.NET Framework 4

**Secure Data Access (ADO.NET)**

To write secure ADO.NET code, you have to understand the security mechanisms available in the underlying data store, or database. You also need to consider the security implications of other features or components that your application may contain.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifAuthentication, Authorization and Permissions

When connecting to Microsoft SQL Server, you can use Windows Authentication, also known as Integrated Security, which uses the identity of the current active Windows user rather than passing a user ID and password. Using Windows Authentication is highly recommended because user credentials are not exposed in the connection string. If you cannot use Windows Authentication to connect to SQL Server, then consider creating connection strings at run time using the [SqlConnectionStringBuilder](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlconnectionstringbuilder.aspx).

The credentials used for authentication need to be handled differently based on the type of application. For example, in a Windows Forms application, the user can be prompted to supply authentication information, or the user's Windows credentials can be used. However, a Web application often accesses data using credentials supplied by the application itself rather than by the user.

Once users have been authenticated, the scope of their actions depends on the permissions that have been granted to them. Always follow the principle of least privilege and grant only permissions that are absolutely necessary.

For more information, see the following resources.

|  |  |
| --- | --- |
| **Resource** | **Description** |
| [Protecting Connection Information (ADO.NET)](http://msdn.microsoft.com/en-us/library/89211k9b.aspx) | Describes security best practices and techniques for protecting connection information, such as using protected configuration to encrypt connection strings. |
| [Recommendations for Data Access Strategies](http://msdn.microsoft.com/en-us/library/8fxztkff.aspx) | Provides recommendations for accessing data and performing database operations. |
| [Connection String Builders (ADO.NET)](http://msdn.microsoft.com/en-us/library/ms254947.aspx) | Describes how to build connection strings from user input at run time. |
| [Overview of SQL Server Security (ADO.NET)](http://msdn.microsoft.com/en-us/library/bb669078.aspx) | Describes the SQL Server security architecture. |

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifParameterized Commands and SQL Injection

Using parameterized commands helps guard against SQL injection attacks, in which an attacker "injects" a command into a SQL statement that compromises security on the server. Parameterized commands guard against a SQL injection attack by ensuring that values received from an external source are passed as values only, and not part of the Transact-SQL statement. As a result, Transact-SQL commands inserted into a value are not executed at the data source. Rather, they are evaluated solely as a parameter value. In addition to the security benefits, parameterized commands provide a convenient method for organizing values passed with a Transact-SQL statement or to a stored procedure.

For more information on using parameterized commands, see the following resources.

|  |  |
| --- | --- |
| **Resource** | **Description** |
| [DataAdapter Parameters (ADO.NET)](http://msdn.microsoft.com/en-us/library/bbw6zyha.aspx) | Describes how to use parameters with a **DataAdapter**. |
| [Modifying Data with Stored Procedures (ADO.NET)](http://msdn.microsoft.com/en-us/library/59x02y99.aspx) | Describes how to specify parameters and obtain a return value. |
| [Managing Permissions with Stored Procedures in SQL Server (ADO.NET)](http://msdn.microsoft.com/en-us/library/bb669058.aspx) | Describes how to use SQL Server stored procedures to encapsulate data access. |

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifScript Exploits

A script exploit is another form of injection that uses malicious characters inserted into a Web page. The browser does not validate the inserted characters and will process them as part of the page.

For more information, see the following resources.

|  |  |
| --- | --- |
| **Resource** | **Description** |
| [Script Exploits Overview](http://msdn.microsoft.com/en-us/library/w1sw53ds.aspx) | Describes how to guard against scripting and SQL statement exploits. |

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifProbing Attacks

Attackers often use information from an exception, such as the name of your server, database, or table, to mount an attack on your system. Because exceptions can contain specific information about your application or data source, you can help keep your application and data source better protected by only exposing essential information to the client.

For more information, see the following resources.

|  |  |
| --- | --- |
| **Resource** | **Description** |
| [Exception Handling Fundamentals](http://msdn.microsoft.com/en-us/library/2w8f0bss.aspx) | Describes the basic forms of try/catch/finally structured exception handling. |
| [Best Practices for Handling Exceptions](http://msdn.microsoft.com/en-us/library/seyhszts.aspx) | Describes best practices for handling exceptions. |

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifProtecting Microsoft Access and Excel Data Sources

Microsoft Access and Microsoft Excel can act as a data store for an ADO.NET application when security requirements are minimal or nonexistent. Their security features are effective for deterrence, but should not be relied upon to do more than discourage meddling by uninformed users. The physical data files for Access and Excel exist on the file system, and must be accessible to all users. This makes them vulnerable to attacks that could result in theft or data loss since the files can be easily copied or altered. When robust security is required, use SQL Server or another server-based database where the physical data files are not readable from the file system.

For more information on protecting Access and Excel data, see the following resources.

|  |  |
| --- | --- |
| **Resource** | **Description** |
| [Security Considerations and Guidance for Access 2007](http://go.microsoft.com/fwlink/?LinkId=98354) | Describes security techniques for Access 2007 such encrypting files, administering passwords, converting databases to the new ACCDB and ACCDE formats, and using other security options. |
| [Overview of Security and Protection in Excel](http://go.microsoft.com/fwlink/?LinkId=98355) | Describes how you can control who can access and change your Excel 2007 data. |
| [Help Protect an Access database with User-Level Security (MDB)](http://go.microsoft.com/fwlink/?LinkId=47697) | Applies to Access 2003. Provides instructions for implementing user-level security to protect data in Access 2003. |
| [Understanding the Role of Workgroup Information Files in Access Security](http://support.microsoft.com/kb/305542) | Explains the role and relationship of the workgroup information file in Access 2003 security. |
| [Frequently Asked Questions About Microsoft Access Security for Microsoft Access versions 2.0 through 2000](http://go.microsoft.com/fwlink/?LinkId=47698) | Downloadable version of the Microsoft Access Security FAQ. |
| [Troubleshoot Security and Protection](http://go.microsoft.com/fwlink/?LinkId=47703) | Presents solutions to common problems with security in Excel 2003. |

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifEnterprise Services

COM+ contains its own security model that relies on Windows NT accounts and process/thread impersonation. The [System.EnterpriseServices](http://msdn.microsoft.com/en-us/library/system.enterpriseservices.aspx) namespace provides wrappers that allow .NET applications to integrate managed code with COM+ security services through the [ServicedComponent](http://msdn.microsoft.com/en-us/library/system.enterpriseservices.servicedcomponent.aspx) class.

For more information, see the following resources.

|  |  |
| --- | --- |
| **Resource** | **Description** |
| [COM+ Role-Based Security and the .NET Framework](http://msdn.microsoft.com/en-us/library/s6y8k15h.aspx) | Discusses how to integrate managed code with COM+ security services. |
| [Writing Serviced Components](http://msdn.microsoft.com/en-us/library/3x7357ez.aspx) | Discusses how to use the classes in the **EnterpriseServices** namespace to create serviced components. |

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifInteroperating with Unmanaged Code

The .NET Framework provides for interaction with unmanaged code, including COM components, COM+ services, external type libraries, and many operating system services. Working with unmanaged code involves going outside the security perimeter for managed code. Both your code and any code that calls it must have unmanaged code permission ([SecurityPermission](http://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermission.aspx) with the [UnmanagedCode](http://msdn.microsoft.com/en-us/library/system.security.permissions.securitypermissionflag.aspx) flag specified). Unmanaged code can introduce unintended security vulnerabilities into your application. Therefore, you should avoid interoperating with unmanaged code unless it is absolutely necessary.

For more information, see the following resources.

|  |  |
| --- | --- |
| **Resource** | **Description** |
| [Interoperating with Unmanaged Code](http://msdn.microsoft.com/en-us/library/sd10k43k.aspx) | Contains topics describing how to expose COM components to the .NET Framework and how to expose .NET Framework components to COM. |
| [Advanced COM Interoperability](http://msdn.microsoft.com/en-us/library/bd9cdfyx.aspx) | Contains advanced topics such as primary interop assemblies, threading and custom marshaling. |

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifSee Also

**Concepts**

[Protecting Connection Information (ADO.NET)](http://msdn.microsoft.com/en-us/library/89211k9b.aspx)

[Connection String Builders (ADO.NET)](http://msdn.microsoft.com/en-us/library/ms254947.aspx)

**Other Resources**

[Securing ADO.NET Applications](http://msdn.microsoft.com/en-us/library/ecb3hak0.aspx)

[SQL Server Security (ADO.NET)](http://msdn.microsoft.com/en-us/library/bb669074.aspx)

[Recommendations for Data Access Strategies](http://msdn.microsoft.com/en-us/library/8fxztkff.aspx)

[ADO.NET Managed Providers and DataSet Developer Center](http://go.microsoft.com/fwlink/?LinkId=217917)

.NET Framework 4

**Protecting Connection Information (ADO.NET)**

Protecting access to your data source is one of the most important goals when securing an application. A connection string presents a potential vulnerability if it is not secured. Storing connection information in plain text or persisting it in memory risks compromising your entire system. Connection strings embedded in your source code can be read using the [Ildasm.exe (MSIL Disassembler)](http://msdn.microsoft.com/en-us/library/f7dy01k1.aspx) to view Microsoft intermediate language (MSIL) in a compiled assembly.

