender2
        {
            [MapValue(1)] Female,
            [MapValue(2)] Male,
            [MapValue(3)] Unknown,
        }
    }
}
```

```
        [MapValue(4)] Other
    }

[Test]
public void Test2()
{
    Gender2 g = Map.ToEnum<Gender2>(2);

    Assert.AreEqual(Gender2.Male, g);
}

public enum Gender3
{
    Female = 1,
    Male   = 2,
    Unknown = 3,
    Other  = 4
}

[Test]
public void Test3()
{
    Gender3 g = Map.ToEnum<Gender3>(2);

    Assert.AreEqual(Gender3.Male, g);
}
}
}
```

Business Logic Toolkit for .NET www.bltoolkit.net

- [Emit](#)

- [ObjectFactory](#)

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

-
- Hello, World! *EmitHelper* in action.
-

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

The **System.Reflection.Emit** namespace provides classes to create dynamic assemblies. Although **Emit** is a powerful tool, it is also extremely hard to use.

Let us take a look at the following example, which demonstrates the "normal" way of emitting code.

HelloWorld.cs

```
using System;
using System.Reflection;
using System.Reflection.Emit;
using System.Threading;

using NUnit.Framework;

namespace Examples.Reflection.Emit
{
    [TestFixture]
    public class HelloWorldNormal
    {
        public interface IHello
        {
            void SayHello(string toWhom);
        }

        [Test]
        public void Test()
        {
            AssemblyName asmName = new AssemblyName("HelloWorld");

            asmName.Name = "HelloWorld";

            AssemblyBuilder asmBuilder =
                Thread.GetDomain().DefineDynamicAssembly(asmName, AssemblyBuilderAccess.Run);
        }
    }
}
```

```

ModuleBuilder    modBuilder    = asmBuilder.
TypeBuilder      typeBuilder    = modBuilder.
    "Hello",
    TypeAttributes.Public,
    typeof(object),
    new Type[] { typeof(IHello) });

MethodBuilder    methodBuilder  = typeBuild
    MethodAttributes.
    typeof(void),
    new Type[] { type

typeBuilder.DefineMethodOverride(methodBu

ILGenerator il = methodBuilder.GetILGener

// string.Format("Hello, {0} World!", toV
//
il.Emit(OpCodes.Ldstr, "Hello, {0} World
il.Emit(OpCodes.Ldarg_1);
il.Emit(OpCodes.Call, typeof(string).GetM

// Console.WriteLine("Hello, World!");
//
il.Emit(OpCodes.Call, typeof(Console).Get
il.Emit(OpCodes.Ret);

Type    type    = typeBuilder.CreateType();

IHello hello = (IHello)Activator.CreateIn

hello.SayHello("Emit");

    }
}
}

```

Note that the Emit method takes one parameter as an OpCode and optionally another one safe.

Fortunately, there is an alternative way. BLToolkit provides a helper class, [EmitHelper](#), which emits commands and allows writing source code that looks similar to **MSIL**.

The following examples show how to use the EmitHelper class with C#, VB.NET, and C++.

HelloWorld.cs

```
using System;

using NUnit.Framework;

using BLToolkit.Reflection;
using BLToolkit.Reflection.Emit;

namespace Examples.Reflection.Emit
{
    [TestFixture]
    public class HelloWorld
    {
        public interface IHello
        {
            void SayHello(string toWhom);
        }

        [Test]
        public void Test()
        {
            EmitHelper emit = new AssemblyBuilderHelper
                .DefineType ("Hello", typeof(object))
                .DefineMethod(typeof(IHello).GetMethod("SayHello"))
                .Emitter;

            emit
                // string.Format("Hello, {0}!", toWhom)
                //
                .ldstr ("Hello, {0}!")
        }
    }
}
```

```

        .ldarg 1
        .call (typeof(string), "Format", t
// Console.WriteLine("Hello, World!")
//
        .call (typeof(Console), "WriteLine
        .ret()
        ;

    Type type = emit.Method.Type.Create();
    IHello hello = (IHello)TypeAccessor.Create
    hello.SayHello("World");
}
}
}

```

Cool!!! Isn't it?

HelloWorld.vb

```

Imports System
Imports NUnit.Framework

Imports BLToolkit.Reflection
Imports BLToolkit.Reflection.Emit

Namespace Examples.Reflection.Emit

    <TestFixture(> _
    Public Class HelloWorld

        Public Interface IHello
            Sub SayHello(ByVal toWhom As String)
        End Interface

