iBATIS.NET

Data Access Objects Developer Guide

Version 1.9.1

March 2007

Authors - Gilles Bayon, Clinton Begin, Roberto Rabe

Table of Contents

Chapter 1. Introduction

1.1. Overview

When developing robust .NET connected applications, it is often a good idea to isolate the specifics of your persistence implementation behind a common API. Data Access Objects allow you to create simple components that provide access to your data without revealing the specifics of the implementation to the rest of your application. Using DAOs, you can allow your application to be dynamically configured to use different persistence mechanisms. If you have a complex application with a number of different databases and persistence approaches involved, DAOs can help you create a consistent API for the rest of your application to use.

Tip

If you would like to get the latest development (unreleased) version of this Guide, please see the iBATIS Wiki FAQ [http://opensource.atlassian.com/confluence/oss/display/IBATIS/]. The FAQ entry explains how you can access our SVN source repository and generate CHM and PDF files of the latest development documentation for the DataAccess framework.

1.2. License Information

1.3. Support

Mailing lists and bug trackers are available (courtesy of Apache Software Foundation) at iBATIS's Apache project page. Just direct your browser to http://ibatis.apache.org/.

1.4. Disclaimer

Chapter 2. Data Access Objects

2.1. Data Access Objects

The iBATIS.NET Data Access Objects API can be used to help hide persistence layer implementation details from the rest of your application by allowing dynamic and pluggable DAO components to be easily swapped in and out. For example, you could have two implementations of a particular DAO, one that uses the iBATIS.NET DataMapper framework and another that uses the NHibernate framework, to persist objects to the database. Another example would be a DAO that provides caching services for another DAO. Depending on the situation (e.g. limited database performance vs. limited memory), either the cache DAO could be plugged in or the standard un-cached DAO could be used. These examples show the convenience, and more importantly, the safety that the DAO pattern provides. The DAO pattern protects your application from possibly being tied to a particular persistence approach. In the event that your current solution becomes unsuitable (or even unavailable), you can simply create new DAO implementations to support a new solution without having to modify any code in the other layers of your application.

Note

The iBATIS.NET DAO Framework and DataMapper Framework are completely separate and are not dependent on each other in any way. Please feel free to use either one separately, or both together.

2.2. The Components of the Data Access Objects API

There are a number of classes that make up the IBatisNet.DataAccess API. Each has a very specific and important role. The following table lists the classes and a brief description. The next sections will provide more detail on how to use these classes.

Table 2.1. IBatisNet.DataAccess API Classes

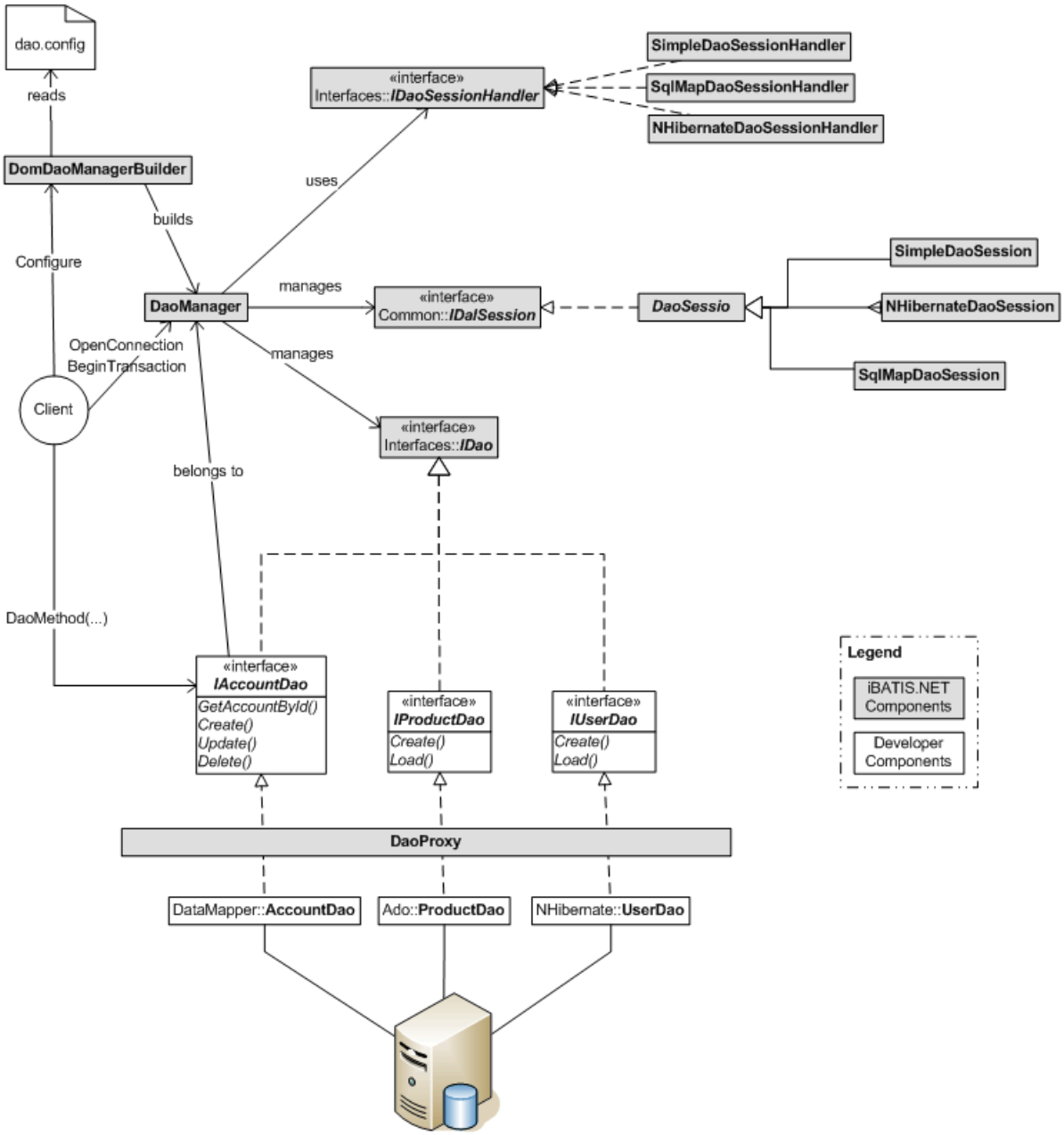
Class/Interface (Patterns)	Description
DomDaoManagerBuilder	Responsible for configuration of the DAO framework (via dao.config), instantiating DAO implementations and IDaoManager instances.
IDaoManager (Facade)	Acting as a façade to the rest of the API.
IDalSession (Marker Interface)	A generic marker interface for a database session. A common implementation would wrap an ADO.NET connection/transaction object.
DataAccessException (Runtime Exception)	All methods and classes in the DAO API throw this exception exclusively. DAO implementations should also throw this exception exclusively and avoid throwing any other exception type by nesting them within the DataAccessException.
IDao (Marker Interface)	A marker interface for all DAO implementations. This interface must be implemented by all DAO classes. This interface does not declare any methods

Class/Interface (Patterns)	Description
	to be implemented and only acts as a marker (i.e. something for the XML-based DaoFactory to identify the class by).

2.3. Session Handler

Table 2.2. Session Handler Implementations

Session Handler	Description
SqlMap	Manages connection and transactions via the DataMapper framework.
NHibernate	Provides easy integration for NHibernate and its associated transaction facilities (SessionFactory, Session, Transaction).
ADONET	Manages connection and transactions via ADO.NET using the basic iBATIS.NET DataSource, IDbConnection, and IDbTransaction interfaces.



iBATIS.NET Data Access Objects

Chapter 3. Configuration

3.1. Installation

There are three steps to using the iBATIS Data Access Objects (DAO) framework with your application for the first time.

1. Setup the distribution
2. Add assembly references
3. Setup XML configuration documents

3.1.1. Setup the Distribution

The official site for iBATIS for .NET is our Apache site <http://ibatis.apache.org/>. The DataAccess framework is availabe in 2 types of distributions: a binary distribution that includes the required DataAccess assemblies and a source distribution that includes a VSN solution. To download either of the distributions, follow the link to the Downloads area on our web site, and select the either the binary or source distribution for the iBATIS .NET DataAccess V1.6.1 or later release (if you download the binary distribution, extract the files using a utility like WinZip or the extractor built into newer versions of Windows and skip ahead to the Add Assembly References section).

The source distribution includes a VSN solution and a number of C# projects. The distribution is in the form of a ZIP archive. You can extract the distribution using a utility like WinZip or the extractor built into newer versions of Windows. We suggest that you create an `ibatisnet` folder in your VSN project directory and extract the distribution there.

