This page is specific to

Microsoft Visual Studio 2010/.NET Framework 4

.NET Framework 4

**Connection Strings and Configuration Files (ADO.NET)**

Embedding connection strings in your application's code can lead to security vulnerabilities and maintenance problems. Unencrypted connection strings compiled into an application's source code can be viewed using the [Ildasm.exe (MSIL Disassembler)](http://msdn.microsoft.com/en-us/library/f7dy01k1.aspx) tool. Moreover, if the connection string ever changes, your application must be recompiled. For these reasons, we recommend storing connection strings in an application configuration file.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifWorking with Application Configuration Files

Application configuration files contain settings that are specific to a particular application. For example, an ASP.NET application can have one or more **web.config** files, and a Windows application can have an optional **app.config** file. Configuration files share common elements, although the name and location of a configuration file vary depending on the application's host.

**The connectionStrings Section**

Connection strings can be stored as key/value pairs in the **connectionStrings** section of the **configuration** element of an application configuration file. Child elements include **add**, **clear**, and **remove**.

The following configuration file fragment demonstrates the schema and syntax for storing a connection string. The **name** attribute is a name that you provide to uniquely identify a connection string so that it can be retrieved at run time. The **providerName** is the invariant name of the .NET Framework data provider, which is registered in the machine.config file.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl74_ctl00_ctl00_code');" \o "Copy Code)

<?xml version='1.0' encoding='utf-8'?>

<configuration>

<connectionStrings>

<clear />

<add name="Name"

providerName="System.Data.ProviderName"

connectionString="Valid Connection String;" />

</connectionStrings>

</configuration>

|  |
| --- |
| **Description: NoteNote** |
| You can save part of a connection string in a configuration file and use the [DbConnectionStringBuilder](http://msdn.microsoft.com/en-us/library/system.data.common.dbconnectionstringbuilder.aspx) class to complete it at run time. This is useful in scenarios where you do not know elements of the connection string ahead of time, or when you do not want to save sensitive information in a configuration file. For more information, see [Connection String Builders (ADO.NET)](http://msdn.microsoft.com/en-us/library/ms254947.aspx). |

**Using External Configuration Files**

External configuration files are separate files that contain a fragment of a configuration file consisting of a single section. The external configuration file is then referenced by the main configuration file. Storing the **connectionStrings** section in a physically separate file is useful in situations where connection strings may be edited after the application is deployed. For example, the standard ASP.NET behavior is to restart an application domain when configuration files are modified, which results in state information being lost. However, modifying an external configuration file does not cause an application restart. External configuration files are not limited to ASP.NET; they can also be used by Windows applications. In addition, file access security and permissions can be used to restrict access to external configuration files. Working with external configuration files at run time is transparent, and requires no special coding.

To store connection strings in an external configuration file, create a separate file that contains only the **connectionStrings** section. Do not include any additional elements, sections, or attributes. This example shows the syntax for an external configuration file.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl74_ctl00_ctl04_code');" \o "Copy Code)

<connectionStrings>

<add name="Name"

providerName="System.Data.ProviderName"

connectionString="Valid Connection String;" />

</connectionStrings>

In the main application configuration file, you use the **configSource** attribute to specify the fully qualified name and location of the external file. This example refers to an external configuration file named connections.config.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl74_ctl00_ctl05_code');" \o "Copy Code)

<?xml version='1.0' encoding='utf-8'?>

<configuration>

<connectionStrings configSource="connections.config"/>

</configuration>

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifRetrieving Connection Strings at Run Time

The .NET Framework 2.0 introduced new classes in the [System.Configuration](http://msdn.microsoft.com/en-us/library/system.configuration.aspx) namespace to simplify retrieving connection strings from configuration files at run time. You can programmatically retrieve a connection string by name or by provider name.

|  |
| --- |
| **Description: NoteNote** |
| The **machine.config** file also contains a **connectionStrings** section, which contains connection strings used by Visual Studio. When retrieving connection strings by provider name from the **app.config** file in a Windows application, the connection strings in **machine.config** get loaded first, and then the entries from **app.config**. Adding **clear** immediately after the **connectionStrings** element removes all inherited references from the data structure in memory, so that only the connection strings defined in the local **app.config** file are considered. |

