**Advanced Remoting**

This page is specific to

**Microsoft Visual Studio 2008/.NET Framework 3.5**

Other versions are also available for the following:

[Microsoft Visual Studio 2003/.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/xec2wbt4(VS.71).aspx)

[Microsoft Visual Studio 2005/.NET Framework 2.0](http://msdn.microsoft.com/en-us/library/xec2wbt4(VS.80).aspx)

[.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/xec2wbt4(VS.85).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).

Advanced remoting scenarios may require you to intercept a remote call at one or more points.

**In This Section**

[Sinks and Sink Chains](http://msdn.microsoft.com/en-us/library/tdzwhfy3.aspx)

Describes the sink chain and how to modify it to extend the functionality of the basic remoting system.

[Extending RealProxy](http://msdn.microsoft.com/en-us/library/scx1w94y.aspx)

Describes how to extend the [RealProxy](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.proxies.realproxy.aspx) class to intercept calls at the moment of invocation.

[Advanced Design Issues](http://msdn.microsoft.com/en-us/library/y1611dh0.aspx)

Describes issues related to deploying a remote type library on a client.

**Sinks and Sink Chains**

This page is specific to

**Microsoft Visual Studio 2008/.NET Framework 3.5**

Other versions are also available for the following:

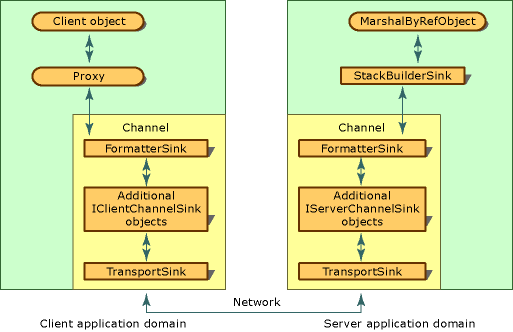
[Microsoft Visual Studio 2003/.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/tdzwhfy3(VS.71).aspx)

[Microsoft Visual Studio 2005/.NET Framework 2.0](http://msdn.microsoft.com/en-us/library/tdzwhfy3(VS.80).aspx)

[.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/tdzwhfy3(VS.85).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).

Clients make method calls on remote objects by sending messages to the remote application domain. This is accomplished by a set of channel objects. The client application domain contains a client channel and the remote application domain contains a remote channel. Each channel is composed of a series of channel sinks that are linked together in a chain. The following illustration shows the structure of a basic channel sink chain.



Channels send each message along a chain of channel sink objects prior to sending or after receiving a message. This sink chain contains sinks required for basic channel functionality, such as formatter, transport, or stackbuilder sinks, but you can customize the channel sink chain to perform special tasks with a message or a stream. Each channel sink implements either [IClientChannelSink](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.channels.iclientchannelsink.aspx) or [IServerChannelSinkfrlrfSystemRuntimeRemotingChannelsIServerChannelSinkClassTopic](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.channels.iserverchannelsink.aspx). The first channel sink on the client must also implement [IMessageSinkfrlrfSystemRuntimeRemotingMessagingIMessageSinkClassTopic](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.messaging.imessagesink.aspx). It typically implements [IClientFormatterSink](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.channels.iclientformattersink.aspx) (which inherits from both **IMessageSink**, [IChannelSinkBase](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.channels.ichannelsinkbase.aspx), and **IClientChannelSink**) and is called a formatter sink because it transforms the incoming message into a stream (an [IMessagefrlrfSystemRuntimeRemotingMessagingIMessageClassTopic](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.messaging.imessage.aspx) object).

The channel sink chain processes any message that is sent to or from an application domain. A channel sink has access to the message being processed and subsequent processing uses the message that is returned to the system after processing. This is a natural place to implement a logging service or any sort of filter.

