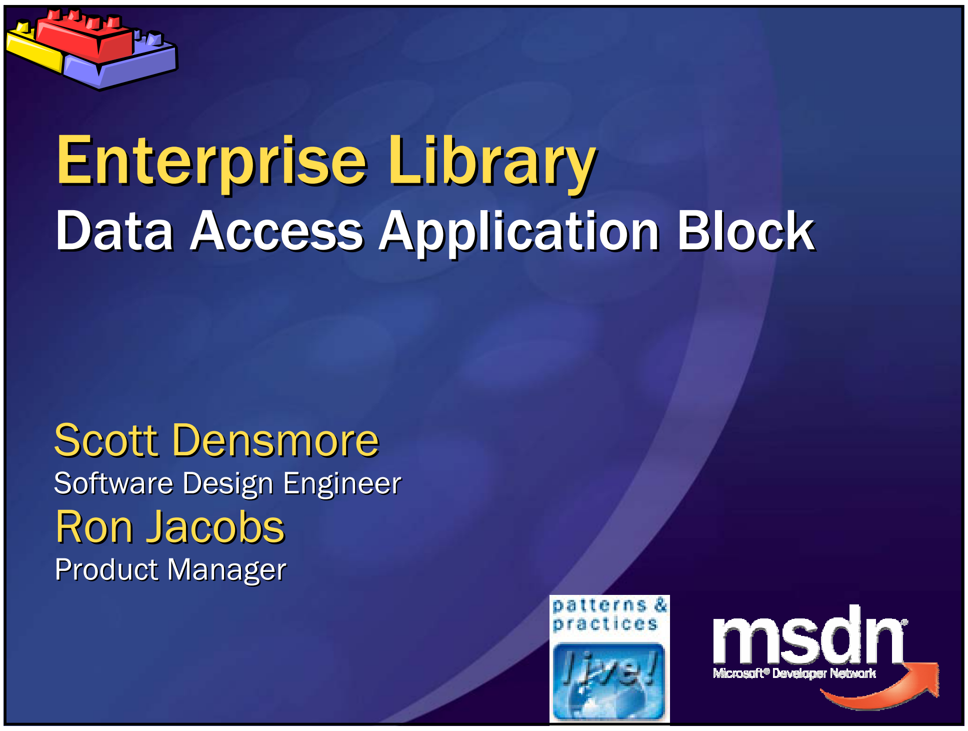
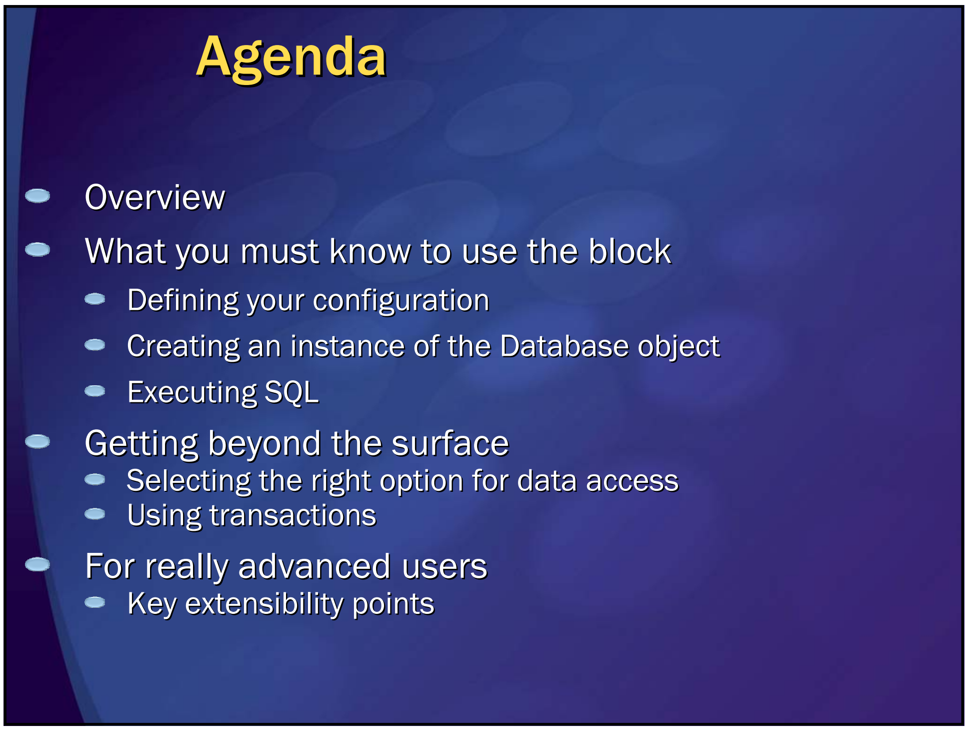


Enterprise Library
Data Access Application Block

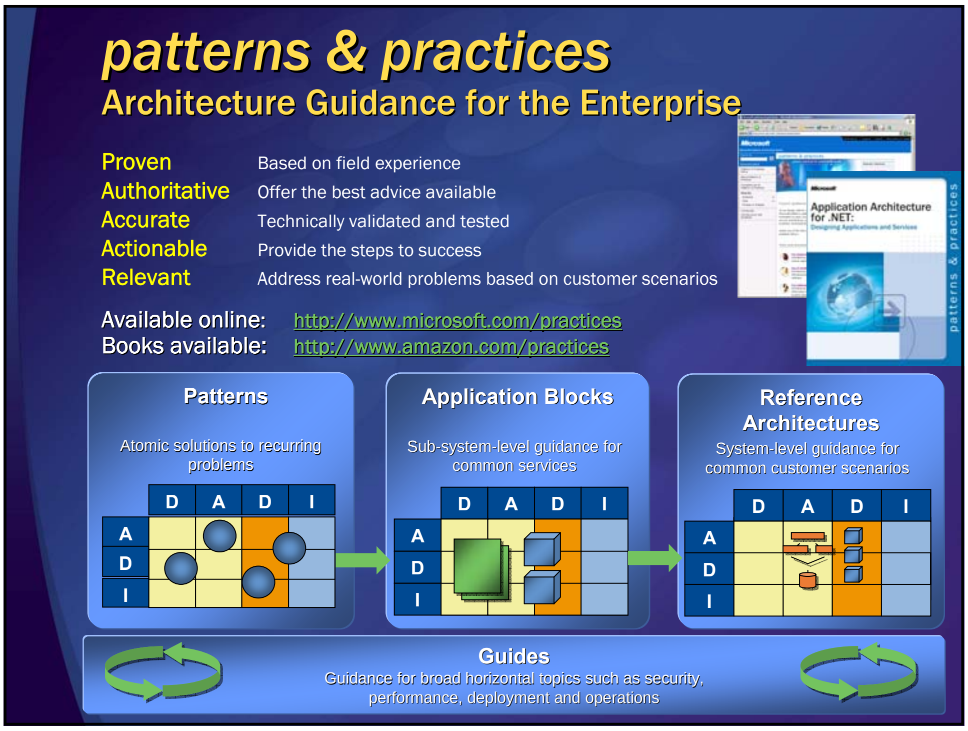
Scott Densmore
Software Design Engineer
Ron Jacobs
Product Manager

patterns & practices
live!
msdn
Microsoft® Developer Network



Agenda

- Overview
- What you must know to use the block
  - Defining your configuration
  - Creating an instance of the Database object
  - Executing SQL
- Getting beyond the surface
  - Selecting the right option for data access
  - Using transactions
- For really advanced users
  - Key extensibility points



patterns & practices

Architecture Guidance for the Enterprise

Proven	Based on field experience
Authoritative	Offer the best advice available
Accurate	Technically validated and tested
Actionable	Provide the steps to success
Relevant	Address real-world problems based on customer scenarios

Available online: http://www.microsoft.com/practices
Books available: http://www.amazon.com/practices

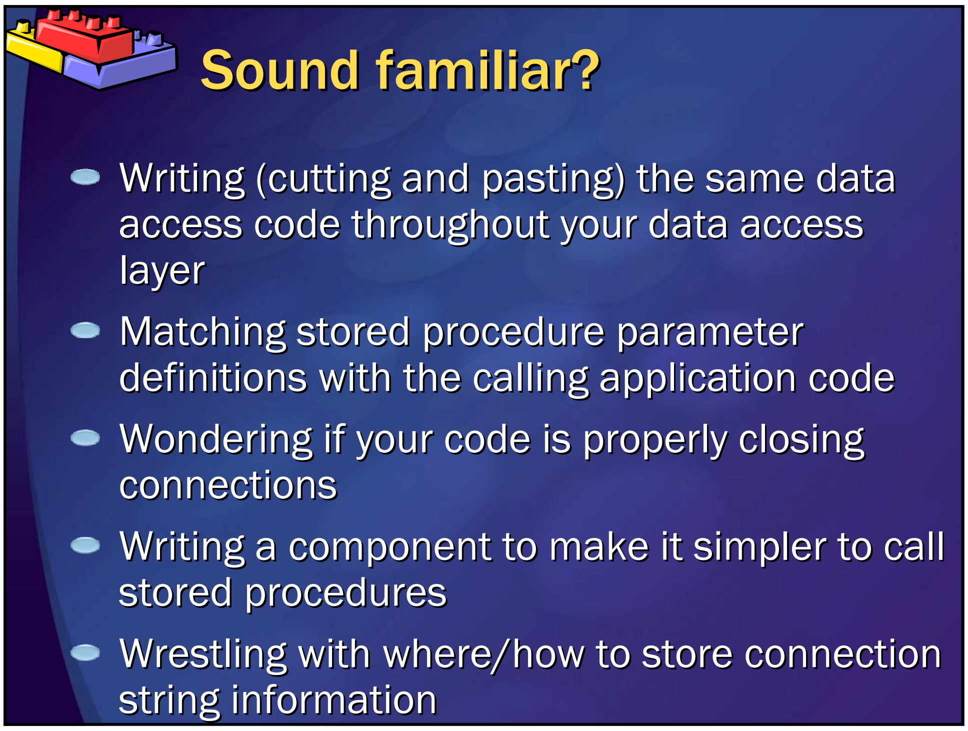
Patterns
Atomic solutions to recurring problems

Application Blocks
Sub-system-level guidance for common services

Reference Architectures
System-level guidance for common customer scenarios

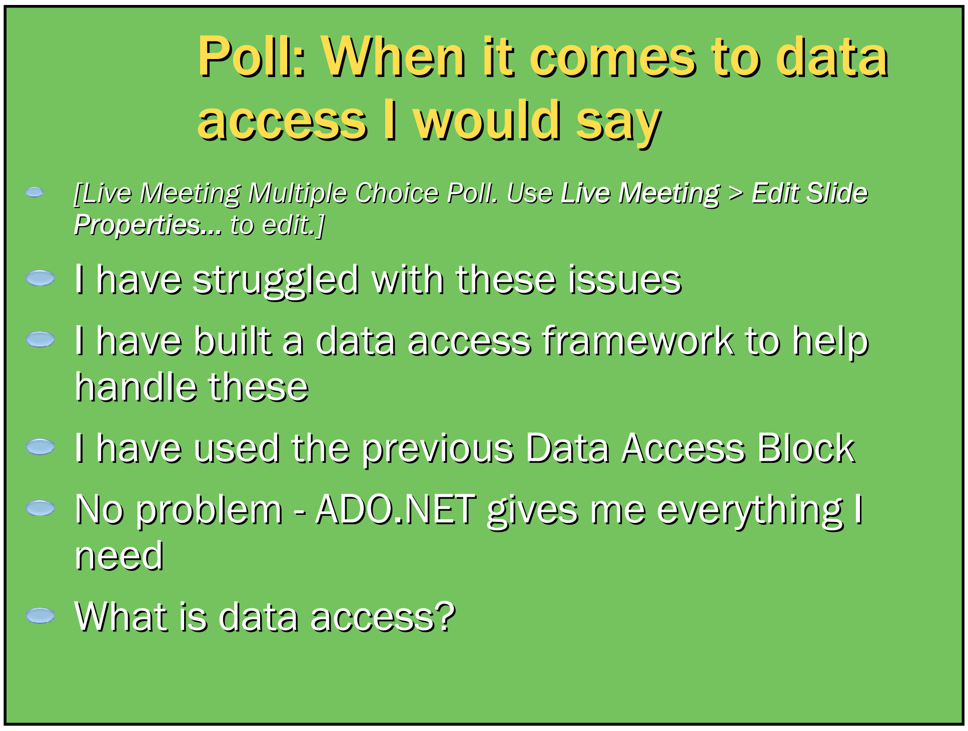
D A D I
A D I
Application Architecture for .NET: Designing Applications and Services
patterns & practices

Guides
Guidance for broad horizontal topics such as security, performance, deployment and operations



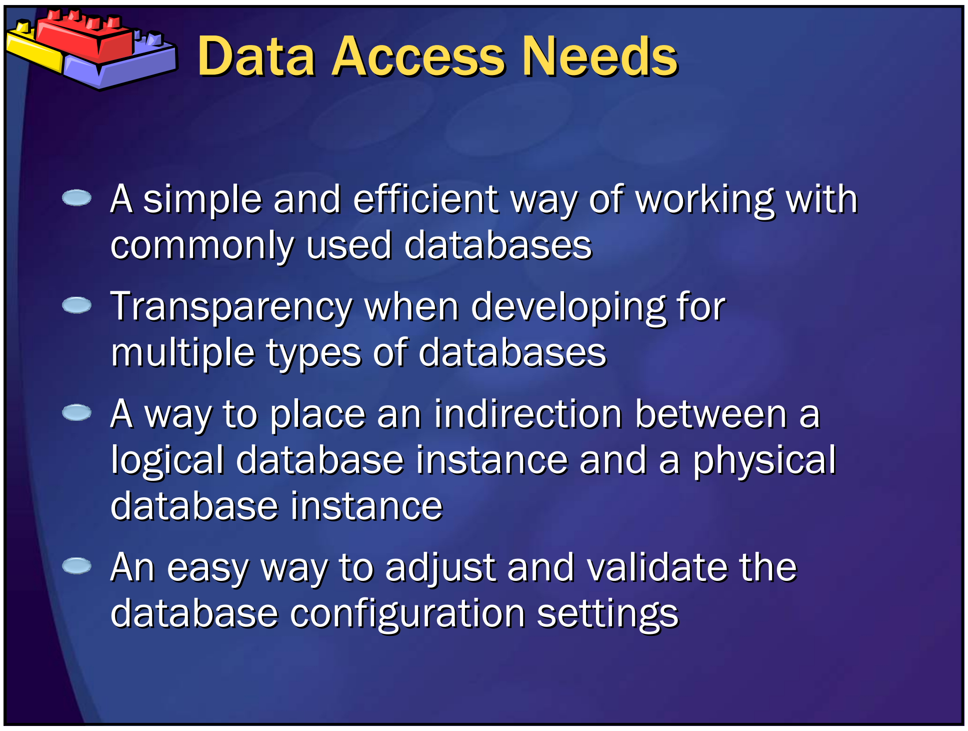
Sound familiar?