Under the distribution's `source` folder are eight folders that make up the iBATIS.NET distribution, as shown in the following table:

Table 3.1. Folders found in the iBATIS.NET source distribution

Folder name	Description
External-Bin	Dependency assemblies provided for your convenience.
IBatisNet.Common	Assembly of classes shared by DataAccess and DataMapper
IBatisNet.Common.Test	Test project for IBatisNet.Common that can be used with NUnit
IBatisNet.DataAccess	The Data Access Objects framework
IBatisNet.DataAccess.Extensions	Contains a C# project for extensions to the DataAccess framework such as NHibernate support
IBatisNet.DataAccess.Test	Test project for the DataAccess framework that can be used with NUnit
iBatisNet.DataMapper	The DataMapper framework (see seperate

Folder name	Description
	DataMapper Guide)
IBatisNet.DataMapper.Test	Test project for the DataMapper that can be used with NUnit

You can load the `IBatisNet.sln` solution file into VSN and build the solution to generate the needed assemblies. There are seven projects in the solution, and all should succeed. The assemblies we need will be created under `\source\IBatisNet.DataAccess\bin\Debug`. The created assemblies are :

1. `IBatisNet.Common.dll`
2. `iBatisNet.DataAccess.dll`

The core DataAccess framework has external dependencies on

1. `Castle.DynamicProxy.dll` (creating proxies)

The dependency is found in the `External-Bin` folder and can also be found in the `bin\Debug` folder after building the solution. In addition, the framework's `IBatisNet.DataAccess.Extensions` and `IBatisNet.DataAccess.Test` projects have dependencies on NHibernate.

Tip

If you will not be using NHibernate and have a problem building the solution due to that dependency, simply remove the IBatisNet.DataAccess.Extensions and IBatisNet.DataAccess.Test projects from the solution before building.

3.1.2. Add Assembly References

Switching to your own solution, open the project that will be using the iBATIS.NET DAO Framework. Depending on how your organize your solutions, this might not be the project for your Windows or Web application. It may be a library project that your application project references. You need to add two references to your project:

1. IBatisNet.DataAccess.dll
2. IBatisNet.Common.dll

Additionally, the DAO framework has the following external dependencies:

Table 3.2. Dependencies

Name	Description	Assembly DLLs
Castle dynamic proxy 1.1.5.0	Dynamic proxy generator	Castle.DynamicProxy.dll

3.1.3. Setup XML configuration documents

You will need to add two or more XML file items to your Windows or Web application project (and Test project if you have one). These files are:

- dao.config - The Data Access configuration file that is used to specify your DAOs, providers.config file location, and data source information.
- providers.config - A file used by the framework to look up the definition of your selected database provider.
- SqlMap.config and Data Map definition XML files - Required when using the iBATIS DataAccess framework in conjunction with the iBATIS DataMapper framework

The `dao.config` and `providers.config` files must be placed in a location where the framework can find them at runtime. Depending on the type of project you have, the default expected location of these 2 files will be different, as shown in Table 3.3. However, your project is not limited to using just these locations. The DataAccess framework provides other options for placing these files in locations that are more suitable for your project instead of using the default locations. These options are covered later in this guide.

Table 3.3. Where to place the dao.config file

Windows, Library, or Test projects (using NUnit or equivalent)	Place with the assembly (.dll) files and the `app.config` file
Web projects	Place in the project root, with the `web.config` file

3.1.4. Visual Studio.NET Integration

The configuration file (dao.config) is associated to a schema. The benefits of associating an XML document with a schema are to validate the document (which is done at runtime) and to use editing features such as IntelliSense/content completion assistance.

To allow association of the schemas in VS.NET XML editor to your configuration file, you should add the schema file (`DaoConfig.xsd` to either your VS.NET project or in your VS.NET installation directory. The VS.NET directory will be either

```
C:\Program Files\Microsoft Visual Studio 8\Xml\Schemas for VS.NET 2005
```

or

```
C:\Program Files\Microsoft Visual Studio .NET 2003\Common7\Packages\schemas\xml for VS.NET 2003
```

or

```
C:\Program Files\Microsoft Visual Studio .NET\Common7\Packages\schemas\xml for VS.NET 2002
```

depending on your version of VS.NET. It is typically easier to place the file in the well known location under the VS.NET installation directory than to copy the XSD file for each project you create.

Once you have registered the schema with VS.NET you will be enough to get IntelliSense and validation of the configuration file from within VS.NET.

Figure 3.1. IntelliSense example

3.2. Logging Activity

The iBATIS DataAccess framework records its interaction with the database through an internal logging mechanism patterned after Apache Log4Net. The internal logging mechanism can use one of the three built-in loggers (NoOpLogger, ConsoleOutLogger, TraceLogger) or external logging packages such as Apache Log4Net. In order for iBATIS to generate log messages, the application's config file (App.Config or Web.Config) must contain an appropriate configSection node:

Example 3.1. iBATIS Configuration Section Handler for logging

```
<configSections>
 <sectionGroup name="iBATIS">
  <section name="logging" type="IBatisNet.Common.Logging.ConfigurationSectionHandler, IBatisNet.Common" />
 </sectionGroup>
</configSections>
```

The application's config file must declare one logger implementation. See the examples below on how to configure one of the three built-in loggers.

```
<iBATIS>
 <logging>
  <logFactoryAdapter type="IBatisNet.Common.Logging.Impl.ConsoleOutLoggerFA, IBatisNet.Common">
   <arg key="showLogName" value="true" />
   <arg key="showDataTime" value="true" />
   <arg key="level" value="ALL" />
   <arg key="dateTimeFormat" value="yyyy/MM/dd HH:mm:ss:SSS" />
  </logFactoryAdapter>
 </logging>
</iBATIS>
```

```
<iBATIS>
 <logging>
  <logFactoryAdapter type="IBatisNet.Common.Logging.Impl.NoOpLoggerFA, IBatisNet.Common" />
 </logging>
</iBATIS>
```

```
<iBATIS>
 <logging>
  <logFactoryAdapter type="IBatisNet.Common.Logging.Impl.TraceLoggerFA, IBatisNet.Common" />
 </logging>
</iBATIS>
```

To configure iBATIS to use another logger implementation, simple specify the appropriate logFactoryAdapter type. To use Apache Log4Net with the iBATIS framework, use the following configuration setting:

```
<iBATIS>
 <logging>
  <logFactoryAdapter type="IBatisNet.Common.Logging.Impl.Log4NetLoggerFA, IBatisNet.Common.Logging.Log4Net">
   <arg key="configType" value="inline" />
  </logFactoryAdapter>
 </logging>
</iBATIS>
```

```
<iBATIS>
 <logging>
  <logFactoryAdapter type="IBatisNet.Common.Logging.Impl.Log4NetLoggerFA, IBatisNet.Common.Logging.Log4Net">
   <arg key="configType" value="file" />
   <arg key="configFile" value="log4Net.config" />
  </logFactoryAdapter>
 </logging>
</iBATIS>
```

The Log4NetLoggerFA supports the following values for the configTypes argument:

Table 3.4. Valid configType values

configType	Description
inline	log4net node will use the log4net node in the App.Config/Web.Config file when it is configured
file	(also requires configFile argument) - log4net will use an external file for its configuration
file-watch	(also requires configFile argument) - log4net will use an external file for its configuration and will re-configure itself if this file changes
external	iBATIS will not attempt to configure log4net.