Security vulnerabilities involving connection strings can arise based on the type of authentication used, how connection strings are persisted in memory and on disk, and the techniques used to construct them at run time.

Use Windows Authentication

To help limit access to your data source, you must secure connection information such as user ID, password, and data source name. In order to avoid exposing user information, we recommend using Windows authentication (sometimes referred to as *integrated security*) wherever possible. Windows authentication is specified in a connection string by using the Integrated Security or Trusted\_Connection keywords, eliminating the need to use a user ID and password. When using Windows authentication, users are authenticated by Windows, and access to server and database resources is determined by granting permissions to Windows users and groups.

For situations where it is not possible to use Windows authentication, you must use extra care because user credentials are exposed in the connection string. In an ASP.NET application, you can configure a Windows account as a fixed identity that is used to connect to databases and other network resources. You enable impersonation in the identity element in the **web.config** file and specify a user name and password.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl17_ctl00_ctl00_code');" \o "Copy Code)

<identity impersonate="true"

userName="MyDomain\UserAccount"

password="\*\*\*\*\*" />

The fixed identity account should be a low-privilege account that has been granted only necessary permissions in the database. In addition, you should encrypt the configuration file so that the user name and password are not exposed in clear text.

Do Not Use Universal Data Link (UDL) files

Avoid storing connection strings for an [OleDbConnection](http://msdn.microsoft.com/en-us/library/system.data.oledb.oledbconnection.aspx) in a Universal Data Link (UDL) file. UDLs are stored in clear text and cannot be encrypted. A UDL file is an external file-based resource to your application, and it cannot be secured or encrypted using the .NET Framework.

Avoid Injection Attacks with Connection String Builders

A connection string injection attack can occur when dynamic string concatenation is used to build connection strings based on user input. If the user input is not validated and malicious text or characters not escaped, an attacker can potentially access sensitive data or other resources on the server. To address this problem, ADO.NET 2.0 introduced new connection string builder classes to validate connection string syntax and ensure that additional parameters are not introduced. For more information, see [Connection String Builders (ADO.NET)](http://msdn.microsoft.com/en-us/library/ms254947.aspx).

Use Persist Security Info=False

The default value for **Persist Security Info** is false; we recommend using this default in all connection strings. Setting **Persist Security Info** to **true** or **yes** allows security-sensitive information, including the user ID and password, to be obtained from a connection after it has been opened. When **Persist Security Info** is set to **false** or **no**, security information is discarded after it is used to open the connection, ensuring that an untrusted source does not have access to security-sensitive information.

Encrypt Configuration Files

You can also store connection strings in configuration files, which eliminates the need to embed them in your application's code. Configuration files are standard XML files for which the .NET Framework has defined a common set of elements. Connection strings in configuration files are typically stored inside the **<connectionStrings>** element in the **app.config** for a Windows application, or the **web.config** file for an ASP.NET application. For more information on the basics of storing, retrieving and encrypting connection strings from configuration files, see [Connection Strings and Configuration Files (ADO.NET)](http://msdn.microsoft.com/en-us/library/ms254494.aspx).

Visual Studio 2010

**Recommendations for Data Access Strategies**

[This documentation is for preview only, and is subject to change in later releases. Blank topics are included as placeholders.]

ADO.NET assumes a model for data access in which you open a connection, get data or perform an operation, and then close the connection. ADO.NET provides two basic strategies for how you work with this model. One model is store data in a dataset, which is an in-memory cache of records you can work with while disconnected from the data source. To use a dataset, you create an instance of it and then use a data adapter to fill it from the data source. You then work with the data in the dataset — for example, by binding controls to dataset members. For more information, see [Working with Datasets in Visual Studio](http://msdn.microsoft.com/en-us/library/8bw9ksd6.aspx).

An alternative strategy is to perform operations directly against the database. In this model, you use a TableAdapter query or data command that includes an SQL statement or a reference to a stored procedure. You can then execute the query or command to perform the operation. For more information, see [Fetching Data into Your Application](http://msdn.microsoft.com/en-us/library/ms171918.aspx).

Storing Data in Datasets

A common model for data access in Visual Studio applications is to store data in datasets and use TableAdapters or data adapters to read and write data in the database. The advantages of the dataset model are:

* Working with multiple tables A dataset can contain multiple tables of results, which it maintains as discrete objects. You can work with the tables individually or navigate between them as parent-child tables.
* Manipulating data from multiple sources The tables in a dataset can represent data from many different sources (for example, from different databases, from XML files, spreadsheets, and so on, all in the same dataset). Once the data is in the dataset, you can manipulate it and relate it in a homogeneous format as if it had come from a single source.
* Moving data between tiers in a distributed application By keeping data in a dataset, you can easily move it between the presentation tier, business tier, and data tier of your applications.
* Data exchange with other applications A dataset provides a powerful way to exchange with other components of your application and with other applications. Datasets include extensive support for features such as serializing data as XML and reading and writing XML Schemas.
* Data binding If you are working with forms, it is usually easier to bind controls to data in a dataset than it is to programmatically load data values into the control after executing a command.
* Maintaining records for reuse A dataset allows you to work with the same records repeatedly without requerying the database. Using dataset facilities, you can filter and sort records, and you can use the dataset as a source of data if you are paging.
* Ease of programming When you work with a dataset, you can generate a class file that represents its structure as objects (for example, a Customers table in the dataset can be accessed as the dataset.Customers object). This makes it easier, clearer, and less error-prone to program with, and is supported by Visual Studio tools such as IntelliSense, the Data Adapter Configuration wizard, and so on.

Performing Database Operations Directly

Alternatively, you can interact with the database directly. In this model, you use a data command object that includes an SQL statement or a reference to a stored procedure. You can then execute the command to perform the operation. For more information, see [Commands and Parameters (ADO.NET)](http://msdn.microsoft.com/en-us/library/ms254953.aspx).

|  |
| --- |
| **Security noteSecurity Note** |
| When using data commands with a [CommandType](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlcommand.commandtype.aspx) property set to [Text](http://msdn.microsoft.com/en-us/library/system.data.commandtype.aspx), carefully check information that is sent from a client before passing it to your database. Malicious users might try to send (inject) modified or additional SQL statements in an effort to gain unauthorized access or damage the database. Before you transfer user input to a database, you should always verify that the information is valid. A best practice is to always use parameterized queries or stored procedures when possible. |

Performing database operations directly has specific advantages, which include:

* Extra functionality As noted, there are some operations, such as executing DDL commands, that you can only accomplish by executing data commands.
* More control over execution By using commands (and a data reader, if you are reading data), you get more direct control over how and when an SQL statement or stored procedure is executed and what becomes of the results or return values.
* Less overhead By reading and writing directly to the database, you can bypass storing data in a dataset. Because the dataset requires memory, you can reduce some overhead in your application. This is especially true in situations where you intend to use the data only once, such as displaying search results in a Web page. In that case, creating and filling a dataset might be an unnecessary step in displaying the data.
* Less programming in some instances In a few instances, particularly Web applications, there is some extra programming required to save the state of a dataset. For example, in Web Forms pages, the page is recreated with each round trip; unless you add programming to save and restore a dataset, it, too, is discarded and recreated with each round trip. If you use a data reader to read directly from the database, you avoid the extra steps required to manage the dataset.

Recommendations for Accessing Data

The following sections provide recommendations for which data-access strategy to use with specific types of applications.

### Windows Forms

In general, in a Windows Form, use a dataset. Windows Forms are typically used on rich clients where the form is not created and discarded (along with its data) with each user operation, as with Web Forms. Windows Forms applications also traditionally offer data-access scenarios that benefit from maintaining a cache of records, such as displaying records one by one in the form. For more information see, [Creating Data Applications](http://msdn.microsoft.com/en-us/library/h0y4a0f6.aspx).

Specifically, use dataset under the following circumstances:

* If you are working with the same records repeatedly, such as allowing a user to navigate between records.
* If you are using the Windows Forms data-binding architecture, which is specifically designed to work with datasets.
* For any of the other reasons listed under Web Forms above.