- Writing (cutting and pasting) the same data access code throughout your data access layer
- Matching stored procedure parameter definitions with the calling application code
- Wondering if your code is properly closing connections
- Writing a component to make it simpler to call stored procedures
- Wrestling with where/how to store connection string information



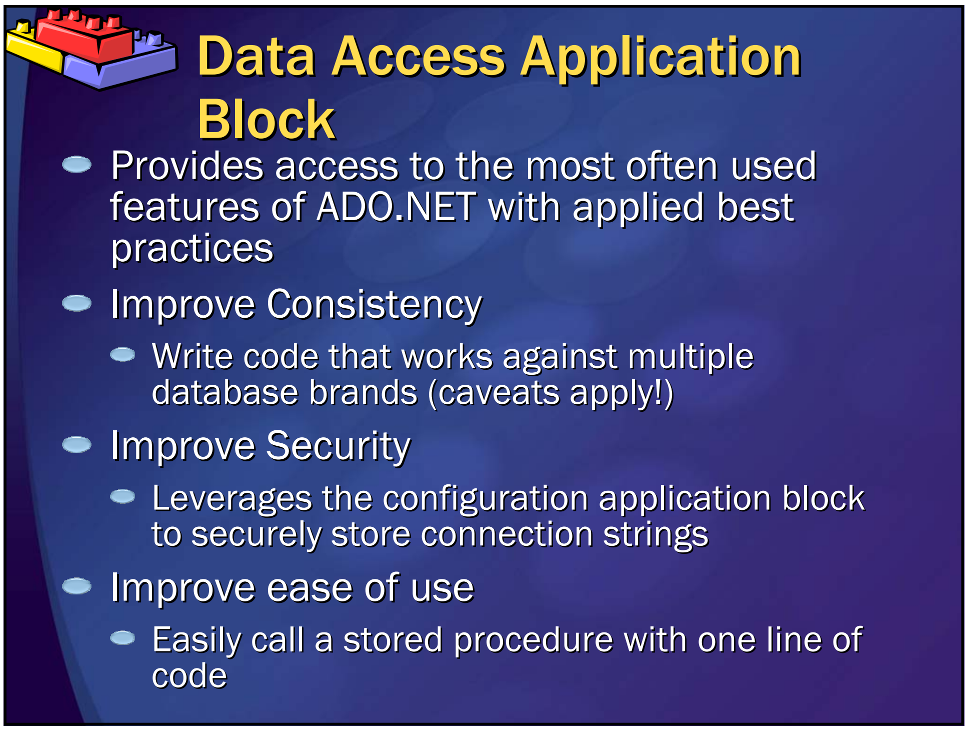
Poll: When it comes to data access I would say

- *[Live Meeting Multiple Choice Poll. Use Live Meeting > Edit Slide Properties... to edit.]*
- I have struggled with these issues
- I have built a data access framework to help handle these
- I have used the previous Data Access Block
- No problem - ADO.NET gives me everything I need
- What is data access?



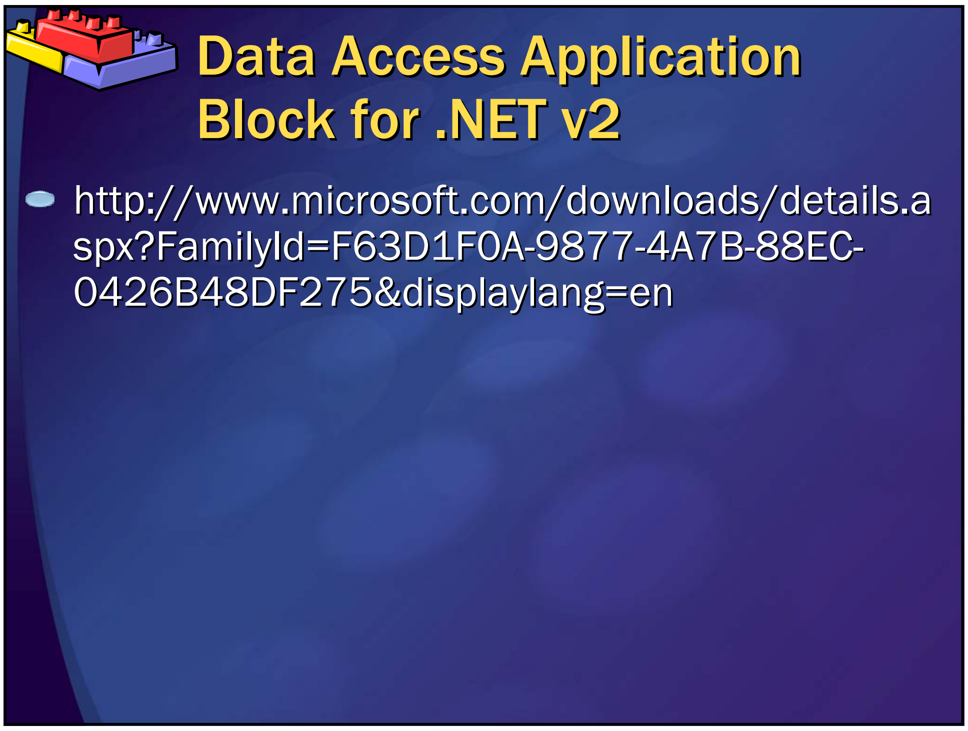
Data Access Needs

- A simple and efficient way of working with commonly used databases
- Transparency when developing for multiple types of databases
- A way to place an indirection between a logical database instance and a physical database instance
- An easy way to adjust and validate the database configuration settings



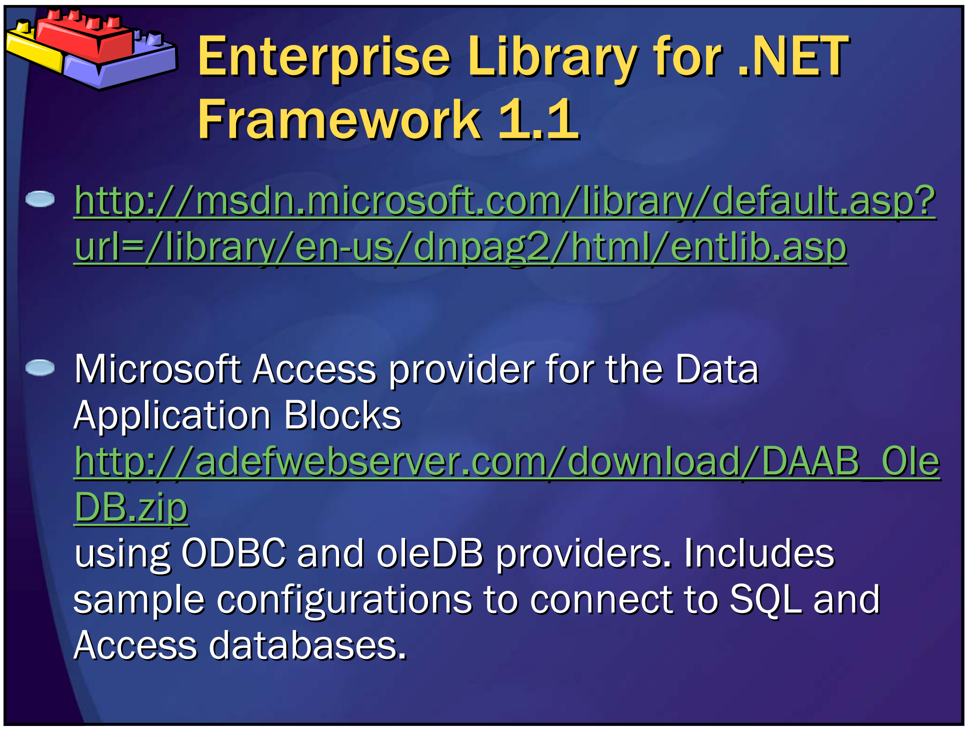
Data Access Application Block

- Provides access to the most often used features of ADO.NET with applied best practices
- Improve Consistency
  - Write code that works against multiple database brands (caveats apply!)
- Improve Security
  - Leverages the configuration application block to securely store connection strings
- Improve ease of use
  - Easily call a stored procedure with one line of code



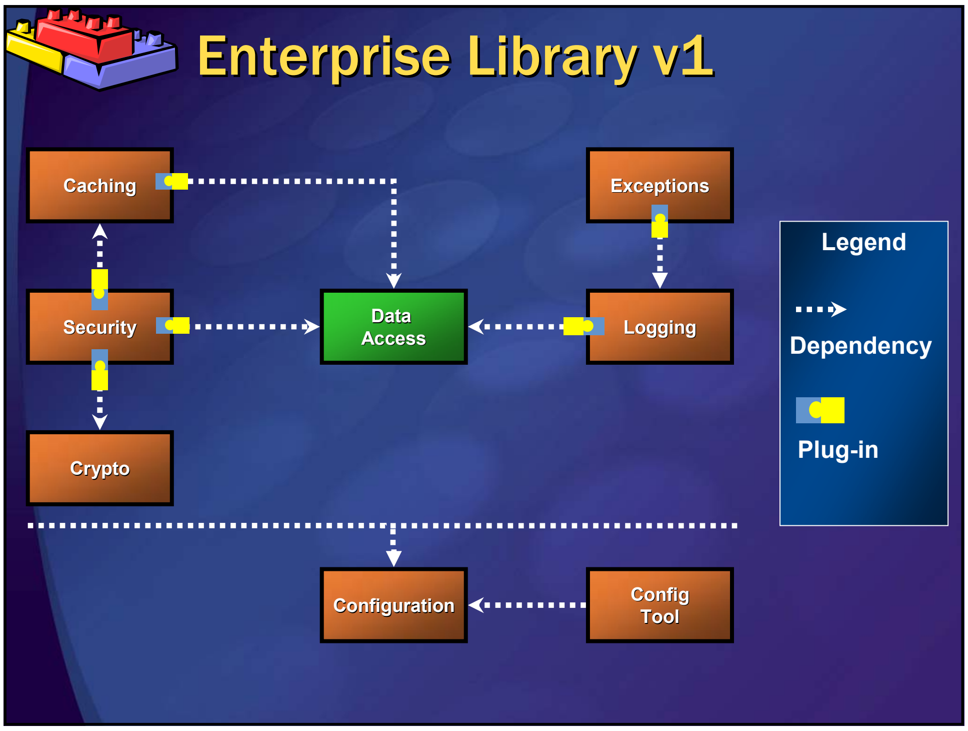
Data Access Application Block for .NET v2

- http://www.microsoft.com/downloads/details.aspx?FamilyId=F63D1F0A-9877-4A7B-88EC-0426B48DF275&displaylang=en

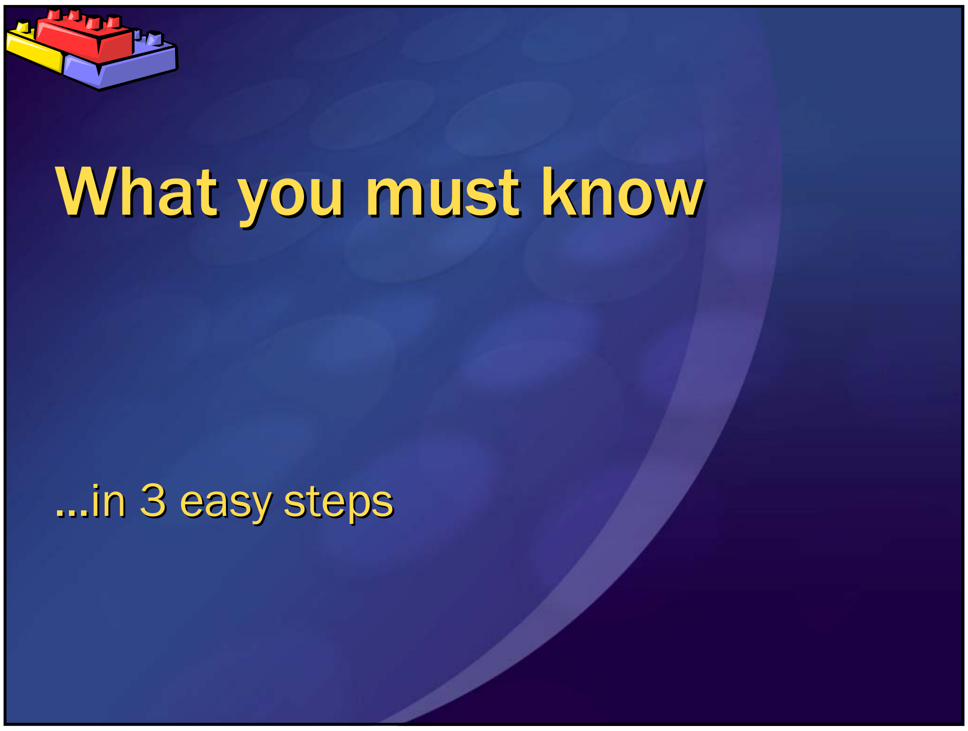


Enterprise Library for .NET Framework 1.1

- http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag2/html/entlib.asp
- Microsoft Access provider for the Data Application Blocks http://adefwebserver.com/download/DAAB_OleDB.zip using ODBC and oleDB providers. Includes sample configurations to connect to SQL and Access databases.

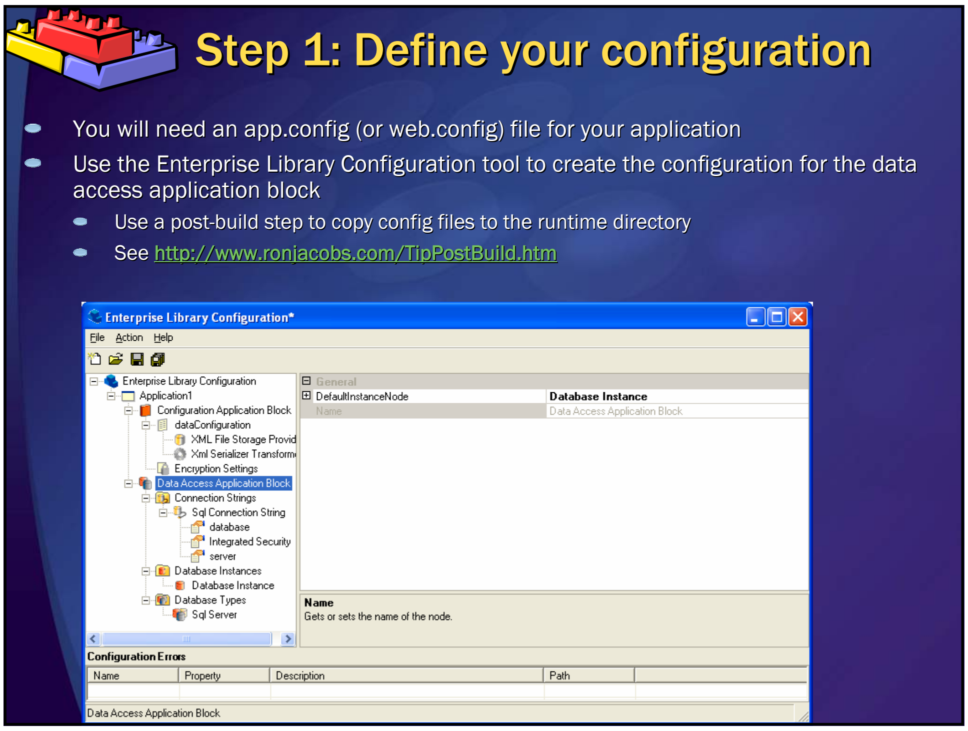


Enterprise Library v1



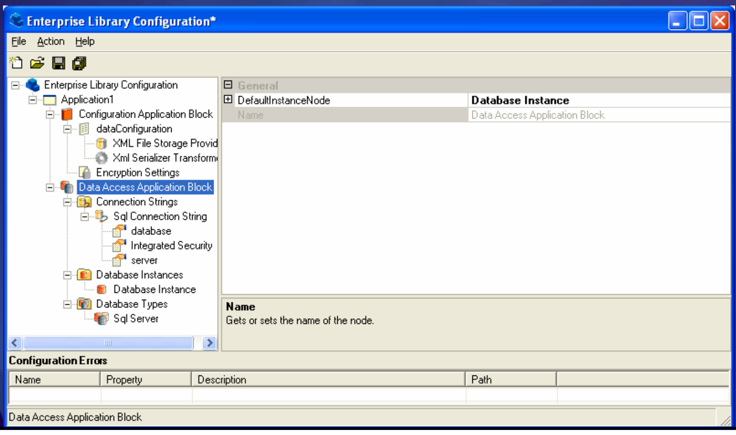
What you must know

…in 3 easy steps



Step 1: Define your configuration

- You will need an app.config (or web.config) file for your application
- Use the Enterprise Library Configuration tool to create the configuration for the data access application block
  - Use a post-build step to copy config files to the runtime directory
  - See http://www.ronjacobs.com/TipPostBuild.htm



```
Enterprise Library Configuration
  Application1
    Configuration Application Block
      dataConfiguration
        XML File Storage Provid
        Xml Serializer Transform
      Encryption Settings
    Data Access Application Block
      Connection Strings
        Sql Connection String
          database
          Integrated Security
          server
      Database Instances
        Database Instance
      Database Types
        Sql Server
```

