Contents

.NET Framework Guide

10/16/2019 • 2 minutes to read • Edit Online

NOTE

This .NET Framework content set includes information for .NET Framework versions 4.5 through 4.8. To download the .NET Framework, see Installing the .NET Framework. For a list of new features and changes in the NET Framework, see What's New in the .NET Framework. For a list of supported platforms, see .NET Framework System Requirements. For documentation about older versions of the .NET Framework, see .NET previous versions documentation.

The .NET Framework is a development platform for building apps for web, Windows, Windows Phone, Windows Server, and Microsoft Azure. It consists of the common language runtime (CLR) and the .NET Framework class library, which includes a broad range of functionality and support for many industry standards.

The .NET Framework provides many services, including memory management, type and memory safety, security, networking, and application deployment. It provides easy-to-use data structures and APIs that abstract the lower-level Windows operating system. You can use different programming languages with the .NET Framework, including C#, F#, and Visual Basic.

For a general introduction to the .NET Framework for both users and developers, see Getting Started. For an introduction to the architecture and key features of the .NET Framework, see the overview.

The .NET Framework can be used with Docker and with Windows Containers.

Installation

The .NET Framework comes with Windows, enabling you to run .NET Framework applications. You may need a later version of the .NET Framework than the one that comes with your Windows version. For more information, see Install the .NET Framework on Windows.

See Repair the .NET Framework to learn how to repair your .NET Framework installation if you're experiencing errors when installing the .NET Framework.

For more detailed information on downloading the .NET Framework, see Install the .NET Framework for developers.

In this section

- What's New
  Describes key new features and changes in the latest versions of the .NET Framework. Includes lists of obsolete types and members, and provides a guide for migrating your applications from the previous version of the .NET Framework.
- Get Started
  Provides a comprehensive overview of the .NET Framework and links to additional resources.
- Installation Guide
  Provides resources and guidance about .NET Framework installation and troubleshooting.
- Migration Guide
  Provides resources and a list of changes you need to consider if you're migrating your application to a new version of the .NET Framework.

- Development Guide
  Provides a guide to all key technology areas and tasks for application development, including creating, configuring, debugging, securing, and deploying your application, and information about dynamic programming, interoperability, extensibility, memory management, and threading.
- Tools
  Describes the tools that help you develop, configure, and deploy applications by using .NET Framework technologies.
- Additional class libraries and APIs
  Provides documentation for private .NET Framework APIs used by Microsoft tools.

See also

- .NET Framework Class Library

What's new in the .NET Framework

10/16/2019 • 86 minutes to read • Edit Online

This article summarizes key new features and improvements in the following versions of the .NET Framework:

- .NET Framework 4.8
- .NET Framework 4.7.2
- .NET Framework 4.7.1
- .NET Framework 4.7
- .NET Framework 4.6.2
- .NET Framework 4.6.1
- .NET 2015 and .NET Framework 4.6
- .NET Framework 4.5.2
- .NET Framework 4.5.1
- .NET Framework 4.5

This article does not provide comprehensive information about each new feature and is subject to change. For general information about the .NET Framework, see Getting Started. For supported platforms, see System Requirements. For download links and installation instructions, see Installation Guide.

> **NOTE**
> The .NET Framework team also releases features out of band with NuGet to expand platform support and to introduce new functionality, such as immutable collections and SIMD-enabled vector types. For more information, see Additional Class Libraries and APIs and The .NET Framework and Out-of-Band Releases. See a complete list of NuGet packages for the .NET Framework.

Introducing .NET Framework 4.8

.NET Framework 4.8 builds on previous versions of the .NET Framework 4.x by adding many new fixes and several new features while remaining a very stable product.

Downloading and installing .NET Framework 4.8

You can download .NET Framework 4.8 from the following locations:

- .NET Framework 4.8 Web Installer
- NET Framework 4.8 Offline Installer

.NET Framework 4.8 can be installed on Windows 10, Windows 8.1, Windows 7 SP1, and the corresponding server platforms starting with Windows Server 2008 R2 SP1. You can install .NET Framework 4.8 by using either the web installer or the offline installer. The recommended way for most users is to use the web installer.

You can target .NET Framework 4.8 in Visual Studio 2012 or later by installing the .NET Framework 4.8 Developer Pack.

What's new in .NET Framework 4.8

.NET Framework 4.8 introduces new features in the following areas:

- Base classes
- Windows Communication Foundation (WCF)
- Windows Presentation Foundation (WPF)
- Common language runtime

Improved accessibility, which allows an application to provide an appropriate experience for users of Assistive Technology, continues to be a major focus of .NET Framework 4.8. For information on accessibility improvements in .NET Framework 4.8, see What's new in accessibility in the .NET Framework.

Base classes

Reduced FIPS impact on Cryptography. In previous versions of the .NET Framework, managed cryptographic provider classes such as SHA256Managed throw a CryptographicException when the system cryptographic libraries are configured in "FIPS mode". These exceptions are thrown because the managed versions of the cryptographic provider classes, unlike the system cryptographic libraries, have not undergone FIPS (Federal Information Processing Standards) 140-2 certification. Because few developers have their development machines in FIPS mode, the exceptions are commonly thrown in production systems.

By default in applications that target .NET Framework 4.8, the following managed cryptography classes no longer throw a CryptographicException in this case:

- MD5Cng
- MD5CryptoServiceProvider
- RC2CryptoServiceProvider

- RijndaelManaged
- RIPEMD160Managed
- SHA256Managed

Instead, these classes redirect cryptographic operations to a system cryptography library. This change effectively removes a potentially confusing difference between developer environments and production environments and makes native components and managed components operate under the same cryptographic policy. Applications that depend on these exceptions can restore the previous behavior by setting the AppContext switch `Switch.System.Security.Cryptography.UseLegacyFipsThrow` to `true`. For more information, see Managed cryptography classes do not throw a CryptographyException in FIPS mode.

Use of updated version of ZLib

Starting with .NET Framework 4.5, the clrcompression.dll assembly uses ZLib, a native external library for data compression, in order to provide an implementation for the deflate algorithm. The .NET Framework 4.8, clrcompression.dll is updated to use ZLib Version 1.2.11, which includes several key improvements and fixes.

Windows Communication Foundation (WCF)

Introduction of ServiceHealthBehavior

Health endpoints are widely used by orchestration tools to manage services based on their health status. Health checks can also be used by monitoring tools to track and provide notifications about the availability and performance of a service.

ServiceHealthBehavior is a WCF service behavior that extends IServiceBehavior. When added to the ServiceDescription.Behaviors collection, a service behavior does the following:

- Returns service health status with HTTP response codes. You can specify in a query string the HTTP status code for a HTTP/GET health probe request.
- Publishes information about service health. Service-specific details, including service state, throttle counts, and capacity can be displayed by using an HTTP/GET request with the `?health` query string. Ease of access to such information is important when troubleshooting a misbehaving WCF service.

There are two ways to expose the health endpoint and publish WCF service health information:

- Through code. For example:

```
ServiceHost host = new ServiceHost(typeof(Service1),
                   new Uri("http://contoso:81/Service1"));
ServiceHealthBehavior healthBehavior =
    host.Description.Behaviors.Find<ServiceHealthBehavior>();
healthBehavior ??= new ServiceHealthBehavior();
host.Description.Behaviors.Add(healthBehavior);
```

```
Dim host As New ServiceHost(GetType(Service1),
            New Uri("http://contoso:81/Service1"))
Dim healthBehavior As ServiceHealthBehavior =
   host.Description.Behaviors.Find(Of ServiceHealthBehavior)()
If healthBehavior Is Nothing Then
   healthBehavior = New ServiceHealthBehavior()
End If
host.Description.Behaviors.Add(healthBehavior)
```

- By using a configuration file. For example:

```
<behaviors>
  <serviceBehaviors>
    <behavior name="DefaultBehavior">
      <serviceHealth httpsGetEnabled="true"/>
    </behavior>
  </serviceBehaviors>
</behaviors>
```

A service's health status can be queried by using query parameters such as `OnServiceFailure`, `OnDispatcherFailure`, `OnListenerFailure`, `OnThrottlePercentExceeded`), and an HTTP response code can be specified for each query parameter. If the HTTP response code is omitted for a query parameter, a 503 HTTP response code is used by default. For example:

- OnServiceFailure: `https://contoso:81/Service1?health&OnServiceFailure=450`

  A 450 HTTP response status code is returned when ServiceHost.State is greater than CommunicationState.Opened. Query parameters and examples:

- OnDispatcherFailure: `https://contoso:81/Service1?health&OnDispatcherFailure=455`

  A 455 HTTP response status code is returned when the state of any of the channel dispatchers is greater than CommunicationState.Opened.

- OnListenerFailure: `https://contoso:81/Service1?health&OnListenerFailure=465`

A 465 HTTP response status code is returned when the state of any of the channel listeners is greater than CommunicationState.Opened.

- OnThrottlePercentExceeded: `https://contoso:81/Service1?health&OnThrottlePercentExceeded= 70:350,95:500`

  Specifies the percentage {1 – 100} that triggers the response and its HTTP response code {200 – 599}. In this example:

  - If the percentage is greater than 95, a 500 HTTP response code is returned.
  - If the percentage or between 70 and 95, 350 is returned.
  - Otherwise, 200 is returned.

The service health status can be displayed either in HTML by specifying a query string like `https://contoso:81/Service1?health` or in XML by specifying a query string like `https://contoso:81/Service1?health&Xml`. A query string like `https://contoso:81/Service1?health&NoContent` returns an empty HTML page.

Windows Presentation Foundation (WPF)

High DPI enhancements

In .NET Framework 4.8, WPF adds support for Per-Monitor V2 DPI Awareness and Mixed-Mode DPI scaling. See High DPI Desktop Application Development on Windows for additional information about high DPI development.

.NET Framework 4.8 improves support for hosted HWNDs and Windows Forms interoperation in High-DPI WPF applications on platforms that support Mixed-Mode DPI scaling (starting with Windows 10 April 2018 Update). When hosted HWNDs or Windows Forms controls are created as Mixed-Mode DPI-scaled windows by calling SetThreadDpiHostingBehavior and SetThreadDpiAwarenessContext, they can be hosted in a Per-Monitor V2 WPF application and are sized and scaled appropriately. Such hosted content is not rendered at the native DPI; instead, the operating system scales the hosted content to the appropriate size. The support for Per-Monitor v2 DPI awareness mode also allows WPF controls to be hosted (i.e., parented) in a native window in a high-DPI application.

To enable support for Mixed-Mode High DPI scaling, you can set the following AppContext switches the application configuration file:

```
<runtime>
   <AppContextSwitchOverrides value = "Switch.System.Windows.DoNotScaleForDpiChanges=false;
Switch.System.Windows.DoNotUsePresentationDpiCapabilityTier2OrGreater=false"/>
</runtime>
```

Common language runtime

The runtime in .NET Framework 4.8 includes the following changes and improvements:

Improvements to the JIT compiler. The Just-in-time (JIT) compiler in .NET Framework 4.8 is based on the JIT compiler in .NET Core 2.1. Many of the optimizations and all of the bug fixes made to the .NET Core 2.1 JIT compiler are included in the .NET Framework 4.8 JIT compiler.

NGEN improvements. The runtime has improved its memory management for Native Image Generator (NGEN) images so that data mapped from NGEN images are not memory-resident. This reduces the surface area available to attacks that attempt to execute arbitrary code by modifying memory that will be executed.

Antimalware scanning for all assemblies. In previous versions of the .NET Framework, the runtime scans all assemblies loaded from disk using either Windows Defender or third-party antimalware software. However, assemblies loaded from other sources, such as by the Assembly.Load(Byte[]) method, are not scanned and can potentially contain undetected malware. Starting with .NET Framework 4.8 running on Windows 10, the runtime triggers a scan by antimalware solutions that implement the Antimalware Scan Interface (AMSI).

What's new in .NET Framework 4.7.2

.NET Framework 4.7.2 includes new features in the following areas:

- Base classes
- ASP.NET
- Networking
- SQL
- WPF
- ClickOnce

A continuing focus in .NET Framework 4.7.2 is improved accessibility, which allows an application to provide an appropriate experience for users of Assistive Technology. For information on accessibility improvements in .NET Framework 4.7.2, see What's new in accessibility in the .NET Framework.

Base classes

.NET Framework 4.7.2 features a large number of cryptographic enhancements, better decompression support for ZIP archives, and additional collection APIs.

New overloads of RSA.Create and DSA.Create

The DSA.Create(DSAParameters) and RSA.Create(RSAParameters) methods let you supply key parameters when instantiating a new DSA or RSA key. They allow you to replace code like the following:

```
// Before .NET Framework 4.7.2
using (RSA rsa = RSA.Create())
{
   rsa.ImportParameters(rsaParameters);
   // Other code to execute using the RSA instance.
}
```

```
' Before .NET Framework 4.7.2
Using rsa = RSA.Create()
   rsa.ImportParameters(rsaParameters)
   ' Other code to execute using the rsa instance.
End Using
```

with code like this:

```
// Starting with .NET Framework 4.7.2
using (RSA rsa = RSA.Create(rsaParameters))
{
   // Other code to execute using the rsa instance.
}
```

```
' Starting with .NET Framework 4.7.2
Using rsa = RSA.Create(rsaParameters)
   ' Other code to execute using the rsa instance.
End Using
```

The DSA.Create(Int32) and RSA.Create(Int32) methods let you generate new DSA or RSA keys with a specific key size. For example:

```
using (DSA dsa = DSA.Create(2048))
{
   // Other code to execute using the dsa instance.
}
```

```
Using dsa = DSA.Create(2048)
   ' Other code to execute using the dsa instance.
End Using
```

Rfc2898DeriveBytes constructors accept a hash algorithm name

The Rfc2898DeriveBytes class has three new constructors with a HashAlgorithmName parameter that identifies the HMAC algorithm to use when deriving keys. Instead of using SHA-1, developers should use a SHA-2-based HMAC like SHA-256, as shown in the following example:

```
private static byte[] DeriveKey(string password, out int iterations, out byte[] salt,
                                out HashAlgorithmName algorithm)
{
   iterations = 100000;
   algorithm = HashAlgorithmName.SHA256;

   const int SaltSize = 32;
   const int DerivedValueSize = 32;

   using (Rfc2898DeriveBytes pbkdf2 = new Rfc2898DeriveBytes(password, SaltSize,
                                                             iterations, algorithm))
   {
      salt = pbkdf2.Salt;
      return pbkdf2.GetBytes(DerivedValueSize);
   }
}
```

```
Private Shared Function DeriveKey(password As String, ByRef iterations As Integer,
                                  ByRef salt AS Byte(), ByRef algorithm As HashAlgorithmName) As Byte()
   iterations = 100000
   algorithm = HashAlgorithmName.SHA256

   Const SaltSize As Integer = 32
   Const  DerivedValueSize As Integer = 32

   Using pbkdf2 = New Rfc2898DeriveBytes(password, SaltSize, iterations, algorithm)
      salt = pbkdf2.Salt
      Return pbkdf2.GetBytes(DerivedValueSize)
   End Using
End Function
```

Support for ephemeral keys

PFX import can optionally load private keys directly from memory, bypassing the hard drive. When the new X509KeyStorageFlags.EphemeralKeySet flag is specified in an X509Certificate2 constructor or one of the overloads of the X509Certificate2.Import method, the private keys will be loaded as ephemeral keys. This prevents the keys from being visible on the disk. However:

- Since the keys are not persisted to disk, certificates loaded with this flag are not good candidates to add to an X509Store.
- Keys loaded in this manner are almost always loaded via Windows CNG. Therefore, callers must access the private key by calling extension methods, such as cert.GetRSAPrivateKey(). The X509Certificate2.PrivateKey property does not function.
- Since the legacy X509Certificate2.PrivateKey property does not work with certificates, developers should perform rigorous testing before switching to ephemeral keys.

Programmatic creation of PKCS#10 certification signing requests and X.509 public key certificates

Starting with .NET Framework 4.7.2, workloads can generate certificate signing requests (CSRs), which allows certificate request generation to be staged into existing tooling. This is frequently useful in test scenarios.

For more information and code examples, see "Programmatic creation of PKCS#10 certification signing requests and X.509 public key certificates" in the .NET Blog.

New SignerInfo members

Starting with .NET Framework 4.7.2, the SignerInfo class exposes more information about the signature. You can retrieve the value of the System.Security.Cryptography.Pkcs.SignerInfo.SignatureAlgorithm property to determine the signature algorithm used by the signer. SignerInfo.GetSignature can be called to get a copy of the cryptographic signature for this signer.

Leaving a wrapped stream open after CryptoStream is disposed

Starting with .NET Framework 4.7.2, the CryptoStream class has an additional constructor that allows Dispose to not close the wrapped stream. To leave the wrapped stream open after the CryptoStream instance is disposed, call the new CryptoStream constructor as follows:

```
var cStream = new CryptoStream(stream, transform, mode, leaveOpen: true);
```

```
Dim cStream = New CryptoStream(stream, transform, mode, leaveOpen:=true)
```

Decompression changes in DeflateStream

Starting with .NET Framework 4.7.2, the implementation of decompression operations in the DeflateStream class has changed to use native Windows APIs by default. Typically, this results in a substantial performance improvement.

Support for decompression by using Windows APIs is enabled by default for applications that target .NET Framework 4.7.2. Applications that target earlier versions of .NET Framework but are running under .NET Framework 4.7.2 can opt into this behavior by adding the following AppContext switch to the application configuration file:

```
<AppContextSwitchOverrides value="Switch.System.IO.Compression.DoNotUseNativeZipLibraryForDecompression=false" />
```

Additional collection APIs

.NET Framework 4.7.2 adds a number of new APIs to the SortedSet<T> and HashSet<T> types. These include:

- `TryGetValue` methods, which extend the try pattern used in other collection types to these two types. The methods are:
  - public bool HashSet<T>.TryGetValue(T equalValue, out T actualValue)
  - public bool SortedSet<T>.TryGetValue(T equalValue, out T actualValue)
- `Enumerable.To*` extension methods, which convert a collection to a HashSet<T>:
  - public static HashSet<TSource> ToHashSet<TSource>(this IEnumerable<TSource> source)
  - public static HashSet<TSource> ToHashSet<TSource>(this IEnumerable<TSource> source, IEqualityComparer<TSource> comparer)
- New HashSet<T> constructors that let you set the collection's capacity, which yields a performance benefit when you know the size of the HashSet<T> in advance:
  - public HashSet(int capacity)
  - public HashSet(int capacity, IEqualityComparer<T> comparer)

The ConcurrentDictionary<TKey,TValue> class includes new overloads of the AddOrUpdate and GetOrAdd methods to retrieve a value from the dictionary or to add it if it is not found, and to add a value to the dictionary or to update it if it already exists.

```
public TValue AddOrUpdate<TArg>(TKey key, Func<TKey, TArg, TValue> addValueFactory, Func<TKey, TValue, TArg, TValue> updateValueFactory, TArg
factoryArgument)

public TValue GetOrAdd<TArg>(TKey key, Func<TKey, TArg, TValue> valueFactory, TArg factoryArgument)
```

```
Public AddOrUpdate(Of TArg)(key As TKey, addValueFactory As Func(Of TKey, TArg, TValue), updateValueFactory As Func(Of TKey, TValue, TArg, TValue),
factoryArgument As TArg) As TValue

Public GetOrAdd(Of TArg)(key As TKey, valueFactory As Func(Of TKey, TArg, TValue), factoryArgument As TArg) As TValue
```

ASP.NET

Support for dependency injection in Web Forms

Dependency injection (DI) decouples objects and their dependencies so that an object's code no longer needs to be changed just because a dependency has changed. When developing ASP.NET applications that target .NET Framework 4.7.2, you can:

- Use setter-based, interface-based, and constructor-based injection in handlers and modules, Page instances, and user controls of ASP.NET web application projects.
- Use setter-based and interface-based injection in handlers and modules, Page instances, and user controls of ASP.NET web site projects.
- Plug in different dependency injection frameworks.

Support for same-site cookies

SameSite prevents a browser from sending a cookie along with a cross-site request. .NET Framework 4.7.2 adds a HttpCookie.SameSite property whose value is a System.Web.SameSiteMode enumeration member. If its value is SameSiteMode.Strict or SameSiteMode.Lax, ASP.NET adds the `SameSite` attribute to the set-cookie header. SameSite support applies to HttpCookie objects, as well as to FormsAuthentication and System.Web.SessionState cookies.

You can set SameSite for an HttpCookie object as follows:

```
var c = new HttpCookie("secureCookie", "same origin");
c.SameSite = SameSiteMode.Lax;
```

```
Dim c As New HttpCookie("secureCookie", "same origin")
c.SameSite = SameSiteMode.Lax
```

You can also configure SameSite cookies at the application level by modifying the web.config file:

```
<system.web>
   <httpCookies sameSite="Strict" />
</system.web>
```

You can add SameSite for FormsAuthentication and System.Web.SessionState cookies by modifying the web config file:

```
<system.web>
   <authentication mode="Forms">
      <forms cookieSameSite="Lax">
         <!-- ...   -->
      </forms>
   <authentication />
   <sessionSate cookieSameSite="Lax"></sessionState>
</system.web>
```

Networking

Implementation of HttpClientHandler properties

.NET Framework 4.7.1 added eight properties to the System.Net.Http.HttpClientHandler class. However, two threw a PlatformNotSupportedException. .NET Framework 4.7.2 now provides an implementation for these properties. The properties are:

- CheckCertificateRevocationList
- SslProtocols

SQLClient

Support for Azure Active Directory Universal Authentication and Multi-Factor authentication

Growing compliance and security demands require that many customers use multi-factor authentication (MFA). In addition, current best practices discourage including user passwords directly in connection strings. To support these changes, .NET Framework 4.7.2 extends SQLClient connection strings by adding a new value, "Active Directory Interactive", for the existing "Authentication" keyword to support MFA and Azure AD Authentication. The new interactive method supports native and federated Azure AD users as well as Azure AD guest users. When this method is used, the MFA authentication imposed by Azure AD is supported for SQL databases. In addition, the authentication process requests a user password to adhere to security best practices.

In previous versions of the .NET Framework, SQL connectivity supported only the SqlAuthenticationMethod.ActiveDirectoryPassword and SqlAuthenticationMethod.ActiveDirectoryIntegrated options. Both of these are part of the non-interactive ADAL protocol, which does not support MFA. With the new SqlAuthenticationMethod.ActiveDirectoryInteractive option, SQL connectivity supports MFA as well as existing authentication methods (password and integrated authentication), which allows users to enter user passwords interactively without persisting passwords in the connection string.

For more information and an example, see "SQL -- Azure AD Universal and Multi-factor Authentication Support" in the .NET Blog.

Support for Always Encrypted version 2

NET Framework 4.7.2 adds supports for enclave-based Always Encrypted. The original version of Always Encrypted is a client-side encryption technology in which encryption keys never leave the client. In enclave-based Always Encrypted, the client can optionally send the encryption keys to a secure enclave, which is a secure computational entity that can be considered part of SQL Server but that SQL Server code cannot tamper with. To support enclave-based Always Encrypted, .NET Framework 4.7.2 adds the following types and members to the System.Data.SqlClient namespace:

- SqlConnectionStringBuilder.EnclaveAttestationUrl, which specifies the Uri for enclave-based Always Encrypted.
- SqlColumnEncryptionEnclaveProvider, which is an abstract class from which all enclave providers are derived.
- SqlEnclaveSession, which encapsulates the state for a given enclave session.
- SqlEnclaveAttestationParameters, which provides the attestation parameters used by SQL Server to get information required to execute a particular Attestation Protocol.

The application configuration file then specifies a concrete implementation of the abstract System.Data.SqlClient.SqlColumnEncryptionEnclaveProvider class that provides the functionality for the enclave provider. For example:

```
<configuration>
  <configSections>
    <section name="SqlColumnEncryptionEnclaveProviders"
type="System.Data.SqlClient.SqlColumnEncryptionEnclaveProviderConfigurationSection,System.Data,Version=4.0.0.0,Culture=neutral,PublicKeyToken=b77a5
c561934e089"/>
  </configSections>
  <SqlColumnEncryptionEnclaveProviders>
    <providers>
      <add name="Azure" type="Microsoft.SqlServer.Management.AlwaysEncrypted.AzureEnclaveProvider,MyApp"/>
      <add name="HGS" type="Microsoft.SqlServer.Management.AlwaysEncrypted.HGSEnclaveProvider,MyApp" />
    </providers>
  </SqlColumnEncryptionEnclaveProviders >
</configuration>
```

The basic flow of enclave-based Always Encrypted is:

1. The user creates an AlwaysEncrypted connection to SQL Server that supported enclave-based Always Encrypted. The driver contacts the attestation service to ensure that it is connecting to right enclave.
2. Once the enclave has been attested, the driver establishes a secure channel with the secure enclave hosted on SQL Server.
3. The driver shares encryption keys authorized by the client with the secure enclave for the duration of the SQL connection.

Windows Presentation Foundation

Finding ResourceDictionaries by Source

Starting with .NET Framework 4.7.2, a diagnostic assistant can locate the ResourceDictionaries that have been created from a given source Uri. (This feature is for use by diagnostic assistants, not by production applications.) A diagnostic assistant such as Visual Studio's "Edit-and-Continue" facility lets its user edit a ResourceDictionary with the intent that the changes be applied to the running application. One step in achieving this is finding all the ResourceDictionaries that the running application has created from the dictionary that's being edited. For example, an application can declare a ResourceDictionary whose content is copied from a given source URI:

```
<ResourceDictionary Source="MyRD.xaml">
```

A diagnostic assistant that edits the original markup in *MyRD.xaml* can use the new feature to locate the dictionary. The feature is implemented by a new static method, ResourceDictionaryDiagnostics.GetResourceDictionariesForSource. The diagnostic assistant calls the new method using an absolute Uri that identifies the original markup, as illustrated by the following code:

```
IEnumerable<ResourceDictionary> dictionaries = ResourceDictionaryDiagnostics.GetResourceDictionariesForSource(new
Uri("pack://application:,,,/MyApp;component/MyRD.xaml"));
```

```
Dim dictionaries As IEnumerable(Of ResourceDictionary) = ResourceDictionaryDiagnostics.GetResourceDictionariesForSource(New
Uri("pack://application:,,,/MyApp;component/MyRD.xaml"))
```

The method returns an empty enumerable unless VisualDiagnostics is enabled and the `ENABLE_XAML_DIAGNOSTICS_SOURCE_INFO` environment variable is set.

Finding ResourceDictionary owners

Starting with .NET Framework 4.7.2, a diagnostic assistant can locate the owners of a given ResourceDictionary. (The feature is for use by diagnostic assistants and not by production applications.) Whenever a change is made to a ResourceDictionary, WPF automatically finds all DynamicResource references that might be affected by the change.