Use a TableAdapter query or data command under the following circumstances:

* If you are getting a scalar value from the database
* If you are performing a non-query operation, such as a DDL command.
* If you are getting read-only data to display in a form — for example, creating a report. Stated differently, if there is no need to keep the data available after accessing it, use a data command.

### Web Forms

In general, use data commands; to fetch data, use a data reader. Because Web Forms pages and their controls and components are recreated each time the page makes a round trip, it often is not efficient to create and fill a dataset each time, unless you also intend to cache it between round trips.

Use dataset under the following circumstances:

* You want to work with multiple separate tables or tables from different data sources.
* You are exchanging data with another application or a component such as an XML Web service.
* You need to perform extensive processing with each record you get from the database. If you use a data command and data reader, processing each record as you read it can result in the connection being held open for a long period, which in turn can affect the performance and scalability of your application.
* If your data processing involves interdependent records (for example, looking up information in related records).
* If you want to perform XML operations such as XSLT transformations on the data.
* If you prefer the ease of programming provided by datasets.

### XML Web Services

XML Web services are ASP.NET Web applications, and therefore use the same model as Web Forms pages: the XML Web service is created and discarded each time a call is made to it. This suggests that the data-access model for an XML Web service is largely the same as it is for Web Forms. However, XML Web services are often middle-tier objects, and an important part of their purpose is often to exchange data with other applications across the Web.

Use a dataset if:

* Your XML Web service sends and receives data — for example, sending it as the return value of a method and receiving it as a method argument. This is a fundamental choice in XML Web services; even if there are other reasons you might consider using a data command, data exchange with other components almost always means that you should use a dataset.
* For any of the reasons listed above for Web Forms.

Use a data command (and if appropriate, a data reader) under the following circumstances:

* The XML Web service is retrieving a scalar value.
* The XML Web service is performing a non-query operation, such as a DDL command.
* The XML Web service is calling a stored procedure to execute logic within the database.

.NET Framework 4

**Connection String Builders (ADO.NET)**

In earlier versions of ADO.NET, compile-time checking of connection strings with concatenated string values did not occur, so that at run time, an incorrect keyword generated an [ArgumentException](http://msdn.microsoft.com/en-us/library/system.argumentexception.aspx). Each of the .NET Framework data providers supported different syntax for connection string keywords, which made constructing valid connection strings difficult if done manually. To address this problem, ADO.NET 2.0 introduced new connection string builders for each .NET Framework data provider. Each data provider includes a strongly typed connection string builder class that inherits from [DbConnectionStringBuilder](http://msdn.microsoft.com/en-us/library/system.data.common.dbconnectionstringbuilder.aspx). The following table lists the .NET Framework data providers and their associated connection string builder classes.

|  |  |
| --- | --- |
| **Provider** | **ConnectionStringBuilder class** |
| [System.Data.SqlClient](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.aspx) | [System.Data.SqlClient..::.SqlConnectionStringBuilder](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlconnectionstringbuilder.aspx) |
| [System.Data.OleDb](http://msdn.microsoft.com/en-us/library/system.data.oledb.aspx) | [System.Data.OleDb..::.OleDbConnectionStringBuilder](http://msdn.microsoft.com/en-us/library/system.data.oledb.oledbconnectionstringbuilder.aspx) |
| [System.Data.Odbc](http://msdn.microsoft.com/en-us/library/system.data.odbc.aspx) | [System.Data.Odbc..::.OdbcConnectionStringBuilder](http://msdn.microsoft.com/en-us/library/system.data.odbc.odbcconnectionstringbuilder.aspx) |
| [System.Data.OracleClient](http://msdn.microsoft.com/en-us/library/system.data.oracleclient.aspx) | [System.Data.OracleClient..::.OracleConnectionStringBuilder](http://msdn.microsoft.com/en-us/library/system.data.oracleclient.oracleconnectionstringbuilder.aspx) |

Connection String Injection Attacks

A connection string injection attack can occur when dynamic string concatenation is used to build connection strings that are based on user input. If the string is not validated and malicious text or characters not escaped, an attacker can potentially access sensitive data or other resources on the server. For example, an attacker could mount an attack by supplying a semicolon and appending an additional value. The connection string is parsed by using a "last one wins" algorithm, and the hostile input is substituted for a legitimate value.

The connection string builder classes are designed to eliminate guesswork and protect against syntax errors and security vulnerabilities. They provide methods and properties corresponding to the known key/value pairs permitted by each data provider. Each class maintains a fixed collection of synonyms and can translate from a synonym to the corresponding well-known key name. Checks are performed for valid key/value pairs and an invalid pair throws an exception. In addition, injected values are handled in a safe manner.

The following example demonstrates how the [SqlConnectionStringBuilder](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlconnectionstringbuilder.aspx) handles an inserted extra value for the Initial Catalog setting.

Visual Basic

Dim builder As New System.Data.SqlClient.SqlConnectionStringBuilder

builder("Data Source") = "(local)"

builder("Integrated Security") = True

builder("Initial Catalog") = "AdventureWorks;NewValue=Bad"

Console.WriteLine(builder.ConnectionString)

C#

System.Data.SqlClient.SqlConnectionStringBuilder builder =

new System.Data.SqlClient.SqlConnectionStringBuilder();

builder["Data Source"] = "(local)";

builder["integrated Security"] = true;

builder["Initial Catalog"] = "AdventureWorks;NewValue=Bad";

Console.WriteLine(builder.ConnectionString);

The output shows that the [SqlConnectionStringBuilder](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlconnectionstringbuilder.aspx) handled this correctly by escaping the extra value in double quotation marks instead of appending it to the connection string as a new key/value pair.

data source=(local);Integrated Security=True;

initial catalog="AdventureWorks;NewValue=Bad"

Building Connection Strings from Configuration Files

If certain elements of a connection string are known beforehand, they can be stored in a configuration file and retrieved at run time to construct a complete connection string. For example, the name of the database might be known in advance, but not the name of the server. Or you might want a user to supply a name and password at run time without being able to inject other values into the connection string.

One of the overloaded constructors for a connection string builder takes a [String](http://msdn.microsoft.com/en-us/library/system.string.aspx) as an argument, which enables you to supply a partial connection string that can then be completed from user input. The partial connection string can be stored in a configuration file and retrieved at run time.

|  |
| --- |
| **NoteNote** |
| The [System.Configuration](http://msdn.microsoft.com/en-us/library/system.configuration.aspx) namespace allows programmatic access to configuration files that use the [WebConfigurationManager](http://msdn.microsoft.com/en-us/library/system.web.configuration.webconfigurationmanager.aspx) for Web applications and the [ConfigurationManager](http://msdn.microsoft.com/en-us/library/system.configuration.configurationmanager.aspx) for Windows applications. For more information about working with connection strings and configuration files, see [Connection Strings and Configuration Files (ADO.NET)](http://msdn.microsoft.com/en-us/library/ms254494.aspx). |

### Example

This example demonstrates retrieving a partial connection string from a configuration file and completing it by setting the [DataSource](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlconnectionstringbuilder.datasource.aspx), [UserID](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlconnectionstringbuilder.userid.aspx), and [Password](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlconnectionstringbuilder.password.aspx) properties of the [SqlConnectionStringBuilder](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlconnectionstringbuilder.aspx). The configuration file is defined as follows.

<connectionStrings>

<clear/>

<add name="partialConnectString"

connectionString="Initial Catalog=Northwind;"

providerName="System.Data.SqlClient" />

</connectionStrings>

|  |
| --- |
| **NoteNote** |
| You must set a reference to the System.Configuration.dll in your project for the code to run. |

Visual Basic

Private Sub BuildConnectionString(ByVal dataSource As String, \_

ByVal userName As String, ByVal userPassword As String)

' Retrieve the partial connection string named databaseConnection

' from the application's app.config or web.config file.

Dim settings As ConnectionStringSettings = \_

ConfigurationManager.ConnectionStrings("partialConnectString")

If Not settings Is Nothing Then

' Retrieve the partial connection string.

Dim connectString As String = settings.ConnectionString

Console.WriteLine("Original: {0}", connectString)

' Create a new SqlConnectionStringBuilder based on the

' partial connection string retrieved from the config file.

Dim builder As New SqlConnectionStringBuilder(connectString)

' Supply the additional values.

builder.DataSource = dataSource

builder.UserID = userName

builder.Password = userPassword

Console.WriteLine("Modified: {0}", builder.ConnectionString)

End If

End Sub

C#

private static void BuildConnectionString(string dataSource,

string userName, string userPassword)

{

// Retrieve the partial connection string named databaseConnection

// from the application's app.config or web.config file.

