AppDomain.Load Method

Reference

Feedback

In this article

Definition

Namespace: System

Assembly: System.Runtime.dll

Loads an Assembly into this application domain.

Overloads

Expand table

Load(Byte[])	Loads the Assembly with a common object file format (COFF) based image containing an emitted Assembly.
Load(AssemblyName)	Loads an Assembly given its AssemblyName.
Load(String)	Loads an Assembly given its display name.
Load(Byte[], Byte[])	Loads the Assembly with a common object file format (COFF) based image containing an emitted Assembly. The raw bytes representing the symbols for the Assembly are also loaded.

Load(Byte[])

Source: AppDomain.cs

Loads the Assembly with a common object file format (COFF) based image containing an emitted Assembly.

```
public:
 System::Reflection::Assembly ^ Load(cli::array
<System::Byte> ^ rawAssembly);
```

```
public System.Reflection.Assembly Load (byte[]
rawAssembly);
```

```

member this.Load : byte[] ->
System.Reflection.Assembly
```

```

Public Function Load (rawAssembly As Byte()) As
Assembly
```

Parameters

`rawAssembly` Byte[]

An array of type `byte` that is a COFF-based image containing an emitted assembly.

Returns

Assembly

The loaded assembly.

Exceptions

ArgumentNullException

`rawAssembly` is `null`.

BadImageFormatException

`rawAssembly` is not a valid assembly for the currently loaded runtime.

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

FileLoadException

An assembly or module was loaded twice with two different evidences.

Examples

The following sample demonstrates the use of loading a raw assembly.