A diagnostic assistant such as Visual Studio's "Edit-and-Continue" facility may want extend this to handle StaticResource references. The first step in this process is to find the owners of the dictionary; that is, to find all the objects whose `Resources` property refers to the dictionary (either directly, or

indirectly via the ResourceDictionary.MergedDictionaries property). Three new static methods implemented on the System.Windows.Diagnostics.ResourceDictionaryDiagnostics class, one for each of the base types that has a `Resources` property, support this step:

- `public static IEnumerable<FrameworkElement> GetFrameworkElementOwners(ResourceDictionary dictionary);`
- `public static IEnumerable<FrameworkContentElement> GetFrameworkContentElementOwners(ResourceDictionary dictionary);`
- `public static IEnumerable<Application> GetApplicationOwners(ResourceDictionary dictionary);`

These methods return an empty enumerable unless VisualDiagnostics is enabled and the `ENABLE_XAML_DIAGNOSTICS_SOURCE_INFO` environment variable is set.

Finding StaticResource references

A diagnostic assistant can now receive a notification whenever a StaticResource reference is resolved. (The feature is for use by diagnostic assistants, not by production applications.) A diagnostic assistant such as Visual Studio's "Edit-and-Continue" facility may want to update all uses of a resource when its value in a ResourceDictionary changes. WPF does this automatically for DynamicResource references, but it intentionally does not do so for StaticResource references. Starting with .NET Framework 4.7.2, the diagnostic assistant can use these notifications to locate those uses of the static resource.

The notification is implemented by the new ResourceDictionaryDiagnostics.StaticResourceResolved event:

```
public static event EventHandler<StaticResourceResolvedEventArgs> StaticResourceResolved;
```

```
Public Shared Event StaticResourceResolved As EventHandler(Of StaticResourceResolvedEventArgs)
```

This event is raised whenever the runtime resolves a StaticResource reference. The StaticResourceResolvedEventArgs arguments describe the resolution, and indicate the object and property that host the StaticResource reference and the ResourceDictionary and key used for the resolution:

```
public class StaticResourceResolvedEventArgs : EventArgs
{
   public Object TargetObject { get; }

   public Object TargetProperty { get; }

   public ResourceDictionary ResourceDictionary { get; }

   public object ResourceKey { get; }
}
```

```
Public Class StaticResourceResolvedEventArgs : Inherits EventArgs
   Public ReadOnly Property TargetObject As Object
   Public ReadOnly Property TargetProperty As Object
   Public ReadOnly Property ResourceDictionary As ResourceDictionary
   Public ReadOnly Property ResourceKey As Object
End Class
```

The event is not raised (and its `add` accessor is ignored) unless VisualDiagnostics is enabled and the `ENABLE_XAML_DIAGNOSTICS_SOURCE_INFO` environment variable is set.

ClickOnce

HDPI-aware applications for Windows Forms, Windows Presentation Foundation (WPF), and Visual Studio Tools for Office (VSTO) can all be deployed by using ClickOnce. If the following entry is found in the application manifest, deployment will succeed under .NET Framework 4.7.2:

```
<windowsSettings>
   <dpiAware xmlns="http://schemas.microsoft.com/SMI/2005/WindowsSettings">true</dpiAware>
</windowsSettings>
```

For Windows Forms application, the previous workaround of setting DPI awareness in the application configuration file rather than the application manifest is no longer necessary for ClickOnce deployment to succeed.

What's new in .NET Framework 4.7.1

.NET Framework 4.7.1 includes new features in the following areas:

- Base classes
- Common language runtime (CLR)
- Networking
- ASP.NET

In addition, a major focus in .NET Framework 4.7.1 is improved accessibility, which allows an application to provide an appropriate experience for users of Assistive Technology. For information on accessibility improvements in .NET Framework 4.7.1, see What's new in accessibility in the .NET Framework.

Base classes

Support for .NET Standard 2.0

.NET Standard defines a set of APIs that must be available on each .NET implementation that supports that version of the standard. .NET Framework 4.7.1 fully supports .NET Standard 2.0 and adds about 200 APIs that are defined in .NET Standard 2.0 and are missing from .NET Framework 4.6.1, 4.6.2, and 4.7. (Note that these versions of the .NET Framework support .NET Standard 2.0 only if additional .NET Standard support files are also deployed on the target system.) For more information, see "BCL - .NET Standard 2.0 Support" in the .NET Framework 4.7.1 Runtime and Compiler Features blog post.

Support for configuration builders

Configuration builders allow developers to inject and build configuration settings for applications dynamically at run time. Custom configuration builders can be used to modify existing data in a configuration section or to build a configuration section entirely from scratch. Without configuration builders, .config files are static, and their settings are defined some time before an application is launched.

To create a custom configuration builder, you derive your builder from the abstract ConfigurationBuilder class and override its ConfigurationBuilder.ProcessConfigurationSection and ConfigurationBuilder.ProcessRawXml. You also define your builders in your .config file. For more information, see the "Configuration Builders" section in the .NET Framework 4.7.1 ASP.NET and Configuration Features blog post.

Run-time feature detection

The System.Runtime.CompilerServices.RuntimeFeature class provides a mechanism for determine whether a predefined feature is supported on a given .NET implementation at compile time or run time. At compile time, a compiler can check whether a specified field exists to determine whether the feature is supported; if so, it can emit code that takes advantage of that feature. At run time, an application can call the RuntimeFeature.IsSupported method before emitting code at runtime. For more information, see Add helper method to describe features supported by the runtime.

Value tuple types are serializable

Starting with .NET Framework 4.7.1, System.ValueTuple and its associated generic types are marked as Serializable, which allows binary serialization. This should make migrating Tuple types, such as Tuple<T1,T2,T3> and Tuple<T1,T2,T3,T4>, to value tuple types easier. For more information, see "Compiler -- ValueTuple is Serializable" in the .NET Framework 4.7.1 Runtime and Compiler Features blog post.

Support for read-only references

.NET Framework 4.7.1 adds the System.Runtime.CompilerServices.IsReadOnlyAttribute. This attribute is used by language compilers to mark members that have read-only ref return types or parameters. For more information, see "Compiler -- Support for ReadOnlyReferences" in the .NET Framework 4.7.1 Runtime and Compiler Features blog post. For information on ref return values, see Ref return values and ref locals (C# Guide) and Ref return values (Visual Basic).

Common language runtime (CLR)

Garbage collection performance improvements

Changes to garbage collection (GC) in .NET Framework 4.7.1 improve overall performance, especially for Large Object Heap (LOH) allocations. In .NET Framework 4.7.1, separate locks are used for Small Object Heap (SOH) and LOH allocations, which allows LOH allocations to occur when Background GC (BGC) is sweeping the SOH. As a result, applications that make a large number of LOH allocations should see a reduction in allocation lock contention and improved performance. For more information, see the "Runtime -- GC Performance Improvements" section in the .NET Framework 4.7.1 Runtime and Compiler Features blog post.

Networking

SHA-2 support for Message.HashAlgorithm

In .NET Framework 4.7 and earlier versions, the Message.HashAlgorithm property supported values of HashAlgorithm.Md5 and HashAlgorithm.Sha only. Starting with .NET Framework 4.7.1, HashAlgorithm.Sha256, HashAlgorithm.Sha384, and HashAlgorithm.Sha512 are also supported. Whether this value is actually used depends on MSMQ, since the Message instance itself does no hashing but simply passes on values to MSMQ. For more information, see the "SHA-2 support for Message.HashAlgorithm" section in the .NET Framework 4.7.1 ASP.NET and Configuration features blog post.

ASP.NET

Execution steps in ASP.NET applications

ASP.NET processes requests in a predefined pipeline that includes 23 events. ASP.NET executes each event handler as an execution step. In versions of ASP.NET up to .NET Framework 4.7, ASP.NET can't flow the execution context due to switching between native and managed threads. Instead, ASP.NET selectively flows only the HttpContext. Starting with .NET Framework 4.7.1, the HttpApplication.OnExecuteRequestStep(Action<HttpContextBase,Action>) method also allows modules to restore ambient data. This feature is targeted at libraries concerned with tracing, profiling, diagnostics, or transactions, for example, that care about the execution flow of the application. For more information, see the "ASP.NET Execution Step Feature" in the .NET Framework 4.7.1 ASP.NET and Configuration Features blog post.

ASP.NET HttpCookie parsing

.NET Framework 4.7.1 includes a new method, HttpCookie.TryParse, that provides a standardized way to create an HttpCookie object from a string and accurately assign cookie values such as expiration date and path. For more information, see "ASP.NET HttpCookie parsing" in the .NET Framework 4.7.1 ASP.NET and Configuration Features blog post.

SHA-2 hash options for ASP.NET forms authentication credentials

In .NET Framework 4.7 and earlier versions, ASP.NET allowed developers to store user credentials with hashed passwords in configuration files using either MD5 or SHA1. Starting with .NET Framework 4.7.1, ASP.NET also supports new secure SHA-2 hash options such as SHA256, SHA384, and SHA512. SHA1 remains the default, and a non-default hash algorithm can be defined in the web configuration file. For example:

```
<system.web>
    <authentication mode="Forms">
        <forms loginUrl="~/login.aspx">
          <credentials passwordFormat="SHA512">
            <user name="jdoe"
password="6D003E98EA1C7F04ABF8FCB375388907B7F3EE06F278DB966BE960E7CBBD103DF30CA6D61F7E7FD981B2E4E3A64D43C836A4BEDCA165C33B163E6BCDC538A664" />
          </credentials>
        </forms>
    </authentication>
</system.web>
```

What's new in .NET Framework 4.7

.NET Framework 4.7 includes new features in the following areas:

- Base classes
- Networking
- ASP.NET
- Windows Communication Foundation (WCF)
- Windows Forms
- Windows Presentation Foundation (WPF)

For a list of new APIs added to .NET Framework 4.7, see .NET Framework 4.7 API Changes on GitHub. For a list of feature improvements and bug fixes in .NET Framework 4.7, see .NET Framework 4.7 List of Changes on GitHub. For additional information, see Announcing the .NET Framework 4.7 in the .NET blog.

Base classes

.NET Framework 4.7 improves serialization by the DataContractJsonSerializer:

Enhanced functionality with Elliptic Curve Cryptography (ECC)*

In .NET Framework 4.7, `ImportParameters(ECParameters)` methods were added to the ECDsa and ECDiffieHellman classes to allow for an object to represent an already-established key. An `ExportParameters(Boolean)` method was also added for exporting the key using explicit curve parameters.

.NET Framework 4.7 also adds support for additional curves (including the Brainpool curve suite), and has added predefined definitions for ease-of-creation through the new Create and Create factory methods.

You can see an example of .NET Framework 4.7 cryptography improvements on GitHub.

Better support for control characters by the DataContractJsonSerializer

In .NET Framework 4.7, the DataContractJsonSerializer serializes control characters in conformity with the ECMAScript 6 standard. This behavior is enabled by default for applications that target .NET Framework 4.7, and is an opt-in feature for applications that are running under .NET Framework 4.7 but target a previous version of the .NET Framework. For more information, see Retargeting Changes in the .NET Framework 4.7.

Networking

.NET Framework 4.7 adds the following network-related feature:

Default operating system support for TLS protocols*

The TLS stack, which is used by System.Net.Security.SslStream and up-stack components such as HTTP, FTP, and SMTP, allows developers to use the default TLS protocols supported by the operating system. Developers need no longer hard-code a TLS version.

ASP.NET

In .NET Framework 4.7, ASP.NET includes the following new features:

Object Cache Extensibility

Starting with .NET Framework 4.7, ASP.NET adds a new set of APIs that allow developers to replace the default ASP.NET implementations for in-memory object caching and memory monitoring. Developers can now replace any of the following three components if the ASP.NET implementation is not adequate:

- **Object Cache Store**. By using the new cache providers configuration section, developers can plug in new implementations of an object cache for an ASP.NET application by using the new **ICacheStoreProvider** interface.
- **Memory monitoring**. The default memory monitor in ASP.NET notifies applications when they are running close to the configured private bytes limit for the process, or when the machine is low on total available physical RAM. When these limits are near, notifications are fired. For some applications, notifications are fired too close to the configured limits to allow for useful reactions. Developers can now write their own memory monitors to replace the default by using the ApplicationMonitors.MemoryMonitor property.
- **Memory Limit Reactions**. By default, ASP.NET attempts to trim the object cache and periodically call GC.Collect when the private byte process limit is near. For some applications, the frequency of calls to GC.Collect or the amount of cache that is trimmed are inefficient. Developers can now replace or supplement the default behavior by subscribing **IObserver** implementations to the application's memory monitor.

Windows Communication Foundation (WCF)

Windows Communication Foundation (WCF) adds the following features and changes:

Ability to configure the default message security settings to TLS 1.1 or TLS 1.2

Starting with .NET Framework 4.7, WCF allows you to configure TSL 1.1 or TLS 1.2 in addition to SSL 3.0 and TSL 1.0 as the default message security protocol. This is an opt-in setting; to enable it, you must add the following entry to your application configuration file:

```
<runtime>
   <AppContextSwitchOverrides
value="Switch.System.ServiceModel.DisableUsingServicePointManagerSecurityProtocols=false;Switch.System.Net.DontEnableSchUseStrongCrypto=false" />
</runtime>
```

Improved reliability of WCF applications and WCF serialization

WCF includes a number of code changes that eliminate race conditions, thereby improving performance and the reliability of serialization options. These include:

- Better support for mixing asynchronous and synchronous code in calls to **SocketConnection.BeginRead** and **SocketConnection.Read**.
- Improved reliability when aborting a connection with **SharedConnectionListener** and **DuplexChannelBinder**.
- Improved reliability of serialization operations when calling the FormatterServices.GetSerializableMembers(Type) method.
- Improved reliability when removing a waiter by calling the **ChannelSynchronizer.RemoveWaiter** method.

Windows Forms

In .NET Framework 4.7, Windows Forms improves support for high DPI monitors.

High DPI support

Starting with applications that target .NET Framework 4.7, the .NET Framework features high DPI and dynamic DPI support for Windows Forms applications. High DPI support improves the layout and appearance of forms and controls on high DPI monitors. Dynamic DPI changes the layout and appearance of forms and controls when the user changes the DPI or display scale factor of a running application.

High DPI support is an opt-in feature that you configure by defining a <System.Windows.Forms.ConfigurationSection> section in your application configuration file. For more information on adding high DPI support and dynamic DPI support to your Windows Forms application, see High DPI Support in Windows Forms.

Windows Presentation Foundation (WPF)

In .NET Framework 4.7, WPF includes the following enhancements:

Support for a touch/stylus stack based on Windows WM_POINTER messages

You now have the option of using a touch/stylus stack based on WM_POINTER messages instead of the Windows Ink Services Platform (WISP). This is an opt-in feature in the .NET Framework. For more information, see Retargeting Changes in the .NET Framework 4.7.

New implementation for WPF printing APIs

WPF's printing APIs in the System.Printing.PrintQueue class call the Windows Print Document Package API instead of the deprecated XPS Print API. For the impact of this change on application compatibility, see Retargeting Changes in the .NET Framework 4.7.

What's new in .NET Framework 4.6.2

The .NET Framework 4.6.2 includes new features in the following areas:

- ASP.NET
- Character categories
- Cryptography
- SqlClient
- Windows Communication Foundation
- Windows Presentation Foundation (WPF)
- Windows Workflow Foundation (WF)
- ClickOnce
- Converting Windows Forms and WPF apps to UWP apps
- Debugging improvements

For a list of new APIs added to .NET Framework 4.6.2, see .NET Framework 4.6.2 API Changes on GitHub. For a list of feature improvements and bug fixes in .NET Framework 4.6.2, see .NET Framework 4.6.2 List of Changes on GitHub. For additional information, see Announcing .NET Framework 4.6.2 in the .NET blog.

ASP.NET

In the .NET Framework 4.6.2, ASP.NET includes the following enhancements:

Improved support for localized error messages in data annotation validators

Data annotation validators enable you to perform validation by adding one or more attributes to a class property. The attribute's

ValidationAttribute.ErrorMessage element defines the text of the error message if validation fails. Starting with the .NET Framework 4.6.2, ASP.NET makes it easy to localize error messages. Error messages will be localized if:

1. The ValidationAttribute.ErrorMessage is provided in the validation attribute.
2. The resource file is stored in the App_LocalResources folder.
3. The name of the localized resources file has the form `DataAnnotation.Localization.{`*name*`}.resx`, where *name* is a culture name in the format *languageCode*`-`*country/regionCode* or *languageCode*.
4. The key name of the resource is the string assigned to the ValidationAttribute.ErrorMessage attribute, and its value is the localized error message.

For example, the following data annotation attribute defines the default culture's error message for an invalid rating.

```
public class RatingInfo
{
   [Required(ErrorMessage = "The rating must be between 1 and 10.")]
   [Display(Name = "Your Rating")]
   public int Rating { get; set; }
}
```

```
Public Class RatingInfo
   <Required(ErrorMessage = "The rating must be between 1 and 10.")>
   <Display(Name = "Your Rating")>
   Public Property Rating As Integer = 1
End Class
```

You can then create a resource file, DataAnnotation.Localization.fr.resx, whose key is the error message string and whose value is the localized error message. The file must be found in the `App.LocalResources` folder. For example, the following is the key and its value in a localized French (fr) language error message:

NAME	VALUE
The rating must be between 1 and 10.	La note doit être comprise entre 1 et 10.

In addition, data annotation localization is extensible. Developers can plug in their own string localizer provider by implementing the IStringLocalizerProvider interface to store localization string somewhere other than in a resource file.

Async support with session-state store providers

ASP.NET now allows task-returning methods to be used with session-state store providers, thereby allowing ASP.NET apps to get the scalability benefits of async. To supports asynchronous operations with session state store providers, ASP.NET includes a new interface, System.Web.SessionState.ISessionStateModule, which inherits from IHttpModule and allows developers to implement their own session-state module and async session store providers. The interface is defined as follows:

```
public interface ISessionStateModule : IHttpModule {
    void ReleaseSessionState(HttpContext context);
    Task ReleaseSessionStateAsync(HttpContext context);
}
```

```
Public Interface ISessionStateModule : Inherits IHttpModule
   Sub ReleaseSessionState(context As HttpContext)
   Function ReleaseSessionStateAsync(context As HttpContext) As Task
End Interface
```

In addition, the SessionStateUtility class includes two new methods, IsSessionStateReadOnly and IsSessionStateRequired, that can be used to support asynchronous operations.

Async support for output-cache providers

Starting with the .NET Framework 4.6.2, task-returning methods can be used with output-cache providers to provide the scalability benefits of async. Providers that implement these methods reduce thread-blocking on a web server and improve the scalability of an ASP.NET service.

The following APIs have been added to support asynchronous output-cache providers:

- The System.Web.Caching.OutputCacheProviderAsync class, which inherits from System.Web.Caching.OutputCacheProvider and allows developers to implement an asynchronous output-cache provider.
- The OutputCacheUtility class, which provides helper methods for configuring the output cache.
- 18 new methods in the System.Web.HttpCachePolicy class. These include GetCacheability, GetCacheExtensions, GetETag, GetETagFromFileDependencies, GetMaxAge, GetMaxAge, GetNoStore, GetNoTransforms, GetOmitVaryStar, GetProxyMaxAge, GetRevalidation, GetUtcLastModified, GetVaryByCustom, HasSlidingExpiration, and IsValidUntilExpires.
- 2 new methods in the System.Web.HttpCacheVaryByContentEncodings class: GetContentEncodings and SetContentEncodings.

- 2 new methods in the System.Web.HttpCacheVaryByHeaders class: GetHeaders and SetHeaders.
- 2 new methods in the System.Web.HttpCacheVaryByParams class: GetParams and SetParams.
- In the System.Web.Caching.AggregateCacheDependency class, the GetFileDependencies method.
- In the CacheDependency, the GetFileDependencies method.

Character categories

Characters in the .NET Framework 4.6.2 are classified based on the Unicode Standard, Version 8.0.0. In .NET Framework 4.6 and .NET Framework 4.6.1, characters were classified based on Unicode 6.3 character categories.

Support for Unicode 8.0 is limited to the classification of characters by the CharUnicodeInfo class and to types and methods that rely on it. These include the StringInfo class, the overloaded Char.GetUnicodeCategory method, and the character classes recognized by the .NET Framework regular expression engine. Character and string comparison and sorting is unaffected by this change and continues to rely on the underlying operating system or, on Windows 7 systems, on character data provided by the .NET Framework.

For changes in character categories from Unicode 6.0 to Unicode 7.0, see The Unicode Standard, Version 7.0.0 at The Unicode Consortium website. For changes from Unicode 7.0 to Unicode 8.0, see The Unicode Standard, Version 8.0.0 at The Unicode Consortium website.

Cryptography

Support for X509 certificates containing FIPS 186-3 DSA

The .NET Framework 4.6.2 adds support for DSA (Digital Signature Algorithm) X509 certificates whose keys exceed the FIPS 186-2 1024-bit limit.

In addition to supporting the larger key sizes of FIPS 186-3, the .NET Framework 4.6.2 allows computing signatures with the SHA-2 family of hash algorithms (SHA256, SHA384, and SHA512). FIPS 186-3 support is provided by the new System.Security.Cryptography.DSACng class.

In keeping with recent changes to the RSA class in .NET Framework 4.6 and the ECDsa class in .NET Framework 4.6.1, the DSA abstract base class in .NET Framework 4.6.2 has additional methods to allow callers to use this functionality without casting. You can call the DSACertificateExtensions.GetDSAPrivateKey extension method to sign data, as the following example shows.

```
public static byte[] SignDataDsaSha384(byte[] data, X509Certificate2 cert)
{
    using (DSA dsa = cert.GetDSAPrivateKey())
    {
        return dsa.SignData(data, HashAlgorithmName.SHA384);
    }
}
```

```
Public Shared Function SignDataDsaSha384(data As Byte(), cert As X509Certificate2) As Byte()
    Using DSA As DSA = cert.GetDSAPrivateKey()
        Return DSA.SignData(data, HashAlgorithmName.SHA384)
    End Using
End Function
```

And you can call the DSACertificateExtensions.GetDSAPublicKey extension method to verify signed data, as the following example shows.

```
public static bool VerifyDataDsaSha384(byte[] data, byte[] signature, X509Certificate2 cert)
{
    using (DSA dsa = cert.GetDSAPublicKey())
    {
        return dsa.VerifyData(data, signature, HashAlgorithmName.SHA384);
    }
}
```

```
 Public Shared Function VerifyDataDsaSha384(data As Byte(), signature As Byte(), cert As X509Certificate2) As Boolean
    Using dsa As DSA = cert.GetDSAPublicKey()
        Return dsa.VerifyData(data, signature, HashAlgorithmName.SHA384)
    End Using
End Function
```

Increased clarity for inputs to ECDiffieHellman key derivation routines

.NET Framework 3.5 added support for Elliptic Curve Diffie-Hellman Key Agreement with three different Key Derivation Function (KDF) routines. The inputs to the routines, and the routines themselves, were configured via properties on the ECDiffieHellmanCng object. But since not every routine read every input property, there was ample room for confusion on the past of the developer.

To address this in the .NET Framework 4.6.2, the following three methods have been added to the ECDiffieHellman base class to more clearly represent these KDF routines and their inputs:

ECDIFFIEHELLMAN METHOD	DESCRIPTION

ECDIFFIEHELLMAN METHOD	DESCRIPTION
DeriveKeyFromHash(ECDiffieHellmanPublicKey, HashAlgorithmName, Byte[], Byte[])	Derives key material using the formula HASH(secretPrepend || *x* || secretAppend) HASH(secretPrepend OrElse *x* OrElse secretAppend) where *x* is the computed result of the EC Diffie-Hellman algorithm.
DeriveKeyFromHmac(ECDiffieHellmanPublicKey, HashAlgorithmName, Byte[], Byte[], Byte[])	Derives key material using the formula HMAC(hmacKey, secretPrepend || *x* || secretAppend) HMAC(hmacKey, secretPrepend OrElse *x* OrElse secretAppend) where *x* is the computed result of the EC Diffie-Hellman algorithm.
DeriveKeyTls(ECDiffieHellmanPublicKey, Byte[], Byte[])	Derives key material using the TLS pseudo-random function (PRF) derivation algorithm.

Support for persisted-key symmetric encryption

The Windows cryptography library (CNG) added support for storing persisted symmetric keys and using hardware-stored symmetric keys, and the .NET Framework 4.6.2 made it possible for developers to make use of this feature. Since the notion of key names and key providers is implementation-specific, using this feature requires utilizing the constructor of the concrete implementation types instead of the preferred factory approach (such as calling `Aes.Create`).

Persisted-key symmetric encryption support exists for the AES (AesCng) and 3DES (TripleDESCng) algorithms. For example:

```
public static byte[] EncryptDataWithPersistedKey(byte[] data, byte[] iv)
{
    using (Aes aes = new AesCng("AesDemoKey", CngProvider.MicrosoftSoftwareKeyStorageProvider))
    {
        aes.IV = iv;

        // Using the zero-argument overload is required to make use of the persisted key
        using (ICryptoTransform encryptor = aes.CreateEncryptor())
        {
            if (!encryptor.CanTransformMultipleBlocks)
            {
                throw new InvalidOperationException("This is a sample, this case wasn't handled...");
            }

            return encryptor.TransformFinalBlock(data, 0, data.Length);
        }
    }
}
```

```
Public Shared Function EncryptDataWithPersistedKey(data As Byte(), iv As Byte()) As Byte()
    Using Aes As Aes = New AesCng("AesDemoKey", CngProvider.MicrosoftSoftwareKeyStorageProvider)
        Aes.IV = iv

        ' Using the zero-argument overload Is required to make use of the persisted key
        Using encryptor As ICryptoTransform = Aes.CreateEncryptor()
            If Not encryptor.CanTransformMultipleBlocks Then
                Throw New InvalidOperationException("This is a sample, this case wasn't handled...")
            End If
            Return encryptor.TransformFinalBlock(data, 0, data.Length)
        End Using
    End Using
End Function
```

SignedXml support for SHA-2 hashing

The .NET Framework 4.6.2 adds support to the SignedXml class for RSA-SHA256, RSA-SHA384, and RSA-SHA512 PKCS#1 signature methods, and SHA256, SHA384, and SHA512 reference digest algorithms.

The URI constants are all exposed on SignedXml:

SIGNEDXML FIELD	CONSTANT
XmlDsigSHA256Url	"http://www.w3.org/2001/04/xmlenc#sha256"
XmlDsigRSASHA256Url	"http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"
XmlDsigSHA384Url	"http://www.w3.org/2001/04/xmldsig-more#sha384"

SIGNEDXML FIELD	CONSTANT
XmlDsigRSASHA384Url	"http://www.w3.org/2001/04/xmldsig-more#rsa-sha384"
XmlDsigSHA512Url	"http://www.w3.org/2001/04/xmlenc#sha512"
XmlDsigRSASHA512Url	"http://www.w3.org/2001/04/xmldsig-more#rsa-sha512"

Any programs that have registered a custom SignatureDescription handler into CryptoConfig to add support for these algorithms will continue to function as they did in the past, but since there are now platform defaults, the CryptoConfig registration is no longer necessary.