**Working with the Configuration Classes**

Starting with the .NET Framework 2.0, [ConfigurationManager](http://msdn.microsoft.com/en-us/library/system.configuration.configurationmanager.aspx) is used when working with configuration files on the local computer, replacing the deprecated [ConfigurationSettings](http://msdn.microsoft.com/en-us/library/system.configuration.configurationsettings.aspx). [WebConfigurationManager](http://msdn.microsoft.com/en-us/library/system.web.configuration.webconfigurationmanager.aspx) is used to work with ASP.NET configuration files. It is designed to work with configuration files on a Web server, and allows programmatic access to configuration file sections such as **system.web**.

|  |
| --- |
| **Description: NoteNote** |
| Accessing configuration files at run time requires granting permissions to the caller; the required permissions depend on the type of application, configuration file, and location. For more information, see [Using the Configuration Classes](http://msdn.microsoft.com/en-us/library/ms228063.aspx) and [WebConfigurationManager](http://msdn.microsoft.com/en-us/library/system.web.configuration.webconfigurationmanager.aspx) for ASP.NET applications, and [ConfigurationManager](http://msdn.microsoft.com/en-us/library/system.configuration.configurationmanager.aspx) for Windows applications. |

You can use the [ConnectionStringSettingsCollection](http://msdn.microsoft.com/en-us/library/system.configuration.connectionstringsettingscollection.aspx) to retrieve connection strings from application configuration files. It contains a collection of [ConnectionStringSettings](http://msdn.microsoft.com/en-us/library/system.configuration.connectionstringsettings.aspx) objects, each of which represents a single entry in the **connectionStrings** section. Its properties map to connection string attributes, allowing you to retrieve a connection string by specifying the name or the provider name.

|  |  |
| --- | --- |
| **Property** | **Description** |
| [Name](http://msdn.microsoft.com/en-us/library/system.configuration.connectionstringsettings.name.aspx) | The name of the connection string. Maps to the **name** attribute. |
| [ProviderName](http://msdn.microsoft.com/en-us/library/system.configuration.connectionstringsettings.providername.aspx) | The fully qualified provider name. Maps to the **providerName** attribute. |
| [ConnectionString](http://msdn.microsoft.com/en-us/library/system.configuration.connectionstringsettings.connectionstring.aspx) | The connection string. Maps to the **connectionString** attribute. |

**Example: Listing All Connection Strings**

This example iterates through the **ConnectionStringSettings** collection and displays the [Name](http://msdn.microsoft.com/en-us/library/system.configuration.connectionstringsettings.name.aspx), [ProviderName](http://msdn.microsoft.com/en-us/library/system.configuration.connectionstringsettings.providername.aspx), and [ConnectionString](http://msdn.microsoft.com/en-us/library/system.configuration.connectionstringsettings.connectionstring.aspx) properties in the console window.

|  |
| --- |
| **Description: NoteNote** |
| System.Configuration.dll is not included in all project types, and you may need to set a reference to it in order to use the configuration classes. The name and location of a particular application configuration file varies by the type of application and the hosting process. |

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl75_ctl00_ctl18_code');" \o "Copy Code)

Imports System.Configuration

Class Program

Shared Sub Main()

GetConnectionStrings()

Console.ReadLine()

End Sub

Private Shared Sub GetConnectionStrings()

Dim settings As ConnectionStringSettingsCollection = \_

ConfigurationManager.ConnectionStrings

If Not settings Is Nothing Then

For Each cs As ConnectionStringSettings In settings

Console.WriteLine(cs.Name)

Console.WriteLine(cs.ProviderName)

Console.WriteLine(cs.ConnectionString)

Next

End If

End Sub

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl75_ctl00_ctl19_code');" \o "Copy Code)

using System.Configuration;

class Program

{

static void Main()

{

GetConnectionStrings();

Console.ReadLine();

}

static void GetConnectionStrings()

{

ConnectionStringSettingsCollection settings =

ConfigurationManager.ConnectionStrings;

if (settings != null)

{

foreach(ConnectionStringSettings cs in settings)

{

Console.WriteLine(cs.Name);

Console.WriteLine(cs.ProviderName);

Console.WriteLine(cs.ConnectionString);

}

}

}

}

**Example: Retrieving a Connection String by Name**

This example demonstrates how to retrieve a connection string from a configuration file by specifying its name. The code creates a [ConnectionStringSettings](http://msdn.microsoft.com/en-us/library/system.configuration.connectionstringsettings.aspx) object, matching the supplied input parameter to the [ConnectionStrings](http://msdn.microsoft.com/en-us/library/system.configuration.configurationmanager.connectionstrings.aspx) name. If no matching name is found, the function returns **null** (**Nothing** in Visual Basic).