For this code example to run, you must provide the fully qualified assembly name. For information about how to obtain the fully qualified assembly name, see Assembly Names.

```
using namespace System;
using namespace System::IO;
using namespace System::Reflection;
using namespace System::Reflection::Emit;
void InstantiateMyType( AppDomain^ domain )
{
   try
   {

      // You must supply a valid fully qualified assembly
      domain->CreateInstance( "Assembly text name, Versic
   }
   catch ( Exception^ e )
   {
      Console::WriteLine( e->Message );
   }

}


// Loads the content of a file to a Byte array.
array<Byte>^ loadFile( String^ filename )
{
   FileStream^ fs = gcnew FileStream( filename,FileMode::
   array<Byte>^buffer = gcnew array<Byte>((int)fs->Length
   fs->Read( buffer, 0, buffer->Length );
   fs->Close();
   return buffer;
```

```
}


// Creates a dynamic assembly with symbol information
// and saves them to temp.dll and temp.pdb
void EmitAssembly( AppDomain^ domain )
{
   AssemblyName^ assemblyName = gcnew AssemblyName;
   assemblyName->Name = "MyAssembly";
   AssemblyBuilder^ assemblyBuilder = domain->DefineDynam
   ModuleBuilder^ moduleBuilder = assemblyBuilder->Define
   TypeBuilder^ typeBuilder = moduleBuilder->DefineType(
   ConstructorBuilder^ constructorBuilder = typeBuilder-:
   ILGenerator^ ilGenerator = constructorBuilder->GetILGe
   ilGenerator->EmitWriteLine( "MyType instantiated!" );
   ilGenerator->Emit( OpCodes::Ret );
   typeBuilder->CreateType();
   assemblyBuilder->Save( "temp.dll" );
}

ref class Resolver
{
public:
   static Assembly^ MyResolver( Object^ sender, ResolveEv
   {
      AppDomain^ domain = dynamic_cast<AppDomain^>(sender

      // Once the files are generated, this call is
      // actually no longer necessary.
      EmitAssembly( domain );
      array<Byte>^rawAssembly = loadFile( "temp.dll" );
      array<Byte>^rawSymbolStore = loadFile( "temp.pdb" )
      Assembly^ assembly = domain->Load( rawAssembly, raw
      return assembly;
   }

};

int main()
{
   AppDomain^ currentDomain = AppDomain::CurrentDomain;
   InstantiateMyType( currentDomain ); // Failed!
   currentDomain->AssemblyResolve += gcnew ResolveEventHa
   InstantiateMyType( currentDomain ); // OK!
}
```

```
using System;
using System.IO;
using System.Reflection;
using System.Reflection.Emit;

class LoadRawSnippet {
   public static void Main() {
      AppDomain currentDomain = AppDomain.CurrentDomain;

      InstantiateMyType(currentDomain);   // Failed!

      currentDomain.AssemblyResolve += new ResolveEventHa

      InstantiateMyType(currentDomain);   // OK!
   }

   static void InstantiateMyType(AppDomain domain) {
      try {
     // You must supply a valid fully qualified assembly
         domain.CreateInstance("Assembly text name, Versi
      } catch (Exception e) {
         Console.WriteLine(e.Message);
      }
   }

   // Loads the content of a file to a byte array.
   static byte[] loadFile(string filename) {
      FileStream fs = new FileStream(filename, FileMode.O
      byte[] buffer = new byte[(int) fs.Length];
      fs.Read(buffer, 0, buffer.Length);
      fs.Close();

      return buffer;
   }

   static Assembly MyResolver(object sender, ResolveEvent
      AppDomain domain = (AppDomain) sender;

      // Once the files are generated, this call is
      // actually no longer necessary.
      EmitAssembly(domain);

      byte[] rawAssembly = loadFile("temp.dll");
      byte[] rawSymbolStore = loadFile("temp.pdb");
      Assembly assembly = domain.Load(rawAssembly, rawSym
```

```
      return assembly;
   }

   // Creates a dynamic assembly with symbol information
   // and saves them to temp.dll and temp.pdb
   static void EmitAssembly(AppDomain domain) {
      AssemblyName assemblyName = new AssemblyName();
      assemblyName.Name = "MyAssembly";

      AssemblyBuilder assemblyBuilder = domain.DefineDyna
      ModuleBuilder moduleBuilder = assemblyBuilder.Defir
      TypeBuilder typeBuilder = moduleBuilder.DefineType(

      ConstructorBuilder constructorBuilder = typeBuilder
      ILGenerator ilGenerator = constructorBuilder.GetILG
      ilGenerator.EmitWriteLine("MyType instantiated!");
      ilGenerator.Emit(OpCodes.Ret);

      typeBuilder.CreateType();

      assemblyBuilder.Save("temp.dll");
   }
}
```

```
open System
open System.IO
open System.Reflection
open System.Reflection.Emit

let instantiateMyType (domain: AppDomain) =
    try
        // You must supply a valid fully qualified assemb
        domain.CreateInstance("Assembly text name, Versic
        |> ignore
    with e ->
        printfn $"{e.Message}"

// Loads the content of a file to a byte array.
let loadFile filename =
    use fs = new FileStream(filename, FileMode.Open)
    let buffer = Array.zeroCreate<byte> (int fs.Length)
    fs.Read(buffer, 0, buffer.Length) |> ignore
    fs.Close()
```

```
    buffer

// Creates a dynamic assembly with symbol information
// and saves them to temp.dll and temp.pdb
let emitAssembly (domain: AppDomain) =
    let assemblyName = AssemblyName()
    assemblyName.Name <- "MyAssembly"

    let assemblyBuilder = domain.DefineDynamicAssembly(as
    let moduleBuilder = assemblyBuilder.DefineDynamicModu
    let typeBuilder = moduleBuilder.DefineType("MyType",

    let constructorBuilder = typeBuilder.DefineConstructo
    let ilGenerator = constructorBuilder.GetILGenerator()
    ilGenerator.EmitWriteLine "MyType instantiated!"
    ilGenerator.Emit OpCodes.Ret

    typeBuilder.CreateType() |> ignore

    assemblyBuilder.Save "temp.dll"

let myResolver (sender: obj) (args: ResolveEventArgs) =
    let domain = sender :?> AppDomain

    // Once the files are generated, this call is
    // actually no longer necessary.
    emitAssembly domain

    let rawAssembly = loadFile "temp.dll"
    let rawSymbolStore = loadFile "temp.pdb"
    domain.Load(rawAssembly, rawSymbolStore)

let currentDomain = AppDomain.CurrentDomain

instantiateMyType currentDomain   // Failed!

currentDomain.add_AssemblyResolve (ResolveEventHandler my

instantiateMyType currentDomain   // OK!
```

```
Imports System.IO
Imports System.Reflection
Imports System.Reflection.Emit
```

```
Module Test

   Sub Main()
      Dim currentDomain As AppDomain = AppDomain.CurrentD

      InstantiateMyType(currentDomain)      ' Failed!

      AddHandler currentDomain.AssemblyResolve, AddressOf

      InstantiateMyType(currentDomain)      ' OK!
   End Sub


   Sub InstantiateMyType(domain As AppDomain)
      Try
     ' You must supply a valid fully qualified assembly n
        domain.CreateInstance("Assembly text name, Versi
      Catch e As Exception
        Console.WriteLine(e.Message)
      End Try
   End Sub


   ' Loads the content of a file to a byte array.
   Function loadFile(filename As String) As Byte()
      Dim fs As New FileStream(filename, FileMode.Open)
      Dim buffer(CInt(fs.Length - 1)) As Byte
      fs.Read(buffer, 0, buffer.Length)
      fs.Close()

      Return buffer
   End Function 'loadFile


   Function MyResolver(sender As Object, args As ResolveE
      Dim domain As AppDomain = DirectCast(sender, AppDom

      ' Once the files are generated, this call is
      ' actually no longer necessary.
      EmitAssembly(domain)

      Dim rawAssembly As Byte() = loadFile("temp.dll")
      Dim rawSymbolStore As Byte() = loadFile("temp.pdb")
      Dim myAssembly As System.Reflection.Assembly = doma

      Return myAssembly
   End Function 'MyResolver
```

```
    ' Creates a dynamic assembly with symbol information
    ' and saves them to temp.dll and temp.pdb
    Sub EmitAssembly(domain As AppDomain)
       Dim assemblyName As New AssemblyName()
       assemblyName.Name = "MyAssembly"

       Dim assemblyBuilder As AssemblyBuilder = domain.Def
       Dim moduleBuilder As ModuleBuilder = assemblyBuilde
       Dim typeBuilder As TypeBuilder = moduleBuilder.Defi

       Dim constructorBuilder As ConstructorBuilder = type
       Dim ilGenerator As ILGenerator = constructorBuilder
       ilGenerator.EmitWriteLine("MyType instantiated!")
       ilGenerator.Emit(OpCodes.Ret)

       typeBuilder.CreateType()

       assemblyBuilder.Save("temp.dll")
    End Sub

 End Module 'Test
```