SqlClient

.NET Framework Data Provider for SQL Server (System.Data.SqlClient) includes the following new features in the .NET Framework 4.6.2:

Connection pooling and timeouts with Azure SQL databases

When connection pooling is enabled and a timeout or other login error occurs, an exception is cached, and the cached exception is thrown on any subsequent connection attempt for the next 5 seconds to 1 minute. For more details, see SQL Server Connection Pooling (ADO.NET).

This behavior is not desirable when connecting to Azure SQL Databases, since connection attempts can fail with transient errors that are typically recovered quickly. To better optimize the connection retry experience, the connection pool blocking period behavior is removed when connections to Azure SQL Databases fail.

The addition of the new `PoolBlockingPeriod` keyword lets you to select the blocking period best suited for your app. Values include:

Auto

The connection pool blocking period for an application that connects to an Azure SQL Database is disabled, and the connection pool blocking period for an application that connects to any other SQL Server instance is enabled. This is the default value. If the Server endpoint name ends with any of the following, they are considered Azure SQL Databases:

- .database.windows.net
- .database.chinacloudapi.cn
- .database.usgovcloudapi.net
- .database.cloudapi.de

AlwaysBlock

The connection pool blocking period is always enabled.

NeverBlock

The connection pool blocking period is always disabled.

Enhancements for Always Encrypted

SQLClient introduces two enhancements for Always Encrypted:

- To improve performance of parameterized queries against encrypted database columns, encryption metadata for query parameters is now cached. With the SqlConnection.ColumnEncryptionQueryMetadataCacheEnabled property set to `true` (which is the default value), if the same query is called multiple times, the client retrieves parameter metadata from the server only once.
- Column encryption key entries in the key cache are now evicted after a configurable time interval, set using the SqlConnection.ColumnEncryptionKeyCacheTtl property.

Windows Communication Foundation

In the .NET Framework 4.6.2, Windows Communication Foundation has been enhanced in the following areas:

WCF transport security support for certificates stored using CNG

WCF transport security supports certificates stored using the Windows cryptography library (CNG). In the .NET Framework 4.6.2, this support is limited to using certificates with a public key that has an exponent no more than 32 bits in length. When an application targets the .NET Framework 4.6.2, this feature is on by default.

For applications that target the .NET Framework 4.6.1 and earlier but are running on the .NET Framework 4.6.2, this feature can be enabled by adding the following line to the <runtime> section of the app.config or web.config file.

```
<AppContextSwitchOverrides
    value="Switch.System.ServiceModel.DisableCngCertificates=false"
/>
```

This can also be done programmatically with code like the following:

```
private const string DisableCngCertificates = @"Switch.System.ServiceModel.DisableCngCertificates";
AppContext.SetSwitch(disableCngCertificates, false);
```

```
Const DisableCngCertificates As String = "Switch.System.ServiceModel.DisableCngCertificates"
AppContext.SetSwitch(disableCngCertificates, False)
```

Better support for multiple daylight saving time adjustment rules by the DataContractJsonSerializer class

Customers can use an application configuration setting to determine whether the DataContractJsonSerializer class supports multiple adjustment rules for a single time zone. This is an opt-in feature. To enable it, add the following setting to your app.config file:

```
<runtime>
     <AppContextSwitchOverrides value="Switch.System.Runtime.Serialization.DoNotUseTimeZoneInfo=false" />
</runtime>
```

When this feature is enabled, a DataContractJsonSerializer object uses the TimeZoneInfo type instead of the TimeZone type to deserialize date and time data. TimeZoneInfo supports multiple adjustment rules, which makes it possible to work with historic time zone data; TimeZone does not.

For more information on the TimeZoneInfo structure and time zone adjustments, see Time Zone Overview.

NetNamedPipeBinding best match

WCF has a new app setting that can be set on client applications to ensure they always connect to the service listening on the URI that best matches the one that they request. With this app setting set to `false` (the default), it is possible for clients using NetNamedPipeBinding to attempt to connect to a service listening on a URI that is a substring of the requested URI.

For example, a client tries to connect to a service listening at `net.pipe://localhost/Service1`, but a different service on that machine running with administrator privilege is listening at `net.pipe://localhost`. With this app setting set to `false`, the client would attempt to connect to the wrong service. After setting the app setting to `true`, the client will always connect to the best matching service.

> **NOTE**
> Clients using NetNamedPipeBinding find services based on the service's base address (if it exists) rather than the full endpoint address. To ensure this setting always works the service should use a unique base address.

To enable this change, add the following app setting to your client application's App.config or Web.config file:

```
<configuration>
    <appSettings>
        <add key="wcf:useBestMatchNamedPipeUri" value="true" />
    </appSettings>
</configuration>
```

SSL 3.0 is not a default protocol

When using NetTcp with transport security and a credential type of certificate, SSL 3.0 is no longer a default protocol used for negotiating a secure connection. In most cases, there should be no impact to existing apps, because TLS 1.0 is included in the protocol list for NetTcp. All existing clients should be able to negotiate a connection using at least TLS 1.0. If Ssl3 is required, use one of the following configuration mechanisms to add it to the list of negotiated protocols.

- The SslStreamSecurityBindingElement.SslProtocols property
- The TcpTransportSecurity.SslProtocols property
- The <transport> section of the <netTcpBinding> section
- The <sslStreamSecurity> section of the <customBinding> section

Windows Presentation Foundation (WPF)

In the .NET Framework 4.6.2, Windows Presentation Foundation has been enhanced in the following areas:

Group sorting

An application that uses a CollectionView object to group data can now explicitly declare how to sort the groups. Explicit sorting addresses the problem of non-intuitive ordering that occurs when an app dynamically adds or removes groups, or when it changes the value of item properties involved in grouping. It can also improve the performance of the group creation process by moving comparisons of the grouping properties from the sort of the full collection to the sort of the groups.

To support group sorting, the new GroupDescription.SortDescriptions and GroupDescription.CustomSort properties describe how to sort the collection of groups produced by the GroupDescription object. This is analogous to the way the identically named ListCollectionView properties describe how to sort the data items.

Two new static properties of the PropertyGroupDescription class, CompareNameAscending and CompareNameDescending, can be used for the most common cases.

For example, the following XAML groups data by age, sort the age groups in ascending order, and group the items within each age group by last name.

```
<GroupDescriptions>
     <PropertyGroupDescription
         PropertyName="Age"
         CustomSort=
              "{x:Static PropertyGroupDescription.CompareNamesAscending}"/>
     </PropertyGroupDescription>
</GroupDescriptions>

<SortDescriptions>
     <SortDescription PropertyName="LastName"/>
</SortDescriptions>
```

Soft keyboard support

Soft Keyboard support enables focus tracking in a WPF applications by automatically invoking and dismissing the new Soft Keyboard in Windows 10 when the touch input is received by a control that can take textual input.

In previous versions of the .NET Framework, WPF applications cannot opt into the focus tracking without disabling WPF pen/touch gesture support. As a result, WPF applications must choose between full WPF touch support or rely on Windows mouse promotion.

Per-monitor DPI

To support the recent proliferation of high-DPI and hybrid-DPI environments for WPF apps, WPF in the .NET Framework 4.6.2 enables per-monitor awareness. See the samples and developer guide on GitHub for more information about how to enable your WPF app to become per-monitor DPI aware.

In previous versions of the .NET Framework, WPF apps are system-DPI aware. In other words, the application's UI is scaled by the OS as appropriate, depending on the DPI of the monitor on which the app is rendered. ,

For apps running under the .NET Framework 4.6.2, you can disable per-monitor DPI changes in WPF apps by adding a configuration statement to the <runtime> section of your application configuration file, as follows:

```
<runtime>
   <AppContextSwitchOverrides value="Switch.System.Windows.DoNotScaleForDpiChanges=false"/>
</runtime>
```

Windows Workflow Foundation (WF)

In the .NET Framework 4.6.2, Windows Workflow Foundation has been enhanced in the following area:

Support for C# expressions and IntelliSense in the Re-hosted WF Designer

Starting with the .NET Framework 4.5, WF supports C# expressions in both the Visual Studio Designer and in code workflows. The Re-hosted Workflow Designer is a key feature of WF that allows for the Workflow Designer to be in an application outside Visual Studio (for example, in WPF). Windows Workflow Foundation provides the ability to support C# expressions and IntelliSense in the Re-hosted Workflow Designer. For more information, see the Windows Workflow Foundation blog.

`Availability of IntelliSense when a customer rebuilds a workflow project from Visual Studio` In versions of the .NET Framework prior to the .NET Framework 4.6.2, WF Designer IntelliSense is broken when a customer rebuilds a workflow project from Visual Studio. While the project build is successful, the workflow types are not found on the designer, and warnings from IntelliSense for the missing workflow types appear in the **Error List** window. The .NET Framework 4.6.2 addresses this issue and makes IntelliSense available.

Workflow V1 applications with Workflow Tracking on now run under FIPS-mode

Machines with FIPS Compliance Mode enabled can now successfully run a workflow Version 1-style application with Workflow tracking on. To enable this scenario, you must make the following change to your app.config file:

```
<add key="microsoft:WorkflowRuntime:FIPSRequired" value="true" />
```

If this scenario is not enabled, running the application continues to generate an exception with the message, "This implementation is not part of the Windows Platform FIPS validated cryptographic algorithms."

Workflow Improvements when using Dynamic Update with Visual Studio Workflow Designer

The Workflow Designer, FlowChart Activity Designer, and other Workflow Activity Designers now successfully load and display workflows that have been saved after calling the DynamicUpdateServices.PrepareForUpdate method. In versions of the .NET Framework before .NET Framework 4.6.2, loading a XAML file in Visual Studio for a workflow that has been saved after calling DynamicUpdateServices.PrepareForUpdate can result in the following issues:

- The Workflow Designer can't load the XAML file correctly (when the ViewStateData.Id is at the end of the line).
- Flowchart Activity Designer or other Workflow Activity Designers may display all objects in their default locations as opposed to attached property values.

ClickOnce

ClickOnce has been updated to support TLS 1.1 and TLS 1.2 in addition to the 1.0 protocol, which it already supports. ClickOnce automatically detects which protocol is required; no extra steps within the ClickOnce application are required to enable TLS 1.1 and 1.2 support.

Converting Windows Forms and WPF apps to UWP apps

Windows now offers capabilities to bring existing Windows desktop apps, including WPF and Windows Forms apps, to the Universal Windows Platform (UWP). This technology acts as a bridge by enabling you to gradually migrate your existing code base to UWP, thereby bringing your app to all Windows 10 devices.

Converted desktop apps gain an app identity similar to the app identity of UWP apps, which makes UWP APIs accessible to enable features such as Live Tiles and notifications. The app continues to behave as before and runs as a full trust app. Once the app is converted, an app container process can be added to the existing full trust process to add an adaptive user interface. When all functionality is moved to the app container process, the full trust process can be removed and the new UWP app can be made available to all Windows 10 devices.

Debugging improvements

The *unmanaged debugging API* has been enhanced in the .NET Framework 4.6.2 to perform additional analysis when a NullReferenceException is thrown so that it is possible to determine which variable in a single line of source code is `null`. To support this scenario, the following APIs have been added to the unmanaged debugging API.

- The ICorDebugCode4, ICorDebugVariableHome, and ICorDebugVariableHomeEnum interfaces, which expose the native homes of managed variables. This enables debuggers to do some code flow analysis when a NullReferenceException occurs and to work backwards to determine the managed variable that corresponds to the native location that was `null`.
- The ICorDebugType2::GetTypeID method provides a mapping for ICorDebugType to COR_TYPEID, which allows the debugger to obtain a COR_TYPEID without an instance of the ICorDebugType. Existing APIs on COR_TYPEID can then be used to determine the class layout of the type.

What's new in .NET Framework 4.6.1

The .NET Framework 4.6.1 includes new features in the following areas:

- Cryptography
- ADO.NET
- Windows Presentation Foundation (WPF)
- Windows Workflow Foundation
- Profiling
- NGen

For more information on the .NET Framework 4.6.1, see the following topics:

- .NET Framework 4.6.1 list of changes
- Application Compatibility in 4.6.1
- .NET Framework API diff (on GitHub)

Cryptography: Support for X509 certificates containing ECDSA

.NET Framework 4.6 added RSACng support for X509 certificates. The .NET Framework 4.6.1 adds support for ECDSA (Elliptic Curve Digital Signature Algorithm) X509 certificates.

ECDSA offers better performance and is a more secure cryptography algorithm than RSA, providing an excellent choice where Transport Layer Security (TLS) performance and scalability is a concern. The .NET Framework implementation wraps calls into existing Windows functionality.

The following example code shows how easy it is to generate a signature for a byte stream by using the new support for ECDSA X509 certificates included in the .NET Framework 4.6.1.

```
using System;
using System.Security.Cryptography;
using System.Security.Cryptography.X509Certificates;

public class Net461Code
{
    public static byte[] SignECDsaSha512(byte[] data, X509Certificate2 cert)
    {
        using (ECDsa privateKey = cert.GetECDsaPrivateKey())
        {
            return privateKey.SignData(data, HashAlgorithmName.SHA512);
        }
    }

    public static byte[] SignECDsaSha512(byte[] data, ECDsa privateKey)
    {
        return privateKey.SignData(data, HashAlgorithmName.SHA512);
    }
}
```

```
Imports System.Security.Cryptography
Imports System.Security.Cryptography.X509Certificates

Public Class Net461Code
    Public Shared Function SignECDsaSha512(data As Byte(), cert As X509Certificate2) As Byte()
        Using privateKey As ECDsa = cert.GetECDsaPrivateKey()
            Return privateKey.SignData(data, HashAlgorithmName.SHA512)
        End Using
    End Function

    Public Shared Function SignECDsaSha512(data As Byte, privateKey As ECDsa) As Byte()
        Return privateKey.SignData(data, HashAlgorithmName.SHA512)
    End Function
End Class
```

This offers a marked contrast to the code needed to generate a signature in .NET Framework 4.6.

```
using System;
using System.Security.Cryptography;
using System.Security.Cryptography.X509Certificates;

public class Net46Code
{
    public static byte[] SignECDsaSha512(byte[] data, X509Certificate2 cert)
    {
        // This would require using cert.Handle and a series of p/invokes to get at the
        // underlying key, then passing that to a CngKey object, and passing that to
        // new ECDsa(CngKey).  It's a lot of work.
        throw new Exception("That's a lot of work...");
    }

    public static byte[] SignECDsaSha512(byte[] data, ECDsa privateKey)
    {
        // This way works, but SignData probably better matches what you want.
        using (SHA512 hasher = SHA512.Create())
        {
            byte[] signature1 = privateKey.SignHash(hasher.ComputeHash(data));
        }

        // This might not be the ECDsa you got!
        ECDsaCng ecDsaCng = (ECDsaCng)privateKey;
        ecDsaCng.HashAlgorithm = CngAlgorithm.Sha512;
        return ecDsaCng.SignData(data);
    }
}
```

```
Imports System.Security.Cryptography
Imports System.Security.Cryptography.X509Certificates

Public Class Net46Code
    Public Shared Function SignECDsaSha512(data As Byte(), cert As X509Certificate2) As Byte()
        ' This would require using cert.Handle and a series of p/invokes to get at the
        ' underlying key, then passing that to a CngKey object, and passing that to
        ' new ECDsa(CngKey).  It's a lot of work.
        Throw New Exception("That's a lot of work...")
    End Function

    Public Shared Function SignECDsaSha512(data As Byte(), privateKey As ECDsa) As Byte()
        ' This way works, but SignData probably better matches what you want.
        Using hasher As SHA512 = SHA512.Create()
            Dim signature1 As Byte() = privateKey.SignHash(hasher.ComputeHash(data))
        End Using

        ' This might not be the ECDsa you got!
        Dim ecDsaCng As ECDsaCng = CType(privateKey, ECDsaCng)
        ecDsaCng.HashAlgorithm = CngAlgorithm.Sha512
        Return ecDsaCng.SignData(data)
    End Function
End Class
```

ADO.NET

The following have been added to ADO.NET:

Always Encrypted support for hardware protected keys

ADO.NET now supports storing Always Encrypted column master keys natively in Hardware Security Modules (HSMs). With this support, customers can leverage asymmetric keys stored in HSMs without having to write custom column master key store providers and registering them in applications.

Customers need to install the HSM vendor-provided CSP provider or CNG key store providers on the app servers or client computers in order to access Always Encrypted data protected with column master keys stored in a HSM.

Improved MultiSubnetFailover connection behavior for AlwaysOn

SqlClient now automatically provides faster connections to an AlwaysOn Availability Group (AG). It transparently detects whether your application is connecting to an AlwaysOn availability group (AG) on a different subnet and quickly discovers the current active server and provides a connection to the server. Prior to this release, an application had to set the connection string to include `"MultisubnetFailover=true"` to indicate that it was connecting to an AlwaysOn Availability Group. Without setting the connection keyword to `true`, an application might experience a timeout while connecting to an AlwaysOn Availability Group. With this release, an application does *not* need to set MultiSubnetFailover to `true` anymore. For more information about SqlClient support for Always On Availability Groups, see SqlClient Support for High Availability, Disaster Recovery.

Windows Presentation Foundation (WPF)

Windows Presentation Foundation includes a number of improvements and changes.

Improved performance

The delay in firing touch events has been fixed in the .NET Framework 4.6.1. In addition, typing in a RichTextBox control no longer ties up the render thread during fast input.

Spell checking improvements

The spell checker in WPF has been updated on Windows 8.1 and later versions to leverage operating system support for spell-checking additional languages. There is no change in functionality on Windows versions prior to Windows 8.1.

As in previous versions of the .NET Framework, the language for a TextBox control ora RichTextBox block is detected by looking for information in the following order:

- `xml:lang`, if it is present.
- Current input language.
- Current thread culture.

For additional information on language support in WPF, see the WPF blog post on .NET Framework 4.6.1 features.

Additional support for per-user custom dictionaries

In .NET Framework 4.6.1, WPF recognizes custom dictionaries that are registered globally. This capability is available in addition to the ability to register them per-control.

In previous versions of WPF, custom dictionaries did not recognize Excluded Words and AutoCorrect lists. They are supported on Windows 8.1 and Windows 10 through the use of files that can be placed under the `%AppData%\Microsoft\Spelling\<language tag>` directory. The following rules apply to these files:

- The files should have extensions of .dic (for added words), .exc (for excluded words), or .acl (for AutoCorrect).
- The files should be UTF-16 LE plaintext that starts with the Byte Order Mark (BOM).
- Each line should consist of a word (in the added and excluded word lists), or an autocorrect pair with the words separated by a vertical bar ("|") (in the AutoCorrect word list).
- These files are considered read-only and are not modified by the system.

> **NOTE**
> These new file-formats are not directly supported by the WPF spell checking APIs, and the custom dictionaries supplied to WPF in applications should continue to use .lex files.

Samples

There are a number of WPF samples on the Microsoft/WPF-Samples GitHub repository. Help us improve our samples by sending us a pull-request or opening a GitHub issue.

DirectX extensions

WPF includes a NuGet package that provides new implementations of D3DImage that make it easy for you to interoperate with DX10 and Dx11 content. The code for this package has been open sourced and is available on GitHub.

Windows Workflow Foundation: Transactions

The Transaction.EnlistPromotableSinglePhase method can now use a distributed transaction manager other than MSDTC to promote the transaction. You do this by specifying a GUID transaction promoter identifier to the new Transaction.EnlistPromotableSinglePhase(IPromotableSinglePhaseNotification, Guid) overload . If this operation is successful, there are limitations placed on the capabilities of the transaction. Once a non-MSDTC transaction promoter is enlisted, the following methods throw a TransactionPromotionException because these methods require promotion to MSDTC:

- Transaction.EnlistDurable
- TransactionInterop.GetDtcTransaction
- TransactionInterop.GetExportCookie
- TransactionInterop.GetTransmitterPropagationToken

Once a non-MSDTC transaction promoter is enlisted, it must be used for future durable enlistments by using protocols that it defines. The Guid of the transaction promoter can be obtained by using the PromoterType property. When the transaction promotes, the transaction promoter provides a Byte array that represents the promoted token. An application can obtain the promoted token for a non-MSDTC promoted transaction with the GetPromotedToken method.

Users of the new Transaction.EnlistPromotableSinglePhase(IPromotableSinglePhaseNotification, Guid) overload must follow a specific call sequence in order for the promotion operation to complete successfully. These rules are documented in the method's documentation.

Profiling

The unmanaged profiling API has been enhanced as follows:

- Better support for accessing PDBs in the ICorProfilerInfo7 interface.

  In ASP.NET Core, it is becoming much more common for assemblies to be compiled in-memory by Roslyn. For developers making profiling tools, this means that PDBs that historically were serialized on disk may no longer be present. Profiler tools often use PDBs to map code back to source lines for tasks such as code coverage or line-by-line performance analysis. The ICorProfilerInfo7 interface now includes two new methods, ICorProfilerInfo7::GetInMemorySymbolsLength and ICorProfilerInfo7::ReadInMemorySymbols, to provide these profiler tools with access to the in-memory PDB data, By using the new APIs, a profiler can obtain the contents of an in-memory PDB as a byte array and then process it or serialize it to disk.

- Better instrumentation with the ICorProfiler interface.

  Profilers that are using the `ICorProfiler` APIs ReJit functionality for dynamic instrumentation can now modify some metadata. Previously such tools could instrument IL at any time, but metadata could only be modified at module load time. Because IL refers to metadata, this limited the kinds of instrumentation that could be done. We have lifted some of those limits by adding the ICorProfilerInfo7::ApplyMetaData method to support a subset of metadata edits after the module loads, in particular by adding new `AssemblyRef`, `TypeRef`, `TypeSpec`, `MemberRef`, `MemberSpec`, and `UserString` records. This change makes a much broader range of on-the-fly instrumentation possible.

Native Image Generator (NGEN) PDBs

Cross-machine event tracing allows customers to profile a program on Machine A and look at the profiling data with source line mapping on Machine B. Using previous versions of the .NET Framework, the user would copy all the modules and native images from the profiled machine to the analysis machine that contains the IL PDB to create the source-to-native mapping. While this process may work well when the files are relatively small, such as for phone applications, the files can be very large on desktop systems and require significant time to copy.

With Ngen PDBs, NGen can create a PDB that contains the IL-to-native mapping without a dependency on the IL PDB. In our cross-machine event tracing scenario, all that is needed is to copy the native image PDB that is generated by Machine A to Machine B and to use Debug Interface Access APIs to read the IL PDB's source-to-IL mapping and the native image PDB's IL-to-native mapping. Combining both mappings provides a source-to-native mapping. Since the native image PDB is much smaller than all the modules and native images, the process of copying from Machine A to Machine B is much faster.

What's new in .NET 2015

.NET 2015 introduces the .NET Framework 4.6 and .NET Core. Some new features apply to both, and other features are specific to .NET Framework 4.6 or .NET Core.

- **ASP.NET Core**

  .NET 2015 includes ASP.NET Core, which is a lean .NET implementation for building modern cloud-based apps. ASP.NET Core is modular so you can include only those features that are needed in your application. It can be hosted on IIS or self-hosted in a custom process, and you can run apps with different versions of the .NET Framework on the same server. It includes a new environment configuration system that is designed for cloud deployment.

  MVC, Web API, and Web Pages are unified into a single framework called MVC 6. You build ASP.NET Core apps through tools in Visual Studio 2015 or later. Your existing applications will work on the new .NET Framework; however to build an app that uses MVC 6 or SignalR 3, you must use the project system in Visual Studio 2015 or later.

  For information, see ASP.NET Core.

- **ASP.NET Updates**

  - **Task-based API for Asynchronous Response Flushing**

    ASP.NET now provides a simple task-based API for asynchronous response flushing, HttpResponse.FlushAsync, that allows responses to be flushed asynchronously by using your language's `async/await` support.

  - **Model binding supports task-returning methods**

    In the .NET Framework 4.5, ASP.NET added the Model Binding feature that enabled an extensible, code-focused approach to CRUD-based data operations in Web Forms pages and user controls. The Model Binding system now supports Task-returning model binding methods. This feature allows Web Forms developers to get the scalability benefits of async with the ease of the data-binding system when using newer versions of ORMs, including the Entity Framework.

    Async model binding is controlled by the `aspnet:EnableAsyncModelBinding` configuration setting.

```
<appSettings>
    <add key=" aspnet:EnableAsyncModelBinding" value="true|false" />
</appSettings>
```

On apps the target the .NET Framework 4.6, it defaults to `true`. On apps running on the .NET Framework 4.6 that target an earlier version of the .NET Framework, it is `false` by default. It can be enabled by setting the configuration setting to `true`.

- **HTTP/2 Support (Windows 10)**

  HTTP/2 is a new version of the HTTP protocol that provides much better connection utilization (fewer round-trips between client and server), resulting in lower latency web page loading for users. Web pages (as opposed to services) benefit the most from HTTP/2, since the protocol optimizes for multiple artifacts being requested as part of a single experience. HTTP/2 support has been added to ASP.NET in .NET Framework 4.6. Because networking functionality exists at multiple layers, new features were required in Windows, in IIS, and in ASP.NET to enable HTTP/2. You must be running on Windows 10 to use HTTP/2 with ASP.NET.

  HTTP/2 is also supported and on by default for Windows 10 Universal Windows Platform (UWP) apps that use the System.Net.Http.HttpClient API.

  In order to provide a way to use the PUSH_PROMISE feature in ASP.NET applications, a new method with two overloads, PushPromise(String) and PushPromise(String, String, NameValueCollection), has been added to the HttpResponse class.

  > **NOTE**
  > While ASP.NET Core supports HTTP/2, support for the PUSH PROMISE feature has not yet been added.

  The browser and the web server (IIS on Windows) do all the work. You don't have to do any heavy-lifting for your users.

  Most of the major browsers support HTTP/2, so it's likely that your users will benefit from HTTP/2 support if your server supports it.

- **Support for the Token Binding Protocol**

  Microsoft and Google have been collaborating on a new approach to authentication, called the Token Binding Protocol. The premise is that authentication tokens (in your browser cache) can be stolen and used by criminals to access otherwise secure resources (e.g. your bank account) without requiring your password or any other privileged knowledge. The new protocol aims to mitigate this problem.

  The Token Binding Protocol will be implemented in Windows 10 as a browser feature. ASP.NET apps will participate in the protocol, so that authentication tokens are validated to be legitimate. The client and the server implementations establish the end-to-end protection specified by the protocol.

- **Randomized string hash algorithms**

  .NET Framework 4.5 introduced a randomized string hash algorithm. However, it was not supported by ASP.NET because of some ASP.NET features depended on a stable hash code. In .NET Framework 4.6, randomized string hash algorithms are now supported. To enable this feature, use the `aspnet:UseRandomizedStringHashAlgorithm` config setting.

```
<appSettings>
    <add key="aspnet:UseRandomizedStringHashAlgorithm" value="true|false" />
</appSettings>
```

- **ADO.NET**

  ADO .NET now supports the Always Encrypted feature available in SQL Server 2016 Community Technology Preview 2 (CTP2). With Always Encrypted, SQL Server can perform operations on encrypted data, and best of all the encryption key resides with the application inside the customer's trusted environment and not on the server. Always Encrypted secures customer data so DBAs do not have access to plain text data. Encryption and decryption of data happens transparently at the driver level, minimizing changes that have to be made to existing applications. For details, see Always Encrypted (Database Engine) and Always Encrypted (client development).