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl75_ctl00_ctl22_code');" \o "Copy Code)

' Retrieves a connection string by name.

' Returns Nothing if the name is not found.

Private Shared Function GetConnectionStringByName( \_

ByVal name As String) As String

' Assume failure

Dim returnValue As String = Nothing

' Look for the name in the connectionStrings section.

Dim settings As ConnectionStringSettings = \_

ConfigurationManager.ConnectionStrings(name)

' If found, return the connection string.

If Not settings Is Nothing Then

returnValue = settings.ConnectionString

End If

Return returnValue

End Function

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl75_ctl00_ctl23_code');" \o "Copy Code)

// Retrieves a connection string by name.

// Returns null if the name is not found.

static string GetConnectionStringByName(string name)

{

// Assume failure.

string returnValue = null;

// Look for the name in the connectionStrings section.

ConnectionStringSettings settings =

ConfigurationManager.ConnectionStrings[name];

// If found, return the connection string.

if (settings != null)

returnValue = settings.ConnectionString;

return returnValue;

}

**Example: Retrieving a Connection String by Provider Name**

This example demonstrates how to retrieve a connection string by specifying the provider-invariant name in the format *System.Data.ProviderName*. The code iterates through the [ConnectionStringSettingsCollection](http://msdn.microsoft.com/en-us/library/system.configuration.connectionstringsettingscollection.aspx) and returns the connection string for the first [ProviderName](http://msdn.microsoft.com/en-us/library/system.configuration.connectionstringsettings.providername.aspx) found. If the provider name is not found, the function returns **null** (**Nothing** in Visual Basic).

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl75_ctl00_ctl26_code');" \o "Copy Code)

' Retrieve a connection string by specifying the providerName.

' Assumes one connection string per provider in the config file.

Private Shared Function GetConnectionStringByProvider( \_

ByVal providerName As String) As String

'Return Nothing on failure.

Dim returnValue As String = Nothing

' Get the collection of connection strings.

Dim settings As ConnectionStringSettingsCollection = \_

ConfigurationManager.ConnectionStrings

' Walk through the collection and return the first

' connection string matching the providerName.

If Not settings Is Nothing Then

For Each cs As ConnectionStringSettings In settings

If cs.ProviderName = providerName Then

returnValue = cs.ConnectionString

Exit For

End If

Next

End If

Return returnValue

End Function

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl75_ctl00_ctl27_code');" \o "Copy Code)

// Retrieve a connection string by specifying the providerName.

// Assumes one connection string per provider in the config file.

static string GetConnectionStringByProvider(string providerName)

{

// Return null on failure.

string returnValue = null;

// Get the collection of connection strings.

ConnectionStringSettingsCollection settings =

ConfigurationManager.ConnectionStrings;

// Walk through the collection and return the first

// connection string matching the providerName.

if (settings != null)

{

foreach (ConnectionStringSettings cs in settings)

{

if (cs.ProviderName == providerName)

returnValue = cs.ConnectionString;

break;

}

}

return returnValue;

}

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifEncrypting Configuration File Sections Using Protected Configuration

ASP.NET 2.0 introduced a new feature, called *protected configuration*, that enables you to encrypt sensitive information in a configuration file. Although primarily designed for ASP.NET, protected configuration can also be used to encrypt configuration file sections in Windows applications. For a detailed description of the protected configuration capabilities, see [Encrypting Configuration Information Using Protected Configuration](http://msdn.microsoft.com/en-us/library/53tyfkaw.aspx).

The following configuration file fragment shows the **connectionStrings** section after it has been encrypted. The **configProtectionProvider** specifies the protected configuration provider used to encrypt and decrypt the connection strings. The **EncryptedData** section contains the cipher text.

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

<connectionStrings configProtectionProvider="DataProtectionConfigurationProvider">

<EncryptedData>

<CipherData>

<CipherValue>AQAAANCMnd8BFdERjHoAwE/Cl+sBAAAAH2... </CipherValue>

</CipherData>

</EncryptedData>

</connectionStrings>

When the encrypted connection string is retrieved at run time, the .NET Framework uses the specified provider to decrypt the **CipherValue** and make it available to your application. You do not need to write any additional code to manage the decryption process.