Remarks

For information that is common to all overloads of this method, see the Load(AssemblyName) method overload.

Beginning with the .NET Framework 4, the trust level of an assembly that is loaded by using this method is the same as the trust level of the application domain.

Applies to

▼

Load(AssemblyName)

Source: AppDomain.cs

Loads an Assembly given its AssemblyName.

```
public:
 System::Reflection::Assembly ^
Load(System::Reflection::AssemblyName ^ assemblyRef);
```

```
public System.Reflection.Assembly Load
(System.Reflection.AssemblyName assemblyRef);
```

```
member this.Load : System.Reflection.AssemblyName ->
System.Reflection.Assembly
```

```
Public Function Load (assemblyRef As AssemblyName) As
Assembly
```

Parameters

`assemblyRef` AssemblyName

An object that describes the assembly to load.

Returns

Assembly

The loaded assembly.

Exceptions

ArgumentNullException

`assemblyRef` is `null`.

FileNotFoundException

`assemblyRef` is not found.

BadImageFormatException

`assemblyRef` is not a valid assembly for the currently loaded runtime.

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

FileLoadException

An assembly or module was loaded twice with two different evidences.

Remarks

This method should be used only to load an assembly into the current application domain. This method is provided as a convenience for interoperability callers who cannot call the static Assembly.Load method. To load assemblies into other application domains, use a method such as CreateInstanceAndUnwrap.

If a version of the requested assembly is already loaded, this method returns the loaded assembly, even if a different version is requested.

Supplying a partial assembly name for `assemblyRef` is not recommended. (A partial name omits one or more of culture, version, or public key token. For overloads that take a string instead of an AssemblyName object, "MyAssembly, Version=1.0.0.0" is an example of a partial name and "MyAssembly, Version=1.0.0.0, Culture=neutral, PublicKeyToken=18ab3442da84b47" is an example of a full name.) Using partial names has a negative effect on performance. In addition, a partial assembly name can load an assembly from the global assembly cache only if there is an exact

copy of the assembly in the application base directory (BaseDirectory or AppDomainSetup.ApplicationBase).