Each channel sink processes the stream and then passes the stream to the next channel sink, which means that sinks before or after a specific sink should know what to do with the stream passed to them.

|  |
| --- |
| **tdzwhfy3.note(en-us,VS.90).gifNote:** |
| Message sinks must not throw exceptions. One way a message sink can control this is by wrapping method code in try-catch blocks. |

Channel sink providers (objects that implement the [IClientChannelSinkProviderfrlrfSystemRuntimeRemotingChannelsIClientChannelSinkProviderClassTopic](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.channels.iclientchannelsinkprovider.aspx), [IClientFormatterSinkProviderfrlrfSystemRuntimeRemotingChannelsIClientFormatterSinkProviderClassTopic](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.channels.iclientformattersinkprovider.aspx), or [IServerChannelSinkProvider](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.channels.iserverchannelsinkprovider.aspx) interface) are responsible for creating the channel sinks that process .NET remoting messages. When a remote type is activated, the channel sink provider is retrieved from the channel and the **CreateSink** method is called on the sink provider to retrieve the first channel sink from the chain.

Channel sinks are responsible for transporting messages between the client and the server. Channel sinks are also linked together in a chain. When the **CreateSink** method is called on a sink provider, it should do the following:

* Create a channel sink.
* Call **CreateSink** on the next sink provider in the chain.
* Ensure that the next sink and the current one are linked together.
* Return its sink to the caller.

Channel sinks are responsible for forwarding all calls made on them to the next sink in the chain and should provide a mechanism for storing a reference to the next sink.

Channel sinks have great flexibility in what they send down the sink chain. For example, security sinks that negotiate authentication before sending the actual serialized original message can hold onto the complete channel message, replace the content stream with their own content, and send it down the sink chain and on to the remote application domain. On the return journey, the security sink can intercept the reply message, creating a conversation with the corresponding security sinks in the remote application domain. Once an agreement is reached, the originating security sink can send the original content stream on to the remote application domain.

## Message Processing in the Channel Sink Chain

Once the .NET remoting system locates a channel that can process the message, the channel passes the message to the formatter channel sink by calling

[SyncProcessMessage](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.messaging.imessagesink.syncprocessmessage.aspx) (or [AsyncProcessMessage](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.messaging.imessagesink.asyncprocessmessage.aspx)). The formatter sink creates the transport header array and calls [GetRequestStream](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.channels.iclientchannelsink.getrequeststream.aspx) on the next sink. This call is forwarded down the sink chain, and any sink can create a request stream that is passed back to the formatter sink. If **GetRequestStream** returns a **null** reference (**Nothing** in Visual Basic), the formatter sink creates its own sink to use for serialization. Once this call returns, the message is serialized and the appropriate message processing method is called on the first channel sink in the sink chain.

Sinks cannot write data into the stream but can read from the stream or pass a new stream along where required. Sinks can also add headers to the header array (if they have not previously called **GetRequestStream** on the next sink) and add themselves to the sink stack before forwarding the call to the next sink. (The sync stack is used to allow asynchronous calls to call back to the caller when they are completed.) When the call reaches the transport sink at the end of the chain, the transport sink sends the headers and serialized message over the channel to the server where the entire process is reversed. The transport sink (on the server) retrieves the headers and serialized message from the server side of the stream and forwards these through the sink chain until the formatter sink is reached. The formatter sink deserializes the message and forwards it to the .NET remoting system where the message is turned back into a method call and is invoked on the server object.

## Creating Channel Sink Chains

To create a new channel sink, you must implement and configure the .NET remoting system to recognize an **IServerChannelSinkProvider** or **IClientChannelSinkProvider** implementation, which can create your custom **IClientChannelSink** or **IServerChannelSink** implementation or retrieve the next sink in the chain. You can use the [BaseChannelSinkWithProperties](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.channels.basechannelsinkwithproperties.aspx) abstract class to help implement your custom channel sinks.

### Building a Channel Sink Provider

Applications can provide server or client channel sink providers as parameters when constructing a channel. Channel sink providers should be stored in a chain and it is the responsibility of the developer to chain all channel sink providers together before passing the outer one to the channel constructor. The channel sink provider implements a **Next** property for this purpose. The following code example illustrates how to build a client channel sink provider. A complete example is available at [Remoting Example: Channel Sink Provider](http://msdn.microsoft.com/en-us/library/2ckwbt8a.aspx).