- **64-bit JIT Compiler for managed code**

  .NET Framework 4.6 features a new version of the 64-bit JIT compiler (originally code-named RyuJIT). The new 64-bit compiler provides significant performance improvements over the older 64-bit JIT compiler. The new 64-bit compiler is enabled for 64-bit processes running on top of .NET Framework 4.6. Your app will run in a 64-bit process if it is compiled as 64-bit or AnyCPU and is running on a 64-bit operating system. While care has been taken to make the transition to the new compiler as transparent as possible, changes in behavior are possible. We would like to hear directly about any issues encountered when using the new JIT compiler. Please contact us through Microsoft Connect if you encounter an issue that may be related to the new 64-bit JIT compiler.

  The new 64-bit JIT compiler also includes hardware SIMD acceleration features when coupled with SIMD-enabled types in the System.Numerics namespace, which can yield good performance improvements.

- **Assembly loader improvements**

  The assembly loader now uses memory more efficiently by unloading IL assemblies after a corresponding NGEN image is loaded. This change decreases virtual memory, which is particularly beneficial for large 32-bit apps (such as Visual Studio), and also saves physical memory.

- **Base class library changes**

Many new APIs have been added around to .NET Framework 4.6 to enable key scenarios. These include the following changes and additions:

- **IReadOnlyCollection<T> implementations**

  Additional collections implement IReadOnlyCollection<T> such as Queue<T> and Stack<T>.

- **CultureInfo.CurrentCulture and CultureInfo.CurrentUICulture**

  The CultureInfo.CurrentCulture and CultureInfo.CurrentUICulture properties are now read-write rather than read-only. If you assign a new CultureInfo object to these properties, the current thread culture defined by the `Thread.CurrentThread.CurrentCulture` property and the current UI thread culture defined by the `Thread.CurrentThread.CurrentUICulture` properties also change.

- **Enhancements to garbage collection (GC)**

  The GC class now includes TryStartNoGCRegion and EndNoGCRegion methods that allow you to disallow garbage collection during the execution of a critical path.

  A new overload of the GC.Collect(Int32, GCCollectionMode, Boolean, Boolean) method allows you to control whether both the small object heap and the large object heap are swept and compacted or swept only.

- **SIMD-enabled types**

  The System.Numerics namespace now includes a number of SIMD-enabled types, such as Matrix3x2, Matrix4x4, Plane, Quaternion, Vector2, Vector3, and Vector4.

  Because the new 64-bit JIT compiler also includes hardware SIMD acceleration features, there are especially significant performance improvements when using the SIMD-enabled types with the new 64-bit JIT compiler.

- **Cryptography updates**

  The System.Security.Cryptography API is being updated to support the Windows CNG cryptography APIs. Previous versions of the .NET Framework have relied entirely on an earlier version of the Windows Cryptography APIs as the basis for the System.Security.Cryptography implementation. We have had requests to support the CNG API, since it supports modern cryptography algorithms, which are important for certain categories of apps.

  .NET Framework 4.6 includes the following new enhancements to support the Windows CNG cryptography APIs:

  - A set of extension methods for X509 Certificates, `System.Security.Cryptography.X509Certificates.RSACertificateExtensions.GetRSAPublicKey(System.Security.Cryptography.X509Certificates.X509Certificate2)` and `System.Security.Cryptography.X509Certificates.RSACertificateExtensions.GetRSAPrivateKey(System.Security.Cryptography.X509Certificates.X509Certificate2)`, that return a CNG-based implementation rather than a CAPI-based implementation when possible. (Some smartcards, etc., still require CAPI, and the APIs handle the fallback).

  - The System.Security.Cryptography.RSACng class, which provides a CNG implementation of the RSA algorithm.

  - Enhancements to the RSA API so that common actions no longer require casting. For example, encrypting data using an X509Certificate2 object requires code like the following in previous versions of the .NET Framework.

    ```
    RSACryptoServiceProvider rsa = (RSACryptoServiceProvider)cert.PrivateKey;
    byte[] oaepEncrypted = rsa.Encrypt(data, true);
    byte[] pkcs1Encrypted = rsa.Encrypt(data, false);
    ```

    ```
    Dim rsa As RSACryptoServiceProvider = CType(cert.PrivateKey, RSACryptoServiceProvider)
    Dim oaepEncrypted() As Byte = rsa.Encrypt(data, True)
    Dim pkcs1Encrypted() As Byte = rsa.Encrypt(data, False)
    ```

    Code that uses the new cryptography APIs in .NET Framework 4.6 can be rewritten as follows to avoid the cast.

    ```
    RSA rsa = cert.GetRSAPrivateKey();
    if (rsa == null)
       throw new InvalidOperationException("An RSA certificate was expected");

    byte[] oaepEncrypted = rsa.Encrypt(data, RSAEncryptionPadding.OaepSHA1);
    byte[] pkcs1Encrypted = rsa.Encrypt(data, RSAEncryptionPadding.Pkcs1);
    ```

    ```
    Dim rsa As RSA = cert.GetRSAPrivateKey()
    If rsa Is Nothing Then
       Throw New InvalidOperationException("An RSA certificate was expected")
     End If

    Dim oaepEncrypted() As Byte = rsa.Encrypt(data, RSAEncryptionPadding.OaepSHA1)
    Dim pkcs1Encrypted() As Byte = rsa.Encrypt(data, RSAEncryptionPadding.Pkcs1)
    ```

- **Support for converting dates and times to or from Unix time**

The following new methods have been added to the DateTimeOffset structure to support converting date and time values to or from Unix time:

- DateTimeOffset.FromUnixTimeSeconds
- DateTimeOffset.FromUnixTimeMilliseconds
- DateTimeOffset.ToUnixTimeSeconds
- DateTimeOffset.ToUnixTimeMilliseconds

- **Compatibility switches**

  The new AppContext class adds a new compatibility feature that enables library writers to provide a uniform opt-out mechanism for new functionality for their users. It establishes a loosely-coupled contract between components in order to communicate an opt-out request. This capability is typically important when a change is made to existing functionality. Conversely, there is already an implicit opt-in for new functionality.

  With AppContext, libraries define and expose compatibility switches, while code that depends on them can set those switches to affect the library behavior. By default, libraries provide the new functionality, and they only alter it (that is, they provide the previous functionality) if the switch is set.

  An application (or a library) can declare the value of a switch (which is always a Boolean value) that a dependent library defines. The switch is always implicitly `false`. Setting the switch to `true` enables it. Explicitly setting the switch to `false` provides the new behavior.

```
AppContext.SetSwitch("Switch.AmazingLib.ThrowOnException", true);
```

```
AppContext.SetSwitch("Switch.AmazingLib.ThrowOnException", True)
```

  The library must check if a consumer has declared the value of the switch and then appropriately act on it.

```
if (!AppContext.TryGetSwitch("Switch.AmazingLib.ThrowOnException", out shouldThrow))
{
    // This is the case where the switch value was not set by the application.
    // The library can choose to get the value of shouldThrow by other means.
    // If no overrides nor default values are specified, the value should be 'false'.
    // A false value implies the latest behavior.
}

// The library can use the value of shouldThrow to throw exceptions or not.
if (shouldThrow)
{
    // old code
}
else
{
    // new code
}
```

```
If Not AppContext.TryGetSwitch("Switch.AmazingLib.ThrowOnException", shouldThrow) Then
    ' This is the case where the switch value was not set by the application.
    ' The library can choose to get the value of shouldThrow by other means.
    ' If no overrides nor default values are specified, the value should be 'false'.
    ' A false value implies the latest behavior.
End If

' The library can use the value of shouldThrow to throw exceptions or not.
If shouldThrow Then
    ' old code
Else
    ' new code
End If
```

  It's beneficial to use a consistent format for switches, since they are a formal contract exposed by a library. The following are two obvious formats.

  - *Switch.namespace.switchname*
  - *Switch.library.switchname*

- **Changes to the task-based asynchronous pattern (TAP)**

  For apps that target the .NET Framework 4.6, Task and Task<TResult> objects inherit the culture and UI culture of the calling thread. The behavior of apps that target previous versions of the .NET Framework, or that do not target a specific version of the .NET Framework, is unaffected. For more information, see the "Culture and task-based asynchronous operations" section of the CultureInfo class topic.

  The System.Threading.AsyncLocal<T> class allows you to represent ambient data that is local to a given asynchronous control flow, such as an `async` method. It can be used to persist data across threads. You can also define a callback method that is notified whenever the

ambient data changes either because the AsyncLocal<T>.Value property was explicitly changed, or because the thread encountered a context transition.

Three convenience methods, Task.CompletedTask, Task.FromCanceled, and Task.FromException, have been added to the task-based asynchronous pattern (TAP) to return completed tasks in a particular state.

The NamedPipeClientStream class now supports asynchronous communication with its new ConnectAsync. method.

- **EventSource now supports writing to the Event log**

  You now can use the EventSource class to log administrative or operational messages to the event log, in addition to any existing ETW sessions created on the machine. In the past, you had to use the Microsoft.Diagnostics.Tracing.EventSource NuGet package for this functionality. This functionality is now built-into .NET Framework 4.6.

  Both the NuGet package and .NET Framework 4.6 have been updated with the following features:

  - **Dynamic events**

    Allows events defined "on the fly" without creating event methods.

  - **Rich payloads**

    Allows specially attributed classes and arrays as well as primitive types to be passed as a payload

  - **Activity tracking**

    Causes Start and Stop events to tag events between them with an ID that represents all currently active activities.

  To support these features, the overloaded Write method has been added to the EventSource class.

- **Windows Presentation Foundation (WPF)**

  - **HDPI improvements**

    HDPI support in WPF is now better in the .NET Framework 4.6. Changes have been made to layout rounding to reduce instances of clipping in controls with borders. By default, this feature is enabled only if your TargetFrameworkAttribute is set to .NET 4.6. Applications that target earlier versions of the framework but are running on the .NET Framework 4.6 can opt in to the new behavior by adding the following line to the <runtime> section of the app.config file:

    ```
    <AppContextSwitchOverrides
    value="Switch.MS.Internal.DoNotApplyLayoutRoundingToMarginsAndBorderThickness=false"
    />
    ```

    WPF windows straddling multiple monitors with different DPI settings (Multi-DPI setup) are now completely rendered without blacked-out regions. You can opt out of this behavior by adding the following line to the `<appSettings>` section of the app.config file to disable this new behavior:

    ```
    <add key="EnableMultiMonitorDisplayClipping" value="true"/>
    ```

    Support for automatically loading the right cursor based on DPI setting has been added to System.Windows.Input.Cursor.

  - **Touch is better**

    Customer reports on Connect that touch produces unpredictable behavior have been addressed in the .NET Framework 4.6. The double tap threshold for Windows Store applications and WPF applications is now the same in Windows 8.1 and above.

  - **Transparent child window support**

    WPF in the .NET Framework 4.6 supports transparent child windows in Windows 8.1 and above. This allows you to create non-rectangular and transparent child windows in your top-level windows. You can enable this feature by setting the HwndSourceParameters.UsesPerPixelTransparency property to `true`.

- **Windows Communication Foundation (WCF)**

  - **SSL support**

    WCF now supports SSL version TLS 1.1 and TLS 1.2, in addition to SSL 3.0 and TLS 1.0, when using NetTcp with transport security and client authentication. It is now possible to select which protocol to use, or to disable old lesser secure protocols. This can be done either by setting the SslProtocols property or by adding the following to a configuration file.

```
<netTcpBinding>
    <binding>
      <security mode= "None|Transport|Message|TransportWithMessageCredential" >
          <transport clientCredentialType="None|Windows|Certificate"
                    protectionLevel="None|Sign|EncryptAndSign"
                    sslProtocols="Ssl3|Tls1|Tls11|Tls12">
            </transport>
      </security>
    </binding>
</netTcpBinding>
```

- **Sending messages using different HTTP connections**

  WCF now allows users to ensure certain messages are sent using different underlying HTTP connections. There are two ways to do this:

  - **Using a connection group name prefix**

    Users can specify a string that WCF will use as a prefix for the connection group name. Two messages with different prefixes are sent using different underlying HTTP connections. You set the prefix by adding a key/value pair to the message's Message.Properties property. The key is "HttpTransportConnectionGroupNamePrefix"; the value is the desired prefix.

  - **Using different channel factories**

    Users can also enable a feature that ensures that messages sent using channels created by different channel factories will use different underlying HTTP connections. To enable this feature, users must set the following `appSetting` to `true`:

    ```
    <appSettings>
        <add key="wcf:httpTransportBinding:useUniqueConnectionPoolPerFactory" value="true" />
    </appSettings>
    ```

- **Windows Workflow Foundation (WWF)**

  You can now specify the number of seconds a workflow service will hold on to an out-of-order operation request when there is an outstanding "non-protocol" bookmark before timing out the request. A "non-protocol" bookmark is a bookmark that is not related to outstanding Receive activities. Some activities create non-protocol bookmarks within their implementation, so it may not be obvious that a non-protocol bookmark exists. These include State and Pick. So if you have a workflow service implemented with a state machine or containing a Pick activity, you will most likely have non-protocol bookmarks. You specify the interval by adding a line like the following to the `appSettings` section of your app.config file:

  ```
  <add key="microsoft:WorkflowServices:FilterResumeTimeoutInSeconds" value="60"/>
  ```

  The default value is 60 seconds. If `value` is set to 0, out-of-order requests are immediately rejected with a fault with text that looks like this:

  ```
  Operation 'Request3|{http://tempuri.org/}IService' on service instance with identifier '2b0667b6-09c8-4093-9d02-f6c67d534292' cannot be
  performed at this time. Please ensure that the operations are performed in the correct order and that the binding in use provides ordered
  delivery guarantees.
  ```

  This is the same message that you receive if an out-of-order operation message is received and there are no non-protocol bookmarks.

  If the value of the `FilterResumeTimeoutInSeconds` element is non-zero, there are non-protocol bookmarks, and the timeout interval expires, the operation fails with a timeout message.

- **Transactions**

  You can now include the distributed transaction identifier for the transaction that has caused an exception derived from TransactionException to be thrown. You do this by adding the following key to the `appSettings` section of your app.config file:

  ```
  <add key="Transactions:IncludeDistributedTransactionIdInExceptionMessage" value="true"/>
  ```

  The default value is `false`.

- **Networking**

  - **Socket reuse**

    Windows 10 includes a new high-scalability networking algorithm that makes better use of machine resources by reusing local ports for outbound TCP connections. .NET Framework 4.6 supports the new algorithm, enabling .NET apps to take advantage of the new behavior. In previous versions of Windows, there was an artificial concurrent connection limit (typically 16,384, the default size of the dynamic port range), which could limit the scalability of a service by causing port exhaustion when under load.

    In the .NET Framework 4.6, two new APIs have been added to enable port reuse, which effectively removes the 64K limit on concurrent connections:

    - The System.Net.Sockets.SocketOptionName enumeration value.

- The ServicePointManager.ReusePort property.

  By default, the ServicePointManager.ReusePort property is `false` unless the `HWRPortReuseOnSocketBind` value of the `HKLM\SOFTWARE\Microsoft\.NETFramework\v4.0.30319` registry key is set to 0x1. To enable local port reuse on HTTP connections, set the ServicePointManager.ReusePort property to `true`. This causes all outgoing TCP socket connections from HttpClient and HttpWebRequest to use a new Windows 10 socket option, SO_REUSE_UNICASTPORT, that enables local port reuse.

  Developers writing a sockets-only application can specify the System.Net.Sockets.SocketOptionName option when calling a method such as Socket.SetSocketOption so that outbound sockets reuse local ports during binding.

- **Support for international domain names and PunyCode**

  A new property, IdnHost, has been added to the Uri class to better support international domain names and PunyCode.

- **Resizing in Windows Forms controls.**

  This feature has been expanded in .NET Framework 4.6 to include the DomainUpDown, NumericUpDown, DataGridViewComboBoxColumn, DataGridViewColumn and ToolStripSplitButton types and the rectangle specified by the Bounds property used when drawing a UITypeEditor.

  This is an opt-in feature. To enable it, set the `EnableWindowsFormsHighDpiAutoResizing` element to `true` in the application configuration (app.config) file:

```
<appSettings>
    <add key="EnableWindowsFormsHighDpiAutoResizing" value="true" />
</appSettings>
```

- **Support for code page encodings**

  .NET Core primarily supports the Unicode encodings and by default provides limited support for code page encodings. You can add support for code page encodings available in .NET Framework but unsupported in .NET Core by registering code page encodings with the Encoding.RegisterProvider method. For more information, see System.Text.CodePagesEncodingProvider.

- **.NET Native**

  Windows apps for Windows 10 that target .NET Core and are written in C# or Visual Basic can take advantage of a new technology that compiles apps to native code rather than IL. They produce apps characterized by faster startup and execution times. For more information, see Compiling Apps with .NET Native. For an overview of .NET Native that examines how it differs from both JIT compilation and NGEN and what that means for your code, see .NET Native and Compilation.

  Your apps are compiled to native code by default when you compile them with Visual Studio 2015 or later. For more information, see Getting Started with .NET Native.

  To support debugging .NET Native apps, a number of new interfaces and enumerations have been added to the unmanaged debugging API. For more information, see the Debugging (Unmanaged API Reference) topic.

- **Open-source .NET Framework packages**

  .NET Core packages such as the immutable collections, SIMD APIs, and networking APIs such as those found in the System.Net.Http namespace are now available as open source packages on GitHub. To access the code, see CoreFx on GitHub. For more information and how to contribute to these packages, see .NET Core and Open-Source, .NET Home Page on GitHub.

What's new in .NET Framework 4.5.2

- **New APIs for ASP.NET apps.** The new HttpResponse.AddOnSendingHeaders and HttpResponseBase.AddOnSendingHeaders methods let you inspect and modify response headers and status code as the response is being flushed to the client app. Consider using these methods instead of the PreSendRequestHeaders and PreSendRequestContent events; they are more efficient and reliable.

  The HostingEnvironment.QueueBackgroundWorkItem method lets you schedule small background work items. ASP.NET tracks these items and prevents IIS from abruptly terminating the worker process until all background work items have completed. This method can't be called outside an ASP.NET managed app domain.

  The new HttpResponse.HeadersWritten and HttpResponseBase.HeadersWritten properties return Boolean values that indicate whether the response headers have been written. You can use these properties to make sure that calls to APIs such as HttpResponse.StatusCode (which throw exceptions if the headers have been written) will succeed.

- **Resizing in Windows Forms controls.** This feature has been expanded. You can now use the system DPI setting to resize components of the following additional controls (for example, the drop-down arrow in combo boxes):

  - ComboBox
  - ToolStripComboBox
  - ToolStripMenuItem
  - Cursor
  - DataGridView
  - DataGridViewComboBoxColumn

  This is an opt-in feature. To enable it, set the `EnableWindowsFormsHighDpiAutoResizing` element to `true` in the application configuration

(app.config) file:

```
<appSettings>
    <add key="EnableWindowsFormsHighDpiAutoResizing" value="true" />
</appSettings>
```

- **New workflow feature.** A resource manager that's using the EnlistPromotableSinglePhase method (and therefore implementing the IPromotableSinglePhaseNotification interface) can use the new Transaction.PromoteAndEnlistDurable method to request the following:
  - Promote the transaction to a Microsoft Distributed Transaction Coordinator (MSDTC) transaction.
  - Replace IPromotableSinglePhaseNotification with an ISinglePhaseNotification, which is a durable enlistment that supports single phase commits.

  This can be done within the same app domain, and doesn't require any extra unmanaged code to interact with MSDTC to perform the promotion. The new method can be called only when there's an outstanding call from System.Transactions to the IPromotableSinglePhaseNotification `Promote` method that's implemented by the promotable enlistment.
- **Profiling improvements.** The following new unmanaged profiling APIs provide more robust profiling:
  - COR_PRF_ASSEMBLY_REFERENCE_INFO Structure
  - COR_PRF_HIGH_MONITOR Enumeration
  - GetAssemblyReferences Method
  - GetEventMask2 Method
  - SetEventMask2 Method
  - AddAssemblyReference Method

  Previous `ICorProfiler` implementations supported lazy loading of dependent assemblies. The new profiling APIs require dependent assemblies that are injected by the profiler to be loadable immediately, instead of being loaded after the app is fully initialized. This change doesn't affect users of the existing `ICorProfiler` APIs.
- **Debugging improvements.** The following new unmanaged debugging APIs provide better integration with a profiler. You can now access metadata inserted by the profiler as well as local variables and code produced by compiler ReJIT requests when dump debugging.
  - SetWriteableMetadataUpdateMode Method
  - EnumerateLocalVariablesEx Method
  - GetLocalVariableEx Method
  - GetCodeEx Method
  - GetActiveReJitRequestILCode Method
  - GetInstrumentedILMap Method
- **Event tracing changes.** .NET Framework 4.5.2 enables out-of-process, Event Tracing for Windows (ETW)-based activity tracing for a larger surface area. This enables Advanced Power Management (APM) vendors to provide lightweight tools that accurately track the costs of individual requests and activities that cross threads. These events are raised only when ETW controllers enable them; therefore, the changes don't affect previously written ETW code or code that runs with ETW disabled.
- **Promoting a transaction and converting it to a durable enlistment**

  Transaction.PromoteAndEnlistDurable is a new API added to .NET Framework 4.5.2 and 4.6:

```
[System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.LinkDemand, Name = "FullTrust")]
public Enlistment PromoteAndEnlistDurable(Guid resourceManagerIdentifier,
                                          IPromotableSinglePhaseNotification promotableNotification,
                                          ISinglePhaseNotification enlistmentNotification,
                                          EnlistmentOptions enlistmentOptions)
```

```
<System.Security.Permissions.PermissionSetAttribute(System.Security.Permissions.SecurityAction.LinkDemand, Name:="FullTrust")>
public Function PromoteAndEnlistDurable(resourceManagerIdentifier As Guid,
                                        promotableNotification As IPromotableSinglePhaseNotification,
                                        enlistmentNotification As ISinglePhaseNotification,
                                        enlistmentOptions As EnlistmentOptions) As Enlistment
```

  The method may be used by an enlistment that was previously created by Transaction.EnlistPromotableSinglePhase in response to the ITransactionPromoter.Promote method. It asks `System.Transactions` to promote the transaction to an MSDTC transaction and to "convert" the promotable enlistment to a durable enlistment. After this method completes successfully, the IPromotableSinglePhaseNotification interface will no longer be referenced by `System.Transactions`, and any future notifications will arrive on the provided ISinglePhaseNotification interface. The enlistment in question must act as a durable enlistment, supporting transaction logging and recovery. Refer to Transaction.EnlistDurable for details. In addition, the enlistment must support ISinglePhaseNotification. This method can *only* be called while processing an ITransactionPromoter.Promote call. If that is not the case, a TransactionException exception is thrown.

What's new in .NET Framework 4.5.1

April 2014 updates:

- Visual Studio 2013 Update 2 includes updates to the Portable Class Library templates to support these scenarios:
  - You can use Windows Runtime APIs in portable libraries that target Windows 8.1, Windows Phone 8.1, and Windows Phone Silverlight 8.1.
  - You can include XAML (Windows.UI.XAML types) in portable libraries when you target Windows 8.1 or Windows Phone 8.1. The following XAML templates are supported: Blank Page, Resource Dictionary, Templated Control, and User Control.
  - You can create a portable Windows Runtime component (.winmd file) for use in Store apps that target Windows 8.1 and Windows Phone 8.1.
  - You can retarget a Windows Store or Windows Phone Store class library like a Portable Class Library.

  For more information about these changes, see Portable Class Library.
- The .NET Framework content set now includes documentation for .NET Native, which is a precompilation technology for building and deploying Windows apps. .NET Native compiles your apps directly to native code, rather than to intermediate language (IL), for better performance. For details, see Compiling Apps with .NET Native.
- The .NET Framework Reference Source provides a new browsing experience and enhanced functionality. You can now browse through the .NET Framework source code online, download the reference for offline viewing, and step through the sources (including patches and updates) during debugging. For more information, see the blog entry A new look for .NET Reference Source.

New features and enhancements in the base classes in .NET Framework 4.5.1 include:

- Automatic binding redirection for assemblies. Starting with Visual Studio 2013, when you compile an app that targets the .NET Framework 4.5.1, binding redirects may be added to the app configuration file if your app or its components reference multiple versions of the same assembly. You can also enable this feature for projects that target older versions of the .NET Framework. For more information, see How to: Enable and Disable Automatic Binding Redirection.
- Ability to collect diagnostics information to help developers improve the performance of server and cloud applications. For more information, see the WriteEventWithRelatedActivityId and WriteEventWithRelatedActivityIdCore methods in the EventSource class.
- Ability to explicitly compact the large object heap (LOH) during garbage collection. For more information, see the GCSettings.LargeObjectHeapCompactionMode property.
- Additional performance improvements such as ASP.NET app suspension, multi-core JIT improvements, and faster app startup after a .NET Framework update. For details, see the .NET Framework 4.5.1 announcement and the ASP.NET app suspend blog post.

Improvements to Windows Forms include:

- Resizing in Windows Forms controls. You can use the system DPI setting to resize components of controls (for example, the icons that appear in a property grid) by opting in with an entry in the application configuration file (app.config) for your app. This feature is currently supported in the following Windows Forms controls:
  - PropertyGrid
  - TreeView
  - Some aspects of the DataGridView (see new features in 4.5.2 for additional controls supported)

  To enable this feature, add a new <appSettings> element to the configuration file (app.config) and set the `EnableWindowsFormsHighDpiAutoResizing` element to `true`:

```
<appSettings>
    <add key="EnableWindowsFormsHighDpiAutoResizing" value="true" />
</appSettings>
```

Improvements when debugging your .NET Framework apps in Visual Studio 2013 include:

- Return values in the Visual Studio debugger. When you debug a managed app in Visual Studio 2013, the Autos window displays return types and values for methods. This information is available for desktop, Windows Store, and Windows Phone apps. For more information, see Examine return values of method calls.
- Edit and Continue for 64-bit apps. Visual Studio 2013 supports the Edit and Continue feature for 64-bit managed apps for desktop, Windows Store, and Windows Phone. The existing limitations remain in effect for both 32-bit and 64-bit apps (see the last section of the Supported Code Changes (C#) article).
- Async-aware debugging. To make it easier to debug asynchronous apps in Visual Studio 2013, the call stack hides the infrastructure code provided by compilers to support asynchronous programming, and also chains in logical parent frames so you can follow logical program execution more clearly. A Tasks window replaces the Parallel Tasks window and displays tasks that relate to a particular breakpoint, and also displays any other tasks that are currently active or scheduled in the app. You can read about this feature in the "Async-aware debugging" section of the .NET Framework 4.5.1 announcement.
- Better exception support for Windows Runtime components. In Windows 8.1, exceptions that arise from Windows Store apps preserve information about the error that caused the exception, even across language boundaries. You can read about this feature in the "Windows Store app development" section of the .NET Framework 4.5.1 announcement.