If the current AppDomain object represents application domain `A`, and the Load method is called from application domain `B`, the assembly is loaded into both application domains. For example, the following code loads `MyAssembly` into the new application domain `ChildDomain` and also into the application domain where the code executes:

```
AppDomain^ ad = AppDomain::CreateDomain("ChildDomain");
ad->Load("MyAssembly");
```

```
AppDomain ad = AppDomain.CreateDomain("ChildDomain");
ad.Load("MyAssembly");
```

```
let ad = AppDomain.CreateDomain "ChildDomain"
ad.Load "MyAssembly"
```

```
Dim ad As AppDomain  = AppDomain.CreateDomain("ChildDomai
ad.Load("MyAssembly")
```

The assembly is loaded into both domains because Assembly does not derive from MarshalByRefObject, and therefore the return value of the Load method cannot be marshaled. Instead, the common language runtime tries to load the assembly into the calling application domain. The assemblies that are loaded into the two application domains might be different if the path settings for the two application domains are different.

> **Note**
>
> If both the AssemblyName.Name property and the AssemblyName.CodeBase property are set, the first attempt to load the assembly uses the display name (including version, culture, and so on, as returned by the Assembly.FullName property). If the file is not found, the CodeBase property is used to search for the assembly. If the assembly is found using CodeBase, the display name is matched against the assembly. If the match fails, a FileLoadException is thrown.

Applies to

▼

Load(String)

Source: AppDomain.cs

Loads an Assembly given its display name.

```
public:
 System::Reflection::Assembly ^ Load(System::String ^
assemblyString);
```

```
public System.Reflection.Assembly Load (string
assemblyString);
```

```
member this.Load : string ->
System.Reflection.Assembly
```

```

Public Function Load (assemblyString As String) As
Assembly
```

Parameters

`assemblyString` String

The display name of the assembly. See FullName.

Returns

Assembly

The loaded assembly.

Exceptions

ArgumentNullException

`assemblyString` is `null`

FileNotFoundException

`assemblyString` is not found.

BadImageFormatException

`assemblyString` is not a valid assembly for the currently loaded runtime.

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

FileLoadException

An assembly or module was loaded twice with two different evidences.

Remarks

For information that is common to all overloads of this method, see the Load(AssemblyName) method overload.

Applies to

▼

Load(Byte[], Byte[])

Source: AppDomain.cs

Loads the Assembly with a common object file format (COFF) based image containing an emitted Assembly. The raw bytes representing the symbols for the Assembly are also loaded.

```
public:
 System::Reflection::Assembly ^ Load(cli::array
<System::Byte> ^ rawAssembly, cli::array
<System::Byte> ^ rawSymbolStore);
```

```
public System.Reflection.Assembly Load (byte[]
rawAssembly, byte[]? rawSymbolStore);
```

```
member this.Load : byte[] * byte[] ->
System.Reflection.Assembly
```

```
Public Function Load (rawAssembly As Byte(),
rawSymbolStore As Byte()) As Assembly
```

Parameters

`rawAssembly` Byte[]

An array of type `byte` that is a COFF-based image containing an emitted assembly.

`rawSymbolStore` Byte[]

An array of type `byte` containing the raw bytes representing the symbols for the assembly.

Returns

Assembly

The loaded assembly.

Exceptions

ArgumentNullException

`rawAssembly` is `null`.

BadImageFormatException

`rawAssembly` is not a valid assembly for the currently loaded runtime.

AppDomainUnloadedException

The operation is attempted on an unloaded application domain.

FileLoadException

An assembly or module was loaded twice with two different evidences.

Examples

The following sample demonstrates the use of loading a raw assembly.

