**.NET Framework Remoting Overview**

This page is specific to

**.NET Framework 3.0**

Other versions are also available for the following:

[Microsoft Visual Studio 2003/.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/kwdt6w2k(VS.71).aspx)

[Microsoft Visual Studio 2005/.NET Framework 2.0](http://msdn.microsoft.com/en-us/library/kwdt6w2k(VS.80).aspx)

**This topic is specific to a legacy technology that is retained for backward compatibility with existing applications and is not recommended for new development. Distributed applications should now be developed using the** [Windows Communication Foundation (WCF)](http://go.microsoft.com/fwlink/?LinkID=127777).

.NET remoting enables you to build widely distributed applications easily, whether the application components are all on one computer or spread out across the entire world. You can build client applications that use objects in other processes on the same computer or on any other computer that is reachable over its network. You can also use .NET remoting to communicate with other application domains in the same process. (For details about programming application domains, see [Programming with Application Domains](http://msdn.microsoft.com/en-us/library/yk22e11a(VS.85).aspx).)

.NET remoting provides an abstract approach to interprocess communication that separates the remotable object from a specific client or server application domain and from a specific mechanism of communication. As a result, it is flexible and easily customizable. You can replace one communication protocol with another, or one serialization format with another without recompiling the client or the server. In addition, the remoting system assumes no particular application model. You can communicate from a Web application, a console application, a Windows Service – from almost anything you want to use. Remoting servers can also be any type of application domain. Any application can host remoting objects and provide its services to any client on its computer or network.

|  |
| --- |
| **NoteNote** |
| For security reasons, it is strongly recommended to expose Remoting endpoints through secure channels. Never expose insecure Remoting endpoints to the Internet. |

To use .NET remoting to build an application in which two components communicate directly across an application domain boundary, you need to build only the following:

* A remotable object.
* A host application domain to listen for requests for that object.
* A client application domain that makes requests for that object.

Even in a complex, multiclient or multiserver application, .NET remoting can be thought of in this way. The host and the client application must also be configured with the remoting infrastructure and you must understand the lifetime and activation issues that the remoting infrastructure introduces.

**How to: Build a Remotable Type**

This page is specific to

**.NET Framework 3.0**

Other versions are also available for the following:

[Microsoft Visual Studio 2003/.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/txct33xt(VS.71).aspx)

[Microsoft Visual Studio 2005/.NET Framework 2.0](http://msdn.microsoft.com/en-us/library/txct33xt(VS.80).aspx)

**This topic is specific to a legacy technology that is retained for backward compatibility with existing applications and is not recommended for new development. Distributed applications should now be developed using the** [Windows Communication Foundation (WCF)](http://go.microsoft.com/fwlink/?LinkID=127777).

To enable objects in other application domains to use an instance of your class, your class must inherit from [MarshalByRefObject](http://msdn.microsoft.com/en-us/library/system.marshalbyrefobject(VS.85).aspx). The following procedure describes how to create a basic object that can be created and invoked from objects executing in another application domain.

**To build a remotable type**

* Define a class that derives from the **MarshalByRefObject** class.

**Example**

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

' RemotableType.vb

Imports System

Public Class RemotableType

Inherits MarshalByRefObject

Public Function SayHello() As String

Console.WriteLine("RemotableType.SayHello() was called!")

Return "Hello, world"

End Function

End Class

// RemotableType.cs

using System;

public class RemotableType : MarshalByRefObject

{

public string SayHello()

{

Console.WriteLine("RemotableType.SayHello() was called!");

return "Hello, world";

}

}

**How to: Build a Hosting Application**

This page is specific to

**.NET Framework 3.0**

Other versions are also available for the following:

[Microsoft Visual Studio 2005/.NET Framework 2.0](http://msdn.microsoft.com/en-us/library/ecc85927(VS.80).aspx)

**This topic is specific to a legacy technology that is retained for backward compatibility with existing applications and is not recommended for new development. Distributed applications should now be developed using the** [Windows Communication Foundation (WCF)](http://go.microsoft.com/fwlink/?LinkID=127777).

By itself, the RemotableType class defined in the [How to: Build a Remotable Type](http://msdn.microsoft.com/en-us/library/txct33xt(VS.85).aspx) topic is not special. To enable objects in other application domains to create instances of this object remotely, you must build a host or listener application to do the following things:

* Choose and register a channel, which is an object that handles the networking protocols and serialization formats on your behalf.
* Register your type with the .NET remoting system so that it can use your channel to listen for requests for your type.