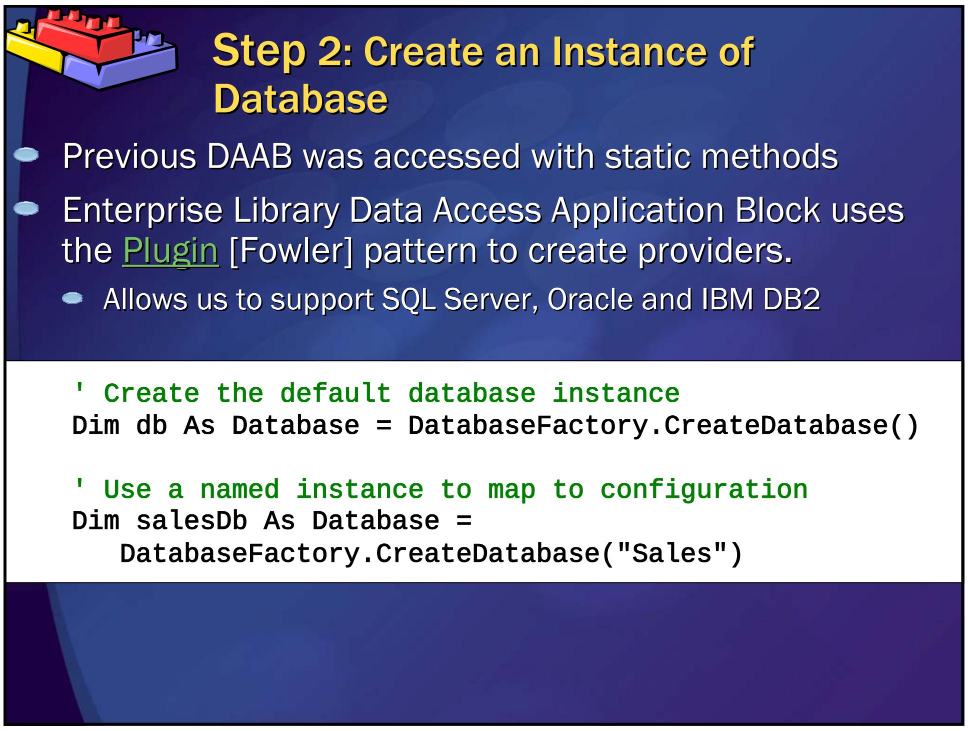
General	
DefaultInstanceNode	Database Instance
Name	Data Access Application Block

Name
Gets or sets the name of the node.

Configuration Errors

Name	Property	Description	Path

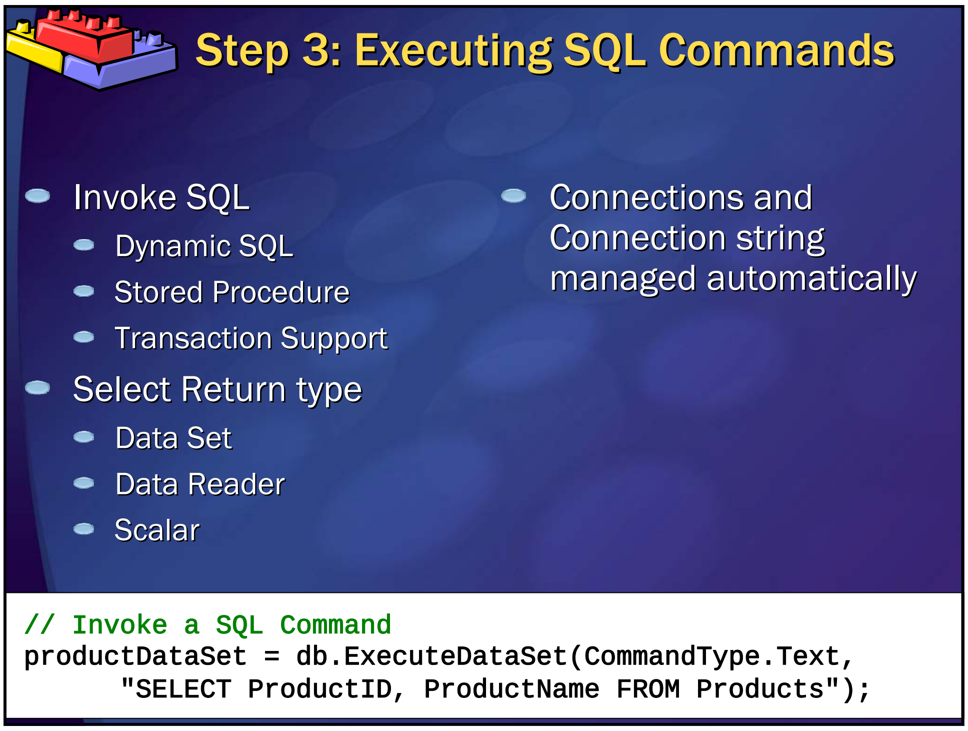
Data Access Application Block



Step 2: Create an Instance of Database

- Previous DAAB was accessed with static methods
- Enterprise Library Data Access Application Block uses the Plugin [Fowler] pattern to create providers.
  - Allows us to support SQL Server, Oracle and IBM DB2

```
' Create the default database instance
Dim db As Database = DatabaseFactory.CreateDatabase()

' Use a named instance to map to configuration
Dim salesDb As Database =
   DatabaseFactory.CreateDatabase("Sales")
```

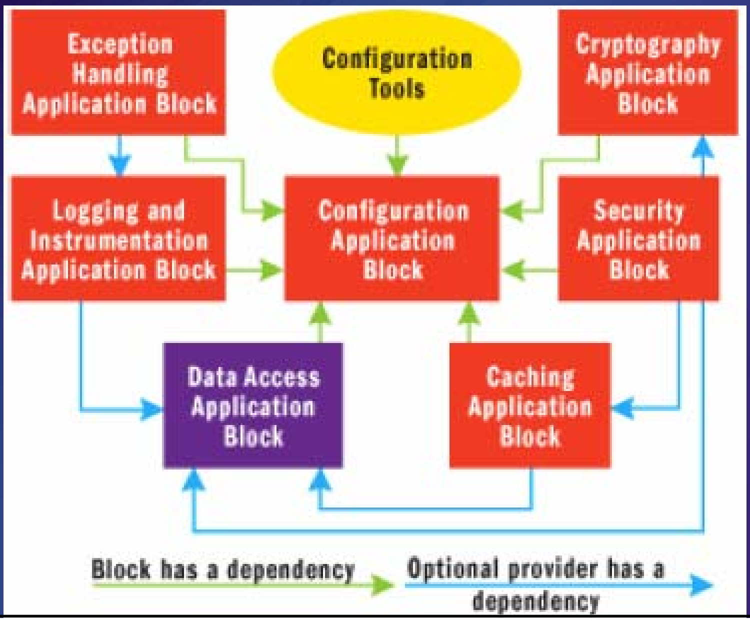


Step 3: Executing SQL Commands

- Invoke SQL
  - Dynamic SQL
  - Stored Procedure
  - Transaction Support
- Select Return type
  - Data Set
  - Data Reader
  - Scalar
- Connections and Connection string managed automatically

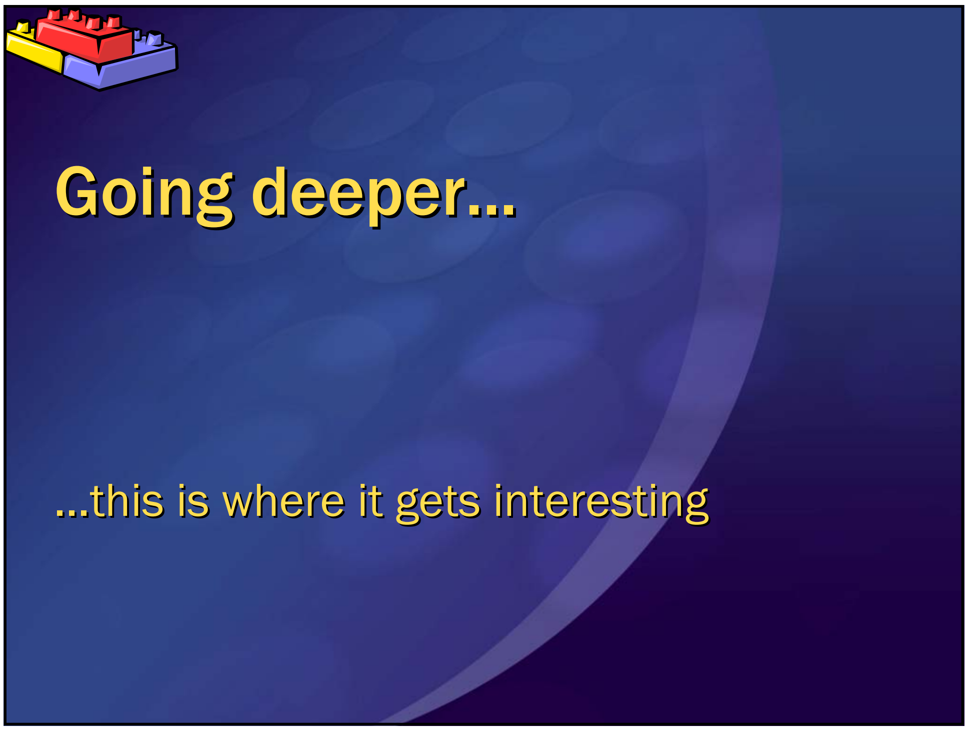
```
// Invoke a SQL Command
productDataSet = db.ExecuteDataSet(CommandType.Text,
      "SELECT ProductID, ProductName FROM Products");
```

Enterprise Library's Application Blocks


Exception Handling Application Block
Configuration Tools
Cryptography Application Block
Logging and Instrumentation Application Block
Configuration Application Block
Security Application Block
Data Access Application Block
Caching Application Block
Block has a dependency
Optional provider has a dependency

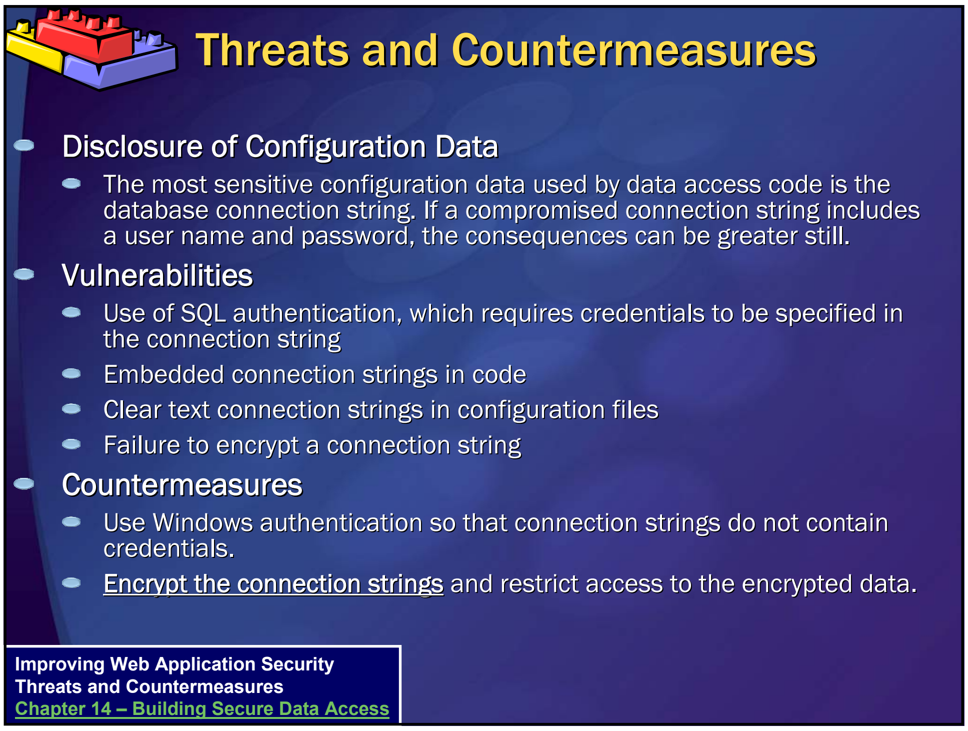
DAAB Get and Save Methods

Method	Description
ExecuteDataSet	Executes a command and returns the results to a newly created DataSet.
ExecuteReader	Executes a command and allows access to the results via a DataReader.
ExecuteScalar	Executes a command and returns the first row's first column's value. This is good for getting a single value from a command.
LoadDataSet	Executes a command and returns the results to an existing DataSet. This is good when you have a DataSet and want to add a new DataTable to it.
ExecuteNonQuery	Executes a command and returns the numbers of rows affected. This is generally used to save data to a database via a stored procedure or a SQL statement.
UpdateDataSet	Accepts a DataSet and saves all modified rows in a specified DataTable to the database. This uses three commands to save the data (InsertCommand, UpdateCommand, DeleteCommand). Each modified row will execute a single command to the database. Transactions can be used or omitted. This method can be instructed on how to behave when an error occurs in a row (for example, continue or throw an exception).



Going deeper…

…this is where it gets interesting



Threats and Countermeasures

- Disclosure of Configuration Data
  - The most sensitive configuration data used by data access code is the database connection string. If a compromised connection string includes a user name and password, the consequences can be greater still.
- Vulnerabilities
  - Use of SQL authentication, which requires credentials to be specified in the connection string
  - Embedded connection strings in code
  - Clear text connection strings in configuration files
  - Failure to encrypt a connection string
- Countermeasures
  - Use Windows authentication so that connection strings do not contain credentials.
  - Encrypt the connection strings and restrict access to the encrypted data.