ConnectionStringSettings settings =

ConfigurationManager.ConnectionStrings["partialConnectString"];

if (null != settings)

{

// Retrieve the partial connection string.

string connectString = settings.ConnectionString;

Console.WriteLine("Original: {0}", connectString);

// Create a new SqlConnectionStringBuilder based on the

// partial connection string retrieved from the config file.

SqlConnectionStringBuilder builder =

new SqlConnectionStringBuilder(connectString);

// Supply the additional values.

builder.DataSource = dataSource;

builder.UserID = userName;

builder.Password = userPassword;

Console.WriteLine("Modified: {0}", builder.ConnectionString);

}

}

.NET Framework 4

**Overview of SQL Server Security (ADO.NET)**

A defense-in-depth strategy, with overlapping layers of security, is the best way to counter security threats. SQL Server provides a security architecture that is designed to allow database administrators and developers to create secure database applications and counter threats. Each version of SQL Server has improved on previous versions of SQL Server with the introduction of new features and functionality. However, security does not ship in the box. Each application is unique in its security requirements. Developers need to understand which combination of features and functionality are most appropriate to counter known threats, and to anticipate threats that may arise in the future.

SQL Server 2005 introduced many improvements to the SQL Server 2000 security framework, but the basic security architecture remains unchanged. A SQL Server instance contains a hierarchical collection of entities, starting with the server. Each server contains multiple databases, and each database contains a collection of securable objects. Every SQL Server securable has associated *permissions* that can be granted to a *principal*, which is an individual, group or process granted access to SQL Server. The SQL Server security framework manages access to securable entities through *authentication* and *authorization*.

* Authentication is the process of logging on to SQL Server by which a principal requests access by submitting credentials that the server evaluates. Authentication establishes the identity of the user or process being authenticated.
* Authorization is the process of determining which securable resources a principal can access, and which operations are allowed for those resources.

The topics in this section cover SQL Server security fundamentals, providing links to the complete documentation in the relevant version of SQL Server Books Online.

In This Section

[Authentication in SQL Server (ADO.NET)](http://msdn.microsoft.com/en-us/library/bb669066.aspx)

Describes logins and authentication in SQL Server and provides links to additional resources.

[Server and Database Roles in SQL Server (ADO.NET)](http://msdn.microsoft.com/en-us/library/bb669065.aspx)

Describes fixed server and database roles, custom database roles, and built-in accounts and provides links to additional resources.

[Ownership and User-Schema Separation in SQL Server (ADO.NET)](http://msdn.microsoft.com/en-us/library/bb669061.aspx)

Describes object ownership and user-schema separation and provides links to additional resources.

[Authorization and Permissions in SQL Server (ADO.NET)](http://msdn.microsoft.com/en-us/library/bb669084.aspx)

Describes granting permissions using the principle of least privilege and provides links to additional resources.

[Data Encryption in SQL Server (ADO.NET)](http://msdn.microsoft.com/en-us/library/bb669072.aspx)

Describes data encryption options in SQL Server and provides links to additional resources.

[CLR Integration Security in SQL Server (ADO.NET)](http://msdn.microsoft.com/en-us/library/bb669064.aspx)

Provides links to CLR integration security resources.

.NET Framework 4

**DataAdapter Parameters (ADO.NET)**

The [DbDataAdapter](http://msdn.microsoft.com/en-us/library/system.data.common.dbdataadapter.aspx) has four properties that are used to retrieve data from and update data to the data source: the [SelectCommand](http://msdn.microsoft.com/en-us/library/system.data.common.dbdataadapter.selectcommand.aspx) property returns data from the data source; and the [InsertCommand](http://msdn.microsoft.com/en-us/library/system.data.common.dbdataadapter.insertcommand.aspx) , [UpdateCommand](http://msdn.microsoft.com/en-us/library/system.data.common.dbdataadapter.updatecommand.aspx), and [DeleteCommand](http://msdn.microsoft.com/en-us/library/system.data.common.dbdataadapter.deletecommand.aspx) properties are used to manage changes at the data source. The **SelectCommand** property must be set before you call the **Fill** method of the **DataAdapter**. The **InsertCommand**, **UpdateCommand**, or **DeleteCommand** properties must be set before the **Update** method of the **DataAdapter** is called, depending on what changes were made to the data in the [DataTable](http://msdn.microsoft.com/en-us/library/system.data.datatable.aspx). For example, if rows have been added, the **InsertCommand** must be set before you call **Update**. When **Update** is processing an inserted, updated, or deleted row, the **DataAdapter** uses the respective **Command** property to process the action. Current information about the modified row is passed to the **Command** object through the **Parameters** collection.

When you update a row at the data source, you call the UPDATE statement, which uses a unique identifier to identify the row in the table be updated. The unique identifier is typically the value of a primary key field. The UPDATE statement uses parameters that contain both the unique identifier and the columns and values to be updated, as shown in the following Transact-SQL statement.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl07_code');" \o "Copy Code)

UPDATE Customers SET CompanyName = @CompanyName

WHERE CustomerID = @CustomerID

|  |
| --- |
| **NoteNote** |
| The syntax for parameter placeholders depends on the data source. This example shows placeholders for a SQL Server data source. Use question mark (?) placeholders for [System.Data.OleDb](http://msdn.microsoft.com/en-us/library/system.data.oledb.aspx) and [System.Data.Odbc](http://msdn.microsoft.com/en-us/library/system.data.odbc.aspx) parameters. |

In this Visual Basic example, the **CompanyName** field is updated with the value of the @CompanyName parameter for the row where **CustomerID** equals the value of the @CustomerIDparameter. The parameters retrieve information from the modified row using the [SourceColumn](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlparameter.sourcecolumn.aspx) property of the [SqlParameter](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlparameter.aspx) object. The following are the parameters for the previous sample UPDATE statement. The code assumes that the variable *adapter* represents a valid [SqlDataAdapter](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldataadapter.aspx) object.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl14_code');" \o "Copy Code)

adapter.Parameters.Add( \_

"@CompanyName", SqlDbType.NChar, 15, "CompanyName")

Dim parameter As SqlParameter = \_

adapter.UpdateCommand.Parameters.Add("@CustomerID", \_

SqlDbType.NChar, 5, "CustomerID")

parameter.SourceVersion = DataRowVersion.Original

The **Add** method of the **Parameters** collection takes the name of the parameter, the data type, the size (if applicable to the type), and the name of the [SourceColumn](http://msdn.microsoft.com/en-us/library/system.data.common.dbparameter.sourcecolumn.aspx) from the **DataTable**. Notice that the [SourceVersion](http://msdn.microsoft.com/en-us/library/system.data.common.dbparameter.sourceversion.aspx) of the @CustomerID parameter is set to **Original**. This guarantees that the existing row in the data source is updated if the value of the identifying column or columns has been changed in the modified [DataRow](http://msdn.microsoft.com/en-us/library/system.data.datarow.aspx). In that case, the **Original** row value would match the current value at the data source, and the **Current** row value would contain the updated value. The **SourceVersion** for the @CompanyName parameter is not set and uses the default, **Current** row value.

|  |
| --- |
| **NoteNote** |
| For both the **Fill** operations of the **DataAdapter** and the **Get** methods of the **DataReader**, the .NET Framework type is inferred from the type returned from the .NET Framework data provider. The inferred .NET Framework types and accessor methods for Microsoft SQL Server, OLE DB, and ODBC data types are described in [Data Type Mappings in ADO.NET](http://msdn.microsoft.com/en-us/library/4e5xt97a.aspx). |

Parameter.SourceColumn, Parameter.SourceVersion

The **SourceColumn** and **SourceVersion** may be passed as arguments to the **Parameter** constructor, or set as properties of an existing **Parameter**. The **SourceColumn** is the name of the [DataColumn](http://msdn.microsoft.com/en-us/library/system.data.datacolumn.aspx) from the [DataRow](http://msdn.microsoft.com/en-us/library/system.data.datarow.aspx) where the value of the **Parameter** will be retrieved. The **SourceVersion** specifies the **DataRow** version that the **DataAdapter** uses to retrieve the value.