Starting with Visual Studio 2013, you can use the Managed Profile Guided Optimization Tool (Mpgo.exe) to optimize Windows 8.x Store apps as well as

desktop apps.

For new features in ASP.NET 4.5.1, see ASP.NET and Web Tools for Visual Studio 2013 Release Notes.

What's new in .NET Framework 4.5

Base classes

- Ability to reduce system restarts by detecting and closing .NET Framework 4 applications during deployment. See Reducing System Restarts During .NET Framework 4.5 Installations.
- Support for arrays that are larger than 2 gigabytes (GB) on 64-bit platforms. This feature can be enabled in the application configuration file. See the <gcAllowVeryLargeObjects> element, which also lists other restrictions on object size and array size.
- Better performance through background garbage collection for servers. When you use server garbage collection in the .NET Framework 4.5, background garbage collection is automatically enabled. See the Background Server Garbage Collection section of the Fundamentals of Garbage Collection topic.
- Background just-in-time (JIT) compilation, which is optionally available on multi-core processors to improve application performance. See ProfileOptimization.
- Ability to limit how long the regular expression engine will attempt to resolve a regular expression before it times out. See the Regex.MatchTimeout property.
- Ability to define the default culture for an application domain. See the CultureInfo class.
- Console support for Unicode (UTF-16) encoding. See the Console class.
- Support for versioning of cultural string ordering and comparison data. See the SortVersion class.
- Better performance when retrieving resources. See Packaging and Deploying Resources.
- Zip compression improvements to reduce the size of a compressed file. See the System.IO.Compression namespace.
- Ability to customize a reflection context to override default reflection behavior through the CustomReflectionContext class.
- Support for the 2008 version of the Internationalized Domain Names in Applications (IDNA) standard when the System.Globalization.IdnMapping class is used on Windows 8.
- Delegation of string comparison to the operating system, which implements Unicode 6.0, when the .NET Framework is used on Windows 8. When running on other platforms, the .NET Framework includes its own string comparison data, which implements Unicode 5.x. See the String class and the Remarks section of the SortVersion class.
- Ability to compute the hash codes for strings on a per application domain basis. See <UseRandomizedStringHashAlgorithm> Element.
- Type reflection support split between Type and TypeInfo classes. See Reflection in the .NET Framework for Windows Store Apps.

Managed Extensibility Framework (MEF)

In the .NET Framework 4.5, the Managed Extensibility Framework (MEF) provides the following new features:

- Support for generic types.
- Convention-based programming model that enables you to create parts based on naming conventions rather than attributes.
- Multiple scopes.
- A subset of MEF that you can use when you create Windows 8.x Store apps. This subset is available as a downloadable package from the NuGet Gallery. To install the package, open your project in Visual Studio, choose **Manage NuGet Packages** from the **Project** menu, and search online for the `Microsoft.Composition` package.

For more information, see Managed Extensibility Framework (MEF).

Asynchronous file operations

In the .NET Framework 4.5, new asynchronous features were added to the C# and Visual Basic languages. These features add a task-based model for performing asynchronous operations. To use this new model, use the asynchronous methods in the I/O classes. See Asynchronous File I/O.

Tools

In the .NET Framework 4.5, Resource File Generator (Resgen.exe) enables you to create a .resw file for use in Windows 8.x Store apps from a .resources file embedded in a .NET Framework assembly. For more information, see Resgen.exe (Resource File Generator).

Managed Profile Guided Optimization (Mpgo.exe) enables you to improve application startup time, memory utilization (working set size), and throughput by optimizing native image assemblies. The command-line tool generates profile data for native image application assemblies. See Mpgo.exe (Managed Profile Guided Optimization Tool). Starting with Visual Studio 2013, you can use Mpgo.exe to optimize Windows 8.x Store apps as well as desktop apps.

Parallel computing

The .NET Framework 4.5 provides several new features and improvements for parallel computing. These include improved performance, increased control, improved support for asynchronous programming, a new dataflow library, and improved support for parallel debugging and performance analysis. See the entry What's New for Parallelism in .NET 4.5 in the Parallel Programming with .NET blog.

Web

ASP.NET 4.5 and 4.5.1 add model binding for Web Forms, WebSocket support, asynchronous handlers, performance enhancements, and many other features. For more information, see the following resources:

- ASP.NET 4.5 and Visual Studio 2012
- ASP.NET and Web Tools for Visual Studio 2013 Release Notes

Networking

The .NET Framework 4.5 provides a new programming interface for HTTP applications. For more information, see the new System.Net.Http and System.Net.Http.Headers namespaces.

Support is also included for a new programming interface for accepting and interacting with a WebSocket connection by using the existing HttpListener and related classes. For more information, see the new System.Net.WebSockets namespace and the HttpListener class.

In addition, the .NET Framework 4.5 includes the following networking improvements:

- RFC-compliant URI support. For more information, see Uri and related classes.
- Support for Internationalized Domain Name (IDN) parsing. For more information, see Uri and related classes.
- Support for Email Address Internationalization (EAI). For more information, see the System.Net.Mail namespace.
- Improved IPv6 support. For more information, see the System.Net.NetworkInformation namespace.
- Dual-mode socket support. For more information, see the Socket and TcpListener classes.

Windows Presentation Foundation (WPF)

In the .NET Framework 4.5, Windows Presentation Foundation (WPF) contains changes and improvements in the following areas:

- The new Ribbon control, which enables you to implement a ribbon user interface that hosts a Quick Access Toolbar, Application Menu, and tabs.
- The new INotifyDataErrorInfo interface, which supports synchronous and asynchronous data validation.
- New features for the VirtualizingPanel and Dispatcher classes.
- Improved performance when displaying large sets of grouped data, and by accessing collections on non-UI threads.
- Data binding to static properties, data binding to custom types that implement the ICustomTypeProvider interface, and retrieval of data binding information from a binding expression.
- Repositioning of data as the values change (live shaping).
- Ability to check whether the data context for an item container is disconnected.
- Ability to set the amount of time that should elapse between property changes and data source updates.
- Improved support for implementing weak event patterns. Also, events can now accept markup extensions.

Windows Communication Foundation (WCF)

In the .NET Framework 4.5, the following features have been added to make it simpler to write and maintain Windows Communication Foundation (WCF) applications:

- Simplification of generated configuration files.
- Support for contract-first development.
- Ability to configure ASP.NET compatibility mode more easily.
- Changes in default transport property values to reduce the likelihood that you will have to set them.
- Updates to the XmlDictionaryReaderQuotas class to reduce the likelihood that you will have to manually configure quotas for XML dictionary readers.
- Validation of WCF configuration files by Visual Studio as part of the build process, so you can detect configuration errors before you run your application.
- New asynchronous streaming support.
- New HTTPS protocol mapping to make it easier to expose an endpoint over HTTPS with Internet Information Services (IIS).
- Ability to generate metadata in a single WSDL document by appending `?singleWSDL` to the service URL.
- Websockets support to enable true bidirectional communication over ports 80 and 443 with performance characteristics similar to the TCP transport.
- Support for configuring services in code.
- XML Editor tooltips.
- ChannelFactory caching support.
- Binary encoder compression support.

- Support for a UDP transport that enables developers to write services that use "fire and forget" messaging. A client sends a message to a service and expects no response from the service.
- Ability to support multiple authentication modes on a single WCF endpoint when using the HTTP transport and transport security.
- Support for WCF services that use internationalized domain names (IDNs).

For more information, see What's New in Windows Communication Foundation.

Windows Workflow Foundation (WF)

In the .NET Framework 4.5, several new features were added to Windows Workflow Foundation (WF), including:

- State machine workflows, which were first introduced as part of .NET Framework 4.0.1 (.NET Framework 4 Platform Update 1). This update included several new classes and activities that enabled developers to create state machine workflows. These classes and activities were updated for the .NET Framework 4.5 to include:
  - The ability to set breakpoints on states.
  - The ability to copy and paste transitions in the workflow designer.
  - Designer support for shared trigger transition creation.
  - Activities for creating state machine workflows, including: StateMachine, State, and Transition.
- Enhanced Workflow Designer features such as the following:
  - Enhanced workflow search capabilities in Visual Studio, including **Quick Find** and **Find in Files**.
  - Ability to automatically create a Sequence activity when a second child activity is added to a container activity, and to include both activities in the Sequence activity.
  - Panning support, which enables the visible portion of a workflow to be changed without using the scroll bars.
  - A new **Document Outline** view that shows the components of a workflow in a tree-style outline view and lets you select a component in the **Document Outline** view.
  - Ability to add annotations to activities.
  - Ability to define and consume activity delegates by using the workflow designer.
  - Auto-connect and auto-insert for activities and transitions in state machine and flowchart workflows.
- Storage of the view state information for a workflow in a single element in the XAML file, so you can easily locate and edit the view state information.
- A NoPersistScope container activity to prevent child activities from persisting.
- Support for C# expressions:
  - Workflow projects that use Visual Basic will use Visual Basic expressions, and C# workflow projects will use C# expressions.
  - C# workflow projects that were created in Visual Studio 2010 and that have Visual Basic expressions are compatible with C# workflow projects that use C# expressions.
- Versioning enhancements:
  - The new WorkflowIdentity class, which provides a mapping between a persisted workflow instance and its workflow definition.
  - Side-by-side execution of multiple workflow versions in the same host, including WorkflowServiceHost.
  - In Dynamic Update, the ability to modify the definition of a persisted workflow instance.
- Contract-first workflow service development, which provides support for automatically generating activities to match an existing service contract.

For more information, see What's New in Windows Workflow Foundation.

.NET for Windows 8.x Store apps

Windows 8.x Store apps are designed for specific form factors and leverage the power of the Windows operating system. A subset of the .NET Framework 4.5 or 4.5.1 is available for building Windows 8.x Store apps for Windows by using C# or Visual Basic. This subset is called .NET for Windows 8.x Store apps and is discussed in an overview in the Windows Dev Center.

Portable Class Libraries

The Portable Class Library project in Visual Studio 2012 (and later versions) enables you to write and build managed assemblies that work on multiple .NET Framework platforms. Using a Portable Class Library project, you choose the platforms (such as Windows Phone and .NET for Windows 8.x Store apps) to target. The available types and members in your project are automatically restricted to the common types and members across these platforms. For more information, see Portable Class Library.

See also

- The .NET Framework and Out-of-Band Releases
- What's new in accessibility in the .NET Framework

- What's New in Visual Studio 2017
- ASP.NET
- What's New in Visual C++

Get started with the .NET Framework

6/5/2019 • 6 minutes to read • Edit Online

The .NET Framework is a runtime execution environment that manages apps that target the .NET Framework. It consists of the common language runtime, which provides memory management and other system services, and an extensive class library, which enables programmers to take advantage of robust, reliable code for all major areas of app development.

> **NOTE**
>
> The .NET Framework is available on Windows systems only. You can use .NET Core to run apps on Windows, MacOS, and Linux.

What is the .NET Framework?

The .NET Framework is a managed execution environment for Windows that provides a variety of services to its running apps. It consists of two major components: the common language runtime (CLR), which is the execution engine that handles running apps, and the .NET Framework Class Library, which provides a library of tested, reusable code that developers can call from their own apps. The services that the .NET Framework provides to running apps include the following:

- Memory management. In many programming languages, programmers are responsible for allocating and releasing memory and for handling object lifetimes. In .NET Framework apps, the CLR provides these services on behalf of the app.
- A common type system. In traditional programming languages, basic types are defined by the compiler, which complicates cross-language interoperability. In the .NET Framework, basic types are defined by the .NET Framework type system and are common to all languages that target the .NET Framework.
- An extensive class library. Instead of having to write vast amounts of code to handle common low-level programming operations, programmers use a readily accessible library of types and their members from the .NET Framework Class Library.
- Development frameworks and technologies. The .NET Framework includes libraries for specific areas of app development, such as ASP.NET for web apps, ADO.NET for data access, Windows Communication Foundation for service-oriented apps, and Windows Presentation Foundation for Windows desktop apps.
- Language interoperability. Language compilers that target the .NET Framework emit an intermediate code named Common Intermediate Language (CIL), which, in turn, is compiled at runtime by the common language runtime. With this feature, routines written in one language are accessible to other languages, and programmers focus on creating apps in their preferred languages.
- Version compatibility. With rare exceptions, apps that are developed by using a particular version of the .NET Framework run without modification on a later version.
- Side-by-side execution. The .NET Framework helps resolve version conflicts by allowing multiple versions of the common language runtime to exist on the same computer. This means that multiple versions of apps can coexist and that an app can run on the version of the .NET Framework with which it was built. Side-by-side execution applies to the .NET Framework version groups 1.0/1.1, 2.0/3.0/3.5, and 4/4.5.x/4.6.x/4.7.x/4.8.
- Multitargeting. By targeting .NET Standard, developers create class libraries that work on multiple .NET Framework platforms supported by that version of the standard. For example, libraries that target the .NET

Standard 2.0 can be used by apps that target the .NET Framework 4.6.1, .NET Core 2.0, and UWP 10.0.16299.

The .NET Framework for users

If you don't develop .NET Framework apps, but you use them, you aren't required to have specific knowledge about the .NET Framework or its operation. For the most part, the .NET Framework is completely transparent to users.

If you're using the Windows operating system, the .NET Framework may already be installed on your computer. In addition, if you install an app that requires the .NET Framework, the app's setup program might install a specific version of the .NET Framework on your computer. In some cases, you may see a dialog box that asks you to install the .NET Framework. If you've just tried to run an app when this dialog box appears and if your computer has Internet access, you can go to a webpage that lets you install the missing version of the .NET Framework. For more information, see the Installation guide.

In general, you shouldn't uninstall versions of the .NET Framework that are installed on your computer. There are two reasons for this:

- If an app that you use depends on a specific version of the .NET Framework, that app may break if that version is removed.
- Some versions of the .NET Framework are in-place updates to earlier versions. For example, the .NET Framework 3.5 is an in-place update to version 2.0, and the .NET Framework 4.8 is an in-place update to versions 4 through 4.7.2. For more information, see .NET Framework Versions and Dependencies.

On Windows versions before Windows 8, if you do choose to remove the .NET Framework, always use **Programs and Features** from Control Panel to uninstall it. Never remove a version of the .NET Framework manually. On Windows 8 and above, the .NET Framework is an operating system component and cannot be independently uninstalled.

Note that multiple versions of the .NET Framework can coexist on a single computer at the same time. This means that you don't have to uninstall previous versions in order to install a later version.

The .NET Framework for developers

If you're a developer, choose any programming language that supports the .NET Framework to create your apps. Because the .NET Framework provides language independence and interoperability, you interact with other .NET Framework apps and components regardless of the language with which they were developed.

To develop .NET Framework apps or components, do the following:

1. If it's not preinstalled on your operating system, install the version of the .NET Framework that your app will target. The most recent production version is the .NET Framework 4.8. It is preinstalled on Windows 10 May 2019 Update, and it is available for download on earlier versions of the Windows operating system. For .NET Framework system requirements, see System Requirements. For information on installing other versions of the .NET Framework, see Installation Guide. Additional .NET Framework packages are released out of band, which means that they're released on a rolling basis outside of any regular or scheduled release cycle. For information about these packages, see The .NET Framework and Out-of-Band Releases.
2. Select the language or languages supported by the .NET Framework that you intend to use to develop your apps. A number of languages are available, including Visual Basic, C#, F#, and C++/CLI from Microsoft. (A programming language that allows you to develop apps for the .NET Framework adheres to the Common Language Infrastructure (CLI) specification.)
3. Select and install the development environment to use to create your apps and that supports your selected programming language or languages. The Microsoft integrated development environment (IDE) for .NET

Framework apps is Visual Studio. It's available in a number of editions.

For more information on developing apps that target the .NET Framework, see the Development Guide.

Related articles

TITLE	DESCRIPTION
Overview	Provides detailed information for developers who build apps that target the .NET Framework.
Installation guide	Provides information about installing the .NET Framework.
The .NET Framework and Out-of-Band Releases	Describes the .NET Framework out of band releases and how to use them in your app.
System Requirements	Lists the hardware and software requirements for running the .NET Framework.
.NET Core and Open-Source	Describes .NET Core in relation to the .NET Framework and how to access the open-source .NET Core projects.
.NET Core documentation	Provides the conceptual and API reference documentation for .NET Core.
.NET Standard	Discusses .NET Standard, a versioned specification that individual .NET implementations support to guarantee that a consistent set of APIs are available on multiple platforms.

See also

- .NET Framework Guide
- What's New
- .NET API Browser
- Development Guide

Installation guide

9/10/2019 • 2 minutes to read • Edit Online

You can install .NET Framework on various Windows versions.

Supported Windows versions

- Windows 10 and Windows Server 2016
- Windows 8.1 and Windows Server 2012 R2
- Windows 8 and Windows Server 2012
- Windows 7 and Windows Server 2008 R2
- Windows Vista and Windows Server 2008

Unsupported Windows versions

- Windows XP and Windows Server 2003

See also

- Download the .NET Framework
- Troubleshoot blocked .NET Framework installations and uninstallations
- Install the .NET Framework for developers
- Deploy the .NET Framework for developers

Migration Guide to the .NET Framework 4.8, 4.7, 4.6, and 4.5

5/15/2019 • 2 minutes to read • Edit Online

If you created your app using an earlier version of the .NET Framework, you can generally upgrade it to .NET Framework 4.5 and its point releases (4.5.1 and 4.5.2), .NET Framework 4.6 and its point releases (4.6.1 and 4.6.2), .NET Framework 4.7 and its point releases (4.7.1 and 4.7.2), or .NET Framework 4.8 easily. Open your project in Visual Studio. If your project was created in an earlier version of Visual Studio, the **Project Compatibility** dialog box automatically opens. For more information about upgrading a project in Visual Studio, see Port, Migrate, and Upgrade Visual Studio Projects and Visual Studio 2019 Platform Targeting and Compatibility.

However, some changes in the .NET Framework require changes to your code. You may also want to take advantage of functionality that is new in .NET Framework 4.5 and its point releases, in .NET Framework 4.6 and its point releases, in .NET Framework 4.7 and its point releases, or in .NET Framework 4.8. Making these types of changes to your app for a new version of the .NET Framework is typically referred to as *migration*. If your app doesn't have to be migrated, you can run it in the .NET Framework 4.5 or a later version without recompiling it.

Migration resources

Review the following documents before you migrate your app from earlier versions of the .NET Framework to version 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, or 4.8:

- See Versions and Dependencies to understand the CLR version underlying each version of the .NET Framework and to review guidelines for targeting your apps successfully.
- Review Application Compatibility to find out about runtime and retargeting changes that might affect your app and how to handle them.
- Review What's Obsolete in the Class Library to determine any types or members in your code that have been made obsolete, and the recommended alternatives.
- See What's New for descriptions of new features that you may want to add to your app.

See also

- Application Compatibility
- Migrating from the .NET Framework 1.1
- Version Compatibility
- Versions and Dependencies
- How to: Configure an app to support .NET Framework 4 or later versions
- What's New
- What's Obsolete in the Class Library
- .NET Framework Version and Assembly Information
- Microsoft .NET Framework Support Lifecycle Policy
- .NET Framework 4 migration issues

.NET Framework Development Guide

9/7/2019 • 2 minutes to read • Edit Online

This section explains how to create, configure, debug, secure, and deploy your .NET Framework apps. The section also provides information about technology areas such as dynamic programming, interoperability, extensibility, memory management, and threading.

In This Section

Application Essentials

Provides information about basic app development tasks, such as programming with app domains and assemblies, using attributes, formatting and parsing base types, using collections, handling events and exceptions, using files and data streams, and using generics.

Data and Modeling

Provides information about how to access data using ADO.NET, Language Integrated Query (LINQ), WCF Data Services, and XML.

Client Applications

Explains how to create Windows-based apps by using Windows Presentation Foundation (WPF) or Windows Forms.

Web Applications with ASP.NET

Provides links to information about using ASP.NET to build enterprise-class web apps with a minimum of coding.

Service-Oriented Applications with WCF

Describes how to use Windows Communication Foundation (WCF) to build service-oriented apps that are secure and reliable.

Building workflows with Windows Workflow Foundation

Provides information about the programming model, samples, and tools for using Windows Workflow Foundation (WF).

Windows Service Applications

Explains how you can use Visual Studio and the .NET Framework to create an app that is installed as a service, and start, stop, and otherwise control its behavior.

Parallel Processing, Concurrency, and Async Programming in .NET

Provides information about managed threading, parallel programming, and asynchronous programming design patterns.

Network Programming in the .NET Framework

Describes the layered, extensible, and managed implementation of Internet services that you can quickly and easily integrate into your apps.

Configuring .NET Framework Apps

Explains how you can use configuration files to change settings without having to recompile your .NET Framework apps.

Compiling Apps with .NET Native

Explains how you can use the .NET Native precompilation technology to build and deploy Windows Store apps. .NET Native compiles apps that are written in managed code (C#) and that target the .NET Framework to native code.

Security
Provides information about the classes and services in the .NET Framework that facilitate secure app development.

Debugging, Tracing, and Profiling
Explains how to test, optimize, and profile .NET Framework apps and the app environment. This section includes information for administrators as well as developers.

Developing for Multiple Platforms
Provides information about how you can use the .NET Framework to build assemblies that can be shared across multiple platforms and multiple devices such as phones, desktop, and web.

Deployment
Explains how to package and distribute your .NET Framework app, and includes deployment guides for both developers and administrators.

Performance
Provides information about caching, lazy initialization, reliability, and ETW events.

Reference

.NET Framework Class Library
Supplies syntax, code examples, and usage information for each class that is contained in the .NET Framework namespaces.

Related Sections

Getting Started
Provides a comprehensive overview of the .NET Framework and links to additional resources.

What's New
Describes key new features and changes in the latest version of the .NET Framework. Includes lists of new and obsolete types and members, and provides a guide for migrating your apps from the previous version of the .NET Framework.

Tools
Describes the tools that help you develop, configure, and deploy apps by using .NET Framework technologies.

.NET samples and tutorials
Provides links to samples and tutorials that help you learn about .NET.

Programming with Application Domains and Assemblies

9/17/2019 • 2 minutes to read • Edit Online

Hosts such as Microsoft Internet Explorer, ASP.NET, and the Windows shell load the common language runtime into a process, create an application domain in that process, and then load and execute user code in that application domain when running a .NET Framework application. In most cases, you do not have to worry about creating application domains and loading assemblies into them because the runtime host performs those tasks.

However, if you are creating an application that will host the common language runtime, creating tools or code you want to unload programmatically, or creating pluggable components that can be unloaded and reloaded on the fly, you will be creating your own application domains. Even if you are not creating a runtime host, this section provides important information on how to work with application domains and assemblies loaded in these application domains.

In This Section

Application Domains and Assemblies How-to Topics
Provides links to all How-to topics found in the conceptual documentation for programming with application domains and assemblies.

Using Application Domains
Provides examples of creating, configuring, and using application domains.

Programming with Assemblies
Describes how to create, sign, and set attributes on assemblies.

Related Sections

Emitting Dynamic Methods and Assemblies
Describes how to create dynamic assemblies.

Assemblies in .NET
Provides a conceptual overview of assemblies.

Application Domains
Provides a conceptual overview of application domains.

Reflection Overview
Describes how to use the **Reflection** class to obtain information about an assembly.

Resources in .NET Apps

9/17/2019 • 3 minutes to read • Edit Online

Nearly every production-quality app has to use resources. A resource is any nonexecutable data that is logically deployed with an app. A resource might be displayed in an app as error messages or as part of the user interface. Resources can contain data in a number of forms, including strings, images, and persisted objects. (To write persisted objects to a resource file, the objects must be serializable.) Storing your data in a resource file enables you to change the data without recompiling your entire app. It also enables you to store data in a single location, and eliminates the need to rely on hard-coded data that is stored in multiple locations.

The .NET Framework and .NET Core provide comprehensive support for the creation and localization of resources. In addition, .NET supports a simple model for packaging and deploying localized resources.

For information about resources in ASP.NET, see ASP.NET Web Page Resources Overview.

Creating and Localizing Resources

In a non-localized app, you can use resource files as a repository for app data, particularly for strings that might otherwise be hard-coded in multiple locations in source code. Most commonly, you create resources as either text (.txt) or XML (.resx) files, and use Resgen.exe (Resource File Generator) to compile them into binary .resources files. These files can then be embedded in the app's executable file by a language compiler. For more information about creating resources, see Creating Resource Files.

You can also localize your app's resources for specific cultures. This enables you to build localized (translated) versions of your apps. When you develop an app that uses localized resources, you designate a culture that serves as the neutral or fallback culture whose resources are used if no suitable resources are available. Typically, the resources of the neutral culture are stored in the app's executable. The remaining resources for individual localized cultures are stored in standalone satellite assemblies. For more information, see Creating Satellite Assemblies.

Packaging and Deploying Resources

You deploy localized app resources in satellite assemblies. A satellite assembly contains the resources of a single culture; it does not contain any app code. In the satellite assembly deployment model, you create an app with one default assembly (which is typically the main assembly) and one satellite assembly for each culture that the app supports. Because the satellite assemblies are not part of the main assembly, you can easily replace or update resources corresponding to a specific culture without replacing the app's main assembly.

Carefully determine which resources will make up your app's default resource assembly. Because it is a part of the main assembly, any changes to it will require you to replace the main assembly. If you do not provide a default resource, an exception will be thrown when the resource fallback process attempts to find it. In a well-designed app, using resources should never throw an exception.

For more information, see the Packaging and Deploying Resources article.

Retrieving Resources

At run time, an app loads the appropriate localized resources on a per-thread basis, based on the culture specified by the CultureInfo.CurrentUICulture property. This property value is derived as follows:

- By directly assigning a CultureInfo object that represents the localized culture to the Thread.CurrentUICulture property.

- If a culture is not explicitly assigned, by retrieving the default thread UI culture from the CultureInfo.DefaultThreadCurrentUICulture property.
- If a default thread UI culture is not explicitly assigned, by retrieving the culture for the current user on the local computer. .NET implementations running on Windows do this by calling the Windows `GetUserDefaultUILanguage` function.

For more information about how the current UI culture is set, see the CultureInfo and CultureInfo.CurrentUICulture reference pages.

You can then retrieve resources for the current UI culture or for a specific culture by using the System.Resources.ResourceManager class. Although the ResourceManager class is most commonly used for retrieving resources, the System.Resources namespace contains additional types that you can use to retrieve resources. These include:

- The ResourceReader class, which enables you to enumerate resources embedded in an assembly or stored in a standalone binary .resources file. It is useful when you don't know the precise names of the resources that are available at run time.
- The ResXResourceReader class, which enables you to retrieve resources from an XML (.resx) file.
- The ResourceSet class, which enables you to retrieve the resources of a specific culture without observing fallback rules. The resources can be stored in an assembly or a standalone binary .resources file. You can also develop an IResourceReader implementation that enables you to use the ResourceSet class to retrieve resources from some other source.
- The ResXResourceSet class, which enables you to retrieve all the items in an XML resource file into memory.