3.2.1. Sample Logging Configurations

The simplest logging configuration is to output log messages to Console.Out:

```
<configuration>
 <configSections>
  <sectionGroup name="iBATIS">
   <section name="logging" type="IBatisNet.Common.Logging.ConfigurationSectionHandler, IBatisNet.Common" />
  </sectionGroup>
 </configSections>
 <iBATIS>
  <logging>
   <logFactoryAdapter type="IBatisNet.Common.Logging.Impl.ConsoleLoggerFA, IBatisNet.Common.Logging" />
  </logging>
 </iBATIS>
</configuration>
```

A common logging configuration is to use Apache Log4Net. To use Log4Net with your own application, you need to provide your own Log4Net configuration. You can do this by adding a configuration file for your assembly that includes a <log4Net> element. The configuration file is named after your assembly but adds a `.config` extension, and is stored in the same folder as your assembly. This is an example of a basic Log4Net configuration block (IBatisNet.DataMapper.Test.dll.Config) that also creates a log4net.txt which contains debug information from log4net. If log4net is not producing output, check the log4net.txt file.

```
<configuration>
 <configSections>
  <sectionGroup name="iBATIS">
   <section name="logging" type="IBatisNet.Common.Logging.ConfigurationSectionHandler, IBatisNet.Common" />
  </sectionGroup>
```

```
  <section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler, log4net" />
 </configSections>
 <appSettings>
  <add key="log4net.Internal.Debug" value="true"/>
 </appSettings>
 <system.diagnostics>
 <trace autoflush="true">
   <listeners>
    <add name="textWriterTraceListener"
     type="System.Diagnostics.TextWriterTraceListener"
     initializeData="C:\\inetpub\\wwwroot\\log4net.txt" />
   </listeners>
  </trace>
 </system.diagnostics>
 <iBATIS>
  <logging>
   <logFactoryAdapter type="IBatisNet.Common.Logging.Impl.Log4NetLoggerFA, IBatisNet.Common.Logging.Log4Net">
    <arg key="configType" value="inline" />
   </logFactoryAdapter>
  </logging>
 </iBATIS>
 <log4net>
  <appender name="FileAppender" type="log4net.Appender.FileAppender">
   <file value="log.txt" />
   <appendToFile value="true" />
   <layout type="log4net.Layout.SimpleLayout" />
  </appender>
  <root>
   <level value="ALL" />
   <appender-ref ref="FileAppender" />
  </root>
 </log4net>
</configuration>
```

3.3. The Configuration File (dao.config)

The DaoManager class is responsible for the configuration of the DAO Framework by parsing a special configuration XML file. The configuration XML file specifies the following items:

- DAO contexts
- properties for configuration
- data source information
- DAO implementations for each associated DAO interface

A DAO context is a grouping of related configuration information and DAO implementations. Usually a context is associated with a single data source such as a relational database or a flat file. By configuring multiple contexts, you can easily centralize the data access configuration for multiple databases. The structure of the DAO configuration file (commonly called `dao.config` but not required) is as follows. Values that you will likely change for your application are highlighted.

Example 3.2.

Example dao.config file

```
<?xml version="1.0" encoding="utf-8"?>
<daoConfig  xmlns="http://ibatis.apache.org/dataAccess"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<providers resource="providers.config">
```

```
<!-- Example ADO.NET DAO Configuration -->
<context id="SimpleDao" default="true">
  <properties resource="properties.config"/>

  <database>
    <!-- Optional ( default ) -->
    <provider name="sqlServer1.1"/>
    <dataSource name="iBatisNet"
                connectionString="data source=${datasource};database=${database};
                                  user id=${userid};password=${password};"/>
  </database>

  <daoFactory>
    <dao interface="IBatisNet.Test.Dao.Interfaces.IAccountDao, IBatisNet.Test"
         implementation="IBatisNet.Test.Dao.Implementations.Ado.AccountDao, IBatisNet.Test"/>
  </daoFactory>
</context>

<!-- Example SQL Maps DAO Configuration -->
<context id="SqlMapDao">
  <properties resource="properties2.config"/>

  <database>
     <provider name="OleDb1.1"/>
     <dataSource name="iBatisNet"
                 connectionString="Provider=SQLOLEDB;Server=${database};database=IBatisNet;
                                   user id=${userid};password=${password};"/>
  </database>

  <daoSessionHandler id="SqlMap">
    <property name="resource" value="SqlMap.config"/>
    <!--
    url and embedded options are also available
    <property name="url" value="C:\iBATIS\IBatisNet.DataAccess.Test\bin\Debug\SqlMap.config"/>
    <property name="embedded" value="bin.Debug.SqlMap.config, IBatisNet.DataAccess.Test"/>
    -->
  </daoSessionHandler>

  <daoFactory>
    <dao interface="IBatisNet.Test.Dao.Interfaces.IAccountDao, IBatisNet.Test"
         implementation="IBatisNet.Test.Dao.Implementations.DataMapper.AccountDao, IBatisNet.Test"/>
  </daoFactory>
</context>

</daoConfig>
```

In the example above, what we end up with is two DAO contexts. DAOs are automatically aware of which context they belong to and therefore which session handler to use. Again, there can be any number of DAO contexts specified in a `dao.config` file, and generally, a context will refer to a single specific data source.

In order to manage multiple configurations for different environments (DEVT, Q/A, PROD), you can also make use of the optional `<properties>` element as shown above. This allows you to use placeholders in the place of literal value elements. If you have a "properties" file with the following values:

Example 3.3. Example properties file

```
<add key="userid" value="IBatisNet" />
<add key="password" value="test" />
<add key="database" value="iBatisNet" />
<add key="datasource" value="(local)\NetSDK" />
```

You can use placeholders defined in that properties file instead of explicit values in the `dao.config` file. For example:

```
<dataSource name="iBatisNet"
            connectionString="data source=${datasource};database=${database};
                              user id=${userid};password=${password};"/>
```

This allows for easier management of different environment configurations. Only one properties resource file may be specified per context.

Continuing with the example above, the provider and the datasource values are given within the `database` element. Following that, an optional `daoSessionHandler` may be specified. If none is specified, the default iBATIS.NET DAO SessionHandler called `SimpleDaoSessionHandler` will be used. In the example, the second context identifies the SessionHandler as `SqlMap`, meaning that the DAO implementations will be using iBATIS.NET Data Mapper SqlMaps instead of basic ADO calls.

Next in the context under `daoFactory`, is the specification of all DAO interfaces and their associated implementations. These mappings link a generic DAO interface to a concrete (specific) implementation. This is where we can see the pluggable nature of Data Access Objects. By simply replacing the implementation class for a given DAO mapping, the persistence approach taken can be completely changed (e.g. from ADO to iBATIS DataMapper SqlMaps to NHibernate).

3.4. Configuration Elements

The sections below describe the elements of the Data Access configuration document for .NET. The `dao.config` file can be validated by the `DaoConfig.xsd` schema provided with the distribution.

3.4.1. type Attributes

In general, when specifying a type attribute within a configuration file, a type attribute value must be a fully qualified type name following the format:

```
type="[namespace.class], [assembly name],
      Version=[version], Culture=[culture],
      PublicKeyToken=[public token]"
```

For example:

```
type="MyProject.Domain.LineItem, MyProject.Domain,
      Version=1.2.3300.0, Culture=neutral,
      PublicKeyToken=b03f455f11d50a3a"
```

The strongly typed name is desired, however, it is also legitimate to use the shorter style assembly type name:

```
type="MyProject.Domain.LineItem, MyProject.Domain"
```

3.4.2. The <providers> Element

Under ADO.NET, a database system is accessed through a provider. A database system can use a custom provider or a generic ODBC provider. iBATIS.NET uses a pluggable approach to installing providers. Each provider is represented by an XML descriptor element. The list of providers you might want to use can be kept in a separate XML descriptor file. The iBATIS.NET distribution includes the standard `providers.config` file with a set of thirteen prewritten provider elements:

- sqlServer1.0 - Microsoft SQL Server 7.0/2000 provider available with .NET Framework 1.0
- sqlServer1.1 -Microsoft SQL Server 7.0/2000 provider available with .NET Framework 1.1
- OleDb1.1 - OleDb provider available with .NET Framework 1.1
- Odbc1.1 - Odbc provider available with .NET Framework 1.1
- oracle9.2 - Oracle provider V9.2.0.401
- oracle10.1 - Oracle provider V10.1.0.301
- oracleClient1.0 - MS Oracle provider V1.0.5 available with .NET Framework 1.1
- ByteFx - ByteFx MySQL provider V0.7.6.15073
- MySql - MySQL provider V1.0.4.20163
- SQLite3 - SQLite.NET provider V0.21.1869.3794
- Firebird1.7 - Firebird SQL .NET provider V1.7.0.33200
- PostgreSql0.7 - Npgsql provider V0.7.0.0
- iDb2.10 - IBM DB2 iSeries provider V10.0.0.0

The `providers.config` file can be found under `\source\IBatisNet.DataAccess.Test\bin\Debug` in the iBATIS.NET source distribution or in the root folder of the .NET DataAccess binary distribution.

A provider may require libraries that you do not have installed,. Therefore, the provider element has an "enabled" attribute that allows you to disable unused providers. One provider can also be marked as the "default" and will be used if another is not specified by your configuration.

The standard `providers.config` file has sqlServer1.1 set as the default and the sqlServer1.0 provider disabled. Aside from sqlServer1.1, OleDb1.1, and Odbc1.1, all other providers are disabled by default. Remember to set the "enabled" attribute to "true" for the provider that you will be using.