The following table shows the [DataRowVersion](http://msdn.microsoft.com/en-us/library/system.data.datarowversion.aspx) enumeration values available for use with **SourceVersion**.

|  |  |
| --- | --- |
| **DataRowVersion Enumeration** | **Description** |
| **Current** | The parameter uses the current value of the column. This is the default. |
| **Default** | The parameter uses the **DefaultValue** of the column. |
| **Original** | The parameter uses the original value of the column. |
| **Proposed** | The parameter uses a proposed value. |

The **SqlClient** code example in the next section defines a parameter for an [UpdateCommand](http://msdn.microsoft.com/en-us/library/system.data.common.dbdataadapter.updatecommand.aspx) in which the **CustomerID** column is used as a **SourceColumn** for two parameters: *@CustomerID* (SET CustomerID = @CustomerID), and *@OldCustomerID* (WHERE CustomerID = @OldCustomerID). The *@CustomerID* parameter is used to update the **CustomerID** column to the current value in the **DataRow**. As a result, the **CustomerID** **SourceColumn** with a **SourceVersion** of **Current** is used. The *@OldCustomerID* parameter is used to identify the current row in the data source. Because the matching column value is found in the **Original** version of the row, the same **SourceColumn** (**CustomerID**) with a **SourceVersion** of **Original** is used.

Working with SqlClient Parameters

The following example demonstrates how to create a [SqlDataAdapter](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldataadapter.aspx) and set the [MissingSchemaAction](http://msdn.microsoft.com/en-us/library/system.data.common.dataadapter.missingschemaaction.aspx) to [AddWithKey](http://msdn.microsoft.com/en-us/library/system.data.missingschemaaction.aspx) in order to retrieve additional schema information from the database. The [SelectCommand](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldataadapter.selectcommand.aspx), [InsertCommand](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldataadapter.insertcommand.aspx), [UpdateCommand](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldataadapter.updatecommand.aspx), and [DeleteCommand](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldataadapter.deletecommand.aspx) properties set and their corresponding [SqlParameter](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlparameter.aspx) objects added to the [Parameters](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlcommand.parameters.aspx) collection. The method returns a **SqlDataAdapter** object.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl74_ctl00_ctl09_code');" \o "Copy Code)

Public Function CreateSqlDataAdapter( \_

ByVal connection As SqlConnection) As SqlDataAdapter

Dim adapter As SqlDataAdapter = New SqlDataAdapter

adapter.MissingSchemaAction = MissingSchemaAction.AddWithKey

' Create the commands.

adapter.SelectCommand = New SqlCommand( \_

"SELECT CustomerID, CompanyName FROM CUSTOMERS", connection)

adapter.InsertCommand = New SqlCommand( \_

"INSERT INTO Customers (CustomerID, CompanyName) " & \_

"VALUES (@CustomerID, @CompanyName)", connection)

adapter.UpdateCommand = New SqlCommand( \_

"UPDATE Customers SET CustomerID = @CustomerID, CompanyName = " & \_

"@CompanyName WHERE CustomerID = @oldCustomerID", connection)

adapter.DeleteCommand = New SqlCommand( \_

"DELETE FROM Customers WHERE CustomerID = @CustomerID", connection)

' Create the parameters.

adapter.InsertCommand.Parameters.Add("@CustomerID", \_

SqlDbType.Char, 5, "CustomerID")

adapter.InsertCommand.Parameters.Add("@CompanyName", \_

SqlDbType.VarChar, 40, "CompanyName")

adapter.UpdateCommand.Parameters.Add("@CustomerID", \_

SqlDbType.Char, 5, "CustomerID")

adapter.UpdateCommand.Parameters.Add("@CompanyName", \_

SqlDbType.VarChar, 40, "CompanyName")

adapter.UpdateCommand.Parameters.Add("@oldCustomerID", \_

SqlDbType.Char, 5, "CustomerID").SourceVersion = \_

DataRowVersion.Original

adapter.DeleteCommand.Parameters.Add("@CustomerID", \_

SqlDbType.Char, 5, "CustomerID").SourceVersion = \_

DataRowVersion.Original

Return adapter

End Function

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl74_ctl00_ctl10_code');" \o "Copy Code)

public static SqlDataAdapter CreateSqlDataAdapter(SqlConnection connection)

{

SqlDataAdapter adapter = new SqlDataAdapter();

adapter.MissingSchemaAction = MissingSchemaAction.AddWithKey;

// Create the commands.

adapter.SelectCommand = new SqlCommand(

"SELECT CustomerID, CompanyName FROM CUSTOMERS", connection);

adapter.InsertCommand = new SqlCommand(

"INSERT INTO Customers (CustomerID, CompanyName) " +

"VALUES (@CustomerID, @CompanyName)", connection);

adapter.UpdateCommand = new SqlCommand(

"UPDATE Customers SET CustomerID = @CustomerID, CompanyName = @CompanyName " +

"WHERE CustomerID = @oldCustomerID", connection);

adapter.DeleteCommand = new SqlCommand(

"DELETE FROM Customers WHERE CustomerID = @CustomerID", connection);

// Create the parameters.

adapter.InsertCommand.Parameters.Add("@CustomerID",

SqlDbType.Char, 5, "CustomerID");

adapter.InsertCommand.Parameters.Add("@CompanyName",

SqlDbType.VarChar, 40, "CompanyName");

adapter.UpdateCommand.Parameters.Add("@CustomerID",

SqlDbType.Char, 5, "CustomerID");

adapter.UpdateCommand.Parameters.Add("@CompanyName",

SqlDbType.VarChar, 40, "CompanyName");

adapter.UpdateCommand.Parameters.Add("@oldCustomerID",

SqlDbType.Char, 5, "CustomerID").SourceVersion =

DataRowVersion.Original;

adapter.DeleteCommand.Parameters.Add("@CustomerID",

SqlDbType.Char, 5, "CustomerID").SourceVersion =

DataRowVersion.Original;

return adapter;

}

OleDb Parameter Placeholders

For the [OleDbDataAdapter](http://msdn.microsoft.com/en-us/library/system.data.oledb.oledbdataadapter.aspx) and [OdbcDataAdapter](http://msdn.microsoft.com/en-us/library/system.data.odbc.odbcdataadapter.aspx) objects, you must use question mark (?) placeholders to identify the parameters.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl75_ctl00_ctl02_code');" \o "Copy Code)

Dim selectSQL As String = \_

"SELECT CustomerID, CompanyName FROM Customers " & \_

"WHERE CountryRegion = ? AND City = ?"

Dim insertSQL AS String = \_

"INSERT INTO Customers (CustomerID, CompanyName) VALUES (?, ?)"

Dim updateSQL AS String = \_

"UPDATE Customers SET CustomerID = ?, CompanyName = ? " & \_

WHERE CustomerID = ?"

Dim deleteSQL As String = "DELETE FROM Customers WHERE CustomerID = ?"

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl75_ctl00_ctl03_code');" \o "Copy Code)

string selectSQL =

"SELECT CustomerID, CompanyName FROM Customers " +

"WHERE CountryRegion = ? AND City = ?";

string insertSQL =

"INSERT INTO Customers (CustomerID, CompanyName) " +

"VALUES (?, ?)";

string updateSQL =

"UPDATE Customers SET CustomerID = ?, CompanyName = ? " +

"WHERE CustomerID = ? ";

string deleteSQL = "DELETE FROM Customers WHERE CustomerID = ?";

The parameterized query statements define which input and output parameters must be created. To create a parameter, use the **Parameters.Add** method or the **Parameter** constructor to specify the column name, data type, and size. For intrinsic data types, such as **Integer**, you do not have to include the size, or you can specify the default size.

The following code example creates the parameters for a SQL statement and then fills a **DataSet**.

OleDb Example

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl76_ctl00_ctl00_code');" \o "Copy Code)

' Assumes that connection is a valid OleDbConnection object.

Dim adapter As OleDbDataAdapter = New OleDbDataAdapter

Dim selectCMD AS OleDbCommand = New OleDbCommand(selectSQL, connection)

adapter.SelectCommand = selectCMD

' Add parameters and set values.

selectCMD.Parameters.Add( \_

"@CountryRegion", OleDbType.VarChar, 15).Value = "UK"

selectCMD.Parameters.Add( \_

"@City", OleDbType.VarChar, 15).Value = "London"

Dim customers As DataSet = New DataSet

adapter.Fill(customers, "Customers")

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl76_ctl00_ctl01_code');" \o "Copy Code)

// Assumes that connection is a valid OleDbConnection object.

OleDbDataAdapter adapter = new OleDbDataAdapter();

OleDbCommand selectCMD = new OleDbCommand(selectSQL, connection);

adapter.SelectCommand = selectCMD;

// Add parameters and set values.

selectCMD.Parameters.Add(

"@CountryRegion", OleDbType.VarChar, 15).Value = "UK";

selectCMD.Parameters.Add(

"@City", OleDbType.VarChar, 15).Value = "London";

DataSet customers = new DataSet();

adapter.Fill(customers, "Customers");

Odbc Parameters

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl77_ctl00_ctl00_code');" \o "Copy Code)

' Assumes that connection is a valid OdbcConnection object.