**Protected Configuration Providers**

Protected configuration providers are registered in the **configProtectedData** section of the **machine.config** file on the local computer, as shown in the following fragment, which shows the two protected configuration providers supplied with the .NET Framework. The values shown here have been truncated for readability.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl76_ctl00_ctl02_code');" \o "Copy Code)

<configProtectedData defaultProvider="RsaProtectedConfigurationProvider">

<providers>

<add name="RsaProtectedConfigurationProvider"

type="System.Configuration.RsaProtectedConfigurationProvider, ... />

<add name="DataProtectionConfigurationProvider"

type="System.Configuration.DpapiProtectedConfigurationProvider, ... />

</providers>

</configProtectedData>

You can configure additional protected configuration providers by adding them to the **machine.config** file. You can also create your own protected configuration provider by inheriting from the [ProtectedConfigurationProvider](http://msdn.microsoft.com/en-us/library/system.configuration.protectedconfigurationprovider.aspx) abstract base class. The following table describes the two configuration files included with the .NET Framework.

|  |  |
| --- | --- |
| **Provider** | **Description** |
| RSAProtectedConfigurationProvider | Uses the RSA encryption algorithm to encrypt and decrypt data. The RSA algorithm can be used for both public key encryption and digital signatures. It is also known as "public key" or asymmetrical encryption because it employs two different keys. You can use the [ASP.NET IIS Registration Tool (Aspnet\_regiis.exe)](http://msdn.microsoft.com/en-us/library/k6h9cz8h.aspx) to encrypt sections in a Web.config file and manage the encryption keys. ASP.NET decrypts the configuration file when it processes the file. The identity of the ASP.NET application must have read access to the encryption key that is used to encrypt and decrypt the encrypted sections. |
| DPAPIProtectedConfigurationProvider | Uses the Windows Data Protection API (DPAPI) to encrypt configuration sections. It uses the Windows built-in cryptographic services and can be configured for either machine-specific or user-account-specific protection. Machine-specific protection is useful for multiple applications on the same server that need to share information. User-account-specific protection can be used with services that run with a specific user identity, such as a shared hosting environment. Each application runs under a separate identity which restricts access to resources such as files and databases. |

Both providers offer strong encryption of data. However, if you are planning to use the same encrypted configuration file on multiple servers, such as a Web farm, only the **RsaProtectedConfigurationProvider** enables you to export the encryption keys used to encrypt the data and import them on another server. For more information, see [Importing and Exporting Protected Configuration RSA Key Containers](http://msdn.microsoft.com/en-us/library/yxw286t2.aspx).

**Using the Configuration Classes**

The [System.Configuration](http://msdn.microsoft.com/en-us/library/system.configuration.aspx) namespace provides classes to work with configuration settings programmatically. The [ConfigurationManager](http://msdn.microsoft.com/en-us/library/system.configuration.configurationmanager.aspx) class provides access to machine, application, and user configuration files. If you are creating an ASP.NET application, you can use the [WebConfigurationManager](http://msdn.microsoft.com/en-us/library/system.web.configuration.webconfigurationmanager.aspx) class, which provides the same functionality while also allowing you to access settings that are unique to ASP.NET applications, such as those found in **<system.web>**.

|  |
| --- |
| **Description: NoteNote** |
| The [System.Security.Cryptography](http://msdn.microsoft.com/en-us/library/system.security.cryptography.aspx) namespace contains classes that provide additional options for encrypting and decrypting data. Use these classes if you require cryptographic services that are not available using protected configuration. Some of these classes are wrappers for the unmanaged Microsoft CryptoAPI, while others are purely managed implementations. For more information, see [Cryptographic Services](http://msdn.microsoft.com/en-us/library/93bskf9z.aspx). |

**App.config Example**

This example demonstrates how to toggle encrypting the **connectionStrings** section in an **app.config** file for a Windows application. In this example, the procedure takes the name of the application as an argument, for example, "MyApplication.exe". The **app.config** file will then be encrypted and copied to the folder that contains the executable under the name of "MyApplication.exe.config".

|  |
| --- |
| **Description: NoteNote** |
| The connection string can only be decrypted on the computer on which it was encrypted. |