    <Test(> _

```

```

Sub Test()
    Dim assemblyHelper As AssemblyBuilderHelper
    Dim typeHelper As TypeBuilderHelper = assemblyHelper.GetTypeBuilder
    Dim methodHelper As MethodBuilderHelper = typeHelper.CreateMethodBuilder
    Dim emit As EmitHelper = methodHelper.Emit

    ' string.Format("Hello, {0} World!", towH
    '
    emit _
        .ldstr("Hello, {0} World!") _
        .ldarg_1 _
        .call(GetType(String), "Format", GetType(
    ' Console.WriteLine("Hello, World!");
    '
    emit _
        .call(GetType(Console), "WriteLine", GetT
        .ret()

    Dim type As Type = typeHelper.Create()

    Dim hello As IHello = TypeAccessor.Create

    hello.SayHello("VB")
End Sub

End Class

End Namespace

```

HelloWorld.cpp

```

#include "stdafx.h"

using namespace System;

using namespace NUnit::Framework;

```

```

using namespace BLToolkit::Reflection;
using namespace BLToolkit::Reflection::Emit;

namespace Examples {
namespace Reflection {
namespace Emit
{
    [TestFixture]
    public ref class HelloWorld
    {
    public:

        interface class IHello
        {
            void SayHello(String ^toWhom);
        };

        [Test]
        void Test()
        {
            AssemblyBuilderHelper ^assembly = gcnew /

            EmitHelper ^emit = assembly
                ->DefineType ("Hello", Object::type:
                ->DefineMethod(IHello::typeid->GetMet
                ->Emitter;

            emit
                // string.Format("Hello, {0} World!",
                //
                ->ldstr ("Hello, {0} World!")
                ->ldarg_1
                ->call (String::typeid, "Format",

                // Console.WriteLine("Hello, World!");
                //

```

```
        ->call      (Console::typeid, "WriteLin  
        ->ret()  
        ;  
  
        Type ^type = emit->Method->Type->Create(  
        IHello ^hello = (IHello^)TypeAccessor::Cr  
        hello->SayHello("C++");  
    }  
};  
}}}
```

Business Logic Toolkit for .NET www.bltoolkit.net

The following example shows how to create objects of different types coming from the s

Typically, to create an object, BLToolkit calls the *TypeAccessor.CreateInstanceEx* meth (property) associated with the creating object's type. If the property value is not null, it

The *ObjectFactory* property can be assigned explicitly at any time or by decorating the t

ObjectFactory.cs

```
using System;
using System.Collections.Generic;

using NUnit.Framework;

using BLToolkit.Data;
using BLToolkit.Mapping;
using BLToolkit.Reflection;

namespace HowTo.Reflection
{
    [TestFixture]
    public class ObjectFactoryTest
    {
        [ObjectFactory(typeof(Person.ObjectFactory))]:
        public class Person
        {
            [MapField("PersonID")]
            public int ID;

            public string LastName;
            public string FirstName;
            public string MiddleName;

            class ObjectFactory : IObjectFactory
            {
```

```

public object CreateInstance(TypeAcc
{
    // Get the object type indicator
    //
    object objectType = context.DataS

    // Target ObjectMapper must be ch
    //
    switch ((string)objectType)
    {
        case "D": context.ObjectMapper
        case "P": context.ObjectMapper
    }

    // Create an object instance.
    // Do not call ObjectMapper.Creat
    //
    return context.ObjectMapper.Type/
    }
}
}

```

```

public class Doctor : Person
{
    public string Taxonomy;
}

```

```

public class Patient : Person
{
    public string Diagnosis;
}

```

```

[Test]
public void Test()
{
    using (DbManager db = new DbManager())
    {

```



```
</configuration>
```

[Create.sql script](#)

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

Abstract types.

- [InitialValues](#) How to specify initial value for an abstract property.
 - [InternalTypes](#) How to use internal abstract types.
 - [XmlSerialization](#) How to serialize an abstract type to xml.
-

© 2010 www.bltoolkit.net
support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

InitialValues.cs