Important

ByteFx is the recommended provider if you are using MySQL. You may download ByteFx from the MySQLNet SourceForge site (http://sf.net/projects/mysqlnet/). If the ByteFx license is acceptable to you, you may install it as a reference within your application and enable the ByteFx provider.

Tip

Be sure to review the `providers.config` file and confirm that the provider you intend to use is enabled! (Set the enabled attribute to true.)

Table 3.5. Expected default locations of the providers.config file

Windows, Library, or Test projects (using NUnit or equivalent)	With the assembly (.dll) files with the `app.config` file
Web projects	In the project base directory, with the `web.config` file

To use the file, you can copy it into your project at the expected default location, give a path to the file relative to the project root directory, specify a url (absolute path) to its location, or make it an embedded resource of your project. If you copy the file into the expected default location, the <providers> element is not required in your dao.config file.

3.4.2.1. <providers> attributes

The <providers> element can accept one of the following attributes to specify the location of the providers.config file.

Table 3.6. Attributes of the <providers> element

Attribute	Description
resource	Specify the file to be loaded from a relative path from the project root directory. Since the root directory is different depending on the project type, it is best to use a properties variable to indicate the relative path. Having that variable defined in a properties file makes it easy to change the path to all your DataAccess configuration resources in one location. `resource="${root}providers.config"`

url	Specify the providers.config to be loaded through an absolute path.
	`url="c:\Web\MyApp\Resources\providers.config"`
	`or`
	`url="file://c:\Web\MyApp\Resources\providers.config"`

embedded	Specify the providers.config file to be loaded as an embedded resource in an assembly. Syntax for the embedded attribute is '*[extendednamespace.]filename, the name of the assembly which contains the embedded resource*' `embedded="Resources.providers.config, MyApp.Data"`

3.4.3. The <context> Element

A DAO context is a grouping of related configuration information and DAO implementations for a specific database and provider.

Table 3.7. Attributes of the <context> element

Attribute	Description
id	Context id
default	Global setting to indicate if this is the default context

3.4.3.1. The <properties> Element

Sometimes the values we use in an XML configuration file occur in more than one element. Often, there are values that change when we move the application from one server to another. To help you manage configuration values, you can specify a standard properties file (with name=value entries) as part of a Data Access configuration. Each named value in the properties file becomes a *shell* variable that can be used throughout the Data Access configuration.

Properties are handy during building, testing, and deployment. Additionally, properties make it easy to reconfigure your application for multiple environments or to use automated tools for configuration such as NAnt.

The <properties> element can only accept one of the following attributes to specify the location of the properties file.

Table 3.8. Attributes of the <properties> element

Attribute	Description
resource	Specify the properties file name files to be loaded from the root directory of the application (relative path) `resource="properties.config"`

url	Specify the properties file to be loaded through an absolute path
	`url="c:\Web\MyApp\Resources\properties.config"`
	`or`
	`url="file://c:\Web\MyApp\Resources\properties.config"`

embedded	Specify the properties file to be loaded as an embedded resource in an assembly. Syntax for the embedded attribute is '*[extendednamespace.]filename, the name of the assembly which contains the embedded resource*' `embedded="Resources.properties.config, MyApp.Data"`

3.4.3.1.1. <property> element and attributes

You can also specify more than one properties file or add property keys and values directly into your `dao.config` file by using <property> elements. For example:

```
<properties>
 <property resource="myProperties.config"/>
 <property resource="anotherProperties.config"/>
 <property key="host" value="ibatis.com" />
</properties>
```

Table 3.9. Attributes of the <property> element

Attribute	Description
resource	Specify the properties file to be loaded from the root directory of the application `resource="properties.config"`

url	Specify the properties file to be loaded through an absolute path.
	`url="c:\Web\MyApp\Resources\properties.config"`
	or
	`url="file://c:\Web\MyApp\Resources\properties.config"`

embedded	Specify the properties file to be loaded as an embedded resource in an assembly. Syntax for the embedded attribute is '*[extendednamespace.]filename, the name of the assembly which contains the embedded resource*' `embedded="Resources.properties.config, MyApp.Data"`
key	Defines a property key (variable) name `key="username"`
value	Defines a value that will be used by the framework in place of the the specified property key/variable `value="mydbuser"`

3.4.3.2. The <database> Element

The <database> element encloses elements that configure the database system for use by the framework. These are the <provider> and <datasource> elements.

3.4.3.2.1. The <provider> Element

The <provider> element specifies a database provider from the `providers.config` file.

If the default provider is being used, the <provider> element is optional. Or, if several providers are available, one may be selected using the provider element without modifying the `providers.config` file.

```
<provider name="OleDb1.1" />
```

Table 3.10. Attributes of the <provider> element

Attribute	Description
name	Name used to uniquely identify the provider

3.4.3.2.2. The <dataSource> Element

The <datasource> element specifies the connection string for a specific database and provider.

Table 3.11. Attributes of the <dataSource> element

Attribute	Description
name	Name used to identify the data source
connectionString	The connection string to the database

3.4.3.3. The <daoSessionHandler> Element

The <daoSessionHandler> element specifies the component that manages transaction and connections for a session.

Table 3.12. Attributes of the <daoSessionHandler> element

Attribute	Description
id	Id used to uniquely identify the handler

3.4.3.3.1. The <property> Element

Some <daoSessionHandler> implementations such as iBATIS DataMapper SqlMaps and NHibernate need additional information for configuration. The <property> element is used to specify this information.

For iBATIS DataMapper SqlMaps, the method for locating and loading the `SqlMap.config` file is used in the "name" attribute. Depending on the chosen method, the corresponding filepath or fully-qualified resource and assembly name is used for the "value" attribute.

Table 3.13. Attributes of the <property> element for use with iBATIS DataMapper SqlMaps

Attribute	Description
name	Indicates the method used to locate and load the `SqlMap.config` file. The available options are "resource", "url", and "embedded" (as previously described for loading the providers and properties configuration files). `name="resource"`
value	Used to specify the filename, fullpath, or fully qualified resource and assembly name for the

Attribute	Description
	`SqlMap.config` file. `value="MySqlMap.config"`

For NHibernate, the "name" and "value" attributes are used to specify various NHibernate configuration details such as "hibernate.dialect", "hibernate.connection.provider", and "mapping".

Table 3.14. Attributes of the <property> element for use with NHibernate

Attribute	Description
name	The name of the property to configure. Example: `name="hibernate.dialect"`
value	The value of the property configuration. Example: `value="NHibernate.Dialect.MsSql2000Dialect"`

3.4.3.4. The <daoFactory> Element

The <daoFactory> element is a grouping of Dao interfaces and implementations.

3.4.3.4.1. The <dao> Element

The <dao> element specifies a Dao interface and one implementation of that interface.

Table 3.15. Attributes of the <dao> element

Attribute	Description
interface	The interface that the Dao must implement. Structure value must be : namespace-qualified name of the class, followed by a comma, followed by (at a bare minimum) the name of the assembly that contains the class. Example: `interface= "IBatisNet.Test.Dao.Interfaces.IAccountDao, IBatisNet.Test"`
implementation	An implementation of the Dao interface. Structure value must be : namespace-qualified name of the class, followed by a comma, followed by (at a bare minimum) the name of the assembly that contains the class. Example: `implementation= "IBatisNet.Test.Dao.Implementations.Ado.AccountDao, IBatisNet.Test"`

Chapter 4. Session Handler Implementations and Configuration

4.1. Overview

A Session Handler implementation is a component that manages transaction and connections for a session. There are currently three implementations of session handlers that come with the framework: Simple, SqlMap, and NHibernate.

4.2. Simple DAO Session Handler (Default) Example Configuration

This implementation uses ADO.NET to provide a connection and transaction via the ADO.NET API.

Example 4.1. Example Simple DAO Session Handler configuration

```
<context id="SimpleDao" default="true">
  <properties resource="database.config"/>

  <database>
    <!-- Optional ( default ) -->
    <provider name="sqlServer1.1"/>
    <dataSource name="iBatisNet"
                connectionString="data source=${datasource};database=${database};
                user id=${userid};password=${password}"/>
  </database>

  <!-- Element daoSessionHandler (ADONET) is Optional ( default ) -->

  <daoFactory>
    <dao interface="IBatisNet.Test.Dao.Interfaces.IAccountDao, IBatisNet.Test"
         implementation="IBatisNet.Test.Dao.Implementations.Ado.AccountDao, IBatisNet.Test"/>
  </daoFactory>
</context>
```

4.3. SqlMap DAO Session Handler Example Configuration

The SqlMap session handler implementation wraps iBATIS Data Mapper SqlMapper session management services for simple use via the DAO framework. All you need to specify is the SqlMap configuration file.