Visual Basic

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

private Function CreateDefaultClientProviderChain() As IClientChannelSinkProvider

Dim chain As New FirstClientFormatterSinkProvider

Dim sink As IClientChannelSinkProvider

sink = chain

sink.Next = New SecondClientFormatterSinkProvider

sink = sink.Next

return chain

End Function

C#

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

private IClientChannelSinkProvider CreateDefaultClientProviderChain(){

IClientChannelSinkProvider chain = new FirstClientFormatterSinkProvider();

IClientChannelSinkProvider sink = chain;

sink.Next = new SecondClientFormatterSinkProvider();

sink = sink.Next;

return chain;

}

|  |
| --- |
| **tdzwhfy3.note(en-us,VS.90).gifNote:** |
| When multiple channel sink providers are provided in a configuration file, the .NET remoting system chains them together in the order in which they are found in the configuration file. The channel sink providers are created when the channel is created during the [Configure](http://msdn.microsoft.com/en-us/library/ms223161.aspx) call. |

### Formatter Sinks

Formatter sinks serialize the channel message into the message stream as an object that implements **IMessage**. Some formatter sink implementations use the system-provided formatter types ([BinaryFormatterfrlrfSystemRuntimeSerializationFormattersBinaryBinaryFormatterClassTopic](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.formatters.binary.binaryformatter.aspx) and [SoapFormatterfrlrfSystemRuntimeSerializationFormattersSoapSoapFormatterClassTopic](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.formatters.soap.soapformatter.aspx)). Other implementations can use their own means to transform the channel message into the stream.

The function of the formatter sink is to generate the necessary headers and serialize the message to the stream. After the formatter sink, the message is forwarded to all sinks in the sink chain through the **SyncProcessMessage** or **AsyncProcessMessage** calls. At this stage the message has already been serialized and cannot be modified.

|  |
| --- |
| **tdzwhfy3.note(en-us,VS.90).gifNote:** |
| Sinks that must create or modify the message itself must be placed in the sink chain prior to the formatter. This is easily achieved by implementing **IClientFormatterSink**, thereby telling the system that it has a reference to the formatter sink. The real formatter sink can then be placed later in the sink chain. |

On the return journey, the formatter sink transforms the message stream back into the channel message object (return message). The first sink on the client must implement the **IClientFormatterSink** interface. When [CreateSink](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.channels.iclientchannelsinkprovider.createsink.aspx)returns to the channel, the reference returned is cast to an **IClientFormatterSink** type so the **SyncProcessMessage** method can be called. Remember **IClientFormatterSink** is derived from **IMessageSink**. If the cast fails, the system raises an exception.

### Custom Channel Sinks

On the client, custom channel sinks are inserted into the chain of objects between the formatter sink and the last transport sink. Inserting a custom channel sink in the client or server channel enables you to process the **IMessage** at one of the following points:

* During the process by which a call represented as a message is converted into a stream and sent over the wire.
* During the process by which a stream is taken off the wire and sent to the last message sink before the remote object on the server or the proxy object (on the client).

Custom sinks can read or write data (depending if the call is outgoing or incoming) to the stream and add additional information to the headers where desired. At this stage, the message has already been serialized by the formatter and cannot be modified. When the message call is forwarded to the transport sink at the end of the chain, the transport sink writes the headers to the stream and forwards the stream to the transport sink on the server using the transport protocol dictated by the channel.

### Transport Sinks

The transport sink is the last sink in the chain on the client, and the first sink in the chain on the server. Besides transporting the serialized message, the transport sink is also responsible for sending the headers to the server and retrieving the headers and the stream when the call returns from the server. These sinks are built into the channel and cannot be extended.