See also

- CultureInfo
- CultureInfo.CurrentUICulture
- Application Essentials
- Creating Resource Files
- Packaging and Deploying Resources
- Creating Satellite Assemblies
- Retrieving Resources

Accessibility

9/17/2019 • 2 minutes to read • Edit Online

> **NOTE**
> This documentation is intended for .NET Framework developers who want to use the managed UI Automation classes defined in the System.Windows.Automation namespace. For the latest information about UI Automation, see Windows Automation API: UI Automation.

Microsoft UI Automation is the new accessibility framework for Microsoft Windows. It addresses the needs of assistive technology products and automated test frameworks by providing programmatic access to information about the user interface (UI). In addition, UI Automation enables control and application developers to make their products accessible.

This documentation describes the UI Automation API for managed code. For information on programming for UI Automation in C++, see UI Automation for Win32 Applications.

In This Section

Accessibility Best Practices
UI Automation Fundamentals
UI Automation Providers for Managed Code
UI Automation Clients for Managed Code
UI Automation Control Patterns
UI Automation Text Pattern
UI Automation Control Types
UI Automation Specification and Community Promise

Related Sections

- Accessibility Samples

Data and modeling in .NET

9/11/2019 • 2 minutes to read • Edit Online

This section provides information on how to access data in the .NET Framework.

In this section

WCF Data Services 4.5

Provides information about how to use WCF Data Services to deploy data services on the Web or an intranet.

ADO.NET

Describes the ADO.NET architecture and how to use the ADO.NET classes to manage application data and interact with data sources, including Microsoft SQL Server, OLE DB data sources, and XML.

Transaction Processing

Discusses the .NET support for transactions.

Related sections

Language Integrated Query (LINQ)

Provides links to relevant documentation for Language Integrated Query (LINQ) using C#.

Language-Integrated Query (LINQ) (Visual Basic)

Provides links to relevant documentation for Language Integrated Query (LINQ) using Visual Basic.

XML Documents and Data

Provides an overview to a comprehensive and integrated set of classes that work with XML documents and data in the .NET Framework.

XML Standards Reference

Provides reference information on XML standards that Microsoft supports.

Developing client applications with the .NET Framework

9/17/2019 • 2 minutes to read • Edit Online

There are several ways to develop Windows-based applications with the .NET Framework. You can use any of these tools and frameworks:

- Universal Windows Platform (UWP)
- Windows Presentation Foundation (WPF)
- Windows Forms

This section contains topics that describe how to create Windows-based applications by using Windows Presentation Foundation or by using Windows Forms. However, you can also create web applications using the .NET Framework, and client applications for computers or devices that you make available through the Microsoft Store.

In this section

Windows Presentation Foundation
Provides information about developing applications by using WPF.

Windows Forms
Provides information about developing applications by using Windows Forms.

Common Client Technologies
Provides information about additional technologies that can be used when developing client applications.

Related sections

Universal Windows Platform
Describes how to create applications for Windows 10 that you can make available to users through the Windows Store.

.NET for UWP apps
Describes the .NET Framework support for Store apps, which can be deployed to Windows computers and devices.

.NET API for Windows Phone Silverlight
Lists the .NET Framework APIs you can use for building apps with Windows Phone Silverlight.

Developing for Multiple Platforms
Describes the different methods you can use the .NET Framework to target multiple client app types.

Get Started with ASP.NET Web Sites
Describes the ways you can develop web apps using ASP.NET.

See also

- .NET Standard
- Overview
- Development Guide

- Windows Service Applications

Windows Presentation Foundation

5/4/2018 • 2 minutes to read • Edit Online

Windows Presentation Foundation (WPF) in Visual Studio provides developers with a unified programming model for building line-of-business desktop applications on Windows.

Create Desktop Applications with Windows Presentation Foundation

Designing XAML in Visual Studio and Blend for Visual Studio

Get Visual Studio

Windows Forms

5/14/2019 • 2 minutes to read • Edit Online

As forms are the base unit of your application, it is essential that you give some thought to their function and design. A form is ultimately a blank slate that you, as a developer, enhance with controls to create a user interface and with code to manipulate data. To that end, Visual Studio provides you with an integrated development environment (IDE) to aid in writing code, as well as a rich control set written with the .NET Framework. By complementing the functionality of these controls with your code, you can easily and quickly develop the solutions you need.

In This Section

Getting Started with Windows Forms
Provides links to topics about how to harness the power of Windows Forms to display data, handle user input, and deploy your applications easily and with more robust security.

Enhancing Windows Forms Applications
Provides links to topics about how to enhance your Windows Forms with a variety of features.

Related Sections

Windows Forms Controls
Contains links to topics that describe Windows Forms controls and show how to implement them.

Windows Forms Data Binding
Contains links to topics that describe the Windows Forms data-binding architecture.

Graphics Overview
Discusses how to create graphics, draw text, and manipulate graphical images as objects using the advanced implementation of the Windows graphics design interface.

ClickOnce Security and Deployment
Discusses the principles of ClickOnce deployment.

Windows Forms/MFC Programming Differences
Discusses the differences between MFC applications and Windows Forms.

Accessing data in Visual Studio
Discusses incorporating data access functionality into your applications.

Windows Forms Applications
Discusses the process of debugging applications created with the Windows Application project template, as well as how to change the Debug and Release configurations.

First look at deployment in Visual Studio
Describes the process by which you distribute a finished application or component to be installed on other computers.

Building Console Applications
Describes the basics of creating a console application using the Console class.

Developing Service-Oriented Applications with WCF

10/15/2019 • 2 minutes to read • Edit Online

This section of the documentation provides information about Windows Communication Foundation (WCF), which is a unified programming model for building service-oriented applications. It enables developers to build secure, reliable, transacted solutions that integrate across platforms and interoperate with existing investments.

In this section

What's New in Windows Communication Foundation 4.5
Discusses features new to Windows Communication Foundation.

WCF Simplification Features
Discusses new features that make writing WCF applications simpler.

Guide to the Documentation
A description of the WCF documentation

Conceptual Overview
Summarizes information about the Windows Communication Foundation (WCF) messaging system and the classes that support its use.

Getting Started Tutorial
A step by step tutorial to create a WCF service and client

Basic WCF Programming
Describes the fundamentals for creating Windows Communication Foundation applications.

WCF Feature Details
Shows topics that let you choose which WCF feature or features you need to employ.

Extending WCF
Describes how to modify and extend WCF runtime components

Guidelines and Best Practices
Provides guidelines for creating Windows Communication Foundation (WCF) applications.

Administration and Diagnostics
Describes the diagnostic features of WCF

System Requirements
Describes system requirements needed to run WCF

Operating System Resources Required by WCF
Describes operating system resources required by WCF

Troubleshooting Setup Issues
Provides guidance for fixing WCF setup issues

Migrating from .NET Remoting to WCF
Compares .NET Remoting to WCF and provides migration guidance for common scenarios.

Using the WCF Development Tools
Describes the Visual Studio Windows Communication Foundation development tools that can assist you in developing your WCFservice.

Windows Communication Foundation Tools

Describes WCF tools designed to make it easier to create, deploy, and manage WCF applications

Windows Communication Foundation Samples

Samples that provide instruction on various aspects of Windows Communication Foundation

Windows Communication Foundation Glossary

Shows a list of terms specific to WCF

General Reference

The section describes the elements that are used to configure Windows Communication Foundation clients and services.

Feedback and Community

Information about how to provide feedback about Windows Communication Foundation

Privacy Information

Information regarding WCF and Privacy

Windows Workflow Foundation

3/9/2019 • 2 minutes to read • Edit Online

This section describes the programming model, samples, and tools of the Windows Workflow Foundation (WF).

In This Section

Guide to the Windows Workflow Documentation

A set of suggested topics to read, depending upon your familiarity (novice to well-acquainted), and requirements.

What's New in Windows Workflow Foundation

Discusses the changes in several development paradigms from previous versions.

What's New in Windows Workflow Foundation in .NET 4.5

Describes the new features in Windows Workflow Foundation in .NET Framework 4.6.1.

Windows Workflow Foundation Feature Specifics

Describes the new features in Windows Workflow Foundation in .NET Framework 4

Windows Workflow Conceptual Overview

A set of topics that discusses the larger concepts behind Windows Workflow Foundation.

Getting Started Tutorial

A set of walkthrough topics that introduce you to programming Windows Workflow Foundation applications.

Windows Workflow Foundation Programming

A set of primer topics that you should understand to become a proficient WF programmer.

Extending Windows Workflow Foundation

A set of topics that discusses how to extend or customize Windows Workflow Foundation to suit your needs.

Windows Workflow Foundation Glossary for .NET Framework 4.5

Defines a list of terms that are specific to WF.

Windows Workflow Samples

Contains sample applications that demonstrate WF features and scenarios.

Develop Windows service apps

9/17/2019 • 2 minutes to read • Edit Online

Using Visual Studio or the .NET Framework SDK, you can easily create services by creating an application that is installed as a service. This type of application is called a Windows service. With framework features, you can create services, install them, and start, stop, and otherwise control their behavior.

> **NOTE**
> In Visual Studio you can create a service in managed code in Visual C# or Visual Basic, which can interoperate with existing C++ code if required. Or, you can create a Windows service in native C++ by using the ATL Project Wizard.

In this section

Introduction to Windows Service Applications

Provides an overview of Windows service applications, the lifetime of a service, and how service applications differ from other common project types.

Walkthrough: Creating a Windows Service Application in the Component Designer

Provides an example of creating a service in Visual Basic and Visual C#.

Service Application Programming Architecture

Explains the language elements used in service programming.

How to: Create Windows Services

Describes the process of creating and configuring Windows services using the Windows service project template.

Related sections

ServiceBase - Describes the major features of the ServiceBase class, which is used to create services.

ServiceProcessInstaller - Describes the features of the ServiceProcessInstaller class, which is used along with the ServiceInstaller class to install and uninstall your services.

ServiceInstaller - Describes the features of the ServiceInstaller class, which is used along with the ServiceProcessInstaller class to install and uninstall your service.

Create Projects from Templates - Describes the projects types used in this chapter and how to choose between them.

64-bit Applications

9/17/2019 • 3 minutes to read • Edit Online

When you compile an application, you can specify that it should run on a Windows 64-bit operating system either as a native application or under WOW64 (Windows 32-bit on Windows 64-bit). WOW64 is a compatibility environment that enables a 32-bit application to run on a 64-bit system. WOW64 is included in all 64-bit versions of the Windows operating system.

Running 32-bit vs. 64-bit Applications on Windows

All applications that are built on the .NET Framework 1.0 or 1.1 are treated as 32-bit applications on a 64-bit operating system and are always executed under WOW64 and the 32-bit common language runtime (CLR). 32-bit applications that are built on the .NET Framework 4 or later versions also run under WOW64 on 64-bit systems.

Visual Studio installs the 32-bit version of the CLR on an x86 computer, and both the 32-bit version and the appropriate 64-bit version of the CLR on a 64-bit Windows computer. (Because Visual Studio is a 32-bit application, when it is installed on a 64-bit system, it runs under WOW64.)

> **NOTE**
> Because of the design of x86 emulation and the WOW64 subsystem for the Itanium processor family, applications are restricted to execution on one processor. These factors reduce the performance and scalability of 32-bit .NET Framework applications that run on Itanium-based systems. We recommend that you use the .NET Framework 4, which includes native 64-bit support for Itanium-based systems, for increased performance and scalability.

By default, when you run a 64-bit managed application on a 64-bit Windows operating system, you can create an object of no more than 2 gigabytes (GB). However, in the .NET Framework 4.5, you can increase this limit. For more information, see the <gcAllowVeryLargeObjects> element.

Many assemblies run identically on both the 32-bit CLR and the 64-bit CLR. However, some programs may behave differently, depending on the CLR, if they contain one or more of the following:

- Structures that contain members that change size depending on the platform (for example, any pointer type).
- Pointer arithmetic that includes constant sizes.
- Incorrect platform invoke or COM declarations that use `Int32` for handles instead of `IntPtr`.
- Code that casts `IntPtr` to `Int32`.

For more information about how to port a 32-bit application to run on the 64-bit CLR, see Migrating 32-bit Managed Code to 64-bit.

General 64-Bit Programming Information

For general information about 64-bit programming, see the following documents:

- For more information about the 64-bit version of the CLR on a 64-bit Windows computer, see the .NET Framework Developer Center on the MSDN website.
- In the Windows SDK documentation, see Programming Guide for 64-bit Windows.

- For information about how to download a 64-bit version of the CLR, see .NET Framework Developer Center Downloads on the MSDN website.
- For information about Visual Studio support for creating 64-bit applications, see Visual Studio IDE 64-Bit Support.

Compiler Support for Creating 64-Bit Applications

By default, when you use the .NET Framework to build an application on either a 32-bit or a 64-bit computer, the application will run on a 64-bit computer as a native application (that is, not under WOW64). The following table lists documents that explain how to use Visual Studio compilers to create 64-bit applications that will run as native, under WOW64, or both.

COMPILER	COMPILER OPTION
Visual Basic	/platform (Visual Basic)
Visual C#	/platform (C# Compiler Options)
Visual C++	You can create platform-agnostic, Microsoft intermediate language (MSIL) applications by using **/clr:safe**. For more information, see /clr (Common Language Runtime Compilation). Visual C++ includes a separate compiler for each 64-bit operating system. For more information about how to use Visual C++ to create native applications that run on a 64-bit Windows operating system, see 64-bit Programming.

Determining the Status of an .exe File or .dll File

To determine whether an .exe file or .dll file is meant to run only on a specific platform or under WOW64, use CorFlags.exe (CorFlags Conversion Tool) with no options. You can also use CorFlags.exe to change the platform status of an .exe file or .dll file. The CLR header of a Visual Studio assembly has the major runtime version number set to 2 and the minor runtime version number set to 5. Applications that have the minor runtime version set to 0 are treated as legacy applications and are always executed under WOW64.

To programmatically query an .exe or .dll to see whether it is meant to run only on a specific platform or under WOW64, use the Module.GetPEKind method.

Developing Web apps with ASP.NET

9/17/2019 • 2 minutes to read • Edit Online

ASP.NET is a .NET Framework technology for creating web apps. For more information on ASP.NET, see:

- ASP.NET documentation
- ASP.NET MVC
- ASP.NET Web Pages
- ASP.NET Web API
- Create an ASP.NET Framework web app in Azure

Developing Web apps with ASP.NET Core

ASP.NET Core is a redesign of ASP.NET 4.x. Some of the benefits ASP.NET Core provides over ASP.NET:

- Cross platform.
- Leaner and more modular.
- A unified story for building web UI and web APIs.

See Why use ASP.NET Core? for an expanded list of benefits.

For more information on ASP.NET Core](/aspnet/core), see:

- Get started with Razor Pages
- Create a Web API
- Create an ASP.NET Core web app in Azure

See also

- Development Guide

Network Programming in the .NET Framework

9/17/2019 • 4 minutes to read • Edit Online

The Microsoft .NET Framework provides a layered, extensible, and managed implementation of Internet services that can be quickly and easily integrated into your applications. Your network applications can build on pluggable protocols to automatically take advantage of new Internet protocols, or they can use a managed implementation of the Windows socket interface to work with the network on the socket level.

In This Section

International Resource Identifier Support in System.Uri
Describes enhancements to the System.Uri class in Version 3.5, 3.0 SP1, and 2.0 SP1 for International Resource Identifier (IRI) and Internationalized Domain Name (IDN) support.

Socket Performance Enhancements in Version 3.5
Describes a set of enhancements to the System.Net.Sockets.Socket class in Version 3.5, 3.0 SP1, and 2.0 SP1 that provide an alternative asynchronous pattern that can be used by specialized high-performance socket applications.

Peer Name Resolution Protocol
Describes support added in Version 3.5 to support the Peer Name Resolution Protocol (PNRP), a serverless and dynamic name registration and name resolution protocol. These new features are supported by the System.Net.PeerToPeer namespace.

Peer-to-Peer Collaboration
Describes support added in Version 3.5 to support the Peer-to-Peer Collaboration that builds on PNRP. These new features are supported by the System.Net.PeerToPeer.Collaboration namespace.

Changes to NTLM authentication for HttpWebRequest in Version 3.5 SP1
Describes security changes made in Version 3.5 SP1 that affect how integrated Windows authentication is handled by the System.Net.HttpWebRequest, System.Net.HttpListener, System.Net.Security.NegotiateStream, and related classes in the System.Net namespace.

Integrated Windows Authentication with Extended Protection
Describes enhancements for extended protection that affect how integrated Windows authentication is handled by the System.Net.HttpWebRequest, System.Net.HttpListener, System.Net.Mail.SmtpClient, System.Net.Security.SslStream, System.Net.Security.NegotiateStream, and related classes in the System.Net and related namespaces.

NAT Traversal using IPv6 and Teredo
Describes enhancements added to the System.Net, System.Net.NetworkInformation, and System.Net.Sockets namespaces to support NAT traversal using IPv6 and Teredo.

Network Isolation for Windows Store Apps
Describes the impact of network isolation when classes in the System.Net, System.Net.Http, and System.Net.Http.Headers namespaces are used in Windows 8.x Store apps.

Network Programming Samples
Links to downloadable network programming samples that use classes in the System.Net, System.Net.Cache, System.Net.Configuration, System.Net.Mail, System.Net.Mime, System.Net.NetworkInformation, System.Net.PeerToPeer, System.Net.Security, System.Net.Sockets namespaces.

Reference

System.Net
Provides a simple programming interface for many of the protocols used on networks today. The System.Net.WebRequest and System.Net.WebResponse classes in this namespace are the basis for pluggable protocols.

System.Net.Cache
Defines the types and enumerations used to define cache policies for resources obtained using the System.Net.WebRequest and System.Net.HttpWebRequest classes.

System.Net.Configuration
Classes that applications use to programmatically access and update configuration settings for the System.Net namespaces.

System.Net.Http

Classes that provides a programming interface for modern HTTP applications.

System.Net.Http.Headers
Provides support for collections of HTTP headers used by the System.Net.Http namespace

System.Net.Mail
Classes to compose and send mail using the SMTP protocol.

System.Net.Mime
Defines types that are used to represent Multipurpose Internet Mail Exchange (MIME) headers used by classes in the System.Net.Mail namespace.

System.Net.NetworkInformation
Classes to programmatically gather information about network events, changes, statistics, and properties.

System.Net.PeerToPeer
Provides a managed implementation of the Peer Name Resolution Protocol (PNRP) for developers.

System.Net.PeerToPeer.Collaboration
Provides a managed implementation of the Peer-to-Peer Collaboration interface for developers.

System.Net.Security
Classes to provide network streams for secure communications between hosts.

System.Net.Sockets
Provides a managed implementation of the Windows Sockets (Winsock) interface for developers who need to help control access to the network.

System.Net.WebSockets
Provides a managed implementation of the WebSocket interface for developers.

System.Uri
Provides an object representation of a uniform resource identifier (URI) and easy access to the parts of the URI.

System.Security.Authentication.ExtendedProtection
Provides support for authentication using extended protection for applications.

System.Security.Authentication.ExtendedProtection.Configuration
Provides support for configuration of authentication using extended protection for applications.

See also

- Transport Layer Security (TLS) best practices with .NET Framework
- Network Programming How-to Topics
- Network Programming Samples
- HttpClient Sample

Configuring Apps by using Configuration Files

9/13/2019 • 4 minutes to read • Edit Online

The .NET Framework, through configuration files, gives developers and administrators control and flexibility over the way applications run. Configuration files are XML files that can be changed as needed. An administrator can control which protected resources an application can access, which versions of assemblies an application will use, and where remote applications and objects are located. Developers can put settings in configuration files, eliminating the need to recompile an application every time a setting changes. This section describes what can be configured and why configuring an application might be useful.

> **NOTE**
>
> Managed code can use the classes in the System.Configuration namespace to read settings from the configuration files, but not to write settings to those files.

This topic describes the syntax of configuration files and provides information about the three types of configuration files: machine, application, and security.

Configuration File Format

Configuration files contain elements, which are logical data structures that set configuration information. Within a configuration file, you use tags to mark the beginning and end of an element. For example, the `<runtime>` element consists of `<runtime>` *child elements* `</runtime>`. An empty element would be written as `<runtime/>` or `<runtime></runtime>`.

As with all XML files, the syntax in configuration files is case-sensitive.

You specify configuration settings using predefined attributes, which are name/value pairs inside an element's start tag. The following example specifies two attributes (`version` and `href`) for the `<codeBase>` element, which specifies where the runtime can locate an assembly (for more information, see Specifying an Assembly's Location).

```
<codeBase version="2.0.0.0"
          href="http://www.litwareinc.com/myAssembly.dll"/>
```

Machine Configuration Files

The machine configuration file, Machine.config, contains settings that apply to an entire computer. This file is located in the %*runtime install path*%\Config directory. Machine.config contains configuration settings for machine-wide assembly binding, built-in remoting channels, and ASP.NET.

The configuration system first looks in the machine configuration file for the **<appSettings>** element and other configuration sections that a developer might define. It then looks in the application configuration file. To keep the machine configuration file manageable, it is best to put these settings in the application configuration file. However, putting the settings in the machine configuration file can make your system more maintainable. For example, if you have a third-party component that both your client and server application uses, it is easier to put the settings for that component in one place. In this case, the machine configuration file is the appropriate place for the settings, so you don't have the same settings in two different files.

> **NOTE**
> Deploying an application using XCOPY will not copy the settings in the machine configuration file.

For more information about how the common language runtime uses the machine configuration file for assembly binding, see How the Runtime Locates Assemblies.

Application Configuration Files

An application configuration file contains settings that are specific to an app. This file includes configuration settings that the common language runtime reads (such as assembly binding policy, remoting objects, and so on), and settings that the app can read.

The name and location of the application configuration file depend on the app's host, which can be one of the following:

- Executable–hosted app.

  These apps have two configuration files: a source configuration file, which is modified by the developer during development, and an output file that is distributed with the app.

  When you develop in Visual Studio, place the source configuration file for your app in the project directory and set its **Copy To Output Directory** property to **Copy always** or **Copy if newer**. The name of the configuration file is the name of the app with a .config extension. For example, an app called myApp.exe should have a source configuration file called myApp.exe.config.

  Visual Studio automatically copies the source configuration file to the directory where the compiled assembly is placed to create the output configuration file, which is deployed with the app. In some cases, Visual Studio may modify the output configuration file; for more information, see the Redirecting assembly versions at the app level section of the Redirecting Assembly Versions article.

- ASP.NET-hosted app.

  For more information about ASP.NET configuration files, see ASP.NET Configuration Settings.

- Internet Explorer-hosted app.

  If an app hosted in Internet Explorer has a configuration file, the location of this file is specified in a `<link>` tag with the following syntax:

  <link rel="*ConfigurationFileName*" href="*location*">

  In this tag, `location` is a URL to the configuration file. This sets the app base. The configuration file must be located on the same website as the app.

Security Configuration Files

Security configuration files contain information about the code group hierarchy and permission sets associated with a policy level. We strongly recommend that you use the Code Access Security Policy tool (Caspol.exe) to modify security policy to ensure that policy changes do not corrupt the security configuration files.

> **NOTE**
> Starting with the .NET Framework 4, the security configuration files are present only if security policy has been changed.

The security configuration files are in the following locations:

- Enterprise policy configuration file: %*runtime-install-path*%\Config\Enterprisesec.config
- Machine policy configuration file: %*runtime-install-path*%\Config\Security.config
- User policy configuration file: %USERPROFILE%\Application data\Microsoft\CLR security config\v*xx.xx*\Security.config

In This Section

How to: Locate Assemblies by Using DEVPATH
Describes how to direct the runtime to use the DEVPATH environment variable when searching for assemblies.

Redirecting Assembly Versions
Describes how to specify the location of an assembly and which version of an assembly to use.

Specifying an Assembly's Location
Describes how to specify where the runtime should search for an assembly.

Configuring Cryptography Classes
Describes how to map an algorithm name to a cryptography class and an object identifier to a cryptography algorithm.

How to: Create a Publisher Policy
Describes when and how you should add a publisher policy file to specify assembly redirection and code base settings.

Configuration File Schema
Describes the schema hierarchy for startup, runtime, network, and other types of configuration settings.

See also

- Configuration File Schema
- Specifying an Assembly's Location
- Redirecting Assembly Versions
- ASP.NET Web Site Administration
- Security Policy Management
- Caspol.exe (Code Access Security Policy Tool)
- Assemblies in .NET

Compiling Apps with .NET Native

9/17/2019 • 2 minutes to read • Edit Online

.NET Native is a precompilation technology for building and deploying Windows apps that is included with Visual Studio 2015 and later versions. It automatically compiles the release version of apps that are written in managed code (C# or Visual Basic) and that target the .NET Framework and Windows 10 to native code.

Typically, apps that target the .NET Framework are compiled to intermediate language (IL). At run time, the just-in-time (JIT) compiler translates the IL to native code. In contrast, .NET Native compiles Windows apps directly to native code. For developers, this means:

- Your apps feature the performance of native code. Usually, performance will be superior to code that is first compiled to IL and then compiled to native code by the JIT compiler.
- You can continue to program in C# or Visual Basic.
- You can continue to take advantage of the resources provided by the .NET Framework, including its class library, automatic memory management and garbage collection, and exception handling.

For users of your apps, .NET Native offers these advantages:

- Faster execution times for the majority of apps and scenarios.
- Faster startup times for the majority of apps and scenarios.
- Low deployment and update costs.
- Optimized app memory usage.

> **IMPORTANT**
> For the vast majority of apps and scenarios, .NET Native offers significantly faster startup times and superior performance when compared to an app compiled to IL or to an NGEN image. However, your results may vary. To ensure that your app has benefited from the performance enhancements of .NET Native, you should compare its performance with that of the non-.NET Native version of your app. For more information, see Performance Session Overview.

But .NET Native involves more than a compilation to native code. It transforms the way that .NET Framework apps are built and executed. In particular:

- During precompilation, required portions of the .NET Framework are statically linked into your app. This allows the app to run with app-local libraries of the .NET Framework, and the compiler to perform global analysis to deliver performance wins. As a result, apps launch consistently faster even after .NET Framework updates.
- The .NET Native runtime is optimized for static precompilation and in the vast majority of cases offers superior performance. At the same time, it retains the core reflection features that developers find so productive.
- .NET Native uses the same back end as the C++ compiler, which is optimized for static precompilation scenarios.

.NET Native is able to bring the performance benefits of C++ to managed code developers because it uses the same or similar tools as C++ under the hood, as shown in this table.

	.NET NATIVE	C++
Libraries	The .NET Framework + Windows Runtime	Win32 + Windows Runtime
Compiler	UTC optimizing compiler	UTC optimizing compiler
Deployed	Ready-to-run binaries	Ready-to-run binaries (ASM)
Runtime	MRT.dll (Minimal CLR Runtime)	CRT.dll (C Runtime)

For Windows apps for Windows 10, you upload .NET Native Code Compilation binaries in app packages (.appx files) to the Windows Store.