The code uses the [OpenExeConfiguration](http://msdn.microsoft.com/en-us/library/system.configuration.configurationmanager.openexeconfiguration.aspx) method to open the **app.config** file for editing, and the [GetSection](http://msdn.microsoft.com/en-us/library/system.configuration.configurationmanager.getsection.aspx) method returns the **connectionStrings** section. The code then checks the [IsProtected](http://msdn.microsoft.com/en-us/library/system.configuration.sectioninformation.isprotected.aspx) property, calling the [ProtectSection](http://msdn.microsoft.com/en-us/library/system.configuration.sectioninformation.protectsection.aspx) to encrypt the section if it is not encrypted. The UnProtectSection()()() method is invoked to decrypt the section. The [Save](http://msdn.microsoft.com/en-us/library/system.configuration.configuration.save.aspx) method completes the operation and saves the changes.

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

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl76_ctl00_ctl20_code');" \o "Copy Code)

Shared Sub ToggleConfigEncryption(ByVal exeConfigName As String)

' Takes the executable file name without the

' .config extension.

Try

' Open the configuration file and retrieve

' the connectionStrings section.

Dim config As Configuration = ConfigurationManager. \_

OpenExeConfiguration(exeConfigName)

Dim section As ConnectionStringsSection = DirectCast( \_

config.GetSection("connectionStrings"), \_

ConnectionStringsSection)

If section.SectionInformation.IsProtected Then

' Remove encryption.

section.SectionInformation.UnprotectSection()

Else

' Encrypt the section.

section.SectionInformation.ProtectSection( \_

"DataProtectionConfigurationProvider")

End If

' Save the current configuration.

config.Save()

Console.WriteLine("Protected={0}", \_

section.SectionInformation.IsProtected)

Catch ex As Exception

Console.WriteLine(ex.Message)

End Try

End Sub

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl76_ctl00_ctl21_code');" \o "Copy Code)

static void ToggleConfigEncryption(string exeConfigName)

{

// Takes the executable file name without the

// .config extension.

try

{

// Open the configuration file and retrieve

// the connectionStrings section.

Configuration config = ConfigurationManager.

OpenExeConfiguration(exeConfigName);

ConnectionStringsSection section =

config.GetSection("connectionStrings")

as ConnectionStringsSection;

if (section.SectionInformation.IsProtected)

{

// Remove encryption.

section.SectionInformation.UnprotectSection();

}

else

{

// Encrypt the section.

section.SectionInformation.ProtectSection(

"DataProtectionConfigurationProvider");

}

// Save the current configuration.

config.Save();

Console.WriteLine("Protected={0}",

section.SectionInformation.IsProtected);

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

**Web.config Example**

This example uses the [OpenWebConfiguration](http://msdn.microsoft.com/en-us/library/system.web.configuration.webconfigurationmanager.openwebconfiguration.aspx) method of the **WebConfigurationManager**. Note that in this case you can supply the relative path to the **Web.config** file by using a tilde. The code requires a reference to the **System.Web.Configuration** class.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl76_ctl00_ctl23_code');" \o "Copy Code)

Shared Sub ToggleWebEncrypt()

' Open the Web.config file.

Dim config As Configuration = WebConfigurationManager. \_

OpenWebConfiguration("~")

' Get the connectionStrings section.

Dim section As ConnectionStringsSection = DirectCast( \_

config.GetSection("connectionStrings"), \_

ConnectionStringsSection)

' Toggle encryption.

If section.SectionInformation.IsProtected Then

section.SectionInformation.UnprotectSection()

Else

section.SectionInformation.ProtectSection( \_

"DataProtectionConfigurationProvider")

End If

' Save changes to the Web.config file.

config.Save()

End Sub

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl76_ctl00_ctl24_code');" \o "Copy Code)

static void ToggleWebEncrypt()

{

// Open the Web.config file.

Configuration config = WebConfigurationManager.

OpenWebConfiguration("~");

// Get the connectionStrings section.

ConnectionStringsSection section =

config.GetSection("connectionStrings")

as ConnectionStringsSection;

// Toggle encryption.

if (section.SectionInformation.IsProtected)

{

section.SectionInformation.UnprotectSection();

}

else

{

section.SectionInformation.ProtectSection(

"DataProtectionConfigurationProvider");

}

// Save changes to the Web.config file.

config.Save();

}