The .NET Framework includes three default channels, [HttpChannel](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.channels.http.httpchannel(VS.85).aspx) (which uses SOAP formatting by default), [TcpChannel](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.channels.tcp.tcpchannel(VS.85).aspx) (which uses binary formatting by default), and [IpcChannel](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.channels.ipc.ipcchannel(VS.85).aspx) (which uses binary formatting by default). [HttpChannel](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.channels.http.httpchannel(VS.85).aspx) can be used through firewalls without opening a port and it supports standard security and authentication protocols. For more information about choosing channels that suit your scenario, see [Channels](http://msdn.microsoft.com/en-us/library/dkfd3wha(VS.85).aspx).

You can build listener applications using any type of application domain — a Windows Forms application, an ASP.NET Web application, a console application, a Windows Service (also known as a Windows NT Service), or any other managed application domain. Because remote configuration is done for each application domain, the application domain must be running to listen for requests.

|  |
| --- |
| **NoteNote** |
| Unlike COM, remoting does not start the host or server application for you. This is an important difference between .NET remoting and remote activation in COM. |

Configuration can be done programmatically or by using an application or machine configuration file.

The remoting system uses the information in this file to listen for and route remote requests to an instance of a remotable type. The file specifies the server-activation mode, the type name and assembly of the type on behalf of which it is to listen, and the object Uniform Resource Identifier (URI) or external name of the object. (For more information about object URIs and remoting, see [Activation URLs](http://msdn.microsoft.com/en-us/library/2ywf02bs(VS.85).aspx).)

|  |
| --- |
| **NoteNote** |
| Although there are only a few settings in the preceding configuration file, most of the problems using .NET remoting occur because some of these settings are either incorrect or do not match the configuration settings for client applications. It is easy to mistype a name, forget a port, or neglect an attribute. If you are having problems with your remoting application, check your configuration settings first. |

Using a configuration file enables you to change the remoting configuration without recompiling your executable, among other things. For information about the configuration of the .NET remoting infrastructure, see [Remoting Settings Schema](http://msdn.microsoft.com/en-us/library/z415cf9a(VS.85).aspx).

|  |
| --- |
| **NoteNote** |
| See [How to: Compile and Run a Basic Remoting Application](http://msdn.microsoft.com/en-us/library/685zyak4(VS.85).aspx) for complete instructions on how to build and run this sample. |

**To implement a simple host application domain that uses a configuration file**

1. Continuing on from the [How to: Build a Remotable Type](http://msdn.microsoft.com/en-us/library/txct33xt(VS.85).aspx), create another directory under remoting and call it host. Create a configuration file for the remote class. The host application must be able to load the configuration for the remote class, and therefore, the configuration file should be saved in the same directory as the host application's assembly, or it is not found and an exception is thrown. The following code shows a configuration file that specifies the remote object is a Singleton, its implementation is a class called RemotableType located in an assembly called RemotableType. Next, an **HttpChannel** is registered listening on port 8989. Save this file in the remoting\listener directory. The filename should follow the pattern of *app-name*.exe.config. In this case it is called listener.exe.config.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl16_code');" \o "Copy Code)

<configuration>

<system.runtime.remoting>

<application>

<service>

<wellknown

mode="Singleton"

type="RemotableType, RemotableType"

objectUri="RemotableType.rem"

/>

</service>

<channels>

<channel ref="http" port="8989"/>

</channels>

</application>

</system.runtime.remoting>

</configuration>

1. Create a new source file for your language of choice. At the top of the source file import the **System.Runtime.Remoting** namespace:

Visual Basic

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

Imports System

Imports System.Runtime.Remoting

C#

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

using System;

using System.Runtime.Remoting;

1. In the Main method, load the configuration file that configures the remote class, display a message letting the user know the host is running, and then wait for a key press. Save this file in the remoting\listener directory.

Visual Basic

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

Public Class Listener

Public Shared Sub Main()

RemotingConfiguration.Configure("Listener.exe.config", False)

Console.WriteLine("Listening for requests. Press enter to exit...")