Dim adapter As OdbcDataAdapter = New OdbcDataAdapter

Dim selectCMD AS OdbcCommand = New OdbcCommand(selectSQL, connection)

adapter.SelectCommand = selectCMD

' Add Parameters and set values.

selectCMD.Parameters.Add("@CountryRegion", OdbcType.VarChar, 15).Value = "UK"

selectCMD.Parameters.Add("@City", OdbcType.VarChar, 15).Value = "London"

Dim customers As DataSet = New DataSet

adapter.Fill(customers, "Customers")

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl77_ctl00_ctl01_code');" \o "Copy Code)

// Assumes that connection is a valid OdbcConnection object.

OdbcDataAdapter adapter = new OdbcDataAdapter();

OdbcCommand selectCMD = new OdbcCommand(selectSQL, connection);

adapter.SelectCommand = selectCMD;

//Add Parameters and set values.

selectCMD.Parameters.Add("@CountryRegion", OdbcType.VarChar, 15).Value = "UK";

selectCMD.Parameters.Add("@City", OdbcType.VarChar, 15).Value = "London";

DataSet customers = new DataSet();

adapter.Fill(customers, "Customers");

|  |
| --- |
| **NoteNote** |
| If a parameter name is not supplied for a parameter, the parameter is given an incremental default name of Parameter*N,* starting with "Parameter1". We recommend that you avoid the Parameter*N* naming convention when you supply a parameter name, because the name that you supply might conflict with an existing default parameter name in the **ParameterCollection**. If the supplied name already exists, an exception is thrown. |

.NET Framework 4

**Modifying Data with Stored Procedures (ADO.NET)**

Stored procedures can accept data as input parameters and can return data as output parameters, result sets, or return values. The sample below illustrates how ADO.NET sends and receives input parameters, output parameters, and return values. The example inserts a new record into a table where the primary key column is an identity column in a SQL Server database.

|  |
| --- |
| **NoteNote** |
| If you are using SQL Server stored procedures to edit or delete data using a [SqlDataAdapter](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldataadapter.aspx), make sure that you do not use SET NOCOUNT ON in the stored procedure definition. This causes the rows affected count returned to be zero, which the **DataAdapter** interprets as a concurrency conflict. In this event, a [DBConcurrencyException](http://msdn.microsoft.com/en-us/library/system.data.dbconcurrencyexception.aspx) will be thrown. |

Example

The sample uses the following stored procedure to insert a new category into the **Northwind** **Categories** table. The stored procedure takes the value in the **CategoryName** column as an input parameter and uses the SCOPE\_IDENTITY() function to retrieve the new value of the identity field, **CategoryID**, and return it in an output parameter. The RETURN statement uses the @@ROWCOUNT function to return the number of rows inserted.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl22_ctl00_ctl00_code');" \o "Copy Code)

CREATE PROCEDURE dbo.InsertCategory

@CategoryName nvarchar(15),

@Identity int OUT

AS

INSERT INTO Categories (CategoryName) VALUES(@CategoryName)

SET @Identity = SCOPE\_IDENTITY()

RETURN @@ROWCOUNT

The following code example uses the InsertCategory stored procedure shown above as the source for the [InsertCommand](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldataadapter.insertcommand.aspx) of the [SqlDataAdapter](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldataadapter.aspx). The @Identity output parameter will be reflected in the [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx) after the record has been inserted into the database when the **Update** method of the [SqlDataAdapter](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldataadapter.aspx) is called. The code also retrieves the return value.

|  |
| --- |
| **NoteNote** |
| When using the [OleDbDataAdapter](http://msdn.microsoft.com/en-us/library/system.data.oledb.oledbdataadapter.aspx), you must specify parameters with a [ParameterDirection](http://msdn.microsoft.com/en-us/library/system.data.parameterdirection.aspx) of **ReturnValue** before the other parameters. |

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl22_ctl00_ctl08_code');" \o "Copy Code)

Private Sub ReturnIdentity(ByVal connectionString As String)

Using connection As SqlConnection = New SqlConnection( \_

connectionString)

' Create a SqlDataAdapter based on a SELECT query.

Dim adapter As SqlDataAdapter = New SqlDataAdapter( \_

"SELECT CategoryID, CategoryName FROM dbo.Categories", \_

connection)

' Create a SqlCommand to execute the stored procedure.

adapter.InsertCommand = New SqlCommand("dbo.InsertCategory", \_

connection)

adapter.InsertCommand.CommandType = CommandType.StoredProcedure

' Create a parameter for the ReturnValue.

Dim parameter As SqlParameter = \_

adapter.InsertCommand.Parameters.Add( \_

"@RowCount", SqlDbType.Int)

parameter.Direction = ParameterDirection.ReturnValue

' Create an input parameter for the CategoryName.

' You do not need to specify direction for input parameters.

adapter.InsertCommand.Parameters.Add( \_

"@CategoryName", SqlDbType.NChar, 15, "CategoryName")

' Create an output parameter for the new identity value.

parameter = adapter.InsertCommand.Parameters.Add( \_

"@Identity", SqlDbType.Int, 0, "CategoryID")

parameter.Direction = ParameterDirection.Output

' Create a DataTable and fill it.

Dim categories As DataTable = New DataTable

adapter.Fill(categories)

' Add a new row.

Dim newRow As DataRow = categories.NewRow()

newRow("CategoryName") = "New Category"

categories.Rows.Add(newRow)

' Update the database.

adapter.Update(categories)

' Retrieve the ReturnValue.

Dim rowCount As Int32 = \_

CInt(adapter.InsertCommand.Parameters("@RowCount").Value)

Console.WriteLine("ReturnValue: {0}", rowCount.ToString())

Console.WriteLine("All Rows:")

Dim row As DataRow

For Each row In categories.Rows

Console.WriteLine(" {0}: {1}", row(0), row(1))

Next

End Using

End Sub

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl22_ctl00_ctl09_code');" \o "Copy Code)

private static void ReturnIdentity(string connectionString)

{

using (SqlConnection connection =

new SqlConnection(connectionString))

{

// Create a SqlDataAdapter based on a SELECT query.

SqlDataAdapter adapter = new SqlDataAdapter(

"SELECT CategoryID, CategoryName FROM dbo.Categories",

connection);

// Create a SqlCommand to execute the stored procedure.

adapter.InsertCommand = new SqlCommand("InsertCategory", connection);

adapter.InsertCommand.CommandType = CommandType.StoredProcedure;

// Create a parameter for the ReturnValue.

SqlParameter parameter = adapter.InsertCommand.Parameters.Add(

"@RowCount", SqlDbType.Int);

parameter.Direction = ParameterDirection.ReturnValue;

// Create an input parameter for the CategoryName.

// You do not need to specify direction for input parameters.

adapter.InsertCommand.Parameters.Add(

"@CategoryName", SqlDbType.NChar, 15, "CategoryName");

// Create an output parameter for the new identity value.

parameter = adapter.InsertCommand.Parameters.Add(

"@Identity", SqlDbType.Int, 0, "CategoryID");

parameter.Direction = ParameterDirection.Output;

// Create a DataTable and fill it.

DataTable categories = new DataTable();

adapter.Fill(categories);

// Add a new row.

DataRow categoryRow = categories.NewRow();

categoryRow["CategoryName"] = "New Beverages";

categories.Rows.Add(categoryRow);

// Update the database.

adapter.Update(categories);

// Retrieve the ReturnValue.

Int32 rowCount =

(Int32)adapter.InsertCommand.Parameters["@RowCount"].Value;

Console.WriteLine("ReturnValue: {0}", rowCount.ToString());

Console.WriteLine("All Rows:");

foreach (DataRow row in categories.Rows)

{

{

Console.WriteLine(" {0}: {1}", row[0], row[1]);

}

}

}

}

.NET Framework 4

**Managing Permissions with Stored Procedures in SQL Server (ADO.NET)**

One method of creating multiple lines of defense around your database is to implement all data access using stored procedures or user-defined functions. You revoke or deny all permissions to underlying objects, such as tables, and grant EXECUTE permissions on stored procedures. This effectively creates a security perimeter around your data and database objects.