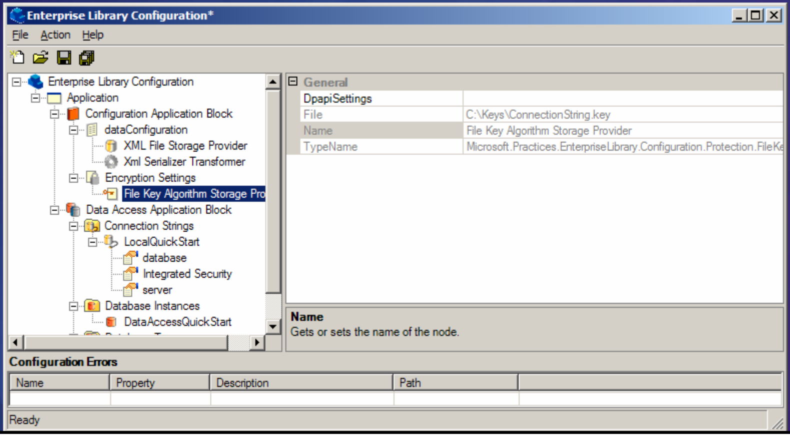
Improving Web Application Security Threats and Countermeasures
Chapter 14 – Building Secure Data Access

Storing Connection Strings

- Enterprise Library provides *applied guidance* through proven practices engineered in code
- Connection strings are managed through configuration with the Configuration Application Block
- With the default XML Storage Provider
  - Connection strings are saved in the file dataConfiguration.config
  - Configuration files are saved as plain text by default
- Enterprise Library cryptography functionality which can be used to encrypt the connection string automatically
  - In just 2 easy steps!

Securing Connection Strings

- The encryption configuration determines *how* the application block configuration will be encrypted



Enterprise Library Configuration*

```
Enterprise Library Configuration
  Application
    Configuration Application Block
      dataConfiguration
        XML File Storage Provider
        Xml Serializer Transformer
      Encryption Settings
        File Key Algorithm Storage Pro
    Data Access Application Block
      Connection Strings
        LocalQuickStart
          database
          Integrated Security
          server
      Database Instances
        DataAccessQuickStart
```

General

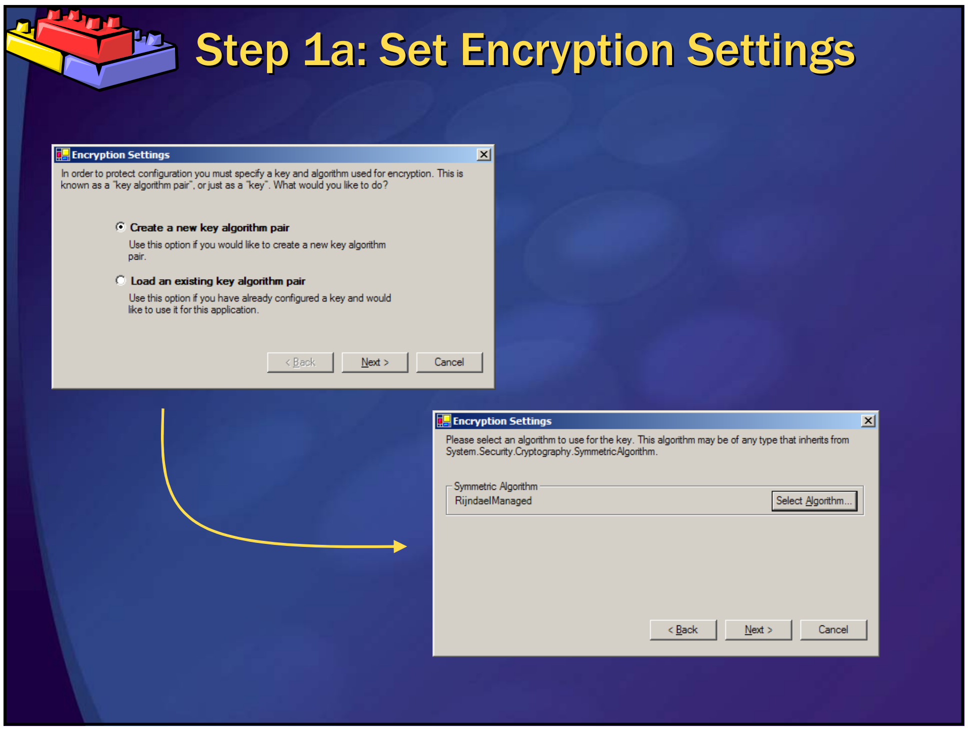
DpapiSettings	
File	C:\Keys\ConnectionString.key
Name	File Key Algorithm Storage Provider
TypeName	Microsoft.Practices.EnterpriseLibrary.Configuration.Protection.FileKe

Name
Gets or sets the name of the node.

Configuration Errors

Name	Property	Description	Path

Ready



Step 1a: Set Encryption Settings

Encryption Settings

In order to protect configuration you must specify a key and algorithm used for encryption. This is known as a "key algorithm pair", or just as a "key". What would you like to do?

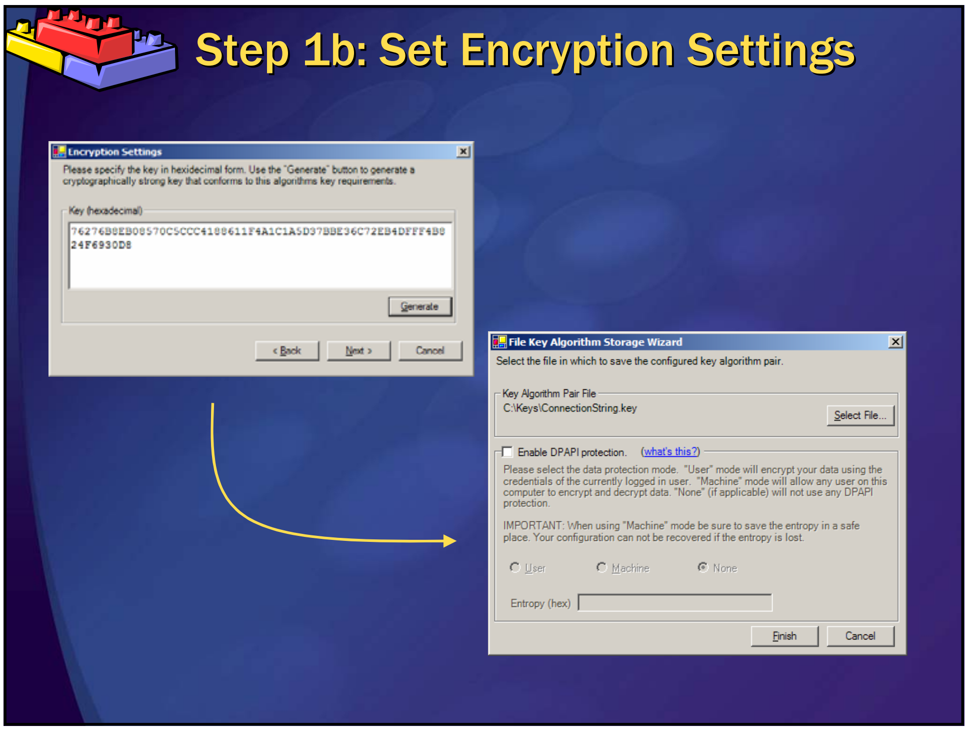
[✓ **Create a new key algorithm pair**]
Use this option if you would like to create a new key algorithm pair.

Load an existing key algorithm pair
Use this option if you have already configured a key and would like to use it for this application.

Encryption Settings

Please select an algorithm to use for the key. This algorithm may be of any type that inherits from System.Security.Cryptography.SymmetricAlgorithm.

Symmetric Algorithm: RijndaelManaged



Step 1b: Set Encryption Settings

Encryption Settings

Please specify the key in hexidecimal form. Use the "Generate" button to generate a cryptographically strong key that conforms to this algorithms key requirements.

Key (hexadecimal):

```
76276B8EB08570C5CCC4188611F4A1C1A5D37BBE36C72EB4DFFF4B8
24F6930D8
```

File Key Algorithm Storage Wizard

Select the file in which to save the configured key algorithm pair.

Key Algorithm Pair File: C:\Keys\ConnectionString.key

Enable DPAPI protection. (what's this?)

Please select the data protection mode. "User" mode will encrypt your data using the credentials of the currently logged in user. "Machine" mode will allow any user on this computer to encrypt and decrypt data. "None" (if applicable) will not use any DPAPI protection.

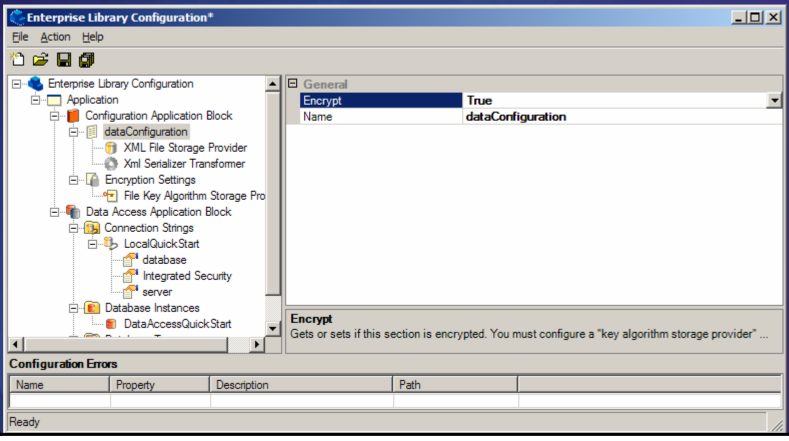
IMPORTANT: When using "Machine" mode be sure to save the entropy in a safe place. Your configuration can not be recovered if the entropy is lost.

User Machine [✓ None]

Entropy (hex):

Step 2: Mark the configuration section as encrypted

- *Whether* to encrypt configuration information is determined by each application block's configuration settings



Enterprise Library Configuration*

File Action Help

- Enterprise Library Configuration
  - Application
    - Configuration Application Block
      - dataConfiguration
        - XML File Storage Provider
        - Xml Serializer Transformer
      - Encryption Settings
        - File Key Algorithm Storage Pro
    - Data Access Application Block
      - Connection Strings
        - LocalQuickStart
          - database
          - Integrated Security
          - server
      - Database Instances
        - DataAccessQuickStart

General	
Encrypt	True
Name	dataConfiguration

Encrypt
Gets or sets if this section is encrypted. You must configure a "key algorithm storage provider" ...

Configuration Errors

Name	Property	Description	Path

Ready

Securing Connection Strings

- From this...

```
<?xml version="1.0" encoding="utf-8"?>
<dataConfiguration>
 <xmlSerializerSection ...>
  <enterpriseLibrary.databaseSettings ...>
   <databaseTypes>
    <databaseType name="Sql Server" type="Microsoft.Practices.EnterpriseLibrary.Data.Sql.SqlDatabase, Microsoft.Practices.EnterpriseLibrary.Data" />
   </databaseTypes>
   <instances>
    <instance name="DataAccessQuickStart" type="Sql Server" connectionString="LocalQuickStart" />
   </instances>
   <connectionStrings>
    <connectionString name="LocalQuickStart">
     <parameters>
      <parameter name="database" value="EntLibQuickStarts" isSensitive="false" />
      <parameter name="Integrated Security" value="True" isSensitive="false" />
      <parameter name="server" value="localhost" isSensitive="false" />
     </parameters>
    </connectionString>
   </connectionStrings>
  </enterpriseLibrary.databaseSettings>
 </xmlSerializerSection>
</dataConfiguration>
```

- To this...

```
滴섢□礪칭□ć告쬤삗□쨌냢▾蚆곤菽□琊恒鵴숚□□□鮞쐯鲇鏖□꼬娧ﻬ넚□□□干說□쟜粢□□□猴□睉□□□栢숀ㄷ쎌ﺻﻢ乞□빨□□
⒝촤긐붇뎟□Ű긂□□呱□□□□□謯礧跶魦□≵□□□ἐ□璉朋뗖鰄ൺ□痴□疼□삳쐺□□□篬흵ﷲ□ﺨﻰ揸□□破壖걶䛇ὶ□□儆첵힙늗□
欯鬌맥쬥ത艺鐇嘒벵□□□쩻□□垿昙□□□ϑ□□□炔膸渶謪颰윤給□□□鹹偩□鷊□峠□鍁롥슨鵄椹렣曡□嫂¢窮鶬锐□젦□□□□僙
□헐铍□K□ŸΊ位ⅿ騺ﻄﻲ馀□訑□ū□□啵□□□褎□弣뗩皻□□□괵撒爂□□□□垘□□毨□넚□□郏□ㄑ五□篥神ㅏ쐺□□伏閃暸□騷罉
뗗□□쵦□倆둂□灕ɽ辰鐣鼚鏓嚾□稛뢎·□□□곮□□펫肐□웰︶뒫僮□□緺□鰦꿇榚ų잿뙥□뒷□闢□□□□攢硴㊒□□□腨뾩蘭□Â퐞
迴졷ﷳ쇠삽렭□□□爅┸□□泒嘲□□䀧慷□쬁炁□登淅됩□□샹□□□□粹救□凿□□□ˆ´□□è섨浱걷旅□펦玪呃켗傓□□硎疡
```

Windows Authentication

- Security

  "When your application connects to a SQL Server database, you have a choice of Windows authentication or SQL authentication. Windows authentication is more secure because it does not send credentials over the network"

- Performance

  "From a security perspective, you should use Windows authentication. From a performance perspective, you should use a fixed service account and avoid impersonation. The fixed service account is typically the process account of the application. By using a fixed service account and a consistent connection string, you help ensure that database connections are pooled efficiently. You also help ensure that the database connections are shared by multiple clients. Using a fixed service account and a consistent connection string is a major factor in helping application scalability."