Console.ReadLine()

End Sub

End Class

public class Listener

{

public static void Main()

{

RemotingConfiguration.Configure("Listener.exe.config", false);

Console.WriteLine("Listening for requests. Press enter to exit...");

Console.ReadLine();

}

}

1. Copy the RemotableType.dll generated in [How to: Build a Remotable Type](http://msdn.microsoft.com/en-us/library/txct33xt(VS.85).aspx) into the remoting\listener directory. The host application must reference this assembly. Compile this class into an executable by typing the following command:

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl21_code');" \o "Copy Code)

vbc /r:RemotableType.dll Listener.vb

csc /noconfig /r:RemotableType.dll Listener.cs

1. You now have an assembly called Listener.exe. Try running it now to see if the configuration succeeds. A security dialog may be displayed if a firewall is currently blocking the 8989 port. If so click the "Unblock" button to temporarily open the firewall on that port.
2. For information about how to use the remotable type, see [How to: Build a Client Application](http://msdn.microsoft.com/en-us/library/y6dc64f2(VS.85).aspx).

**Example**

Visual Basic

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

' Listener.vb

Public Class Listener

Public Shared Sub Main()

RemotingConfiguration.Configure("Listener.exe.config", False)

Console.WriteLine("Listening for requests. Press enter to exit...")

Console.ReadLine()

End Sub

End Class

// Listener.cs

using System;

using System.Runtime.Remoting;

public class Listener

{

public static void Main(string[] args)

{

RemotingConfiguration.Configure("Listener.exe.config", false);

Console.WriteLine("Listening for requests. Press enter to exit...");

Console.ReadLine();

}

}

**How to: Build a Client Application**

This page is specific to

**.NET Framework 3.0**

Other versions are also available for the following:

[Microsoft Visual Studio 2003/.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/y6dc64f2(VS.71).aspx)

[Microsoft Visual Studio 2005/.NET Framework 2.0](http://msdn.microsoft.com/en-us/library/y6dc64f2(VS.80).aspx)

**This topic is specific to a legacy technology that is retained for backward compatibility with existing applications and is not recommended for new development. Distributed applications should now be developed using the** [Windows Communication Foundation (WCF)](http://go.microsoft.com/fwlink/?LinkID=127777).

To build a client of the remote type defined in [How to: Build a Remotable Type](http://msdn.microsoft.com/en-us/library/txct33xt(VS.85).aspx) and hosted by the application created in [Building a Host Application](http://msdn.microsoft.com/en-us/library/8019cc4e(VS.85).aspx), your application must register itself as a client for that remote object and then invoke it as though it were within the client's application domain. The .NET remoting system intercepts your client calls, forwards them to the remote object, and returns the results to your client. The following procedure describes how to build a basic remoting client.

|  |
| --- |
| **NoteNote** |
| See [How to: Compile and Run a Basic Remoting Application](http://msdn.microsoft.com/en-us/library/685zyak4(VS.85).aspx) for complete instructions on how to build and run this sample. |

**To build a basic remoting client**

1. Continuing on from [How to: Build a Hosting Application](http://msdn.microsoft.com/en-us/library/ecc85927(VS.85).aspx), create a new directory under remoting called client. Create a configuration file for the client application as shown in the following code and save the file in the remoting\client directory. The file name should follow the pattern of *app-name*.exe.config. In this case, it is called client.exe.config. The following configuration file tells the remoting system that the type information for the RemotableType remote object can be found in the RemotableType assembly and the object is located at http://localhost:8989/RemotableType.rem.

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

<configuration>

<system.runtime.remoting>

<application>

<client>

<wellknown

type="RemotableType, RemotableType"

url="http://localhost:8989/RemotableType.rem"

/>

</client>

</application>

</system.runtime.remoting>

</configuration>

For details about the URL attribute in this configuration file, see [Activation URLs](http://msdn.microsoft.com/en-us/library/2ywf02bs(VS.85).aspx) . If you want to run this application over a network, you must replace localhost in the client configuration with the name of the remote computer.

|  |
| --- |
| **NoteNote** |
| Although there are only a few settings in the preceding configuration file, most of the problems using .NET remoting occur because some of these settings are either incorrect or do not match the configuration settings for client applications. It is easy to mistype a name, forget a port, or neglect an attribute. If you are having problems with your remoting application, check your configuration settings first. |

1. Create a new source file in the language of your choice. In the main method, call **RemotingConfiguration.Configure** passing in the name of the client configuration file (client.exe.config). Next, instantiate an instance of RemotableType and call its SayHello method. Save the client application as Client.cs or Client.vb in the remoting\client directory.

|  |
| --- |
| **NoteNote** |
| The client application should not be saved in the same directory as that of your Listener.exe application. If it is, you cannot be certain that you are receiving and making use of a remote reference, because assembly and type resolution can occur when applications are in the same directory. |