Stored Procedure Benefits

Stored procedures have the following benefits:

* Data logic and business rules can be encapsulated so that users can access data and objects only in ways that developers and database administrators intend.
* Parameterized stored procedures that validate all user input can be used to thwart SQL injection attacks. If you use dynamic SQL, be sure to parameterize your commands, and never include parameter values directly into a query string.
* Ad hoc queries and data modifications can be disallowed. This prevents users from maliciously or inadvertently destroying data or executing queries that impair performance on the server or the network.
* Errors can be handled in procedure code without being passed directly to client applications. This prevents error messages from being returned that could aid in a probing attack. Log errors and handle them on the server.
* Stored procedures can be written once, and accessed by many applications.
* Client applications do not need to know anything about the underlying data structures. Stored procedure code can be changed without requiring changes in client applications as long as the changes do not affect parameter lists or returned data types.
* Stored procedures can reduce network traffic by combining multiple operations into one procedure call.

Stored Procedure Execution

Stored procedures take advantage of ownership chaining to provide access to data so that users do not need to have explicit permission to access database objects. An ownership chain exists when objects that access each other sequentially are owned by the same user. For example, a stored procedure can call other stored procedures, or a stored procedure can access multiple tables. If all objects in the chain of execution have the same owner, then SQL Server only checks the EXECUTE permission for the caller, not the caller's permissions on other objects. Therefore you need to grant only EXECUTE permissions on stored procedures; you can revoke or deny all permissions on the underlying tables.

Best Practices

Simply writing stored procedures isn't enough to adequately secure your application. You should also consider the following potential security holes.

* Grant EXECUTE permissions on the stored procedures for database roles you want to be able to access the data.
* Revoke or deny all permissions to the underlying tables for all roles and users in the database, including the **public** role. All users inherit permissions from public. Therefore denying permissions to **public** means that only owners and **sysadmin** members have access; all other users will be unable to inherit permissions from membership in other roles.
* Do not add users or roles to the **sysadmin** or **db\_owner** roles. System administrators and database owners can access all database objects.
* Disable the **guest** account. This will prevent anonymous users from connecting to the database. The guest account is disabled by default in new databases.
* Implement error handling and log errors.
* Create parameterized stored procedures that validate all user input. Treat all user input as untrusted.
* Avoid dynamic SQL unless absolutely necessary. Use the Transact-SQL QUOTENAME() function to delimit a string value and escape any occurrence of the delimiter in the input string.

.NET Framework 4 - ASP.NET

**Script Exploits Overview**

From the perspective of a browser, a Web page is simply a long string of characters. The browser processes the string sequentially, displaying some characters while interpreting other characters, such as <b> and <script> according to special rules. If a malicious user can insert some of those special characters into a page, the browser will not know that the characters are not supposed to be there, and it will process them as part of the page.

A simplistic script exploit might work as follows. If an application allows users to post comments about the latest movies for other users to read, the exploit steps might be:

1. The application displays a form where users enter comments. The malicious user writes a comment that includes a <script> block in it.
2. The form is posted and the malicious user's comment is stored in a database.
3. Another user visits the site. When the page is constructed, it reads comments out of the database and puts them into the page. The malicious user's <script> block is written into the page as if it were a text comment.
4. When the second user's browser displays the page, it gets to the <script> block and executes it.

There are other ways that malicious users can exploit script. Most script exploits require the application to accept the malicious input and inject it (or echo it) into a page where it will be executed by the browser. The potential damage from such an exploit depends on the script that is executed. It can be trivial, such as an annoying message that pops up in the browser. But it can also do serious damage by stealing cookies, stealing user input (such as a password), and, if Internet security is lax, running native code on the user's computer.

For information about preventing script exploits, see [How to: Protect Against Script Exploits in a Web Application by Applying HTML Encoding to Strings](http://msdn.microsoft.com/en-us/library/a2a4yykt.aspx).

|  |
| --- |
| **NoteNote** |
| ASP.NET helps protect against script exploits that are disguised as URLs by checking for potentially dangerous strings, such as "<!", "</", and "<?". For more information, see [HtmlEncode](http://msdn.microsoft.com/en-us/library/system.web.httpserverutility.htmlencode.aspx) and [ValidateRequest](http://msdn.microsoft.com/en-us/library/system.web.configuration.pagessection.validaterequest.aspx). |

SQL Statement Exploits

A variation on a script exploit is one that causes malicious SQL statements to be executed. This can occur if an application prompts users for information and then concatenates the user's input into a string representing the SQL statement. For example, an application might prompt for a customer name with the intention of executing a statement, such as the following:

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl19_ctl00_ctl00_code');" \o "Copy Code)

"Select \* From Customers where CustomerName = " & txtCustomerName.Value

But a malicious user who knows something about the database could use the text box to enter an embedded SQL statement with the customer name, resulting in a statement like the following:

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl19_ctl00_ctl01_code');" \o "Copy Code)

Select \* From Customers Where CustomerName = 'a' Delete From

Customers Where CustomerName > ''

When the query is executed, the database is compromised.

Guarding Against Scripting Exploits

The primary defense against scripting exploits is to never trust information coming from a user. Assume that any data posted to your application from a browser can contain malicious script.

Similarly, any time that you write a string into a page, you should assume that the string could contain malicious script (unless you programmatically created the string yourself). For example, when you read strings out of a database, you should assume that they can contain malicious script. The most security-conscious developers distrust even their own databases, on the theory that a malicious user might have found a way to tamper with the database.

ASP.NET provides you with several ways to help protect against scripting exploits:

* ASP.NET performs request validation against query-string and form variables as well as cookie values. By default, if the current [Request](http://msdn.microsoft.com/en-us/library/system.web.httpcontext.request.aspx) contains HTML-encoded elements or certain HTML characters (such as &#151; for an em dash), the ASP.NET page framework raises an error.
* If you want to display strings in your application but do not trust them, apply HTML encoding to them when the strings are written back in a response. For example, with encoding, the tag <b> becomes &lt;b&gt;. You might do this if the strings that you are displaying are from a database whose contents you are not sure that you can trust.
* If you want your application to accept some HTML (for example, some formatting instructions from users), you should encode the HTML at the client before it is submitted to the server. For more information, see [How to: Protect Against Script Exploits in a Web Application by Applying HTML Encoding to Strings](http://msdn.microsoft.com/en-us/library/a2a4yykt.aspx).
* To help protect against SQL statement exploits, never create SQL queries using string concatenation. Instead, use a parameterized query and assign user input to parameter objects.
* Always validate form input against a set of expected values and string formatting/type validation. For example, if a specific form variable is expected to be an integer, use the [TryParse](http://msdn.microsoft.com/en-us/library/system.int32.tryparse.aspx) method to verify that the value really is an integer and use range checking to help ensure that the value is within an acceptable range.

.NET Framework 4

**Exception Handling Fundamentals**

The common language runtime supports an exception handling model based on the concepts of exception objects and protected blocks of code. The runtime creates an object to represent an exception when it occurs. You can also create your own exception classes by deriving classes from the appropriate base exception.

All languages that use the runtime handle exceptions in a similar manner. Each language uses a form of try/catch/finally structured exception handling. This section provides several examples of basic exception handling.

In This Section

[How to: Use the Try/Catch Block to Catch Exceptions](http://msdn.microsoft.com/en-us/library/xtd0s8kd.aspx)

Describes how to use the try/catch block to handle exceptions.

[How to: Use Specific Exceptions in a Catch Block](http://msdn.microsoft.com/en-us/library/3tca6706.aspx)

Describes how to catch specific exceptions.

[How to: Explicitly Throw Exceptions](http://msdn.microsoft.com/en-us/library/xhcbs8fz.aspx)

Describes how to throw exceptions and how to catch exceptions and then throw them again.

[How to: Create User-Defined Exceptions](http://msdn.microsoft.com/en-us/library/87cdya3t.aspx)

Describes how to create your own exception classes.

[Using User-Filtered Handlers](http://msdn.microsoft.com/en-us/library/4dy8x9k9.aspx)

Describes how to set up filtered exceptions.

[How to: Use a Finally Block](http://msdn.microsoft.com/en-us/library/ke0zf0f5.aspx)

Explains how to use the finally statement in an exception block.

.NET Framework 4

**Best Practices for Handling Exceptions**

A well-designed set of error handling code blocks can make a program more robust and less prone to crashing because the application handles such errors. The following list contains suggestions on best practices for handling exceptions:

* Know when to set up a try/catch block. For example, you can programmatically check for a condition that is likely to occur without using exception handling. In other situations, using exception handling to catch an error condition is appropriate.