For this code example to run, you must provide the fully qualified assembly name. For information about how to obtain the fully qualified assembly name, see Assembly Names.

```
using namespace System;
using namespace System::IO;
using namespace System::Reflection;
using namespace System::Reflection::Emit;
void InstantiateMyType( AppDomain^ domain )
{
   try
   {

      // You must supply a valid fully qualified assembly
      domain->CreateInstance( "Assembly text name, Versic
   }
   catch ( Exception^ e )
   {
      Console::WriteLine( e->Message );
   }

}


// Loads the content of a file to a Byte array.
array<Byte>^ loadFile( String^ filename )
{
   FileStream^ fs = gcnew FileStream( filename,FileMode::
   array<Byte>^buffer = gcnew array<Byte>((int)fs->Length
   fs->Read( buffer, 0, buffer->Length );
   fs->Close();
   return buffer;
}


// Creates a dynamic assembly with symbol information
// and saves them to temp.dll and temp.pdb
void EmitAssembly( AppDomain^ domain )
```

```
{
   AssemblyName^ assemblyName = gcnew AssemblyName;
   assemblyName->Name = "MyAssembly";
   AssemblyBuilder^ assemblyBuilder = domain->DefineDynam
   ModuleBuilder^ moduleBuilder = assemblyBuilder->Define
   TypeBuilder^ typeBuilder = moduleBuilder->DefineType(
   ConstructorBuilder^ constructorBuilder = typeBuilder->
   ILGenerator^ ilGenerator = constructorBuilder->GetILGe
   ilGenerator->EmitWriteLine( "MyType instantiated!" );
   ilGenerator->Emit( OpCodes::Ret );
   typeBuilder->CreateType();
   assemblyBuilder->Save( "temp.dll" );
}

ref class Resolver
{
public:
   static Assembly^ MyResolver( Object^ sender, ResolveEv
   {
      AppDomain^ domain = dynamic_cast<AppDomain^>(sender

      // Once the files are generated, this call is
      // actually no longer necessary.
      EmitAssembly( domain );
      array<Byte>^rawAssembly = loadFile( "temp.dll" );
      array<Byte>^rawSymbolStore = loadFile( "temp.pdb" )
      Assembly^ assembly = domain->Load( rawAssembly, raw
      return assembly;
   }

};

int main()
{
   AppDomain^ currentDomain = AppDomain::CurrentDomain;
   InstantiateMyType( currentDomain ); // Failed!
   currentDomain->AssemblyResolve += gcnew ResolveEventHa
   InstantiateMyType( currentDomain ); // OK!
}
```

```
using System;
using System.IO;
using System.Reflection;
using System.Reflection.Emit;
```

```
class LoadRawSnippet {
   public static void Main() {
      AppDomain currentDomain = AppDomain.CurrentDomain;

      InstantiateMyType(currentDomain);   // Failed!

      currentDomain.AssemblyResolve += new ResolveEventHa

      InstantiateMyType(currentDomain);   // OK!
   }

   static void InstantiateMyType(AppDomain domain) {
      try {
     // You must supply a valid fully qualified assembly
         domain.CreateInstance("Assembly text name, Versi
      } catch (Exception e) {
         Console.WriteLine(e.Message);
      }
   }

   // Loads the content of a file to a byte array.
   static byte[] loadFile(string filename) {
      FileStream fs = new FileStream(filename, FileMode.O
      byte[] buffer = new byte[(int) fs.Length];
      fs.Read(buffer, 0, buffer.Length);
      fs.Close();

      return buffer;
   }

   static Assembly MyResolver(object sender, ResolveEvent
      AppDomain domain = (AppDomain) sender;

      // Once the files are generated, this call is
      // actually no longer necessary.
      EmitAssembly(domain);

      byte[] rawAssembly = loadFile("temp.dll");
      byte[] rawSymbolStore = loadFile("temp.pdb");
      Assembly assembly = domain.Load(rawAssembly, rawSym

      return assembly;
   }

   // Creates a dynamic assembly with symbol information
   // and saves them to temp.dll and temp.pdb
   static void EmitAssembly(AppDomain domain) {
```

```
        AssemblyName assemblyName = new AssemblyName();
        assemblyName.Name = "MyAssembly";

        AssemblyBuilder assemblyBuilder = domain.DefineDyna
        ModuleBuilder moduleBuilder = assemblyBuilder.Defir
        TypeBuilder typeBuilder = moduleBuilder.DefineType(

        ConstructorBuilder constructorBuilder = typeBuilder
        ILGenerator ilGenerator = constructorBuilder.GetILG
        ilGenerator.EmitWriteLine("MyType instantiated!");
        ilGenerator.Emit(OpCodes.Ret);

        typeBuilder.CreateType();

        assemblyBuilder.Save("temp.dll");
    }
}
```

```
open System
open System.IO
open System.Reflection
open System.Reflection.Emit

let instantiateMyType (domain: AppDomain) =
    try