Example 4.2. Example SqlMap DAO Session Handler configuration

```
<context id="SqlMapDao">
  <properties resource="database.config"/>

  <database>
    <provider name="OleDb1.1"/>
    <dataSource name="iBatisNet"
                connectionString="Provider=SQLOLEDB;Server=${database};database=IBatisNet;
```

```
                user id=${userid};password=${password};"/>
  </database>

  <daoSessionHandler id="SqlMap">
    <property name="resource" value="SqlMap.config"/>
  </daoSessionHandler>

  <daoFactory>
    <dao interface="IBatisNet.Test.Dao.Interfaces.IAccountDao, IBatisNet.Test"
         implementation="IBatisNet.Test.Dao.Implementations.DataMapper.AccountDao, IBatisNet.Test"/>
 </daoFactory>
</context>
```

4.4. NHibernate DAO Session Handler Example Configuration

Similar to the SqlMap implementation, the NHibernate session handler implementation wraps the NHibernate session management services for simple use via the DAO framework. Basically, the properties specified in the configuration are the same as those that would normally be specified in an NHibernate configuration section. Here is an example configuration:

Example 4.3. Example NHibernate DAO Session Handler configuration

```
<context id="NHibernateDao">
  <properties resource="database.config"/>

  <database>
    <provider name="OleDb1.1"/>
    <dataSource name="iBatisNet"
                connectionString="Provider=SQLOLEDB;Server=${database};database=IBatisNet;
                user id=${userid};password=${password};"/>
  </database>

  <daoSessionHandler id="NHibernate">
    <property name="hibernate.dialect"
              value="NHibernate.Dialect.MsSql2000Dialect"/>
    <property name="hibernate.connection.provider"
              value="NHibernate.Connection.DriverConnectionProvider"/>
    <property name="hibernate.connection.driver_class"
              value="NHibernate.Driver.SqlClientDriver"/>
    <property name="mapping" value="IBatisNet.Test"/>
    <property name="show_sql" value="false"/>
    <property name="use_outer_join" value="true"/>
  </daoSessionHandler>

  <daoFactory>
    <dao interface="IBatisNet.Test.Dao.Interfaces.IAccountDao, IBatisNet.Test"
         implementation="IBatisNet.Test.Dao.Implementations.DataMapper.AccountDao, IBatisNet.Test"/>
  </daoFactory>
</context>
```

Chapter 5. DAO Programming

5.1. Overview

The iBATIS.NET Data Access Objects framework has a number of goals. First, it attempts to hide the details of your persistence layer. This includes hiding all interface, implementation, and exception details of your persistence solution. For example: if your application is using raw ADO, the DAO framework will hide classes like `DataReader`, `DataAdapter`, `Connection`, and `Command`. Similarly, if your application is using the NHibernate object persistence library, the DAO framework will hide classes like `Configuration`, `SessionFactory`, `Session`, and `HibernateException`. All of these implementation details will be hidden behind a consistent DAO interface layer. Furthermore, the number of different data sources that are being used can be hidden from the view of the application.

The second goal of the framework is to simplify the persistence programming model while keeping it more homogeneous at the same time. Different persistence solutions have different programming semantics and behavior. The DAO framework attempts to hide this as much as possible, allowing the service and domain layer of your application to be written in a uniform fashion.

The `DomDaoManagerBuilder` class is responsible for the proper configuration of the DAO framework (via `dao.config` as described in previous sections). After configuration, another class called the `DaoManager` provides the core DAO API for each of your data access contexts. In particular, the `DaoManager` provides methods that allow you to access connection, transaction, and DAO instances.

Note

In prior versions of the DAO framework, the `DaoManager` handled configuration responsibilities. This has been superceded by a new configuration API found within the `DomDaoManagerBuilder` class. Old configuration method signatures have remained the same, but new configuration methods have been added for more flexibility. These new methods support the loading of configuration information through a `Stream`, `Uri`, `FileInfo`, or `XmlDocument` instance.

5.2. Configuration with the DomDaoManagerBuilder

iBATIS offers you a plethora of options for loading your `dao.config` file, such as loading it through a file path, `Stream`, `Uri`, `FileInfo`, or `XmlDocument`. All of these methods are available through the `DomDaoManagerBuilder` API for creating `DaoManager` instances.

The basic `DomDaoManagerBuilder.Configure()` call will look for a file named `dao.config` in your application's root directory. This directory's location differs by project type but is normally the directory where you place your web.config or app.config file.

Example 5.1. Basic Configuration Call

```
DomDaoManagerBuilder builder = new DomDaoManagerBuilder();
builder.Configure();
IDaoManager daoManager = DaoManager.GetInstance("SqlMapDaoContext");
```

If you have named your configuration file something other than `dao.config` or if you have located

your configuration file in a directory other than the application root directory, you can also pass in a relative or absolute file path to the `Configure` method.

Example 5.2. Configuration through an absolute or relative file path

```
/* Configure from a file path.
   Uses a relative resource path from your application root
   or an absolute file path such as "file://c:\dir\a.config" */
DomDaoManagerBuilder builder = new DomDaoManagerBuilder();
builder.Configure(strPath);
IDaoManager daoManager = DaoManager.GetInstance("AnotherContext");
```

Tip

Since the application root directory location differs by project type (Windows, Web, or library), you can use an AppSettings key for defining a relative path to your `dao.config` file. Having this key defined makes it easy to change the path without having to recompile your code:

```
builder.Configure(
        ConfigurationSettings.AppSettings["rootPath"]+"dao.config");
```

Aside from using a simple string filepath, you can also pass in a `FileInfo` or `Uri` instance for the `DomDaoManagerBuilder` to use in locating your `dao.config` file.

Example 5.3. Configuration with a FileInfo or Uri instance

```
/* Configure with FileInfo. */
FileInfo aFileInfo = someSupportClass.GetDynamicFileInfo();
DomDaoManagerBuilder builder = new DomDaoManagerBuilder();
builder.Configure(aFileInfo);
IDaoManager daoManager = DaoManager.GetInstance("NHibernateContext");

/* Configure through a Uri. */
Uri aUri = someSupportClass.GetDynamicUri();
DomDaoManagerBuilder builder = new DomDaoManagerBuilder();
builder.Configure(aUri);
IDaoManager daoManager = DaoManager.GetInstance("SimpleDao");
```

If you find that you already have loaded your DAO configuration information as an `XmlDocument` or `Stream` instance within your application, the `DomDaoManagerBuilder` provides `Configure` overloads for those types as well.

Example 5.4. Configuration with an XmlDocument or Stream

```
/* Configure with an XmlDocument */
XmlDocument anXmlDoc = someSupportClass.GetDynamicXmlDocument();
DomDaoManagerBuilder builder = new DomDaoManagerBuilder();
builder.Configure(anXmlDoc);
IDaoManager daoManager = DaoManager.GetInstance("Petstore");

/* Configure from a stream. */
Stream aStream = someSupportClass.GetDynamicStream();
DomDaoManagerBuilder builder = new DomDaoManagerBuilder();
builder.Configure(aStream);
IDaoManager daoManager = DaoManager.GetInstance("AnotherPetstore");
```

In addition to the straightforward `Configure` methods, the `DomDaoManagerBuilder` provides `ConfigureAndWatch` methods that can be used to monitor changes to the configuration files so that `DaoManagers` can be reconfigured and reloaded on the fly. To use this functionality, you application will need to pass a `ConfigureHandler` (callback delegate) to the `DomDaoManagerBuilder` so that it knows the method for resetting your application's `DaoManager` instances.

Since the configuration files need to be watched for changes, your `dao.config` file must be accessible through the file system. This means that configuration is limited to the three methods shown below.

Example 5.5. DomDaoManagerBuilder ConfigureAndWatch methods

```
/* Configure and monitor the configuration file for modifications
   and automatically reconfigure DaoManagers.
   This basic ConfigureAndWatch method looks for a file with the
   default name of dao.config in the application root directory. */
public void ConfigureAndWatch(ConfigureHandler configureDelegate)

/* Configure and monitor the configuration file for modifications
   and automatically reconfigure DaoManagers.
   Uses a relative path from your application root
   or an absolute file path such as "file:\\c:\dir\a.config" */
public void ConfigureAndWatch( string resource, ConfigureHandler configureDelegate )

/* Configure and monitor the configuration file for modifications
   and automatically reconfigure DaoManagers.
   Uses a FileInfo instance for your config file. */
public void ConfigureAndWatch( FileInfo resource, ConfigureHandler configureDelegate )
```

5.3. Contexts, the DaoManager, and Session Handlers

The `DaoManager` instance that is built from a `dao.config` file is aware of all of the contexts contained within the configuration file. The context basically bundles DAO implementations together with a session handler. The `DaoManager` knows which DAOs and session handler belong to which context. When you request a DAO instance from the `DaoManager`, the proper session handler will be provided with it. Therefore, there is no need to access the context or session handler directly. Your DAO knows how it works.