In This Section

For more information about developing apps with .NET Native Code Compilation, see these topics:

- Getting Started with .NET Native Code Compilation: The Developer Experience Walkthrough
- .NET Native and Compilation: How .NET Native compiles your project to native code.
- Reflection and .NET Native
  - APIs That Rely on Reflection
  - Reflection API Reference
  - Runtime Directives (rd.xml) Configuration File Reference
- Serialization and Metadata
- Migrating Your Windows Store App to .NET Native
- .NET Native General Troubleshooting

Windows Identity Foundation

9/17/2019 • 2 minutes to read • Edit Online

- WIF Extensions
- WIF API Reference
- WIF Configuration Reference
  - WIF Configuration Schema Conventions

Debugging, Tracing, and Profiling

9/17/2019 • 2 minutes to read • Edit Online

To debug a .NET Framework application, the compiler and runtime environment must be configured to enable a debugger to attach to the application and to produce both symbols and line maps, if possible, for the application and its corresponding Microsoft intermediate language (MSIL). After a managed application has been debugged, it can be profiled to boost performance. Profiling evaluates and describes the lines of source code that generate the most frequently executed code, and how much time it takes to execute them.

.NET Framework applications are easily debugged by using Visual Studio, which handles many of the configuration details. If Visual Studio is not installed, you can examine and improve the performance of .NET Framework applications by using the debugging classes in the .NET Framework System.Diagnostics namespace. This namespace includes the Trace, Debug, and TraceSource classes for tracing execution flow, and the Process, EventLog, and PerformanceCounter classes for profiling code.

In This Section

Enabling JIT-Attach Debugging
Shows how to configure the registry to JIT-attach a debug engine to a .NET Framework application.

Making an Image Easier to Debug
Shows how to turn JIT tracking on and optimization off to make an assembly easier to debug.

Tracing and Instrumenting Applications
Describes how to monitor the execution of your application while it is running, and how to instrument it to display how well it is performing or whether something has gone wrong.

Diagnosing Errors with Managed Debugging Assistants
Describes managed debugging assistants (MDAs), which are debugging aids that work in conjunction with the common language runtime (CLR) to provide information on runtime state.

Enhancing Debugging with the Debugger Display Attributes
Describes how the developer of a type can specify what that type will look like when it is displayed in a debugger.

Performance Counters
Describes the counters that you can use to track the performance of an application.

Related Sections

Debug ASP.NET or ASP.NET Core apps in Visual Studio
Provides prerequisites and instructions for how to debug an ASP.NET application during development or after deployment.

Development Guide
Provides a guide to all key technology areas and tasks for application development, including creating, configuring, debugging, securing, and deploying your application, and information about dynamic programming, interoperability, extensibility, memory management, and threading.

Deploying the .NET Framework and Applications

9/17/2019 • 5 minutes to read • Edit Online

This article helps you get started deploying the .NET Framework with your application. Most of the information is intended for developers, OEMs, and enterprise administrators. Users who want to install the .NET Framework on their computers should read Installing the .NET Framework.

Key Deployment Resources

Use the following links to other MSDN topics for specific information about deploying and servicing the .NET Framework.

Setup and deployment

- General installer and deployment information:
  - Installer options:
    - Web installer
    - Offline installer
  - Installation modes:
    - Silent installation
    - Displaying a UI
  - Reducing system restarts during .NET Framework 4.5 installations
  - Troubleshoot blocked .NET Framework installations and uninstallations
- Deploying the .NET Framework with a client application (for developers):
  - Using InstallShield in a setup and deployment project
  - Using a Visual Studio ClickOnce application
  - Creating a WiX installation package
  - Using a custom installer
  - Additional information for developers
- Deploying the .NET Framework (for OEMs and administrators):
  - Windows Assessment and Deployment Kit (ADK)
  - Administrator's guide

Servicing

- For general information, see the .NET Framework blog
- Detecting versions
- Detecting service packs and updates

Features That Simplify Deployment

The .NET Framework provides a number of basic features that make it easier to deploy your applications:

- No-impact applications.

  This feature provides application isolation and eliminates DLL conflicts. By default, components do not affect other applications.

- Private components by default.

  By default, components are deployed to the application directory and are visible only to the containing application.

- Controlled code sharing.

  Code sharing requires you to explicitly make code available for sharing instead of being the default behavior.

- Side-by-side versioning.

  Multiple versions of a component or application can coexist, you can choose which versions to use, and the common language runtime enforces versioning policy.

- XCOPY deployment and replication.

  Self-described and self-contained components and applications can be deployed without registry entries or dependencies.

- On-the-fly updates.

  Administrators can use hosts, such as ASP.NET, to update program DLLs, even on remote computers.

- Integration with the Windows Installer.

  Advertisement, publishing, repair, and install-on-demand are all available when deploying your application.

- Enterprise deployment.

  This feature provides easy software distribution, including using Active Directory.

- Downloading and caching.

  Incremental downloads keep downloads smaller, and components can be isolated for use only by the application for low-impact deployment.

- Partially trusted code.

  Identity is based on the code instead of the user, and no certificate dialog boxes appear.

Packaging and Distributing .NET Framework Applications

Some of the packaging and deployment information for the .NET Framework is described in other sections of the documentation. Those sections provide information about the self-describing units called assemblies, which require no registry entries, strong-named assemblies, which ensure name uniqueness and prevent name spoofing, and assembly versioning, which addresses many of the problems associated with DLL conflicts. The following sections provide information about packaging and distributing .NET Framework applications.

Packaging

The .NET Framework provides the following options for packaging applications:

- As a single assembly or as a collection of assemblies.

With this option, you simply use the .dll or .exe files as they were built.

- As cabinet (CAB) files.

  With this option, you compress files into .cab files to make distribution or download less time consuming.

- As a Windows Installer package or in other installer formats.

  With this option, you create .msi files for use with the Windows Installer, or you package your application for use with some other installer.

Distribution

The .NET Framework provides the following options for distributing applications:

- Use XCOPY or FTP.

  Because common language runtime applications are self-describing and require no registry entries, you can use XCOPY or FTP to simply copy the application to an appropriate directory. The application can then be run from that directory.

- Use code download.

  If you are distributing your application over the Internet or through a corporate intranet, you can simply download the code to a computer and run the application there.

- Use an installer program such as Windows Installer 2.0.

  Windows Installer 2.0 can install, repair, or remove .NET Framework assemblies in the global assembly cache and in private directories.

Installation Location

To determine where to deploy your application's assemblies so they can be found by the runtime, see How the Runtime Locates Assemblies.

Security considerations can also affect how you deploy your application. Security permissions are granted to managed code according to where the code is located. Deploying an application or component to a location where it receives little trust, such as the Internet, limits what the application or component can do. For more information about deployment and security considerations, see Code Access Security Basics.

Related Topics

TITLE	DESCRIPTION
How the Runtime Locates Assemblies	Describes how the common language runtime determines which assembly to use to fulfill a binding request.
Best Practices for Assembly Loading	Discusses ways to avoid problems of type identity that can lead to InvalidCastException, MissingMethodException, and other errors.
Reducing System Restarts During .NET Framework 4.5 Installations	Describes the Restart Manager, which prevents reboots whenever possible, and explains how applications that install the .NET Framework can take advantage of it.
Deployment Guide for Administrators	Explains how a system administrator can deploy the .NET Framework and its system dependencies across a network by using System Center Configuration Manager (SCCM).

TITLE	DESCRIPTION
Deployment Guide for Developers	Explains how developers can install .NET Framework on their users' computers with their applications.
Deploying Applications, Services, and Components	Discusses deployment options in Visual Studio, including instructions for publishing an application using the ClickOnce and Windows Installer technologies.
Publishing ClickOnce Applications	Describes how to package a Windows Forms application and deploy it with ClickOnce to client computers on a network.
Packaging and Deploying Resources	Describes the hub and spoke model that the .NET Framework uses to package and deploy resources; covers resource naming conventions, fallback process, and packaging alternatives.
Deploying an Interop Application	Explains how to ship and install interop applications, which typically include a .NET Framework client assembly, one or more interop assemblies representing distinct COM type libraries, and one or more registered COM components.
How to: Get Progress from the .NET Framework 4.5 Installer	Describes how to silently launch and track the .NET Framework setup process while showing your own view of the setup progress.

See also

- Development Guide

.NET Framework Performance

9/17/2019 • 4 minutes to read • Edit Online

If you want to create apps with great performance, you should design and plan for performance just as you would design any other feature of your app. You can use the tools provided by Microsoft to measure your app's performance, and, if needed, make improvements to memory use, code throughput, and responsiveness. This topic lists the performance analysis tools that Microsoft provides, and provides links to other topics that cover performance for specific areas of app development.

Designing and planning for performance

If you want a great performing app, you must design performance into your app just as you would design any other feature. You should determine the performance-critical scenarios in your app, set performance goals, and measure performance for these app scenarios early and often. Because each app is different and has different performance-critical execution paths, determining those paths early and focusing your efforts enable you to maximize your productivity.

You don't have to be completely familiar with your target platform to create a high-performance app. However, you should develop an understanding of which parts of your target platform are costly in terms of performance. You can do this by measuring performance early in your development process.

To determine the areas that are crucial to performance and to establish your performance goals, always consider the user experience. Startup time and responsiveness are two key areas that will affect the user's perception of your app. If your app uses a lot of memory, it may appear sluggish to the user or affect other apps running on the system, or, in some cases, it could fail the Windows Store or Windows Phone Store submission process. Also, if you determine which parts of your code execute more frequently, you can make sure that these portions of your code are well optimized.

Analyzing performance

As part of your overall development plan, set points during development where you will measure the performance of your app and compare the results with the goals you set previously. Measure your app in the environment and hardware that you expect your users to have. By analyzing your app's performance early and often you can change architectural decisions that would be costly and expensive to fix later in the development cycle. The following sections describe performance tools you can use to analyze your apps and discuss event tracing, which is used by these tools.

Performance tools

Here are some of the performance tools you can use with your .NET Framework apps.

TOOL	DESCRIPTION
Visual Studio Performance Analysis	Use to analyze the CPU usage of your .NET Framework apps that will be deployed to computers that are running the Windows operating system. This tool is available from the **Debug** menu in Visual Studio after you open a project. For more information, see Performance Explorer. **Note:** Use Windows Phone Application Analysis (see next row) when targeting Windows Phone.

TOOL	DESCRIPTION
Windows Phone Application Analysis	Use to analyze the CPU and memory, network data transfer rate, app responsiveness, and battery consumption in your Windows Phone apps. This tool is available from the **Debug** menu for a Windows Phone project in Visual Studio after you install the Windows Phone SDK. For more information, see App profiling for Windows Phone 8.
PerfView	Use to identify CPU and memory-related performance issues. This tool uses event tracing for Windows (ETW) and CLR profiling APIs to provide advanced memory and CPU investigations as well as information about garbage collection and JIT compilation. For more information about how to use PerfView, see the tutorial and help files that are included with the app, Channel 9 video tutorials, and blog posts. For memory-specific issues, see Using PerfView for Memory Investigations.
Windows Performance Analyzer	Use to determine overall system performance such as your app's memory and storage use when multiple apps are running on the same computer. This tool is available from the download center as part of the Windows Assessment and Deployment Kit (ADK) for Windows 8. For more information, see Windows Performance Analyzer.

Event tracing for Windows (ETW)

ETW is a technique that lets you obtain diagnostic information about running code and is essential for many of the performance tools mentioned previously. ETW creates logs when particular events are raised by .NET Framework apps and Windows. With ETW, you can enable and disable logging dynamically, so that you can perform detailed tracing in a production environment without restarting your app. The .NET Framework offers support for ETW events, and ETW is used by many profiling and performance tools to generate performance data. These tools often enable and disable ETW events, so familiarity with them is helpful. You can use specific ETW events to collect performance information about particular components of your app. For more information about ETW support in the .NET Framework, see ETW Events in the Common Language Runtime and ETW Events in Task Parallel Library and PLINQ.

Performance by app type

Each type of .NET Framework app has its own best practices, considerations, and tools for evaluating performance. The following table links to performance topics for specific .NET Framework app types.

APP TYPE	SEE
.NET Framework apps for all platforms	Garbage Collection and Performance Performance Tips
Windows 8.x Store apps written in C++, C#, and Visual Basic	Performance best practices for Windows Store apps using C++, C#, and Visual Basic
Windows Presentation Foundation (WPF)	WPF Performance Suite

APP TYPE	SEE
ASP.NET	ASP.NET Performance Overview

Related Topics

TITLE	DESCRIPTION
Caching in .NET Framework Applications	Describes techniques for caching data to improve performance in your app.
Lazy Initialization	Describes how to initialize objects as-needed to improve performance, particularly at app startup.
Reliability	Provides information about preventing asynchronous exceptions in a server environment.
Writing Large, Responsive .NET Framework Apps	Provides performance tips gathered from rewriting the C# and Visual Basic compilers in managed code, and includes several real examples from the C# compiler.

Dynamic Programming in the .NET Framework

9/17/2019 • 2 minutes to read • Edit Online

This section of the documentation provides information about dynamic programming in the .NET Framework.

In This Section

Reflection

Describes how to use reflection to work with objects at run time.

Emitting Dynamic Methods and Assemblies

Describes how to create methods and assemblies at run time by using Reflection.Emit.

Dynamic Language Runtime Overview

Describes the features of the dynamic language runtime.

Dynamic Source Code Generation and Compilation

Describes how to generate and compile dynamic source code.

Related Sections

Development Guide

Managed Extensibility Framework (MEF)

7/24/2019 • 16 minutes to read • Edit Online

This topic provides an overview of the Managed Extensibility Framework that was introduced in the .NET Framework 4.

What is MEF?

The Managed Extensibility Framework or MEF is a library for creating lightweight, extensible applications. It allows application developers to discover and use extensions with no configuration required. It also lets extension developers easily encapsulate code and avoid fragile hard dependencies. MEF not only allows extensions to be reused within applications, but across applications as well.

The Problem of Extensibility

Imagine that you are the architect of a large application that must provide support for extensibility. Your application has to include a potentially large number of smaller components, and is responsible for creating and running them.

The simplest approach to the problem is to include the components as source code in your application, and call them directly from your code. This has a number of obvious drawbacks. Most importantly, you cannot add new components without modifying the source code, a restriction that might be acceptable in, for example, a Web application, but is unworkable in a client application. Equally problematic, you may not have access to the source code for the components, because they might be developed by third parties, and for the same reason you cannot allow them to access yours.

A slightly more sophisticated approach would be to provide an extension point or interface, to permit decoupling between the application and its components. Under this model, you might provide an interface that a component can implement, and an API to enable it to interact with your application. This solves the problem of requiring source code access, but it still has its own difficulties.

Because the application lacks any capacity for discovering components on its own, it must still be explicitly told which components are available and should be loaded. This is typically accomplished by explicitly registering the available components in a configuration file. This means that assuring that the components are correct becomes a maintenance issue, particularly if it is the end user and not the developer who is expected to do the updating.

In addition, components are incapable of communicating with one another, except through the rigidly defined channels of the application itself. If the application architect has not anticipated the need for a particular communication, it is usually impossible.

Finally, the component developers must accept a hard dependency on what assembly contains the interface they implement. This makes it difficult for a component to be used in more than one application, and can also create problems when you create a test framework for components.

What MEF Provides

Instead of this explicit registration of available components, MEF provides a way to discover them implicitly, via *composition*. A MEF component, called a *part*, declaratively specifies both its dependencies (known as *imports*) and what capabilities (known as *exports*) it makes available. When a part is created, the MEF composition engine satisfies its imports with what is available from other parts.

This approach solves the problems discussed in the previous section. Because MEF parts declaratively specify their

capabilities, they are discoverable at runtime, which means an application can make use of parts without either hard-coded references or fragile configuration files. MEF allows applications to discover and examine parts by their metadata, without instantiating them or even loading their assemblies. As a result, there is no need to carefully specify when and how extensions should be loaded.

In addition to its provided exports, a part can specify its imports, which will be filled by other parts. This makes communication among parts not only possible, but easy, and allows for good factoring of code. For example, services common to many components can be factored into a separate part and easily modified or replaced.

Because the MEF model requires no hard dependency on a particular application assembly, it allows extensions to be reused from application to application. This also makes it easy to develop a test harness, independent of the application, to test extension components.

An extensible application written by using MEF declares an import that can be filled by extension components, and may also declare exports in order to expose application services to extensions. Each extension component declares an export, and may also declare imports. In this way, extension components themselves are automatically extensible.

Where Is MEF Available?

MEF is an integral part of the .NET Framework 4, and is available wherever the .NET Framework is used. You can use MEF in your client applications, whether they use Windows Forms, WPF, or any other technology, or in server applications that use ASP.NET.

MEF and MAF

Previous versions of the .NET Framework introduced the Managed Add-in Framework (MAF), designed to allow applications to isolate and manage extensions. The focus of MAF is slightly higher-level than MEF, concentrating on extension isolation and assembly loading and unloading, while MEF's focus is on discoverability, extensibility, and portability. The two frameworks interoperate smoothly, and a single application can take advantage of both.

SimpleCalculator: An Example Application

The simplest way to see what MEF can do is to build a simple MEF application. In this example, you build a very simple calculator named SimpleCalculator. The goal of SimpleCalculator is to create a console application that accepts basic arithmetic commands, in the form "5+3" or "6-2", and returns the correct answers. Using MEF, you'll be able to add new operators without changing the application code.

To download the complete code for this example, see the SimpleCalculator sample.

> **NOTE**
> The purpose of SimpleCalculator is to demonstrate the concepts and syntax of MEF, rather than to necessarily provide a realistic scenario for its use. Many of the applications that would benefit most from the power of MEF are more complex than SimpleCalculator. For more extensive examples, see the Managed Extensibility Framework on GitHub.

- To start, in Visual Studio, create a new Console Application project and name it `SimpleCalculator`.
- Add a reference to the System.ComponentModel.Composition assembly, where MEF resides.
- Open Module1.vb or Program.cs and add `Imports` or `using` statements for System.ComponentModel.Composition and System.ComponentModel.Composition.Hosting. These two namespaces contain MEF types you will need to develop an extensible application.
- If you're using Visual Basic, add the `Public` keyword to the line that declares the `Module1` module.

Composition Container and Catalogs

The core of the MEF composition model is the *composition container*, which contains all the parts available and performs composition. Composition is the matching up of imports to exports. The most common type of composition container is CompositionContainer, and you'll use this for SimpleCalculator.

If you're using Visual Basic, in Module1.vb, add a public class named `Program`.

Add the following line to the `Program` class in Module1.vb or Program.cs:

```
Dim _container As CompositionContainer
```

```
private CompositionContainer _container;
```

In order to discover the parts available to it, the composition containers makes use of a *catalog*. A catalog is an object that makes available parts discovered from some source. MEF provides catalogs to discover parts from a provided type, an assembly, or a directory. Application developers can easily create new catalogs to discover parts from other sources, such as a Web service.

Add the following constructor to the `Program` class:

```
Public Sub New()
    ' An aggregate catalog that combines multiple catalogs.
     Dim catalog = New AggregateCatalog()

    ' Adds all the parts found in the same assembly as the Program class.
    catalog.Catalogs.Add(New AssemblyCatalog(GetType(Program).Assembly))

    ' Create the CompositionContainer with the parts in the catalog.
    _container = New CompositionContainer(catalog)

    ' Fill the imports of this object.
    Try
        _container.ComposeParts(Me)
    Catch ex As CompositionException
        Console.WriteLine(ex.ToString)
    End Try
End Sub
```

```
private Program()
{
    // An aggregate catalog that combines multiple catalogs.
    var catalog = new AggregateCatalog();
    // Adds all the parts found in the same assembly as the Program class.
    catalog.Catalogs.Add(new AssemblyCatalog(typeof(Program).Assembly));

    // Create the CompositionContainer with the parts in the catalog.
    _container = new CompositionContainer(catalog);

    // Fill the imports of this object.
    try
    {
        this._container.ComposeParts(this);
    }
    catch (CompositionException compositionException)
    {
        Console.WriteLine(compositionException.ToString());
   }
}
```

The call to ComposeParts tells the composition container to compose a specific set of parts, in this case the current instance of `Program`. At this point, however, nothing will happen, since `Program` has no imports to fill.

Imports and Exports with Attributes

First, you have `Program` import a calculator. This allows the separation of user interface concerns, such as the console input and output that will go into `Program`, from the logic of the calculator.

Add the following code to the `Program` class:

```
<Import(GetType(ICalculator))>
Public Property calculator As ICalculator
```

```
[Import(typeof(ICalculator))]
public ICalculator calculator;
```

Notice that the declaration of the `calculator` object is not unusual, but that it is decorated with the ImportAttribute attribute. This attribute declares something to be an import; that is, it will be filled by the composition engine when the object is composed.

Every import has a *contract*, which determines what exports it will be matched with. The contract can be an explicitly specified string, or it can be automatically generated by MEF from a given type, in this case the interface `ICalculator`. Any export declared with a matching contract will fulfill this import. Note that while the type of the `calculator` object is in fact `ICalculator`, this is not required. The contract is independent from the type of the importing object. (In this case, you could leave out the `typeof(ICalculator)`. MEF will automatically assume the contract to be based on the type of the import unless you specify it explicitly.)

Add this very simple interface to the module or `SimpleCalculator` namespace:

```
Public Interface ICalculator
    Function Calculate(ByVal input As String) As String
End Interface
```

```
public interface ICalculator
{
    String Calculate(String input);
}
```

Now that you have defined `ICalculator`, you need a class that implements it. Add the following class to the module or `SimpleCalculator` namespace:

```
<Export(GetType(ICalculator))>
Public Class MySimpleCalculator
   Implements ICalculator

End Class
```

```
[Export(typeof(ICalculator))]
class MySimpleCalculator : ICalculator
{

}
```

Here is the export that will match the import in `Program`. In order for the export to match the import, the export must have the same contract. Exporting under a contract based on `typeof(MySimpleCalculator)` would produce a mismatch, and the import would not be filled; the contract needs to match exactly.

Since the composition container will be populated with all the parts available in this assembly, the `MySimpleCalculator` part will be available. When the constructor for `Program` performs composition on the `Program` object, its import will be filled with a `MySimpleCalculator` object, which will be created for that purpose.

The user interface layer (`Program`) does not need to know anything else. You can therefore fill in the rest of the user interface logic in the `Main` method.

Add the following code to the `Main` method:

```
Sub Main()
    ' Composition is performed in the constructor.
    Dim p As New Program()
    Dim s As String
    Console.WriteLine("Enter Command:")
    While (True)
        s = Console.ReadLine()
        Console.WriteLine(p.calculator.Calculate(s))
    End While
End Sub
```

```
static void Main(string[] args)
{
    // Composition is performed in the constructor.
    var p = new Program();
    String s;
    Console.WriteLine("Enter Command:");
    while (true)
    {
        s = Console.ReadLine();
        Console.WriteLine(p.calculator.Calculate(s));
    }
}
```

This code simply reads a line of input and calls the `Calculate` function of `ICalculator` on the result, which it writes back to the console. That is all the code you need in `Program`. All the rest of the work will happen in the parts.

Further Imports and ImportMany

In order for SimpleCalculator to be extensible, it needs to import a list of operations. An ordinary ImportAttribute attribute is filled by one and only one ExportAttribute. If more than one is available, the composition engine produces an error. To create an import that can be filled by any number of exports, you can use the ImportManyAttribute attribute.

Add the following operations property to the `MySimpleCalculator` class:

```
<ImportMany()>
Public Property operations As IEnumerable(Of Lazy(Of IOperation, IOperationData))
```

```
[ImportMany]
IEnumerable<Lazy<IOperation, IOperationData>> operations;
```

Lazy<T,TMetadata> is a type provided by MEF to hold indirect references to exports. Here, in addition to the exported object itself, you also get *export metadata*, or information that describes the exported object. Each Lazy<T,TMetadata> contains an `IOperation` object, representing an actual operation, and an `IOperationData` object, representing its metadata.

Add the following simple interfaces to the module or `SimpleCalculator` namespace:

```
Public Interface IOperation
    Function Operate(ByVal left As Integer, ByVal right As Integer) As Integer
End Interface

Public Interface IOperationData
    ReadOnly Property Symbol As Char
End Interface
```

```
public interface IOperation
{
     int Operate(int left, int right);
}

public interface IOperationData
{
    Char Symbol { get; }
}
```

In this case, the metadata for each operation is the symbol that represents that operation, such as +, -, *, and so on. To make the addition operation available, add the following class to the module or `SimpleCalculator` namespace:

```
<Export(GetType(IOperation))>
<ExportMetadata("Symbol", "+"c)>
Public Class Add
    Implements IOperation

    Public Function Operate(ByVal left As Integer, ByVal right As Integer) As Integer Implements
IOperation.Operate
        Return left + right
    End Function
End Class
```

```
[Export(typeof(IOperation))]
[ExportMetadata("Symbol", '+')]
class Add: IOperation
{
    public int Operate(int left, int right)
    {
        return left + right;
    }
}
```

The ExportAttribute attribute functions as it did before. The ExportMetadataAttribute attribute attaches metadata, in the form of a name-value pair, to that export. While the `Add` class implements `IOperation`, a class that implements `IOperationData` is not explicitly defined. Instead, a class is implicitly created by MEF with properties based on the names of the metadata provided. (This is one of several ways to access metadata in MEF.)

Composition in MEF is *recursive*. You explicitly composed the `Program` object, which imported an `ICalculator` that turned out to be of type `MySimpleCalculator`. `MySimpleCalculator`, in turn, imports a collection of `IOperation` objects, and that import will be filled when `MySimpleCalculator` is created, at the same time as the imports of

`Program`. If the `Add` class declared a further import, that too would have to be filled, and so on. Any import left unfilled results in a composition error. (It is possible, however, to declare imports to be optional or to assign them default values.)

Calculator Logic

With these parts in place, all that remains is the calculator logic itself. Add the following code in the `MySimpleCalculator` class to implement the `Calculate` method:

```
Public Function Calculate(ByVal input As String) As String Implements ICalculator.Calculate
    Dim left, right As Integer
    Dim operation As Char
    ' Finds the operator.
    Dim fn = FindFirstNonDigit(input)
    If fn < 0 Then
        Return "Could not parse command."
    End If
    operation = input(fn)
    Try
        ' Separate out the operands.
        left = Integer.Parse(input.Substring(0, fn))
        right = Integer.Parse(input.Substring(fn + 1))
    Catch ex As Exception
        Return "Could not parse command."
    End Try
    For Each i As Lazy(Of IOperation, IOperationData) In operations
        If i.Metadata.symbol = operation Then
            Return i.Value.Operate(left, right).ToString()
        End If
    Next
    Return "Operation not found!"
End Function
```

```
public String Calculate(String input)
{
    int left;
    int right;
    char operation;
    // Finds the operator.
    int fn = FindFirstNonDigit(input);
    if (fn < 0) return "Could not parse command.";

    try
    {
        // Separate out the operands.
        left = int.Parse(input.Substring(0, fn));
        right = int.Parse(input.Substring(fn + 1));
    }
    catch
    {
        return "Could not parse command.";
    }

    operation = input[fn];

    foreach (Lazy<IOperation, IOperationData> i in operations)
    {
        if (i.Metadata.Symbol.Equals(operation)) return i.Value.Operate(left, right).ToString();
    }
    return "Operation Not Found!";
}
```

The initial steps parse the input string into left and right operands and an operator character. In the `foreach` loop,

every member of the `operations` collection is examined. These objects are of type Lazy<T,TMetadata>, and their metadata values and exported object can be accessed with the Metadata property and the Value property respectively. In this case, if the `Symbol` property of the `IOperationData` object is discovered to be a match, the calculator calls the `Operate` method of the `IOperation` object and returns the result.