Improving Web Application Security Threats and Countermeasures
Chapter 14 – Building Secure Data Access

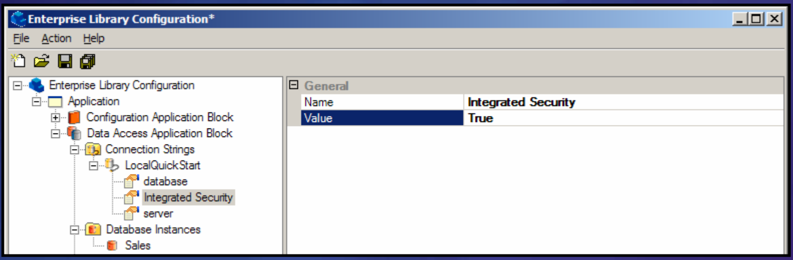
Improving .NET Application Performance and Scalability
Chapter 12 – Improving ADO.NET Performance

Configuring Windows Auth

- Desired connection string

```
"database=Sales;Integrated Security=True;server=localhost;"
```

- Using the Configuration Console



```
Enterprise Library Configuration
  Application
    Configuration Application Block
    Data Access Application Block
      Connection Strings
        LocalQuickStart
          database
          Integrated Security
          server
      Database Instances
        Sales
```

General	
Name	Integrated Security
Value	True

- Sample code

```
Database db = DatabaseFactory.CreateDatabase(“Sales”);
```

Improving .NET Application Performance

- Connections
- Database connections are an expensive and limited resource. Your approach to connection management can significantly affect the overall performance and scalability of your application. Issues to consider include acquiring and releasing connections, pooling, and authentication. To improve database connection performance and scalability, apply the following strategies to your connection management policy:
  - Open and close the connection in the method.
  - Explicitly close connections.
  - Pool connections

Microsoft
Improving .NET Application Performance and Scalability
patterns & practices

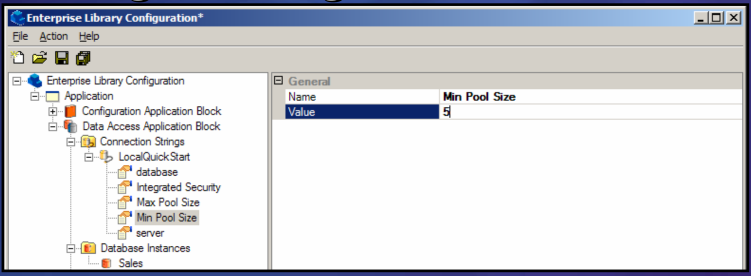
Improving .NET Application Performance and Scalability
Chapter 12 – Improving ADO.NET Performance

Configuring Connection Pooling

- Desired connection string

```
"database=EntLibQuickStarts;Integrated Security=True;
Max Pool Size=75;Min Pool Size=5;server=localhost;"`
```

- Using the Configuration Console



Enterprise Library Configuration*
File Action Help

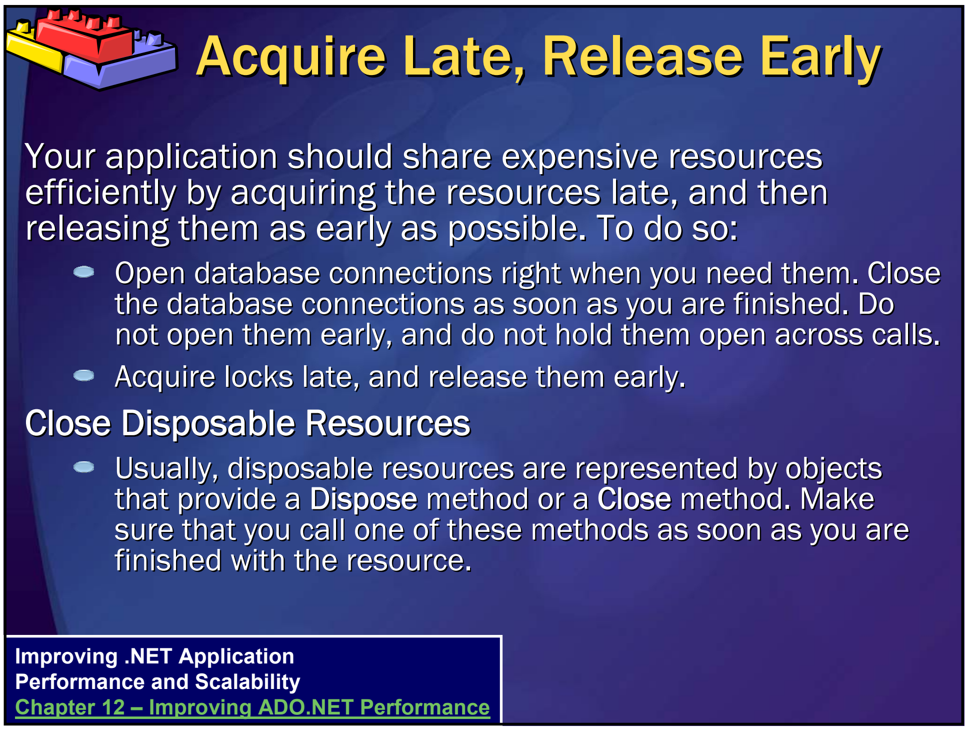
- Enterprise Library Configuration
  - Application
    - Configuration Application Block
    - Data Access Application Block
      - Connection Strings
        - LocalQuickStart
          - database
          - Integrated Security
          - Max Pool Size
          - Min Pool Size
          - server
      - Database Instances
        - Sales

General

Name	Min Pool Size
Value	5

- Sample code

```
Database db = DatabaseFactory.CreateDatabase(“Sales”);
```



Acquire Late, Release Early

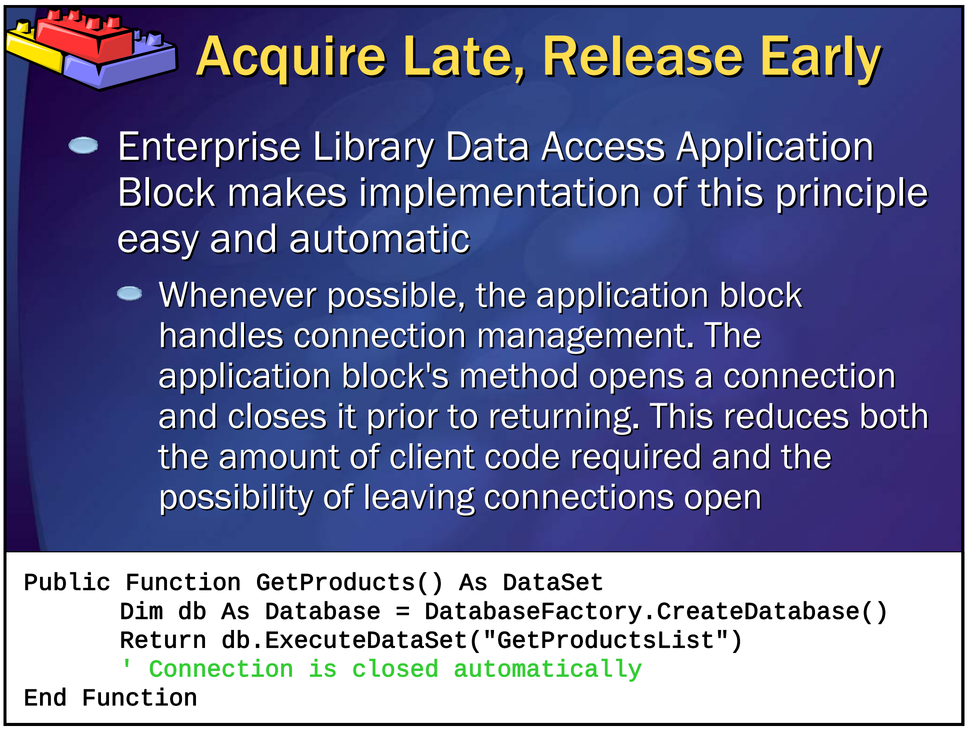
Your application should share expensive resources efficiently by acquiring the resources late, and then releasing them as early as possible. To do so:

- Open database connections right when you need them. Close the database connections as soon as you are finished. Do not open them early, and do not hold them open across calls.
- Acquire locks late, and release them early.

Close Disposable Resources

- Usually, disposable resources are represented by objects that provide a Dispose method or a Close method. Make sure that you call one of these methods as soon as you are finished with the resource.

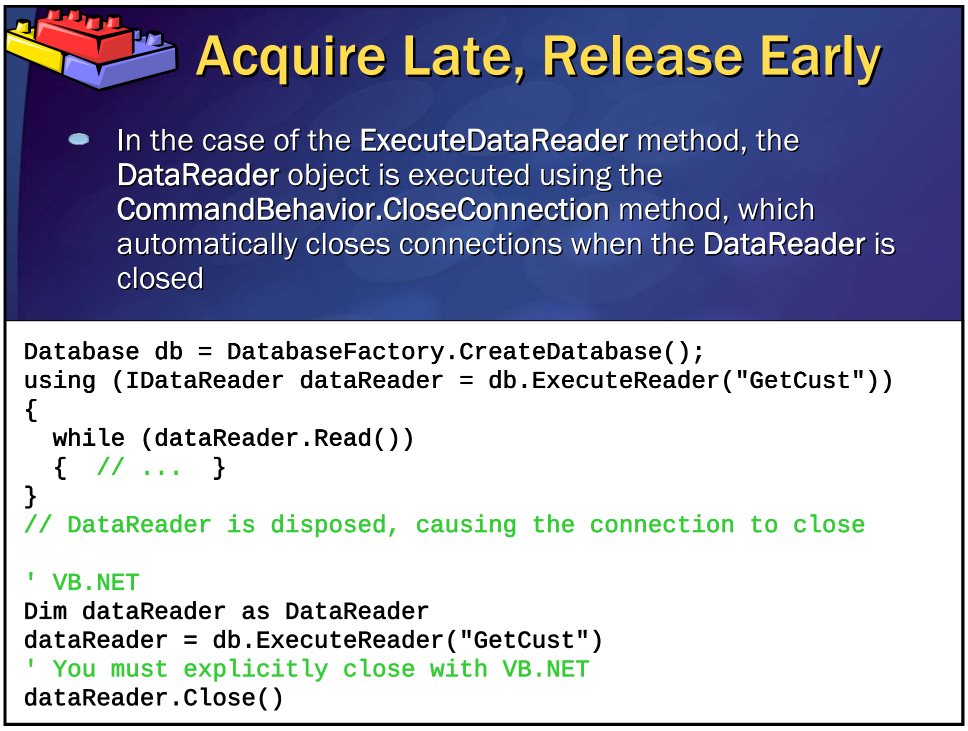
Improving .NET Application Performance and Scalability
Chapter 12 – Improving ADO.NET Performance



Acquire Late, Release Early

- Enterprise Library Data Access Application Block makes implementation of this principle easy and automatic
  - Whenever possible, the application block handles connection management. The application block's method opens a connection and closes it prior to returning. This reduces both the amount of client code required and the possibility of leaving connections open

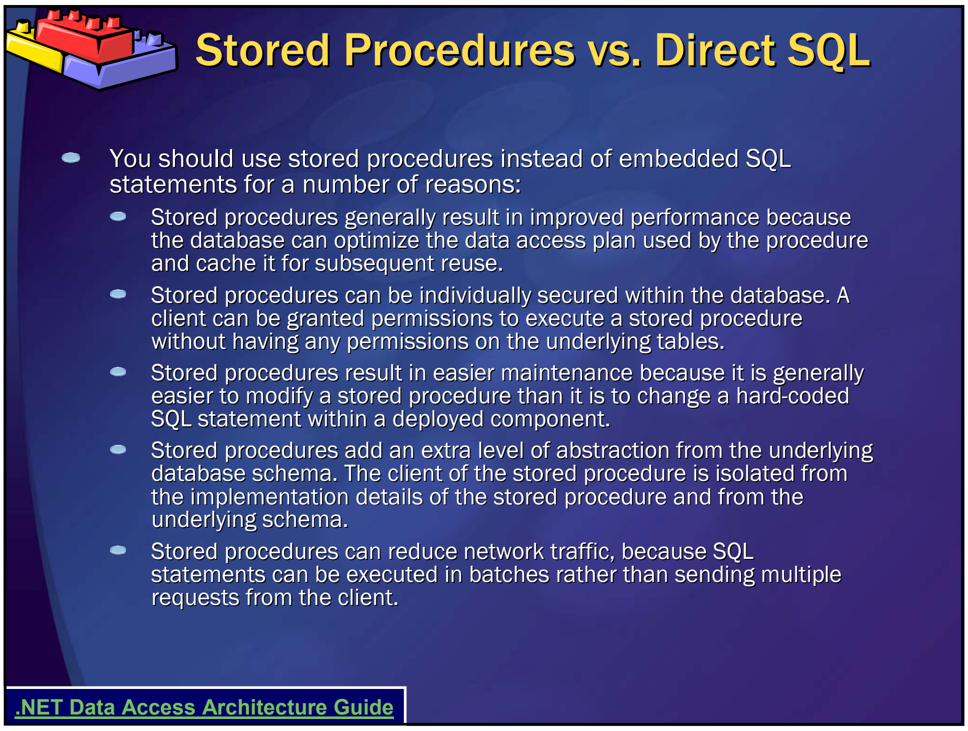
```
Public Function GetProducts() As DataSet
        Dim db As Database = DatabaseFactory.CreateDatabase()
        Return db.ExecuteDataSet("GetProductsList")
        ' Connection is closed automatically
End Function
```



Acquire Late, Release Early

- In the case of the ExecuteDataReader method, the DataReader object is executed using the CommandBehavior.CloseConnection method, which automatically closes connections when the DataReader is closed

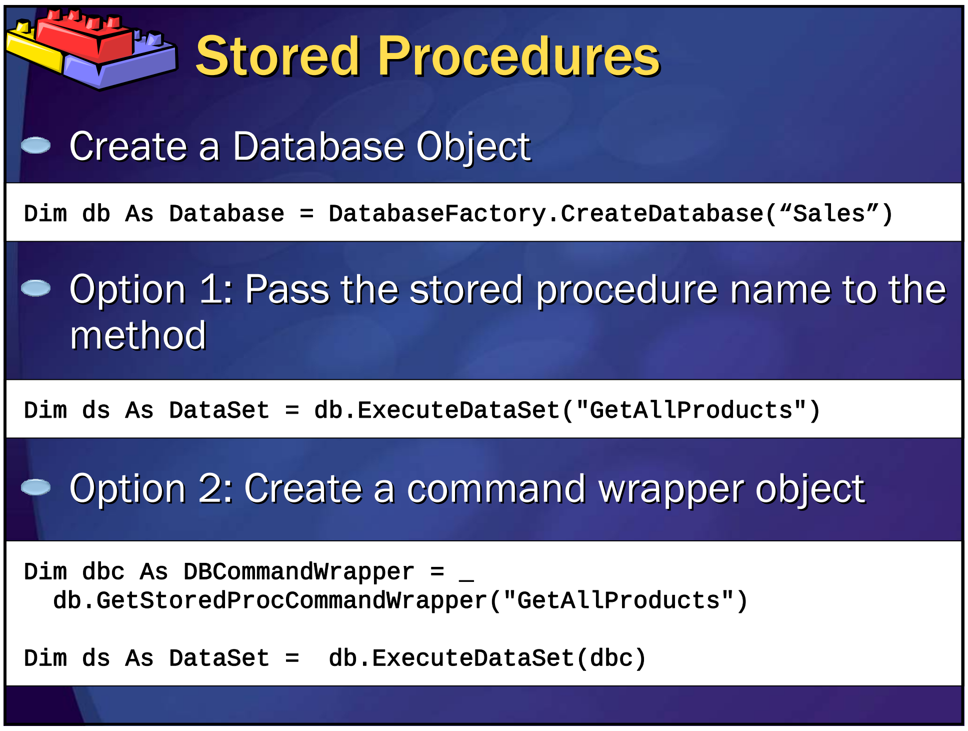
```
Database db = DatabaseFactory.CreateDatabase();
using (IDataReader dataReader = db.ExecuteReader("GetCust"))
{
  while (dataReader.Read())
  {  // ...  }
}
// DataReader is disposed, causing the connection to close

' VB.NET
Dim dataReader as DataReader
dataReader = db.ExecuteReader("GetCust")
' You must explicitly close with VB.NET
dataReader.Close()
```



Stored Procedures vs. Direct SQL

- You should use stored procedures instead of embedded SQL statements for a number of reasons:
  - Stored procedures generally result in improved performance because the database can optimize the data access plan used by the procedure and cache it for subsequent reuse.
  - Stored procedures can be individually secured within the database. A client can be granted permissions to execute a stored procedure without having any permissions on the underlying tables.
  - Stored procedures result in easier maintenance because it is generally easier to modify a stored procedure than it is to change a hard-coded SQL statement within a deployed component.
  - Stored procedures add an extra level of abstraction from the underlying database schema. The client of the stored procedure is isolated from the implementation details of the stored procedure and from the underlying schema.
  - Stored procedures can reduce network traffic, because SQL statements can be executed in batches rather than sending multiple requests from the client.