The following example uses an **if** statement to check whether a connection is closed. You can use this method instead of throwing an exception if the connection is not closed.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl01_code');" \o "Copy Code)

If conn.State <> ConnectionState.Closed Then

conn.Close()

End IF

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl02_code');" \o "Copy Code)

if (conn.State != ConnectionState.Closed)

{

conn.Close();

}

Visual C++

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl03_code');" \o "Copy Code)

if (conn->State != ConnectionState::Closed)

{

conn->Close();

}

In the following example, an exception is thrown if the connection is not closed.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl04_code');" \o "Copy Code)

Try

conn.Close()

Catch ex As InvalidOperationException

Console.WriteLine(ex.GetType().FullName)

Console.WriteLine(ex.Message)

End Try

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl05_code');" \o "Copy Code)

try

{

conn.Close();

}

catch (InvalidOperationException ex)

{

Console.WriteLine(ex.GetType().FullName);

Console.WriteLine(ex.Message);

}

Visual C++

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl06_code');" \o "Copy Code)

try

{

conn->Close();

}

catch (InvalidOperationException^ ex)

{

Console::WriteLine(ex->GetType()->FullName);

Console::WriteLine(ex->Message);

}

The method you choose depends on how often you expect the event to occur. If the event is truly exceptional and is an error (such as an unexpected end-of-file), using exception handling is better because less code is executed in the normal case. If the event happens routinely, using the programmatic method to check for errors is better. In this case, if an exception occurs, the exception will take longer to handle.

* Use try/finally blocks around code that can potentially generate an exception and centralize your catch statements in one location. In this way, the try statement generates the exception, the finally statement closes or deallocates resources, and the catch statement handles the exception from a central location.
* Always order exceptions in catch blocks from the most specific to the least specific. This technique handles the specific exception before it is passed to a more general catch block.
* End exception class names with the word "Exception". For example:

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl07_code');" \o "Copy Code)

Public Class MyFileNotFoundException

Inherits Exception

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl08_code');" \o "Copy Code)

public class MyFileNotFoundException : Exception

{

}

Visual C++

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl09_code');" \o "Copy Code)

public ref class MyFileNotFoundException : public Exception

{

};

* When creating user-defined exceptions, you must ensure that the metadata for the exceptions is available to code executing remotely, including when exceptions occur across application domains. For example, suppose Application Domain A creates Application Domain B, which executes code that throws an exception. For Application Domain A to properly catch and handle the exception, it must be able to find the assembly containing the exception thrown by Application Domain B. If Application Domain B throws an exception that is contained in an assembly under its application base, but not under Application Domain A's application base, Application Domain A will not be able to find the exception and the common language runtime will throw a [FileNotFoundException](http://msdn.microsoft.com/en-us/library/system.io.filenotfoundexception.aspx). To avoid this situation, you can deploy the assembly containing the exception information in two ways:
  + Put the assembly into a common application base shared by both application domains

- or -

* + If the domains do not share a common application base, sign the assembly containing the exception information with a strong name and deploy the assembly into the global assembly cache.
* In C# and C++, use at least the three common constructors when creating your own exception classes. For an example, see [How to: Create User-Defined Exceptions](http://msdn.microsoft.com/en-us/library/87cdya3t.aspx).
* In most cases, use the predefined exceptions types. Define new exception types only for programmatic scenarios. Introduce a new exception class to enable a programmer to take a different action in code based on the exception class.
* For most applications, derive custom exceptions from the [Exception](http://msdn.microsoft.com/en-us/library/system.exception.aspx) class. It was originally thought that custom exceptions should derive from the [ApplicationException](http://msdn.microsoft.com/en-us/library/system.applicationexception.aspx) class; however in practice this has not been found to add significant value.
* Include a localized description string in every exception. When the user sees an error message, it is derived from the description string of the exception that was thrown, rather than from the exception class.
* Use grammatically correct error messages, including ending punctuation. Each sentence in a description string of an exception should end in a period.
* Provide [Exception](http://msdn.microsoft.com/en-us/library/system.exception.aspx) properties for programmatic access. Include extra information in an exception (in addition to the description string) only when there is a programmatic scenario where the additional information is useful.
* Return null for extremely common error cases. For example, [Open](http://msdn.microsoft.com/en-us/library/system.io.file.open.aspx) returns **null** if the file is not found, but throws an exception if the file is locked.
* Design classes so that an exception is never thrown in normal use. For example, a [FileStream](http://msdn.microsoft.com/en-us/library/system.io.filestream.aspx) class exposes another way of determining whether the end of the file has been reached. This avoids the exception that is thrown if you read past the end of the file. The following example shows how to read to the end of the file.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl17_code');" \o "Copy Code)

Class FileRead

Public Sub ReadAll(fileToRead As FileStream)

' This if statement is optional

' as it is very unlikely that

' the stream would ever be null.

If fileToRead Is Nothing Then

Throw New System.ArgumentNullException()

End If

Dim b As Integer

' Set the stream position to the beginning of the file.

fileToRead.Seek(0, SeekOrigin.Begin)

' Read each byte to the end of the file.

For i As Integer = 0 To fileToRead.Length - 1

b = fileToRead.ReadByte()

Console.Write(b.ToString())

' Or do something else with the byte.

Next i

End Sub

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl18_code');" \o "Copy Code)

class FileRead

{

public void ReadAll(FileStream fileToRead)

{

// This if statement is optional

// as it is very unlikely that

// the stream would ever be null.

if (fileToRead == null)

{

throw new System.ArgumentNullException();

}

int b;

// Set the stream position to the beginning of the file.

fileToRead.Seek(0, SeekOrigin.Begin);

// Read each byte to the end of the file.

for (int i = 0; i < fileToRead.Length; i++)

{

b = fileToRead.ReadByte();

Console.Write(b.ToString());

// Or do something else with the byte.

}

}

}

Visual C++

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl19_code');" \o "Copy Code)

class FileRead

{

public:

void ReadAll(FileStream^ fileToRead)

{

// This if statement is optional

// as it is very unlikely that

// the stream would ever be null.

if (fileToRead == nullptr)

{

throw gcnew System::ArgumentNullException();

}

int b;

// Set the stream position to the beginning of the file.

fileToRead->Seek(0, SeekOrigin::Begin);

// Read each byte to the end of the file.

for (int i = 0; i < fileToRead->Length; i++)

{

b = fileToRead->ReadByte();

Console::Write(b.ToString());

// Or do something else with the byte.

}

}

};

Throw an [InvalidOperationException](http://msdn.microsoft.com/en-us/library/system.invalidoperationexception.aspx) if a property set or method call is not appropriate given the object's current state.

Throw an [ArgumentException](http://msdn.microsoft.com/en-us/library/system.argumentexception.aspx) or a class derived from [ArgumentException](http://msdn.microsoft.com/en-us/library/system.argumentexception.aspx) if invalid parameters are passed.

The stack trace begins at the statement where the exception is thrown and ends at the catch statement that catches the exception. Be aware of this fact when deciding where to place a throw statement.

Use exception builder methods. It is common for a class to throw the same exception from different places in its implementation. To avoid excessive code, use helper methods that create the exception and return it. For example:

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl23_code');" \o "Copy Code)

Class FileReader

Private fileName As String

Public Sub New(path As String)

fileName = path

End Sub

Public Function Read(bytes As Integer) As Byte()

Dim results() As Byte = FileUtils.ReadFromFile(fileName, bytes)

If results Is Nothing

Throw NewFileIOException()

End If

Return results

End Function

Function NewFileIOException() As FileReaderException

Dim description As String = "My NewFileIOException Description"

Return New FileReaderException(description)

End Function

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl24_code');" \o "Copy Code)

class FileReader

{

private string fileName;

public FileReader(string path)

{

fileName = path;

}

public byte[] Read(int bytes)

{

byte[] results = FileUtils.ReadFromFile(fileName, bytes);

if (results == null)

{

throw NewFileIOException();

}

return results;

}

FileReaderException NewFileIOException()

{

string description = "My NewFileIOException Description";

return new FileReaderException(description);

}

}

Visual C++

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl25_code');" \o "Copy Code)

ref class FileReader

{

private:

String^ fileName;

public:

FileReader(String^ path)

{

fileName = path;

}

array<Byte>^ Read(int bytes)

{

array<Byte>^ results = FileUtils::ReadFromFile(fileName, bytes);

if (results == nullptr)

{

throw NewFileIOException();

}

return results;

}

FileReaderException^ NewFileIOException()

{

String^ description = "My NewFileIOException Description";

return gcnew FileReaderException(description);

}

};

Alternatively, use the exception's constructor to build the exception. This is more appropriate for global exception classes, such as [ArgumentException](http://msdn.microsoft.com/en-us/library/system.argumentexception.aspx).

* Throw exceptions instead of returning an error code or HRESULT.
* Clean up intermediate results when throwing an exception. Callers should be able assume that there are no side effects when an exception is thrown from a method.