```
using System;

using NUnit.Framework;

using BLToolkit.Reflection;
using BLToolkit.TypeBuilder;

namespace HowTo.TypeBuilder
{
    [TestFixture]
    public class InitialValueTest
    {
        [AttributeUsage(AttributeTargets.Property)]
        public class NewGuidParameterAttribute : Para
        {
            public NewGuidParameterAttribute() : base
            {
            }
        }

        public abstract class TestObject1
        {
            [Parameter("t")] public abstract string
            [Parameter(20)] public abstract string
            [Parameter(54)] public abstract int
            [Parameter(2,2,2)] public abstract DateTime
            [Parameter(222L)] public abstract Decimal
            [Parameter(-2.05)] public abstract Decimal
            [NewGuidParameter] public abstract Guid
        }

        [Test]
```

```
public void Test()
{
    TestObject1 o = (TestObject1)TypeAccessor

    Assert.That(o.Str,      Is.EqualTo("t"));
    Assert.That(o.Int,      Is.EqualTo(54));
    Assert.That(o.Date,     Is.EqualTo(new Da
    Assert.That(o.Decimal1, Is.EqualTo(222m));
    Assert.That(o.Decimal2, Is.EqualTo(-2.05r
    Assert.That(o.Guid,     Is.Not.EqualTo(Gu
}
}
}
```

Business Logic Toolkit for .NET www.bltoolkit.net

InternalTypes.cs

```
using System;
using System.Runtime.CompilerServices;
using NUnit.Framework;

using BLToolkit.Reflection;

// typeof(TargetType).FullName + "." + TypeBuilderCore
//
[assembly: InternalsVisibleTo("HowTo.TypeBuilder.Inte
[assembly: InternalsVisibleTo("HowTo.TypeBuilder.Inte

namespace HowTo.TypeBuilder
{
    [TestFixture]
    public class InternalTypesTest
    {
        internal abstract class TestObject
        {
            public abstract string Value { get; set; }
        }

        [Test]
        public void Test()
        {
            TestObject o = TypeAccessor.CreateInstance
            Assert.IsNotNull(o);
        }
    }
}
```

© 2010 www.bluetooth.net
support@bluetooth.net

Business Logic Toolkit for .NET www.bltoolkit.net

XmlSerialization.cs

```
using System.IO;
using System.Xml.Serialization;

using NUnit.Framework;

using BLToolkit.Reflection;
using BLToolkit.EditableObjects;
using BLToolkit.TypeBuilder;

namespace HowTo.TypeBuilder
{
    [TestFixture]
    public class XmlSerializationTest
    {
        [XmlAttribute(AnonymousType = true)]
        public abstract class MyClassA
        {
            public abstract string ValueA { get; set; }
        }

        [XmlAttribute(AnonymousType = true)]
        [XmlAttribute(typeof(MyClassA))]
        [XmlAttribute(typeof(MyClassC))]
        public abstract class MyClassB
        {
            public abstract string ValueB {
            public abstract MyClassA ValueMyClassA {

            public abstract EditableList<MyClassA> My

        }

        [XmlAttribute("abs:MyClassC")]
    }
}
```

```

public abstract class MyClassC : MyClassA {

[Test]
public void Test()
{
    MyClassB original = TypeAccessor<MyClassB>.Of(MyClassB);
    MyClassB serialized;
    XmlSerializer sr = new XmlSerializer(typeof(MyClassB));

    original.ValueB = "string \";
    original.ValueMyClassA.ValueA = "string \";
    original.MyList.Add(TypeAccessor<MyClassA>.Of(MyClassA));
    original.MyList.Add(TypeAccessor<MyClassA>.Of(MyClassA));

    using (MemoryStream stm = new MemoryStream())
    {
        sr.Serialize(stm, original);
        stm.Position = 0L;
        serialized = (MyClassB)sr.Deserialize(stm);
    }

    Assert.That(serialized.ValueB, Is.EqualTo(original.ValueB));
    Assert.That(serialized.ValueMyClassA.ValueA, Is.EqualTo(original.ValueMyClassA.ValueA));

    Assert.AreEqual(original.MyList.Count, serialized.MyList.Count);
    Assert.That(serialized.MyList[0], Is.TypeOf(MyClassA));
    Assert.That(serialized.MyList[1], Is.TypeOf(MyClassA));
    Assert.That(serialized.MyList[1], Is.TypeOf(MyClassA));

    Assert.AreEqual(serialized.MyList[0].GetHashCode(), original.MyList[0].GetHashCode());
    Assert.AreEqual(serialized.MyList[1].GetHashCode(), original.MyList[1].GetHashCode());
}
}
}
}

```

support@bltoolkit.net

Business Logic Toolkit for .NET www.bltoolkit.net

© 2010 www.bltoolkit.net
support@bltoolkit.net