.NET Data Access Architecture Guide



Stored Procedures

- Create a Database Object

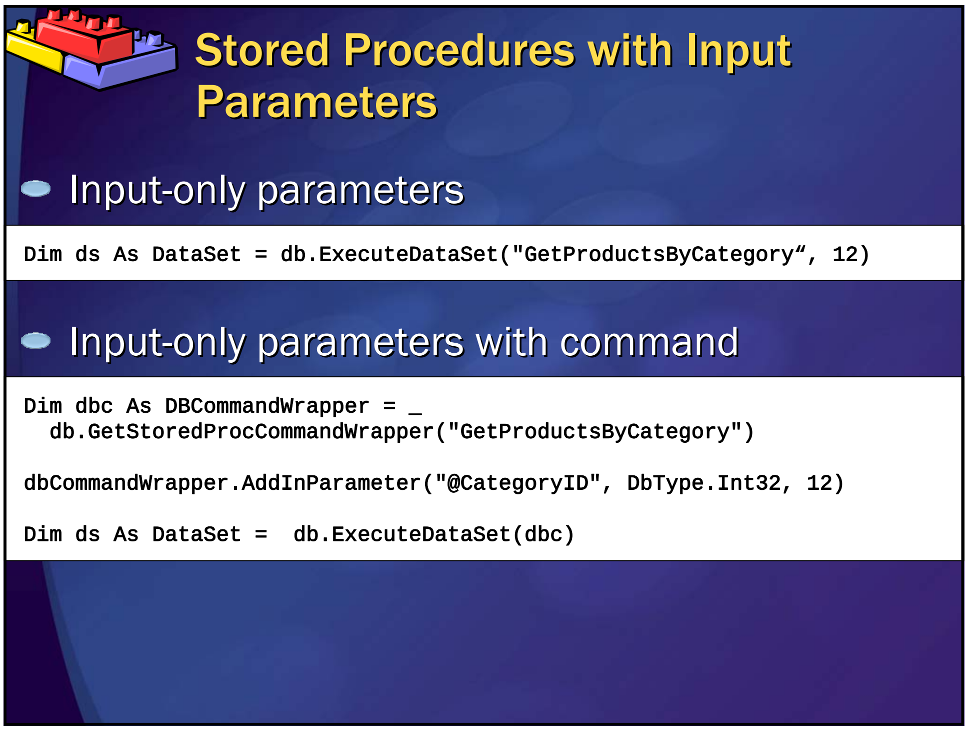
```
Dim db As Database = DatabaseFactory.CreateDatabase(“Sales”)
```

- Option 1: Pass the stored procedure name to the method

```
Dim ds As DataSet = db.ExecuteDataSet("GetAllProducts")
```

- Option 2: Create a command wrapper object

```
Dim dbc As DBCommandWrapper = _
  db.GetStoredProcCommandWrapper("GetAllProducts")

Dim ds As DataSet =  db.ExecuteDataSet(dbc)
```



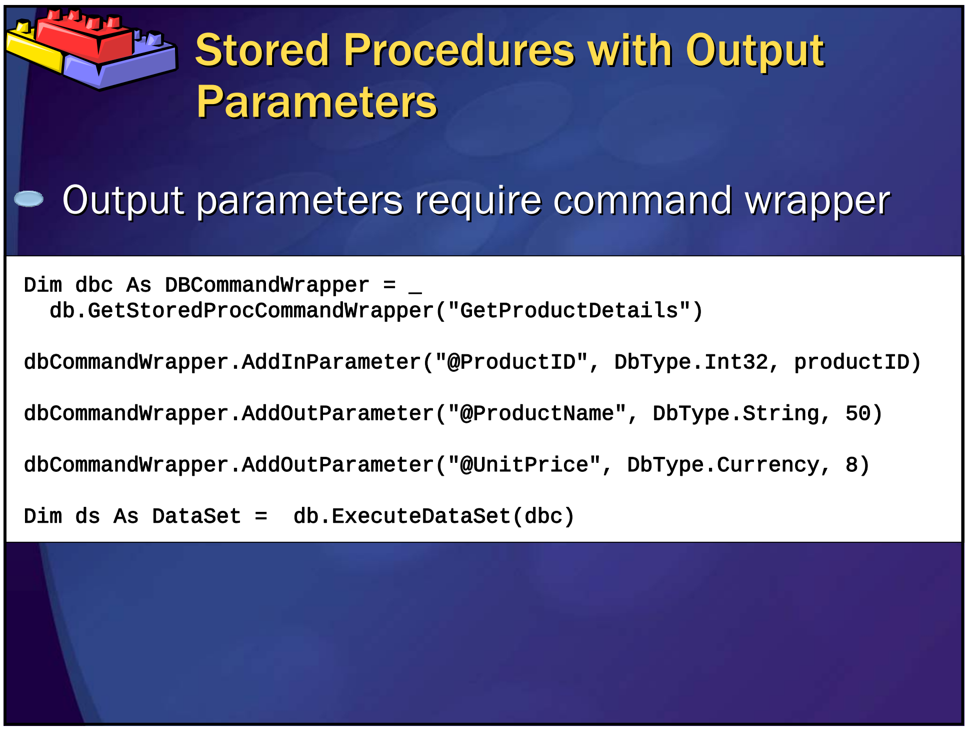
Stored Procedures with Input Parameters

- Input-only parameters

```
Dim ds As DataSet = db.ExecuteDataSet("GetProductsByCategory“, 12)
```

- Input-only parameters with command

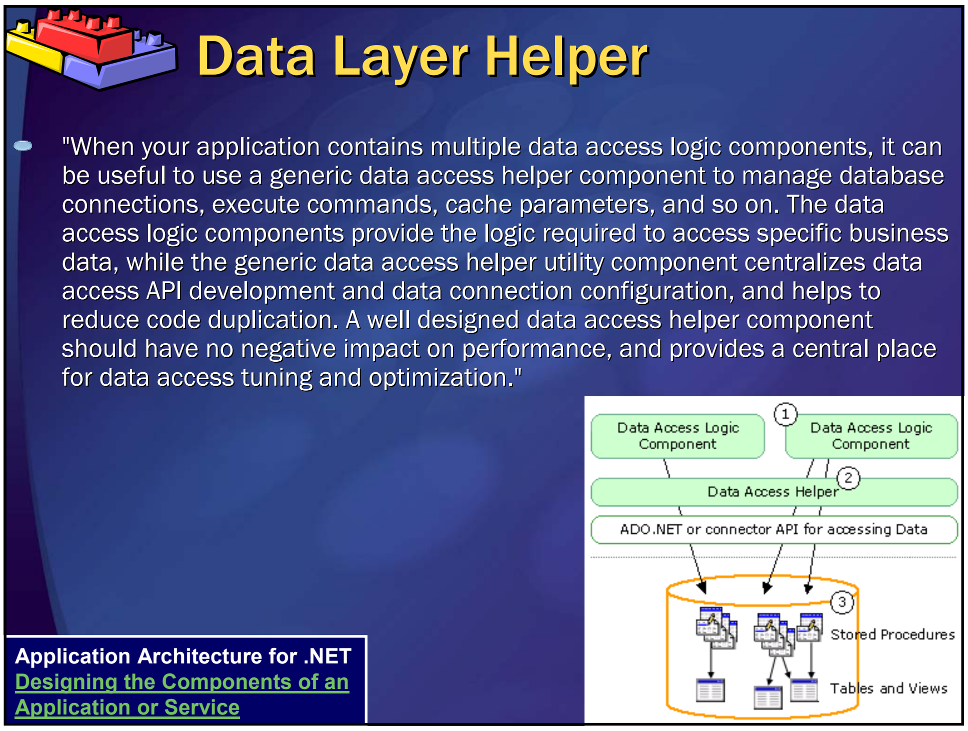
```
Dim dbc As DBCommandWrapper = _
  db.GetStoredProcCommandWrapper("GetProductsByCategory")

dbCommandWrapper.AddInParameter("@CategoryID", DbType.Int32, 12)

Dim ds As DataSet =  db.ExecuteDataSet(dbc)
```



Stored Procedures with Output Parameters

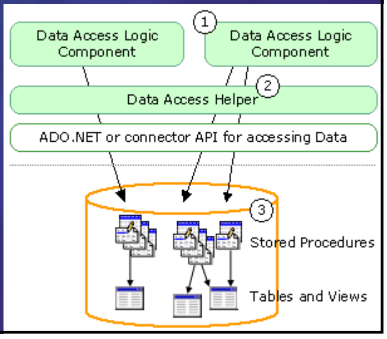
- Output parameters require command wrapper

```
Dim dbc As DBCommandWrapper = _
  db.GetStoredProcCommandWrapper("GetProductDetails")

dbCommandWrapper.AddInParameter("@ProductID", DbType.Int32, productID)

dbCommandWrapper.AddOutParameter("@ProductName", DbType.String, 50)

dbCommandWrapper.AddOutParameter("@UnitPrice", DbType.Currency, 8)

Dim ds As DataSet =  db.ExecuteDataSet(dbc)
```

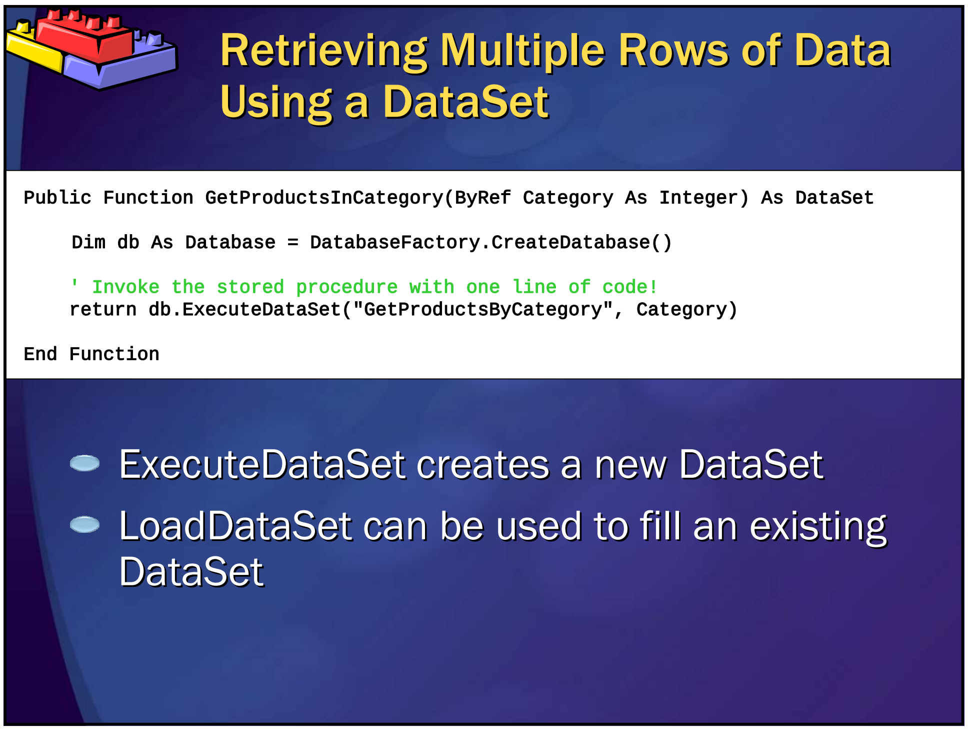


Data Layer Helper

- "When your application contains multiple data access logic components, it can be useful to use a generic data access helper component to manage database connections, execute commands, cache parameters, and so on. The data access logic components provide the logic required to access specific business data, while the generic data access helper utility component centralizes data access API development and data connection configuration, and helps to reduce code duplication. A well designed data access helper component should have no negative impact on performance, and provides a central place for data access tuning and optimization."


Data Access Logic Component
1
Data Access Logic Component
2
Data Access Helper
ADO.NET or connector API for accessing Data
3
Stored Procedures
Tables and Views

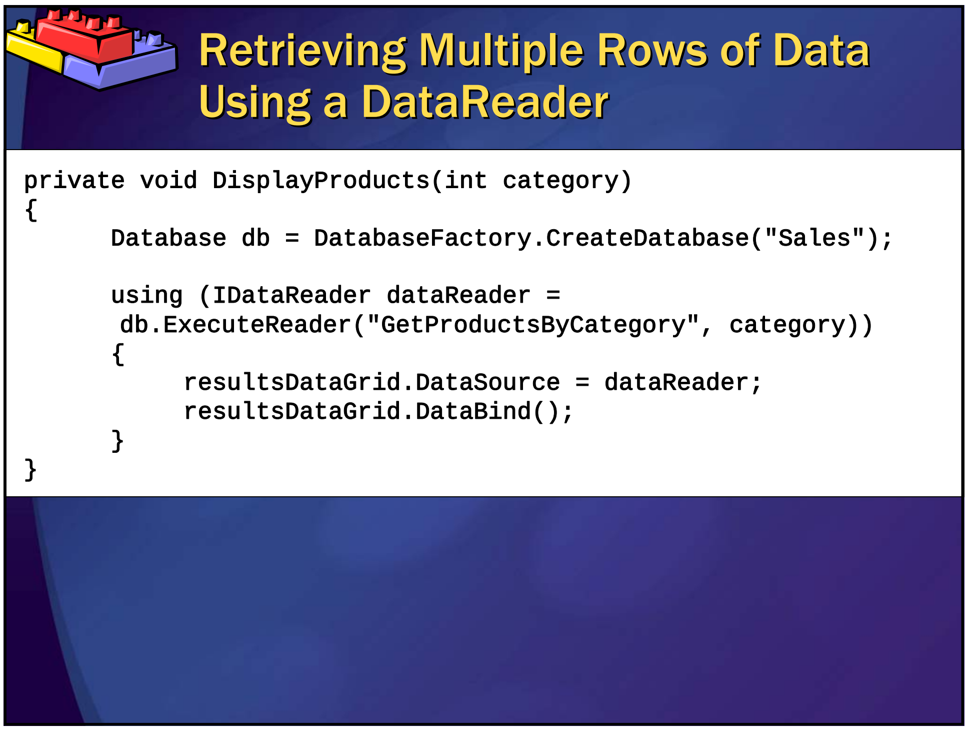
Application Architecture for .NET
Designing the Components of an Application or Service