5.4. Getting a Data Access Object

Once you get a `DaoManager` instance, you can use it to retrieve the DAO implementation by type (as specified in the `dao.config` file in the `daoFactory` section). Getting a DataAccess object is simply a matter of using the `GetDao(type)` method of a `DaoManager` instance (or the more concise indexer method `DaoManager[type]`) For example:

```
[C#]
IDaoManager daoManager = DaoManager.GetInstance("daoContextName");
// gets the dao of type 'ICategoryDao'
ICategoryDao categoryDao = daoManager.GetDao(typeof(ICategoryDao)) as ICategoryDao;
// same thing, just using the indexer
IProductDao productDao = daoManager[typeof(IProductDao)] as IProductDao;
```

5.5. Working with Connection and Transactions

The `DaoManager` provides methods for working with connection and transaction. These methods allow you to demarcate connection/transactions and avoid having to pass transaction objects, like `IDbTransaction`, around to all of your DAOs.

Example 5.6. Example DAO transaction management

```
[C#]
Product p1 = new Product();
Product p2 = new Product();
Category c 1= new Category()
c1.Add(p1);
c2.Add(p2);

IDaoManager daoManager = DaoManager.GetInstance("PetStore");
ICategoryDao categoryDao = daoManager[typeof(ICategoryDao)] as ICategoryDao;
IProductDao productDao = daoManager[typeof(IProductDao)] as IProductDao;

try {
      daoManager.BeginTransaction();
      productDao.Insert(p1);
      productDao.Insert(p2);
      categoryDao.Insert(c1);
      daoManager.CommitTransaction();
}
catch {
        daoManager.RollBackTransaction();
}
```

Calling `BeginTransaction()` lets the `DaoManager` know that you are interested in managing transactions programmatically. It is very important that you guarantee a call to `RollBackTransaction()` if you've called `BeginTransaction()`, which is why it is within the `catch` block. The call to `RollBackTransaction()` will rollback any changes you've made in case an exception is thrown before the call to `CommitTransaction()`. Be aware that if an exception occurs before `BeginTransaction()` completes and is caught in the `catch` block, RollBackTransaction() will also throw another exception since a transaction was not started.

When you deal with a connection and transaction, you can also use the `using` syntax as in the examples below.

```
[C#]
IDaoManager daoManager = DaoManager.GetInstance("PetStore");
IAccountDao accountDao = daoManager[typeof(IAccountDao)] as IAccountDao;

using ( IDalSession session = daoManager.OpenConnection() )
{
   Account account = NewAccount();
   accountDao.Create(account);
}

using ( IDalSession session = daoManager.BeginTransaction() )
{
   Account account = NewAccount();
   Account account2 = accountDao.GetAccountById(1);
   account2.EmailAddress = "someotherAddress@somewhere.com";

   accountDao.Create(account);
   accountDao.Update(account2);

   session.Complete(); // Commit
}
```

5.6. "AutoConnection"-Behavior

In addition to programmatically demarcating a connection, you can allow the `DaoManager` to automatically Open and Close a connection for you. You don't need to do anything special to use the autoconnection behavior, just don't call `OpenConnection()`.

Example 5.7. Example AutoConnection

```
[C#]
// Open/close first Connection
Product product = productDao.GetProduct (5);
product.Description = "New description.";
// Open/close second Connection
productDao.UpdateProduct(product);
// Open/close third Connection
product = productDao.GetProduct (5);
```

Note

If the `UpdateProduct()` method contained more than a single update, you would need to use the `BeginTransaction()` method!

5.7. Distributed transactions

Distributed transactions are transactions that can span multiple resource managers, such as SQL Server and Oracle, and reconcile transactions among them.

iBATIS.NET introduces a new `TransactionScope` class mimicking the new TransactionScope found in the `System.Transactions` namespace (.NET Framework 2.0). This class supports MSMQ, ADO.NET, SqlServer, and DTC transaction models. This is a simple managed interface to COM+'s SWC (Services Without Components) Transactions. It can be used only by developers using .NET 1.1 and Windows XP SP2 or Windows Server 2003 since it implements distributed transactional support using the `ServiceDomain` class.

Usage is simple, as seen in the following example where a code block is made transactional à la Indigo (moving to Indigo will be easier since it is the same API):

```
[C#]
using IBatisNet.Common.Transaction;

using (TransactionScope tx = new TransactionScope())
{
   daoManager.OpenConnection();
   // Transaction will be automatically associated with it
   account = accountDao.GetAccountById(1001);
   account.FirstName = "Gilles";
   accountDao.Update(account);
   daoManager.CloseConnection();

   daoManager2.OpenConnection();
   // Transaction will be automatically associated with it
   joeCool = userDao.Load("joe_cool");
   joeCool.LastLogon = stamp;
   daoManager2.CloseConnection();

   tx.Complete(); // Commit
}
```

It is important to make sure that each instance of this class gets `Close()`'d. The easiest way to ensure that each instance is closed is with the `using` statement in C#. When `using` calls `Dispose` on the transaction scope at the end of the `using` code block, the *ambient* transaction will be commited only if the `Complete()` method has been called.

Note

This class does not support a nested transaction scope with different transaction options.
The next section provides some tips on how to write a DAO like the AccountDao shown in the previous example.

Chapter 6. Implementing the DAO Interface (Creating Your Data Access Objects)

6.1. Interface

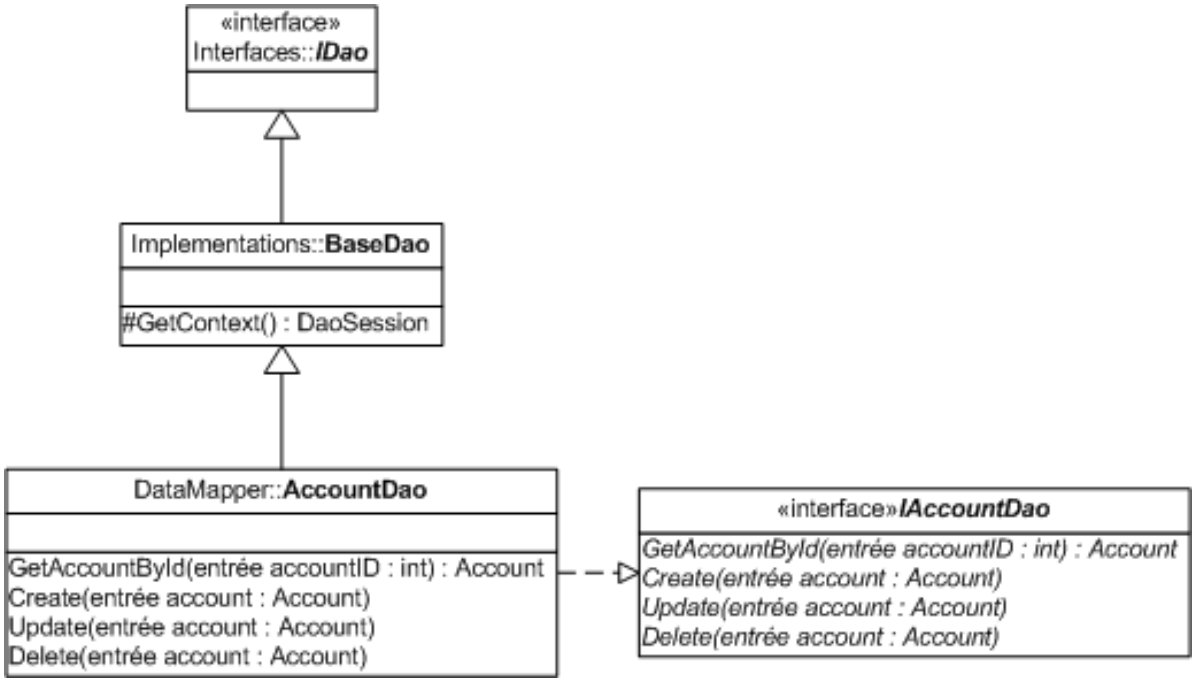
The `IDao` interface is simple and flexible because it does not declare any methods. It is intended to act as a marker interface only (as per the Marker Interface pattern –Grand98). In other words, by extending the `IDao` interface, all that is really achieved for the class that implements your DAO interface is the ability to be instantiated and managed by the `DaoManager`. There are no limitations to the methods that you use in your DAO interfaces. It is recommended for consistency that DAO implementations only throw exceptions of type `DataAccessException` which is an `ApplicationException`. This helps to hide the implementation details of your persistence solution.