1. Visual Basic
2. [Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl12_code');" \o "Copy Code)
3. Public Shared Sub Main()
4. RemotingConfiguration.Configure("Client.exe.config")
5. Dim remoteObject As New RemotableType()
6. Console.WriteLine(remoteObject.SayHello())
7. End Sub 'Main
8. public static void Main(){
9. RemotingConfiguration.Configure("Client.exe.config");
10. RemotableType remoteObject = new RemotableType();
11. Console.WriteLine(remoteObject.SayHello());
12. }
13. Copy the RemotableType.dll assembly from remoting\Type into remoting\client.

|  |
| --- |
| **NoteNote** |
| A common question at this point is "How do I know that the remote object gets called if I am copying the assembly to the client?" This is exactly why we added the call to Console.WriteLine in the RemotableType.SayHello() method. If the remote object is being called the WriteLine occurs in the Listener process, if not the WriteLine occurs in the client process. |

1. Compile client application by typing the following command in the remoting\client directory:

Visual Basic

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

vbc /r:RemotableType.dll Client.vb

csc /noconfig /r:RemotableType.dll Client.cs

1. Open up two command prompts. In one, go to the remoting\listener directory and run listener.exe. In the other, go to the remoting\client directory and run client.exe. The client command-prompt should look like this:

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl15_code');" \o "Copy Code)

C:\tmp\Remoting\client>client

Hello, world

1. The listener command-prompt should look like this:

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl16_code');" \o "Copy Code)

C:\tmp\Remoting\listener>listener

Listening for requests. Press Enter to exit...

RemotableType.SayHello() was called!

1. You can see by the output of the listener that it received a call to RemotableType.SayHello().

**Example**

Visual Basic

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

' Client.vb

Imports System

Imports System.Runtime.Remoting

Public Class Client

Public Shared Sub Main()

RemotingConfiguration.Configure("Client.exe.config")

Dim remoteObject As New RemotableType()

Console.WriteLine(remoteObject.SayHello())

End Sub 'Main

End Class 'Client

// Client.cs

using System;

using System.Runtime.Remoting;

public class Client{

public static void Main(){

RemotingConfiguration.Configure("Client.exe.config");

RemotableType remoteObject = new RemotableType();

Console.WriteLine(remoteObject.SayHello());

}

}

**How to: Compile and Run a Basic Remoting Application**

This page is specific to

**.NET Framework 3.0**

Other versions are also available for the following:

[Microsoft Visual Studio 2003/.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/685zyak4(VS.71).aspx)

[Microsoft Visual Studio 2005/.NET Framework 2.0](http://msdn.microsoft.com/en-us/library/685zyak4(VS.80).aspx)

**This topic is specific to a legacy technology that is retained for backward compatibility with existing applications and is not recommended for new development. Distributed applications should now be developed using the** [Windows Communication Foundation (WCF)](http://go.microsoft.com/fwlink/?LinkID=127777).

The following procedure shows how to use the command-line tools that ship with the .NET Framework SDK to compile the basic remoting application built in the topics [How to: Build a Remotable Type](http://msdn.microsoft.com/en-us/library/txct33xt(VS.85).aspx), [Building a Host Application](http://msdn.microsoft.com/en-us/library/8019cc4e(VS.85).aspx), and [How to: Build a Client Application](http://msdn.microsoft.com/en-us/library/y6dc64f2(VS.85).aspx). The instructions in each of these topics had you create a remoting directory with subdirectories called type, client, and listener.

**To compile and run a basic remoting application**

1. At the command prompt in the remoting\type directory, type the following command:

Visual Basic

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

vbc /t:library RemotableType.vb

[C#]

csc /noconfig /t:library RemotableType.cs

1. Copy RemotableType.dll into the remoting\client and remoting\listener directories.
2. Copy listener.exe.config into the remoting\listener directory.
3. Copy client.exe.config into the remoting\client directory.
4. At the command-prompt in the remoting\listener directory, type the following command:

Visual Basic

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

vbc /r:RemotableType.dll Listener.vb

csc /noconfig /r:RemotableType.dll Listener.cs

1. At the command-prompt in the remoting\client directory, type the following command:

Visual Basic

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

vbc /r:RemotableType.dll client.vb

csc /noconfig /r:RemotableType.dll Client.cs

1. At the command prompt in the remoting\listener directory, type Listener.
2. When the Listener application is running, open a new command prompt in the remoting\client directory and type Client.