Retrieving Multiple Rows of Data Using a DataSet

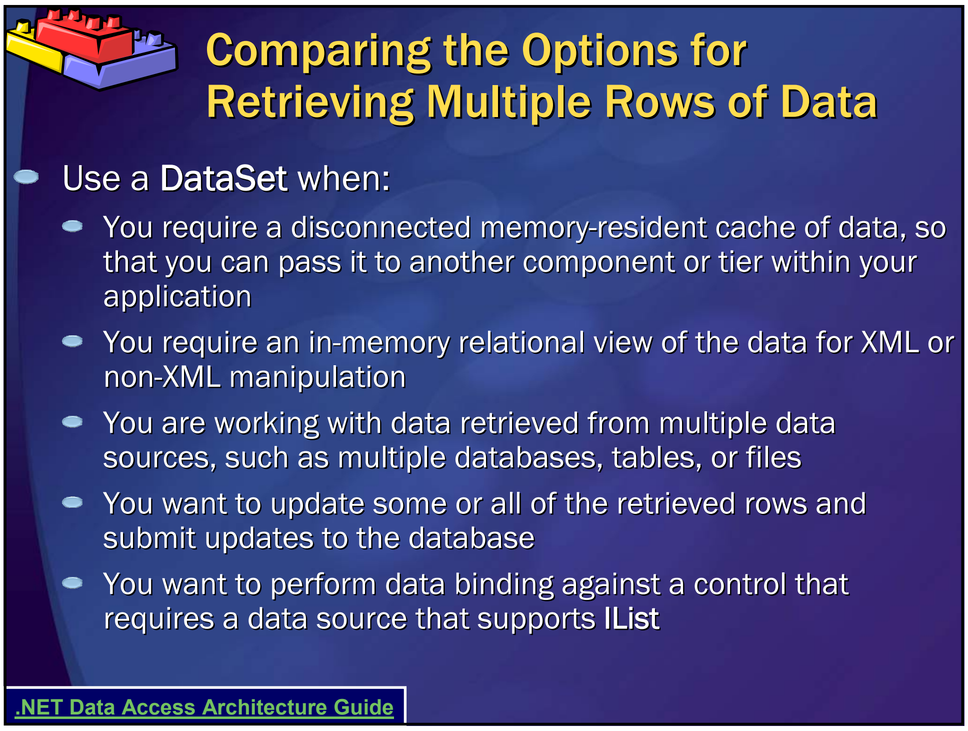
```
Public Function GetProductsInCategory(ByRef Category As Integer) As DataSet

    Dim db As Database = DatabaseFactory.CreateDatabase()

    ' Invoke the stored procedure with one line of code!
    return db.ExecuteDataSet("GetProductsByCategory", Category)

End Function
```

- ExecuteDataSet creates a new DataSet
- LoadDataSet can be used to fill an existing DataSet



Retrieving Multiple Rows of Data Using a DataReader

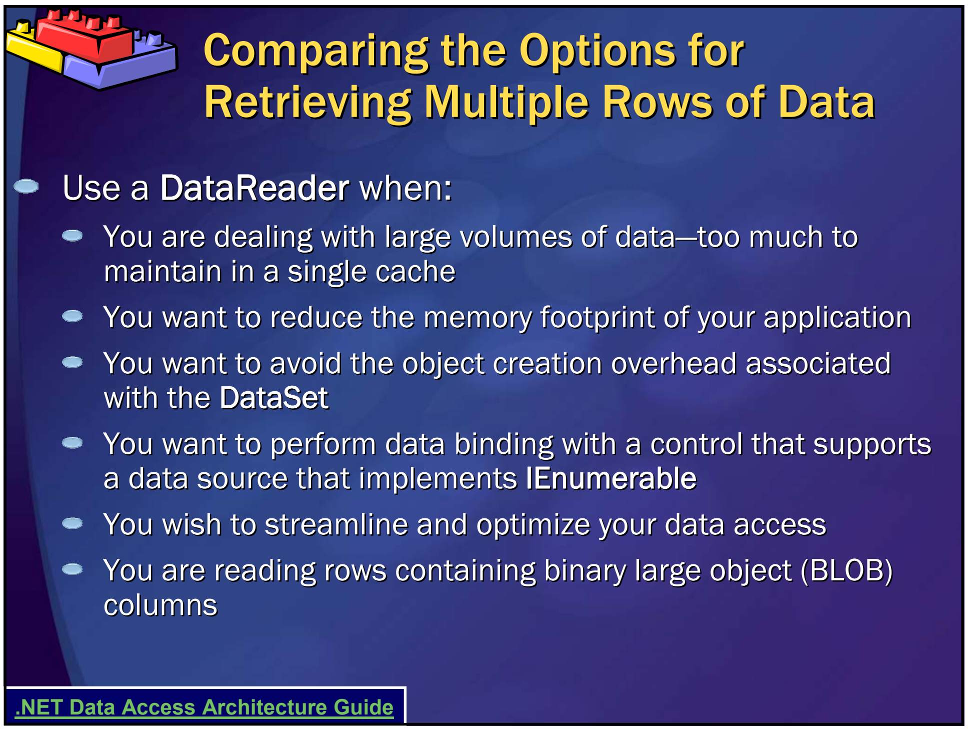
```
private void DisplayProducts(int category)
{
       Database db = DatabaseFactory.CreateDatabase("Sales");

       using (IDataReader dataReader =
        db.ExecuteReader("GetProductsByCategory", category))
       {
             resultsDataGrid.DataSource = dataReader;
             resultsDataGrid.DataBind();
       }
}
```



Comparing the Options for Retrieving Multiple Rows of Data

- Use a **DataSet** when:
  - You require a disconnected memory-resident cache of data, so that you can pass it to another component or tier within your application
  - You require an in-memory relational view of the data for XML or non-XML manipulation
  - You are working with data retrieved from multiple data sources, such as multiple databases, tables, or files
  - You want to update some or all of the retrieved rows and submit updates to the database
  - You want to perform data binding against a control that requires a data source that supports **IList**

.NET Data Access Architecture Guide



Comparing the Options for Retrieving Multiple Rows of Data

- Use a **DataReader** when:
  - You are dealing with large volumes of data—too much to maintain in a single cache
  - You want to reduce the memory footprint of your application
  - You want to avoid the object creation overhead associated with the **DataSet**
  - You want to perform data binding with a control that supports a data source that implements **IEnumerable**
  - You wish to streamline and optimize your data access
  - You are reading rows containing binary large object (BLOB) columns

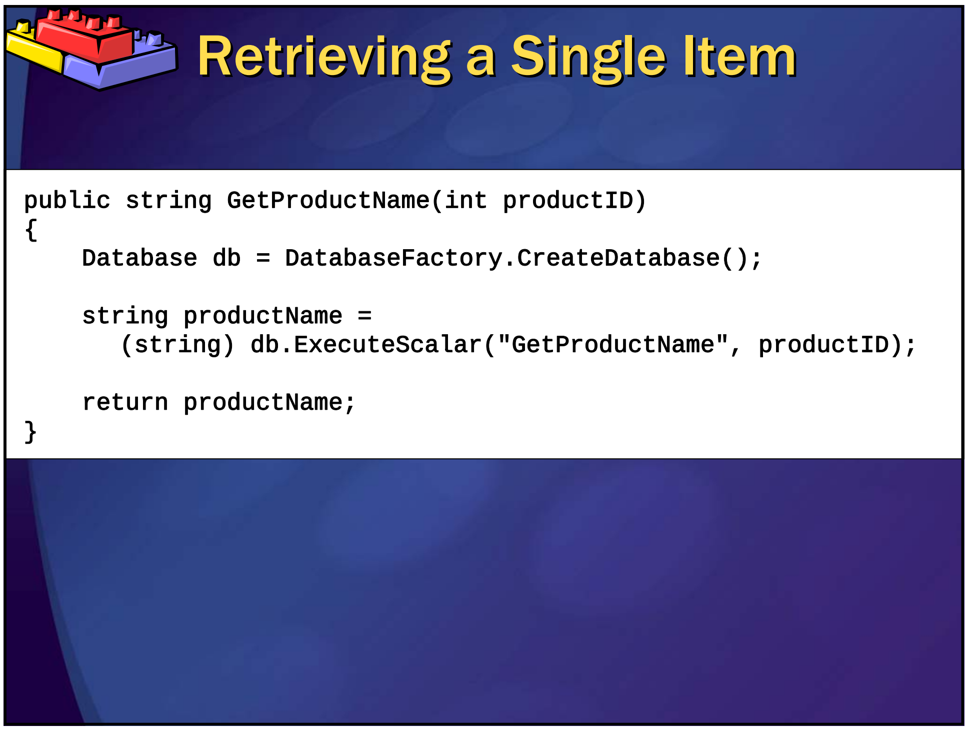
.NET Data Access Architecture Guide

Retrieving a Single Row Using Output Parameters

```
public string GetProductDetails(int productID)
{
    Database db = DatabaseFactory.CreateDatabase();

    string sqlCommand = "GetProductDetails";
    DBCommandWrapper dbCommandWrapper =
        db.GetStoredProcCommandWrapper(sqlCommand);

    dbCommandWrapper.AddInParameter("@ProductID", DbType.Int32, productID);
    dbCommandWrapper.AddOutParameter("@ProductName", DbType.String, 50);
    dbCommandWrapper.AddOutParameter("@UnitPrice", DbType.Currency, 8);

    db.ExecuteNonQuery(dbCommandWrapper);
    string results = string.Format(CultureInfo.CurrentCulture,
        "{0}, {1}, {2:C} ",
        dbCommandWrapper.GetParameterValue("@ProductID"),
        dbCommandWrapper.GetParameterValue("@ProductName"),
        dbCommandWrapper.GetParameterValue("@UnitPrice"));
    return results;
}
```

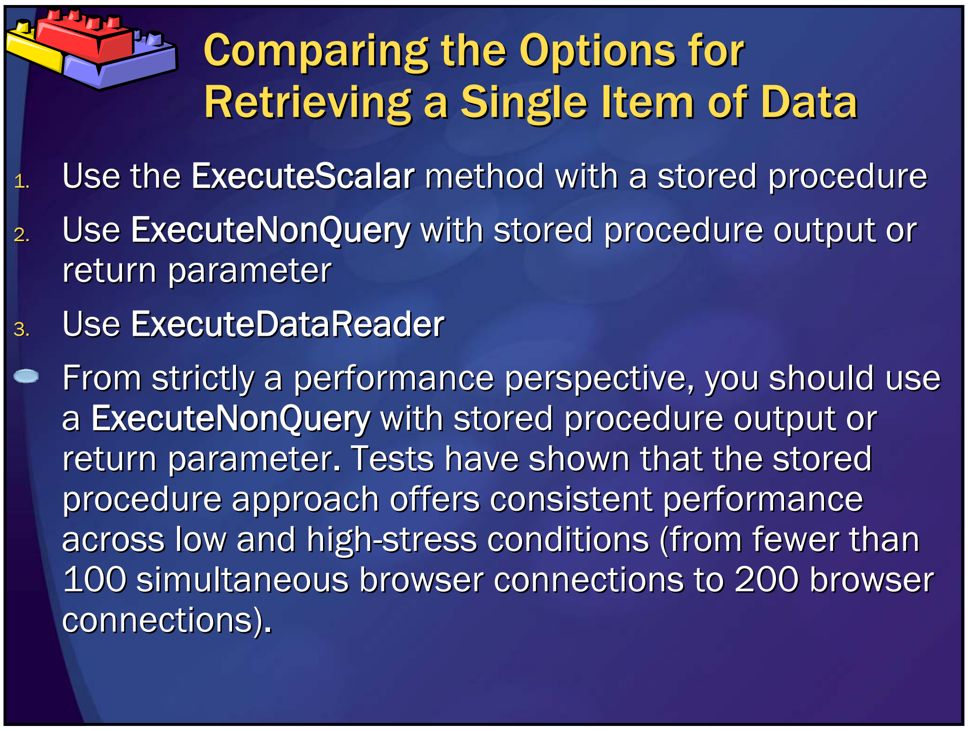
Comparing the Options for Retrieving a Single Row of Data

- You can use a DataSet or DataTable. However, unless you specifically require DataSet/DataTable functionality (for example, with data binding), you should avoid creating these objects.
- If you need to retrieve a single row, use one of the following options:
  1. Use ExecuteNonQuery to obtain stored procedure output parameters
  2. Use ExecuteDataReader to return an object that implements IDataReader



Retrieving a Single Item

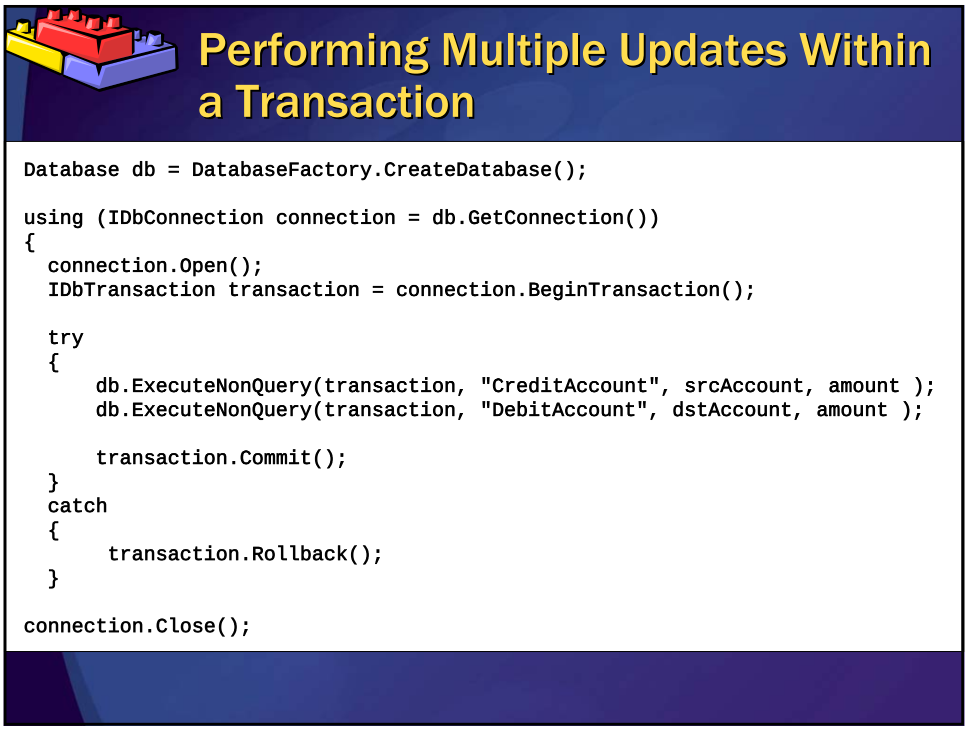
```
public string GetProductName(int productID)
{
    Database db = DatabaseFactory.CreateDatabase();

    string productName =
       (string) db.ExecuteScalar("GetProductName", productID);

    return productName;
}
```



Comparing the Options for Retrieving a Single Item of Data

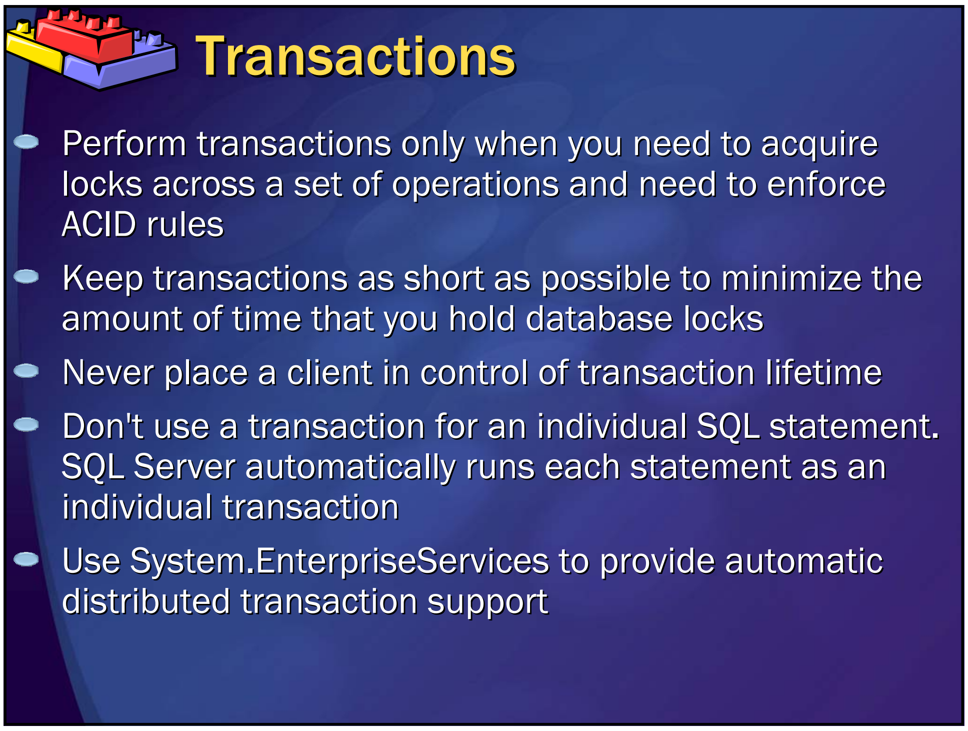
1. Use the **ExecuteScalar** method with a stored procedure
2. Use **ExecuteNonQuery** with stored procedure output or return parameter
3. Use **ExecuteDataReader**

- From strictly a performance perspective, you should use a **ExecuteNonQuery** with stored procedure output or return parameter. Tests have shown that the stored procedure approach offers consistent performance across low and high-stress conditions (from fewer than 100 simultaneous browser connections to 200 browser connections).