To complete the calculator, you also need a helper method that returns the position of the first non-digit character in a string. Add the following helper method to the `MySimpleCalculator` class:

```
Private Function FindFirstNonDigit(ByVal s As String) As Integer
    For i = 0 To s.Length - 1
        If (Not (Char.IsDigit(s(i)))) Then Return i
    Next
    Return -1
End Function
```

```
private int FindFirstNonDigit(string s)
{
    for (int i = 0; i < s.Length; i++)
    {
        if (!(Char.IsDigit(s[i]))) return i;
    }
    return -1;
}
```

You should now be able to compile and run the project. In Visual Basic, make sure that you added the `Public` keyword to `Module1`. In the console window, type an addition operation, such as "5+3", and the calculator returns the results. Any other operator results in the "Operation Not Found!" message.

Extending SimpleCalculator Using A New Class

Now that the calculator works, adding a new operation is easy. Add the following class to the module or `SimpleCalculator` namespace:

```
<Export(GetType(IOperation))>
<ExportMetadata("Symbol", "-"c)>
Public Class Subtract
    Implements IOperation

    Public Function Operate(ByVal left As Integer, ByVal right As Integer) As Integer Implements
IOperation.Operate
        Return left - right
    End Function
End Class
```

```
[Export(typeof(IOperation))]
[ExportMetadata("Symbol", '-')]
class Subtract : IOperation
{
    public int Operate(int left, int right)
    {
        return left - right;
    }
}
```

Compile and run the project. Type a subtraction operation, such as "5-3". The calculator now supports subtraction as well as addition.

Extending SimpleCalculator Using A New Assembly

Adding classes to the source code is simple enough, but MEF provides the ability to look outside an application's own source for parts. To demonstrate this, you will need to modify SimpleCalculator to search a directory, as well as its own assembly, for parts, by adding a DirectoryCatalog.

Add a new directory named `Extensions` to the SimpleCalculator project. Make sure to add it at the project level, and not at the solution level. Then add a new Class Library project to the solution, named `ExtendedOperations`. The new project will compile into a separate assembly.

Open the Project Properties Designer for the ExtendedOperations project and click the **Compile** or **Build** tab. Change the **Build output path** or **Output path** to point to the Extensions directory in the SimpleCalculator project directory (..\SimpleCalculator\Extensions\).

In Module1.vb or Program.cs, add the following line to the `Program` constructor:

```
catalog.Catalogs.Add(New DirectoryCatalog("C:\SimpleCalculator\SimpleCalculator\Extensions"))
```

```
catalog.Catalogs.Add(new DirectoryCatalog("C:\\SimpleCalculator\\SimpleCalculator\\Extensions"));
```

Replace the example path with the path to your Extensions directory. (This absolute path is for debugging purposes only. In a production application, you would use a relative path.) The DirectoryCatalog will now add any parts found in any assemblies in the Extensions directory to the composition container.

In the ExtendedOperations project, add references to SimpleCalculator and System.ComponentModel.Composition. In the ExtendedOperations class file, add an `Imports` or a `using` statement for System.ComponentModel.Composition. In Visual Basic, also add an `Imports` statement for SimpleCalculator. Then add the following class to the ExtendedOperations class file:

```
<Export(GetType(SimpleCalculator.IOperation))>
<ExportMetadata("Symbol", "%"c)>
Public Class Modulo
    Implements IOperation

    Public Function Operate(ByVal left As Integer, ByVal right As Integer) As Integer Implements
IOperation.Operate
        Return left Mod right
    End Function
End Class
```

```
[Export(typeof(SimpleCalculator.IOperation))]
[ExportMetadata("Symbol", '%')]
public class Mod : SimpleCalculator.IOperation
{
    public int Operate(int left, int right)
    {
        return left % right;
    }
}
```

Note that in order for the contract to match, the ExportAttribute attribute must have the same type as the ImportAttribute.

Compile and run the project. Test the new Mod (%) operator.

Conclusion

This topic covered the basic concepts of MEF.

- Parts, catalogs, and the composition container

  Parts and the composition container are the basic building blocks of a MEF application. A part is any object that imports or exports a value, up to and including itself. A catalog provides a collection of parts from a particular source. The composition container uses the parts provided by a catalog to perform composition, the binding of imports to exports.

- Imports and exports

  Imports and exports are the way by which components communicate. With an import, the component specifies a need for a particular value or object, and with an export it specifies the availability of a value. Each import is matched with a list of exports by way of its contract.

Where Do I Go Now?

To download the complete code for this example, see the SimpleCalculator sample.

For more information and code examples, see Managed Extensibility Framework. For a list of the MEF types, see the System.ComponentModel.Composition namespace.

Interoperating with unmanaged code

3/25/2019 • 2 minutes to read • Edit Online

The .NET Framework promotes interaction with COM components, COM+ services, external type libraries, and many operating system services. Data types, method signatures, and error-handling mechanisms vary between managed and unmanaged object models. To simplify interoperation between .NET Framework components and unmanaged code and to ease the migration path, the common language runtime conceals from both clients and servers the differences in these object models.

Code that executes under the control of the runtime is called managed code. Conversely, code that runs outside the runtime is called unmanaged code. COM components, ActiveX interfaces, and Windows API functions are examples of unmanaged code.

In this section

Exposing COM Components to the .NET Framework
Describes how to use COM components from .NET Framework applications.

Exposing .NET Framework Components to COM
Describes how to use .NET Framework components from COM applications.

Consuming Unmanaged DLL Functions
Describes how to call unmanaged DLL functions using platform invoke.

Interop Marshaling
Describes marshaling for COM interop and platform invoke.

How to: Map HRESULTs and Exceptions
Describes the mapping between exceptions and HRESULTs.

COM Wrappers
Describes the wrappers provided by COM interop.

Type Equivalence and Embedded Interop Types
Describes how type information for COM types is embedded in assemblies, and how the common language runtime determines the equivalence of embedded COM types.

How to: Generate Primary Interop Assemblies Using Tlbimp.exe
Describes how to produce primary interop assemblies using *Tlbimp.exe* (Type Library Importer).

How to: Register Primary Interop Assemblies
Describes how to register the primary interop assemblies before you can reference them in your projects.

Registration-Free COM Interop
Describes how COM interop can activate components without using the Windows registry.

How to: Configure .NET Framework-Based COM Components for Registration-Free Activation
Describes how to create an application manifest and how to create and embed a component manifest.

Unmanaged API Reference

9/7/2019 • 2 minutes to read • Edit Online

This section includes information on unmanaged APIs that can be used by managed-code-related applications, such as runtime hosts, compilers, disassemblers, obfuscators, debuggers, and profilers.

In This Section

Common Data Types

Lists the common data types that are used, particularly in the unmanaged profiling and debugging APIs.

ALink

Describes the ALink API, which supports the creation of .NET Framework assemblies and unbound modules.

Authenticode

Supports the Authenticode XrML license creation and verification module.

Constants

Describes the constants that are defined in CorSym.idl.

Custom Interface Attributes

Describes component object model (COM) custom interface attributes.

Debugging

Describes the debugging API, which enables a debugger to debug code that runs in the common language runtime (CLR) environment.

Diagnostics Symbol Store

Describes the diagnostics symbol store API, which enables a compiler to generate symbol information for use by a debugger.

Fusion

Describes the fusion API, which enables a runtime host to access the properties of an application's resources in order to locate the correct versions of those resources for the application.

Hosting

Describes the hosting API, which enables unmanaged hosts to integrate the CLR into their applications.

Metadata

Describes the metadata API, which enables a client such as a compiler to generate or access a component's metadata without the types being loaded by the CLR.

Profiling

Describes the profiling API, which enables a profiler to monitor a program's execution by the CLR.

Strong Naming

Describes the strong naming API, which enables a client to administer strong name signing for assemblies.

WMI and Performance Counters

Describes the APIs that wrap calls to Windows Management Instrumentation (WMI) libraries.

Tlbexp Helper Functions

Describes the two helper functions and interface used by the Type Library Exporter (Tlbexp.exe) during the assembly-to-type-library conversion process.

Related Sections

Development Guide

XAML Services

6/4/2019 • 8 minutes to read • Edit Online

This topic describes the capabilities of a technology set known as .NET Framework XAML Services. The majority of the services and APIs described are located in the assembly System.Xaml, which is an assembly introduced with the .NET Framework 4 set of .NET core assemblies. Services include readers and writers, schema classes and schema support, factories, attributing of classes, XAML language intrinsic support, and other XAML language features.

About This Documentation

Conceptual documentation for .NET Framework XAML Services assumes that you have previous experience with the XAML language and how it might apply to a specific framework, for example Windows Presentation Foundation (WPF) or Windows Workflow Foundation, or a specific technology feature area, for example the build customization features in Microsoft.Build.Framework.XamlTypes. This documentation does not attempt to explain the basics of XAML as a markup language, XAML syntax terminology, or other introductory material. Instead, this documentation focuses on specifically using the .NET Framework XAML Services that are enabled in the System.Xaml assembly library. Most of these APIs are for scenarios of XAML language integration and extensibility. This might include any of the following:

- Extending the capabilities of the base XAML readers or XAML writers (processing the XAML node stream directly; deriving your own XAML reader or XAML writer).
- Defining XAML-usable custom types that do not have specific framework dependencies, and attributing the types to convey their XAML type system characteristics to .NET Framework XAML Services.
- Hosting XAML readers or XAML writers as a component of an application, such as a visual designer or interactive editor for XAML markup sources.
- Writing XAML value converters (markup extensions; type converters for custom types).
- Defining a custom XAML schema context (using alternate assembly-loading techniques for backing type sources; using known-types lookup techniques instead of always reflecting assemblies; using loaded assembly concepts that do not use the CLR `AppDomain` and its associated security model).
- Extending the base XAML type system.
- Using the `Lookup` or `Invoker` techniques to influence the XAML type system and how type backings are evaluated.

If you are looking for introductory material on XAML as a language, you might try XAML Overview (WPF). That topic discusses XAML for an audience that is new both to Windows Presentation Foundation (WPF) and also to using XAML markup and XAML language features. Another useful document is the introductory material in the XAML language specification.

.NET Framework XAML Services and System.Xaml in the .NET Architecture

In previous versions of Microsoft .NET Framework, support for XAML language features was implemented by frameworks that built on Microsoft .NET Framework (Windows Presentation Foundation (WPF), Windows Workflow Foundation and Windows Communication Foundation (WCF)), and therefore varied in its behavior and the API used depending on which specific framework you were using. This included the XAML parser and its object

graph creation mechanism, XAML language intrinsics, serialization support, and so on.

In .NET Framework 4, .NET Framework XAML Services and the System.Xaml assembly define much of what is needed for supporting XAML language features. This includes base classes for XAML readers and XAML writers. The most important feature added to .NET Framework XAML Services that was not present in any of the framework-specific XAML implementations is a type system representation for XAML. The type system representation presents XAML in an object-oriented way that centers on XAML capabilities without taking dependencies on specific capabilities of frameworks.

The XAML type system is not limited by the markup form or run-time specifics of the XAML origin; nor is it limited by any specific backing type system. The XAML type system includes object representations for types, members, XAML schema contexts, XML-level concepts, and other XAML language concepts or XAML intrinsics. Using or extending the XAML type system makes it possible to derive from classes like XAML readers and XAML writers, and extend the functionality of XAML representations into specific features enabled by a framework, a technology, or an application that consumes or emits XAML. The concept of a XAML schema context enables practical object graph write operations from the combination of a XAML object writer implementation, a technology's backing type system as communicated through assembly information in the context, and the XAML node source. For more information on the XAML schema concept. see Default XAML Schema Context and WPF XAML Schema Context.

XAML Node Streams, XAML Readers, and XAML Writers

To understand the role that .NET Framework XAML Services plays in the relationship between the XAML language and specific technologies that use XAML as a language, it is helpful to understand the concept of a XAML node stream and how that concept shapes the API and terminology. The XAML node stream is a conceptual intermediate between a XAML language representation and the object graph that the XAML represents or defines.

- A XAML reader is an entity that processes XAML in some form, and produces a XAML node stream. In the API, a XAML reader is represented by the base class XamlReader.
- A XAML writer is an entity that processes a XAML node stream and produces something else. In the API, a XAML writer is represented by the base class XamlWriter.

The two most common scenarios involving XAML are loading XAML to instantiate an object graph, and saving an object graph from an application or tool and producing a XAML representation (typically in markup form saved as text file). Loading XAML and creating an object graph is often referred to in this documentation as the load path. Saving or serializing an existing object graph to XAML is often referred to in this documentation as the save path.

The most common type of load path can be described as follows:

- Start with a XAML representation, in UTF-encoded XML format and saved as a text file.
- Load that XAML into XamlXmlReader. XamlXmlReader is a XamlReader subclass.
- The result is a XAML node stream. You can access individual nodes of the XAML node stream using XamlXmlReader / XamlReader API. The most typical operation here is to advance through the XAML node stream, processing each node using a "current record" metaphor.
- Pass the resulting nodes from the XAML node stream to a XamlObjectWriter API. XamlObjectWriter is a XamlWriter subclass.
- The XamlObjectWriter writes an object graph, one object at a time, in accordance to progress through the source XAML node stream. This is done with the assistance of a XAML schema context and an implementation that can access the assemblies and types of a backing type system and framework.
- Call Result at the end of the XAML node stream to obtain the root object of the object graph.

The most common type of save path can be described as follows:

- Start with the object graph of an entire application run time, the UI content and state of a run time, or a smaller segment of an overall application's object representation at run time.
- From a logical start object, such as an application root or document root, load the objects into XamlObjectReader. XamlObjectReader is a XamlReader subclass.
- The result is a XAML node stream. You can access individual nodes of the XAML node stream using XamlObjectReader and XamlReader API. The most typical operation here is to advance through the XAML node stream, processing each node using a "current record" metaphor.
- Pass the resulting nodes from the XAML node stream to a XamlXmlWriter API. XamlXmlWriter is a XamlWriter subclass.
- The XamlXmlWriter writes XAML in an XML UTF encoding. You can save this as a text file, as a stream, or in other forms.
- Call Flush to obtain the final output.

For more information about XAML node stream concepts, see Understanding XAML Node Stream Structures and Concepts.

The XamlServices Class

It is not always necessary to deal with a XAML node stream. If you want a basic load path or a basic save path, you can use APIs in the XamlServices class.

- Various signatures of Load implement a load path. You can either load a file or stream, or can load an XmlReader, TextReader or XamlReader that wrap your XAML input by loading with that reader's APIs.
- Various signatures of Save save an object graph and produce output as a stream, file, or XmlWriter/TextWriter instance.
- Transform converts XAML by linking a load path and a save path as a single operation. A different schema context or different backing type system could be used for XamlReader and XamlWriter, which is what influences how the resulting XAML is transformed.

For more information about how to use XamlServices, see XAMLServices Class and Basic XAML Reading or Writing.

XAML Type System

The XAML type system provides the APIs that are required to work with a given individual node of a XAML node stream.

XamlType is the representation for an object - what you are processing between a start object node and end object node.

XamlMember is the representation for a member of an object - what you are processing between a start member node and end member node.

APIs such as GetAllMembers and GetMember and DeclaringType report the relationships between a XamlType and XamlMember.

The default behavior of the XAML type system as implemented by .NET Framework XAML Services is based on the common language runtime (CLR), and static analysis of CLR types in assemblies by using reflection. Therefore, for a specific CLR type, the default implementation of the XAML type system can expose the XAML schema of that type and its members and report it in terms of the XAML type system. In the default XAML type system, the concept of assignability of types is mapped onto CLR inheritance, and the concepts of instances, value types and so on are also mapped to the supporting behaviors and features of the CLR.

Reference for XAML Language Features

To support XAML, .NET Framework XAML Services provides specific implementation of XAML language concepts as defined for the XAML language XAML namespace. These are documented as specific reference pages. The language features are documented from the perspective of how these language features behave when they are processed by a XAML reader or XAML writer that is defined by .NET Framework XAML Services. For more information, see XAML Namespace (x:) Language Features.

.NET Framework Tools

9/17/2019 • 5 minutes to read • Edit Online

The .NET Framework tools make it easier for you to create, deploy, and manage applications and components that target the .NET Framework.

Most of the .NET Framework tools described in this section are automatically installed with Visual Studio. To download Visual Studio, visit the Visual Studio Downloads page.

You can run all the tools from the command line with the exception of the Assembly Cache Viewer (Shfusion.dll). You must access Shfusion.dll from File Explorer.

The best way to run the command-line tools is by using the Developer Command Prompt for Visual Studio. These utilities enable you to run the tools easily, without navigating to the installation folder. For more information, see Command Prompts.

> **NOTE**
> Some tools are specific to either 32-bit computers or 64-bit computers. Be sure to run the appropriate version of the tool for your computer.

In This Section

Al.exe (Assembly Linker)
Generates a file that has an assembly manifest from modules or resource files.

Aximp.exe (Windows Forms ActiveX Control Importer)
Converts type definitions in a COM type library for an ActiveX control into a Windows Forms control.

Caspol.exe (Code Access Security Policy Tool)
Enables you to view and configure security policy for the machine policy level, the user policy level, and the enterprise policy level. In the .NET Framework 4 and later, this tool does not affect code access security (CAS) policy unless the <legacyCasPolicy> element is set to `true`. For more information, see Security Changes.

Cert2spc.exe (Software Publisher Certificate Test Tool)
Creates a Software Publisher's Certificate (SPC) from one or more X.509 certificates. This tool is for testing purposes only.

Certmgr.exe (Certificate Manager Tool)
Manages certificates, certificate trust lists (CTLs), and certificate revocation lists (CRLs).

Clrver.exe (CLR Version Tool)
reports all the installed versions of the common language runtime (CLR) on the computer.

CorFlags.exe (CorFlags Conversion Tool)
Lets you configure the CorFlags section of the header of a portable executable (PE) image.

Fuslogvw.exe (Assembly Binding Log Viewer)
Displays information about assembly binds to help you diagnose why the .NET Framework cannot locate an assembly at run time.

Gacutil.exe (Global Assembly Cache Tool)
Lets you view and manipulate the contents of the global assembly cache and download cache.

Ilasm.exe (IL Assembler)

Generates a portable executable (PE) file from intermediate language (IL). You can run the resulting executable to determine whether the IL performs as expected.

Ildasm.exe (IL Disassembler)

Takes a portable executable (PE) file that contains intermediate language (IL) code and creates a text file that can be input to the IL Assembler (Ilasm.exe).

Installutil.exe (Installer Tool)

Enables you to install and uninstall server resources by executing the installer components in a specified assembly. (Works with classes in the System.Configuration.Install namespace.)

Lc.exe (License Compiler)

Reads text files that contain licensing information and produces a .licenses file that can be embedded in a common language runtime executable as a resource.

Mage.exe (Manifest Generation and Editing Tool)

Lets you create, edit, and sign application and deployment manifests. As a command-line tool, Mage.exe can be run from both batch scripts and other Windows-based applications, including ASP.NET applications.

MageUI.exe (Manifest Generation and Editing Tool, Graphical Client)

Supports the same functionality as the command-line tool Mage.exe, but uses a Windows-based user interface (UI). Supports the same functionality as the command-line tool Mage.exe, but uses a Windows-based user interface (UI).

MDbg.exe (.NET Framework Command-Line Debugger)

Helps tools vendors and application developers find and fix bugs in programs that target the .NET Framework common language runtime. This tool uses the runtime debugging API to provide debugging services.

Mgmtclassgen.exe (Management Strongly Typed Class Generator)

Enables you to generate an early-bound managed class for a specified Windows Management Instrumentation (WMI) class.

Mpgo.exe (Managed Profile Guided Optimization Tool)

Enables you to tune native image assemblies using common end-user scenarios. Mpgo.exe allows the generation and consumption of profile data for native image application assemblies (not the .NET Framework assemblies) using training scenarios selected by the application developer.

Ngen.exe (Native Image Generator)

Improves the performance of managed applications through the use of native images (files containing compiled processor-specific machine code). The runtime can use native images from the cache instead of using the just-in-time (JIT) compiler to compile the original assembly.

Peverify.exe (PEVerify Tool)

Helps you verify whether your Microsoft intermediate language (MSIL) code and associated metadata meet type safety requirements. Helps you verify whether your Microsoft intermediate language (MSIL) code and associated metadata meet type safety requirements.

Regasm.exe (Assembly Registration Tool)

Reads the metadata within an assembly and adds the necessary entries to the registry. This enables COM clients to appear as .NET Framework classes.

Regsvcs.exe (.NET Services Installation Tool)

Loads and registers an assembly, generates and installs a type library into a specified COM+ version 1.0 application, and configures services that you have added programmatically to a class.

Resgen.exe (Resource File Generator)

Converts text (.txt or .restext) files and XML-based resource format (.resx) files to common language runtime

binary (.resources) files that can be embedded in a runtime binary executable or compiled into satellite assemblies.

SecAnnotate.exe (.NET Security Annotator Tool)
Identifies the SecurityCritical and SecuritySafeCritical portions of an assembly. Identifies the `SecurityCritical` and `SecuritySafeCritical` portions of an assembly.

SignTool.exe (Sign Tool)
Digitally signs files, verifies signatures in files, and time-stamps files.

Sn.exe (Strong Name Tool)
Helps create assemblies with strong names. This tool provides options for key management, signature generation, and signature verification.

SOS.dll (SOS Debugging Extension)
Helps you debug managed programs in the WinDbg.exe debugger and in Visual Studio by providing information about the internal common language runtime environment.

SqlMetal.exe (Code Generation Tool)
Generates code and mapping for the LINQ to SQL component of the .NET Framework.

Storeadm.exe (Isolated Storage Tool)
Manages isolated storage; provides options for listing the user's stores and deleting them.

Tlbexp.exe (Type Library Exporter)
Generates a type library that describes the types that are defined in a common language runtime assembly.

Tlbimp.exe (Type Library Importer)
Converts the type definitions found in a COM type library into equivalent definitions in a common language runtime assembly.

Winmdexp.exe (Windows Runtime Metadata Export Tool)
Exports a .NET Framework assembly that is compiled as a .winmdobj file into a Windows Runtime component, which is packaged as a .winmd file that contains both Windows Runtime metadata and implementation information.

Winres.exe (Windows Forms Resource Editor)
Helps you localize user interface (UI) resources (.resx or .resources files) that are used by Windows Forms. You can translate strings, and then size, move, and hide controls to accommodate the localized strings.

Related Sections

WPF Tools
Includes tools such as the isXPS Conformance tool (isXPS.exe) and performance profiling tools.

Windows Communication Foundation Tools
Includes tools that make it easier for you to create, deploy, and manage Windows Communication Foundation (WCF) applications.

Additional class libraries and APIs

10/18/2019 • 2 minutes to read • Edit Online

The .NET Framework is constantly evolving. To improve cross-platform development and introduce new functionality early, new features are released out of band (OOB). This topic lists the OOB projects that we provide documentation for.

In addition, some libraries target specific platforms or implementations of the .NET Framework. For example, the CodePagesEncodingProvider class makes code page encodings available to UWP apps developed using the .NET Framework. This topic lists these libraries as well.

OOB projects

PROJECT	DESCRIPTION
System.Collections.Immutable	Provides collections that are thread safe and guaranteed to never change their contents.
WinHttpHandler	Provides a message handler for HttpClient based on the WinHTTP interface of Windows.
System.Numerics	Provides a library of vector types that can take advantage of SIMD hardware-based acceleration.
System.Threading.Tasks.Dataflow	The TPL Dataflow Library provides dataflow components to help increase the robustness of concurrency-enabled applications.

Platform-specific libraries

PROJECT	DESCRIPTION
CodePagesEncodingProvider	Extends the EncodingProvider class to make code page encodings available to apps that target the Universal Windows Platform.

Private APIs

These APIs support the product infrastructure and are not intended/supported to be used directly from your code.

- Microsoft.SqlServer.Server.SmiOrderProperty.Item Property
- System.Exception.PrepForRemoting Method
- System.Data.SqlTypes.SqlChars.Stream Property
- System.Data.SqlTypes.SqlStreamChars Constructor
- System.Data.SqlTypes.SqlStreamChars.CanSeek Property
- System.Data.SqlTypes.SqlStreamChars.IsNull Property
- System.Data.SqlTypes.SqlStreamChars.Length Property
- System.Data.SqlTypes.SqlStreamChars.Close Method
- System.Data.SqlTypes.SqlStreamChars.Dispose Method

- System.Data.SqlTypes.SqlStreamChars.Flush Method
- System.Data.SqlTypes.SqlStreamChars.Read Method
- System.Data.SqlTypes.SqlStreamChars.Seek Method
- System.Data.SqlTypes.SqlStreamChars.SetLength Method
- System.Data.SqlTypes.SqlStreamChars.Write Method
- System.Net.Connection Class
- System.Net.Connection.m_WriteList Field
- System.Net.ConnectionGroup Class
- System.Net.ConnectionGroup.m_ConnectionList Field
- System.Net.CoreResponseData Class
- System.Net.CoreResponseData.m_ResponseHeaders Field
- System.Net.CoreResponseData.m_StatusCode Field
- System.Net.HttpWebRequest._AutoRedirects Field
- System.Net.HttpWebRequest._CoreResponse Field
- System.Net.HttpWebRequest._HttpResponse Field
- System.Net.ServicePoint.m_ConnectionGroupList Field
- System.Net.ServicePointManager.s_ServicePointTable Field
- System.Windows.Diagnostics.VisualDiagnostics.s_isDebuggerCheckDisabledForTestPurposes Field
- System.Windows.Forms.Design.DataMemberFieldEditor Class
- System.Windows.Forms.Design.DataMemberListEditor Class
- System.Xml.XmlReader.CreateSqlReader Method
- adodb.Connection Interface
- adodb.EventReason Enum
- adodb.EventStatus Enum
- stdole.DISPPARAMS Structure
- stdole.EXCEPINFO Structure
- stdole.IFont.Name Property
- stdole.IFontDisp Interface
- stdole.IPicture.Handle Property
- stdole.IPictureDisp.Handle Property
- stdole.StdFont Interface
- stdole.StdPicture Interface

See also

- The .NET Framework and Out-of-Band Releases