        // You must supply a valid fully qualified assemb
        domain.CreateInstance("Assembly text name, Versic
        |> ignore
    with e ->
        printfn $"{e.Message}"

// Loads the content of a file to a byte array.
let loadFile filename =
    use fs = new FileStream(filename, FileMode.Open)
    let buffer = Array.zeroCreate<byte> (int fs.Length)
    fs.Read(buffer, 0, buffer.Length) |> ignore
    fs.Close()
    buffer

// Creates a dynamic assembly with symbol information
// and saves them to temp.dll and temp.pdb
let emitAssembly (domain: AppDomain) =
    let assemblyName = AssemblyName()
    assemblyName.Name <- "MyAssembly"
```

```
    let assemblyBuilder = domain.DefineDynamicAssembly(as
    let moduleBuilder = assemblyBuilder.DefineDynamicModu
    let typeBuilder = moduleBuilder.DefineType("MyType",

    let constructorBuilder = typeBuilder.DefineConstructo
    let ilGenerator = constructorBuilder.GetILGenerator()
    ilGenerator.EmitWriteLine "MyType instantiated!"
    ilGenerator.Emit OpCodes.Ret

    typeBuilder.CreateType() |> ignore

    assemblyBuilder.Save "temp.dll"

let myResolver (sender: obj) (args: ResolveEventArgs) =
    let domain = sender :?> AppDomain

    // Once the files are generated, this call is
    // actually no longer necessary.
    emitAssembly domain

    let rawAssembly = loadFile "temp.dll"
    let rawSymbolStore = loadFile "temp.pdb"
    domain.Load(rawAssembly, rawSymbolStore)

let currentDomain = AppDomain.CurrentDomain

instantiateMyType currentDomain   // Failed!

currentDomain.add_AssemblyResolve (ResolveEventHandler my

instantiateMyType currentDomain   // OK!
```

```
Imports System.IO
Imports System.Reflection
Imports System.Reflection.Emit

Module Test

   Sub Main()
      Dim currentDomain As AppDomain = AppDomain.CurrentD

      InstantiateMyType(currentDomain)      ' Failed!
```

```
        AddHandler currentDomain.AssemblyResolve, AddressOf

        InstantiateMyType(currentDomain)      ' OK!
    End Sub


    Sub InstantiateMyType(domain As AppDomain)
        Try
       ' You must supply a valid fully qualified assembly r
           domain.CreateInstance("Assembly text name, Versi
        Catch e As Exception
           Console.WriteLine(e.Message)
        End Try
    End Sub


    ' Loads the content of a file to a byte array.
    Function loadFile(filename As String) As Byte()
        Dim fs As New FileStream(filename, FileMode.Open)
        Dim buffer(CInt(fs.Length - 1)) As Byte
        fs.Read(buffer, 0, buffer.Length)
        fs.Close()

        Return buffer
    End Function 'loadFile


    Function MyResolver(sender As Object, args As ResolveE
        Dim domain As AppDomain = DirectCast(sender, AppDom

        ' Once the files are generated, this call is
        ' actually no longer necessary.
        EmitAssembly(domain)

        Dim rawAssembly As Byte() = loadFile("temp.dll")
        Dim rawSymbolStore As Byte() = loadFile("temp.pdb")
        Dim myAssembly As System.Reflection.Assembly = doma

        Return myAssembly
    End Function 'MyResolver


    ' Creates a dynamic assembly with symbol information
    ' and saves them to temp.dll and temp.pdb
    Sub EmitAssembly(domain As AppDomain)
        Dim assemblyName As New AssemblyName()
        assemblyName.Name = "MyAssembly"
```

```
        Dim assemblyBuilder As AssemblyBuilder = domain.Def
        Dim moduleBuilder As ModuleBuilder = assemblyBuilde
        Dim typeBuilder As TypeBuilder = moduleBuilder.Defi

        Dim constructorBuilder As ConstructorBuilder = type
        Dim ilGenerator As ILGenerator = constructorBuilder
        ilGenerator.EmitWriteLine("MyType instantiated!")
        ilGenerator.Emit(OpCodes.Ret)

        typeBuilder.CreateType()

        assemblyBuilder.Save("temp.dll")
    End Sub

End Module 'Test
```

Remarks

For information that is common to all overloads of this method, see the Load(AssemblyName) method overload.

Beginning with the .NET Framework 4, the trust level of an assembly that is loaded by using this method is the same as the trust level of the application domain.

Applies to

▼

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more

.NET feedback

.NET is an open source project. Select a link to provide feedback:

Open a documentation issue

information, see our contributor guide.

Provide product feedback

English (United States) Your Privacy Choices Theme

Manage cookies Previous Versions Blog Contribute Privacy Terms of Use Trademarks