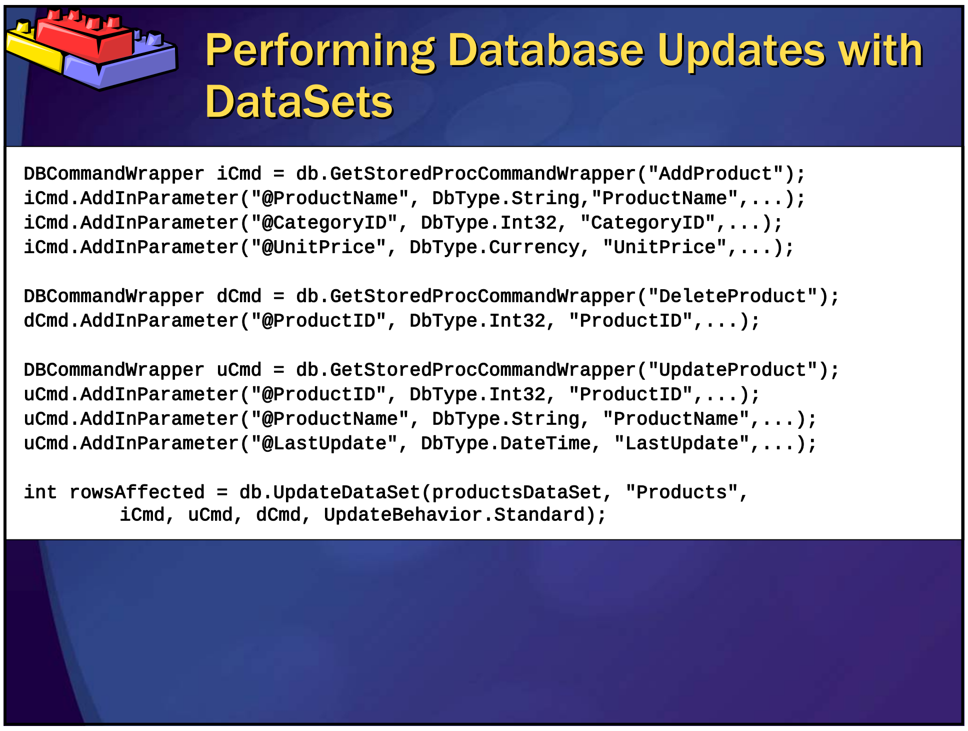
Performing Multiple Updates Within a Transaction

```
Database db = DatabaseFactory.CreateDatabase();

using (IDbConnection connection = db.GetConnection())
{
  connection.Open();
  IDbTransaction transaction = connection.BeginTransaction();

  try
  {
      db.ExecuteNonQuery(transaction, "CreditAccount", srcAccount, amount );
      db.ExecuteNonQuery(transaction, "DebitAccount", dstAccount, amount );

      transaction.Commit();
  }
  catch
  {
       transaction.Rollback();
  }

connection.Close();
```



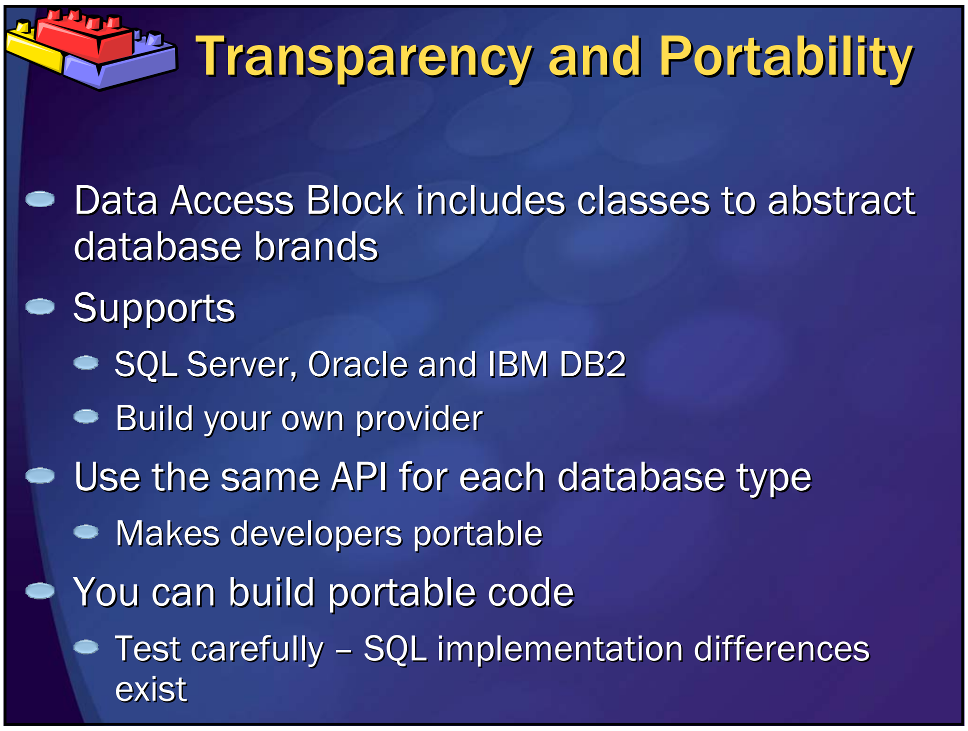
Transactions

- Perform transactions only when you need to acquire locks across a set of operations and need to enforce ACID rules
- Keep transactions as short as possible to minimize the amount of time that you hold database locks
- Never place a client in control of transaction lifetime
- Don't use a transaction for an individual SQL statement. SQL Server automatically runs each statement as an individual transaction
- Use System.EnterpriseServices to provide automatic distributed transaction support



Performing Database Updates with DataSets

```
DBCommandWrapper iCmd = db.GetStoredProcCommandWrapper("AddProduct");
iCmd.AddInParameter("@ProductName", DbType.String,"ProductName",...);
iCmd.AddInParameter("@CategoryID", DbType.Int32, "CategoryID",...);
iCmd.AddInParameter("@UnitPrice", DbType.Currency, "UnitPrice",...);

DBCommandWrapper dCmd = db.GetStoredProcCommandWrapper("DeleteProduct");
dCmd.AddInParameter("@ProductID", DbType.Int32, "ProductID",...);

DBCommandWrapper uCmd = db.GetStoredProcCommandWrapper("UpdateProduct");
uCmd.AddInParameter("@ProductID", DbType.Int32, "ProductID",...);
uCmd.AddInParameter("@ProductName", DbType.String, "ProductName",...);
uCmd.AddInParameter("@LastUpdate", DbType.DateTime, "LastUpdate",...);

int rowsAffected = db.UpdateDataSet(productsDataSet, "Products",
        iCmd, uCmd, dCmd, UpdateBehavior.Standard);
```



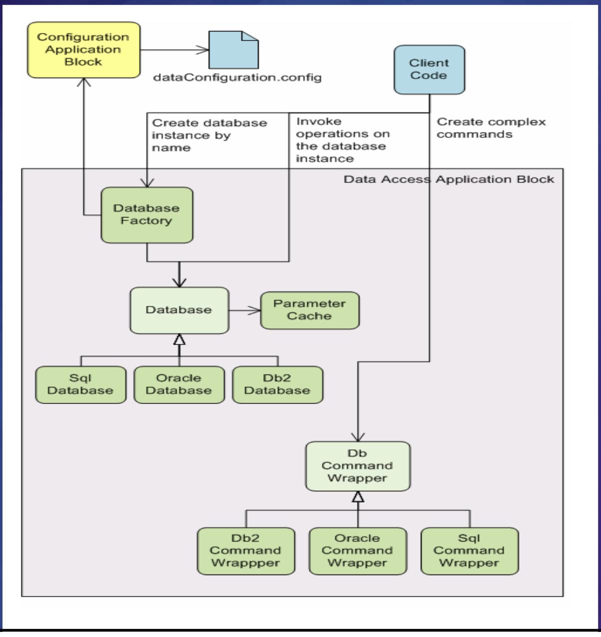
Transparency and Portability

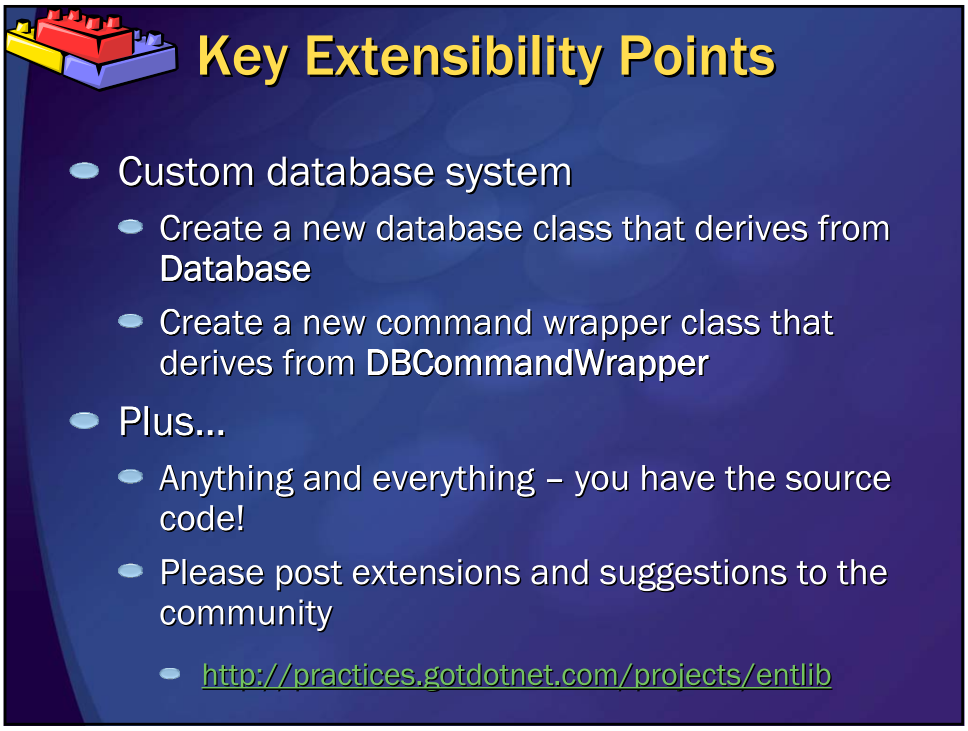
- Data Access Block includes classes to abstract database brands
- Supports
  - SQL Server, Oracle and IBM DB2
  - Build your own provider
- Use the same API for each database type
  - Makes developers portable
- You can build portable code
  - Test carefully – SQL implementation differences exist

SQL Server 2005 Express Edition

- http://www.microsoft.com/sql/editions/express/default.mspx

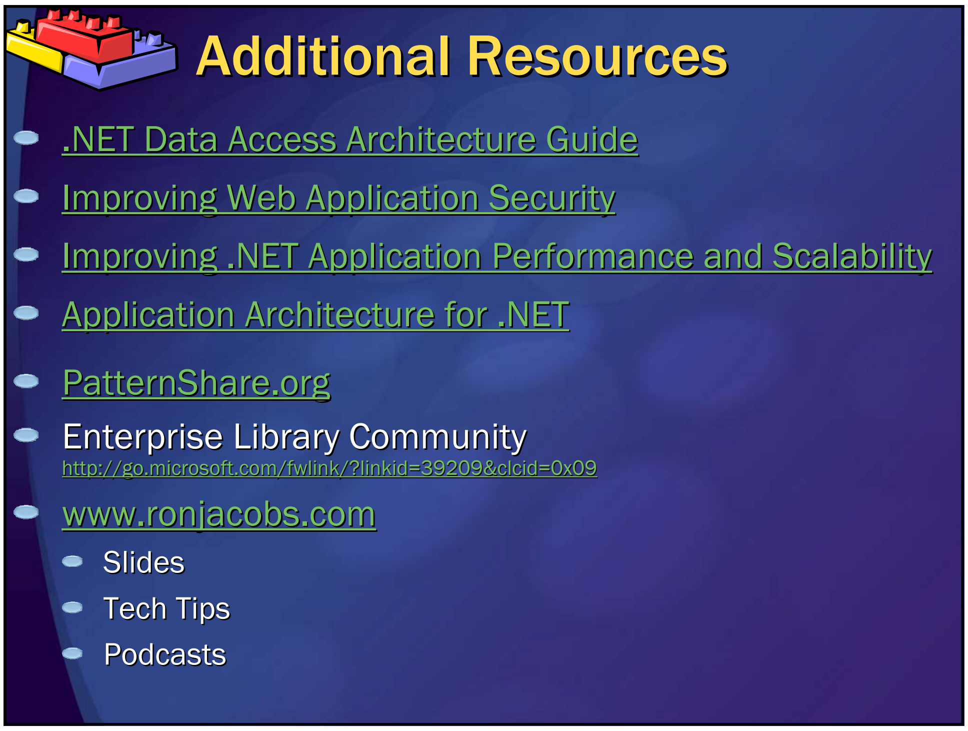
Data Access Diagram





Key Extensibility Points

- Custom database system
  - Create a new database class that derives from **Database**
  - Create a new command wrapper class that derives from **DBCommandWrapper**
- Plus…
  - Anything and everything – you have the source code!
  - Please post extensions and suggestions to the community
    - http://practices.gotdotnet.com/projects/entlib



Additional Resources

- .NET Data Access Architecture Guide
- Improving Web Application Security
- Improving .NET Application Performance and Scalability
- Application Architecture for .NET
- PatternShare.org
- Enterprise Library Community
  http://go.microsoft.com/fwlink/?linkid=39209&clcid=0x09
- www.ronjacobs.com
  - Slides
  - Tech Tips
  - Podcasts

Microsoft patterns & practices

proven practices for predictable results

Announcing: Enterprise Library 1.0

http://www.microsoft.com/practices

Download it Today!

Microsoft®

Your potential. Our passion.™

http://www.microsoft.com/practices