An example of a DAO interface is:

```
[C#]
public interface IAccountDao : IDao {
    Account GetAccountById(int accountID);
    void Create(Account account);
    // DAO Framework code may throw DataAccessException
    void Update(Account account);
    void Delete(Account account);
}
```

6.2. DAO Design Considerations

When implementing your DAO classes for your DAO interfaces, it is recommended to use a design that includes the DAO interface, an abstract (base) class, and a concrete class. The advantage to having the base class is that it can contain common methods that simplify the usage of your persistence approach (wrapping up exception handling, transaction acquisition, etc.).



DAO Implementation Diagram

Notice that the `BaseDao` implements the `IDao` marker and leaves the `IAccountDao` without it. Furthermore, the concrete `AccountDao` implements both by implementing the `IAccountDao` and extending the `BaseDao`. This allows your Account interface API to remain free of the `IDao` marker.

6.3. An Example BaseDao

In case you are still wondering what the `IDao` interface looks like in code, here it is:

```
using System;

namespace IBatisNet.DataAccess.Interfaces {
 public interface IDao {

 }
}
```

It may not look like much, but that's the point! The iBATIS DAO framework uses this interface to interact with your application DAOs since they are marked as implementing it. As mentioned earlier, you don't have to force your DAO interfaces to inherit `IDao` if you create a class like `BaseDao`.

```
using System;
using IBatisNet.Common;
using IBatisNet.DataAccess; //DaoManager, DaoSession
using IBatisNet.DataAccess.Interfaces; //IDao

namespace IBatisNet.Test.Implementations {
 public class BaseDao: IDao {
  protected DaoSession GetContext() {
   IDaoManager daoManager = DaoManager.GetInstance(this);
   return (daoManager.LocalDaoSession as DaoSession);
  }
 }
}
```

The `BaseDao` class implements the `IDao` marker interface and defines a protected `GetContext` method that will be used by DAO classes that inherit from `BaseDao`. By calling `GetContext`, a DAO class gets a `DaoSession` that is managed by the framework's `DaoManager`. The `DaoSession` abstract class uses the `IDalSession` interface to provide an API to manage transactions and to access ADO.NET objects during the session.

```
namespace IBatisNet.DataAccess {
 public abstract class DaoSession : IDalSession {

  // Constructor
  public DaoSession(DaoManager daoManager)

  // Properties
  public abstract DataSource DataSource
  public abstract IDbConnection Connection
  public abstract IDbTransaction Transaction

  // Methods

  // Complete (commit) a transaction
  public abstract void Complete();

  // Open a database connection.
  public abstract void OpenConnection();

  // Close the connection
  public abstract void CloseConnection();

  // Begin a transaction.
  public abstract void BeginTransaction();
```

```
  // Begin a database transaction
  public abstract void BeginTransaction(bool openConnection);

  // Begin a transaction at the data source
  // with the specified IsolationLevel value.
  public abstract void BeginTransaction(IsolationLevel isolationLevel);

  // Begin a transaction on the current connection
  // with the specified IsolationLevel value.
  public abstract void BeginTransaction(bool openConnection, IsolationLevel isolationLevel);

  // Commit the database transaction.
  // and close the connection.
  public abstract void CommitTransaction();

  // Commit the database transaction.
  public abstract void CommitTransaction(bool closeConnection);

  // Rollback a transaction from a pending state
  // and close the connection.
  public abstract void RollBackTransaction();

  // Rolls back a transaction from a pending state.
  public abstract void RollBackTransaction(bool closeConnection);

  public abstract IDbCommand CreateCommand(CommandType commandType);
  public abstract IDataParameter CreateDataParameter();
  public abstract IDbDataAdapter CreateDataAdapter();
  public abstract IDbDataAdapter CreateDataAdapter(IDbCommand command);
}
```

The ADO.NET access provided by the `DaoSession` object gives you the flexibility to sprinkle in ADO-based DAO classes with iBATIS Data Mapper-based DAOs and NHibernate-based DAOs for those special circumstances that often pop up. And you get all this without losing the consistent DAO interface layer or exposing your persistence implementation details.

6.4. Consistency and Hidden Implementations

Let's take a look at a consistency means when using a DAO layer. We will use the previously shown IAccountDao interface to create two example implementations, the first using ADO.NET and the second using the iBATIS Data Mapper framework. After that, we will go through the steps to creating and using an NHibernate-based DAO to use with the framework. (Complete code examples can be found in the IBatisNet-Test project found in the distribution)

6.4.1. ADO.NET AccountDao with the Simple DAO Session Handler

First, we declare that we'll be using types from the `IBatisNet.DataAccess`, `IBatisNet.DataAccess.Exceptions`, and `IBatisNet.DataAccess.Interfaces` namespaces with our DAO. These provide the `DaoSession` and `DataAccessException` classes as well as the `IDao` interface.

```
using System;
using System.Collections;
using System.Data;

using IBatisNet.DataAccess; // DaoSession
using IBatisNet.DataAccess.Exceptions; // DataAccessException
using IBatisNet.DataAccess.Interfaces; // IDao

using IBatisNet.Test.Dao.Interfaces; // IAccountDao
using IBatisNet.Test.Implementations; // BaseDao
using IBatisNet.Test.Domain; // Account
```

The DAO will inherit from our `BaseDao` class and implement our `IAccountDao` interface. SQL queries are written to allow the DAO to be used with the SqlClient provider for MS SQL Server. Additionally, command parameter names are specified.

```
namespace IBatisNet.Test.Dao.Implementations.Ado {

 public class AccountDao : BaseDao, IAccountDao {
  private const string SELECT_ACCOUNT_BY_ID ="select Account_ID, ...";
  private const string INSERT_ACCOUNT = "insert into Accounts (Account_ID, ...";
  private const string UPDATE_ACCOUNT = "update Accounts set Account_FirstName = @Account_FirstName, ...";
  private const string DELETE_ACCOUNT = "delete from Accounts where Account_ID = @Account_ID";

  private const string PARAM_ACCOUNT_ID = "@Account_ID";
  private const string PARAM_ACCOUNT_FIRSTNAME = "@Account_FirstName";
  private const string PARAM_ACCOUNT_LASTNAME = "@Account_LastName";
  private const string PARAM_ACCOUNT_EMAIL = "@Account_Email";
```

Now, let's let's see how this DAO's `Delete(Account account)` method uses the framework.

```
  public void Delete(Account account) {

   IDbCommand command = null;
   DaoSession daoSession = null;

   daoSession = this.GetContext();
   command = daoSession.CreateCommand(CommandType.Text);

   try {
    command.CommandText = DELETE_ACCOUNT;
    IDbDataParameter sqlParameter = command.CreateParameter();
    sqlParameter.ParameterName = PARAM_ACCOUNT_ID;
    sqlParameter.Value = account.Id;
    command.Parameters.Add(  sqlParameter );
    command.ExecuteNonQuery();
    command.Parameters.Clear();
   }
   catch (System.Exception e) {
    throw new
       DataAccessException("Error executing AccountDao.Delete. Cause :"
                           + e.Message, e);
   }
   finally {
    command.Dispose();
   }
  }
```

The DAO obtains an instance of `DaoSession` by calling the `GetContext` method inherited from `BaseDao`. Next, it uses the simple `DaoSession` to get a `command` object to execute the DELETE_ACCOUNT SQL statement. Lastly, if an exception is thrown, the DAO makes sure it is a `DataAccessException`. Pretty straightforward.

6.4.2. Data Mapper AccountDao with the SqlMap DAO Session Handler

The Data Mapper `AccountDao` requires the use of the iBATIS Data Mapper framework. For this reason, the DAO framework provides a `SqlMapDaoSession` handler for Data Mapper-based DAOs.

```
using System;

using IBatisNet.DataMapper; // SqlMapper
using IBatisNet.DataAccess.DaoSessionHandlers; // SqlMapDaoSession
using IBatisNet.DataAccess.Exceptions; // DataAccessException

using IBatisNet.Test.Dao.Interfaces; // IAccountDao
using IBatisNet.Test.Implementations; // BaseDao

using IBatisNet.Test.Domain; // Account
```

As with the ADO.NET implementation, the class will inherit from our `BaseDao` class and implement our `IAccountDao` interface. Since our queries are handled by our work with the Data Mapper framework, we don't need to define any SQL queries in this DAO.

```
namespace IBatisNet.Test.Dao.Implementations.DataMapper {
 public class AccountDao : BaseDao, IAccountDao {
```

Here's how the `Delete(Account account)` method is implemented in conjunction with the Data Mapper framework's `SqlMapper` (see the iBATIS Data Mapper Guide for details on using the `SqlMapper`):

```
  public void Delete(Account account) {
   SqlMapDaoSession sqlMapDaoSession = null;

   try {
    sqlMapDaoSession = (SqlMapDaoSession)this.GetContext();
    SqlMapper sqlMap = sqlMapDaoSession.SqlMap;
    sqlMap.Delete("DeleteAccount", account);
   }
   catch(DataAccessException ex) {
    throw new DataAccessException("Error executing AccountDao.Delete. Cause :"
                                  + ex.Message,ex);
   }
  }
```

In this example, the DAO obtains an instance of the Data Mapper framework's `SqlMapper` through the `SqlMapDaoSession` object. Next, the DAO passes in the appropriate mapped statement name to the `SqlMapper` to delete an `Account`. If an exception is thrown, the DAO makes sure it is a `DataAccessException`. Still straightforward!

6.4.3. Using the AccountDao with the DaoManager

After configuring the framework's `DaoManager` for either the ADO.NET or Data Mapper `AccountDao` that we just created (see the Configuration section), we can now look at how to use them.

```
using System;

using IBatisNet.DataAccess;

using IBatisNet.Test.Dao.Interfaces;
using IBatisNet.Test.Domain;

namespace Examples.Services {

 public class AccountService {

  protected static IDaoManager daoManager = null;

  static AccountService() {
   daoManager = DaoManager.GetInstance();
  }

  public void someMethod () {
   IAccountDao accountDao = (IAccountDao)daoManager[typeof(IAccountDao)];
   Account account = NewAccount(); // get an example Account for us to use
   daoManager.OpenConnection();
   accountDao.Create(account);
   account = accountDao.GetAccountById(1001);
   accountDao.Delete(account);
   daoManager.CloseConnection();
  }
 }
}
```

We get an instance of the `AccountDao` from the `DaoManager` by requesting an instance of the `IAccountDao` type. There is no indication of whether the returned `accountDao` is the ADO.NET or Data Mapper implementation. If we needed to for one reason or another, we could easily change the configuration to switch between the DAO implementations (use the DAO configuration file to switch database platforms, run test DAOs, etc). After getting the `accountDao`, we open a connection to the data source and call the DAO method(s) that we need to execute. A consistent interface for either ADO.NET or Data Mapper DAOs with no persistence implementation details found in our code!

6.4.4. NHibernate DAO with the NHibernate DAO Session Handler

Using the DAO framework with the NHibernate object persistence library is as easy as using the framework with raw ADO.NET or iBATIS Data Maps. To demonstrate, we'll examine an NHibernate-based `UserDao` class.

Like the `SqlMapDaoSession` handler, the `NHibernateDaoSession` handler is found in the `IBatisNet.DataAccess.Extensions.DaoSessionHandlers` namespace.

```
using System;

using NHibernate;

using IBatisNet.DataAccess.Extensions.DaoSessionHandlers; // NHibernateDaoSession
using IBatisNet.DataAccess.Exceptions; // DataAccessException

using IBatisNet.Test.Dao.Interfaces; // IUserDao
using IBatisNet.Test.Implementations; // BaseDao

using IBatisNet.Test.Domain; // User
```

Our example NHibernate `UserDao` class will inherit from our `BaseDao` class and implement our `IUserDao` interface.

```
namespace IBatisNet.Test.Dao.Implementations.NHibernate {
 public class UserDao : BaseDao, IUserDao
```

Here's how two methods are implemented in conjunction with an NHibernate `Session`:

```
  public void Create(User user) {
   NHibernateDaoSession nHibernateDaoSession = null;

   try {
    nHibernateDaoSession = (NHibernateDaoSession)this.GetContext();
    ISession session = nHibernateDaoSession.Session;
    session.Save( user );
   }
   catch(Exception ex) {
    throw new DataAccessException("Error executing UserDao.Create. Cause :"
                                  + ex.Message,ex);
   }
  }

  public User Load(string id) {
   NHibernateDaoSession nHibernateDaoSession = null;
   User user = null;

   try {
    nHibernateDaoSession = (NHibernateDaoSession)this.GetContext();
    ISession session = nHibernateDaoSession.Session;
    user = session.Load(typeof(User),id) as User;
   }
   catch(Exception ex) {
    throw new DataAccessException("Error executing UserDao.Create. Cause :"
                                  + ex.Message,ex);
```

```
  }

  return user;
 }
```

The DAO obtains an NHibernate `Session` through the `NHibernateDaoSession` object and executes the appropriate `Session` methods. If an exception is thrown, the DAO makes sure it is a `DataAccessException`.

Chapter 7. Examples

7.1. NPetshop Application

For a complete example of using the DAO Framework, please visit http://ibatis.apache.org/ and download the NPetshop demo application. Have fun!

Appendix A. iBATIS.NET's `DaoConfig.xsd`

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema
targetNamespace="http://ibatis.apache.org/dataAccess"
elementFormDefault="qualified"
xmlns:mstns="http://tempuri.org/XMLSchema.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://ibatis.apache.org/dataAccess"
xmlns:vs="http://schemas.microsoft.com/Visual-Studio-Intellisense"
vs:friendlyname="iBATIS.NET DataAccess file Configuration Schema"
vs:ishtmlschema="false"
vs:iscasesensitive="true"
vs:requireattributequotes="true"
vs:defaultnamespacequalifier=""
vs:defaultnsprefix="" >

        <xs:element name="providers">
                <xs:complexType>
                        <xs:attribute name="resource" type="xs:string"/>
                        <xs:attribute name="url" type="xs:string"/>
                        <xs:attribute name="embedded" type="xs:string"/>
                </xs:complexType>
        </xs:element>
        <xs:element name="context">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="properties" minOccurs="0"/>
                                <xs:element ref="database"/>
                                <xs:element ref="daoSessionHandler" minOccurs="0"/>
                                <xs:element ref="daoFactory"/>
                        </xs:sequence>
                        <xs:attribute name="id" type="xs:string" use="required"/>
                        <xs:attribute name="default" type="xs:boolean"/>
                </xs:complexType>
        </xs:element>
        <xs:element name="dao">
                <xs:complexType>
                        <xs:attribute name="interface" type="xs:string" use="required"/>
                        <xs:attribute name="implementation" type="xs:string" use="required"/>
                </xs:complexType>
        </xs:element>
        <xs:element name="daoConfig">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="daoSessionHandlers" minOccurs="0"/>
                                <xs:element ref="providers" minOccurs="0"  maxOccurs="1"/>
                                <xs:element ref="context" maxOccurs="unbounded"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
        <xs:element name="daoSessionHandlers">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="handler"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
        <xs:element name="handler">
                <xs:complexType>
                        <xs:simpleContent>
                                <xs:extension base="xs:string">
                                        <xs:attribute name="id" type="xs:string" use="required"/>
                                        <xs:attribute name="implementation" type="xs:string" use="required"/>
                                </xs:extension>
                        </xs:simpleContent>
                </xs:complexType>
        </xs:element>
        <xs:element name="daoFactory">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="dao" minOccurs="1" maxOccurs="unbounded"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
```

```
        <xs:element name="daoSessionHandler">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="property" maxOccurs="unbounded"/>
                        </xs:sequence>
                        <xs:attribute name="id" type="xs:string" use="required"/>
                </xs:complexType>
        </xs:element>
        <xs:element name="dataSource">
                <xs:complexType>
                        <xs:attribute name="name" type="xs:string" use="required"/>
                        <xs:attribute name="connectionString" type="xs:string" use="required"/>
                </xs:complexType>
        </xs:element>
        <xs:element name="database">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="provider" minOccurs="0"/>
                                <xs:element ref="dataSource"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
        <xs:element name="properties">
                <xs:complexType>
                        <xs:attribute name="resource" type="xs:string"/>
                        <xs:attribute name="url" type="xs:string"/>
                        <xs:attribute name="embedded" type="xs:string"/>
                </xs:complexType>
        </xs:element>
        <xs:element name="property">
                <xs:complexType>
                        <xs:attribute name="name" type="xs:string" use="required"/>
                        <xs:attribute name="value" type="xs:string" use="required"/>
                </xs:complexType>
        </xs:element>
        <xs:element name="provider">
                <xs:complexType>
                        <xs:attribute name="name" type="xs:string" use="required"/>
                </xs:complexType>
        </xs:element>
</xs:schema>
```