## Replacing the Default Formatter

Because a channel is an abstract networking mechanism, you can configure the .NET remoting system to combine a system-implemented channel with any formatter you choose. You can do this using the channel constructor that takes an [IDictionary](http://msdn.microsoft.com/en-us/library/system.collections.idictionary.aspx) implementation of channel properties, a formatter on the server, and a formatter on the client. You can also specify the formatter in a configuration file. The following example instructs the .NET remoting configuration system to create an [HttpChannel](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.channels.http.httpchannel.aspx) but use the [BinaryClientFormatterSink](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.channels.binaryclientformattersink.aspx) on the client.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl36_code');" \o "Copy Code)

<configuration>

<system.runtime.remoting>

<application>

<channels>

<channel ref="http">

<clientProviders>

<formatter ref="binary"/>

</clientProviders>

<channels>

</application>

</system.runtime.remoting>

</configuration>

The following code does the same thing programmatically, assuming a remote interface type IServicethat implements GetServerString and GetServerTime:

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl37_code');" \o "Copy Code)

Imports System

Imports System.Collections

Imports System.Runtime.Remoting

Imports System.Runtime.Remoting.Channels

Imports System.Runtime.Remoting.Channels.Http

Public Class ClientProcess

<MTAThread()> \_

Public Shared Sub Main()

' Note that any name/value pairs of configuration attributes can be

' placed in this dictionary (the configuration system calls this same

' constructor).

Dim properties As New Hashtable()

properties("name") = "HttpBinary"

ChannelServices.RegisterChannel(New HttpChannel(properties, New BinaryClientFormatterSinkProvider(), Nothing))

' The last parameter above (Nothing) is the server sink provider chain

' to obtain the default behavior (which includes SOAP and

' binary formatters on the server side).

Dim service As IService = CType(Activator.GetObject(GetType(IService), "http://computer:8080/SAService"), IService)

Console.WriteLine("Server string is: " + service.GetServerString())

Console.WriteLine("Server time is: " + service.GetServerTime())

End Sub

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl38_code');" \o "Copy Code)

using System;

using System.Collections;

using System.Runtime.Remoting;

using System.Runtime.Remoting.Channels;

using System.Runtime.Remoting.Channels.Http;

public class ClientProcess{

public static void Main(string[] Args){

// Note that any name/value pairs of configuration attributes can be

// placed in this dictionary (the configuration system calls this

// same HttpChannel constructor).

IDictionary properties = new Hashtable();

properties["name"] = "HttpBinary";

// The last parameter below is the server sink provider chain

// to obtain the default behavior (which includes SOAP and binary

// formatters) on the server side.

ChannelServices.RegisterChannel(new HttpChannel(properties, new BinaryClientFormatterSinkProvider(), null));

IService service = (IService)Activator.GetObject(typeof(IService),"http://computer:8080/SAService");

Console.WriteLine("Server string is: " + service.GetServerString());

Console.WriteLine("Server time is: " + service.GetServerTime());

}

}

For a complete example of this channel and formatter combination hosted in Internet Information Services (IIS), see [Remoting Example: Hosting in Internet Information Services (IIS)](http://msdn.microsoft.com/en-us/library/c2swb8ah.aspx).

To change this client to use a [TcpChannel](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.channels.tcp.tcpchannel.aspx) object with the [SoapClientFormatterSinkfrlrfSystemRuntimeRemotingChannelsSoapClientFormatterSinkClassTopic](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.channels.soapclientformattersink.aspx) object, you must change only the namespaces and the [RegisterChannel](http://msdn.microsoft.com/en-us/library/2eeyt4ba.aspx)call, as shown in the following code:

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl44_code');" \o "Copy Code)

ChannelServices.RegisterChannel(New TcpChannel(properties, New SoapClientFormatterSinkProvider(), Nothing))

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl45_code');" \o "Copy Code)

ChannelServices.RegisterChannel(new TcpChannel(properties, new SoapClientFormatterSinkProvider(), null));

**Extending RealProxy**

This page is specific to

**Microsoft Visual Studio 2008/.NET Framework 3.5**

Other versions are also available for the following:

[Microsoft Visual Studio 2003/.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/scx1w94y(VS.71).aspx)

[Microsoft Visual Studio 2005/.NET Framework 2.0](http://msdn.microsoft.com/en-us/library/scx1w94y(VS.80).aspx)

[.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/scx1w94y(VS.85).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).

Extending the proxy can be useful when you want to participate in the proxy creation process, marshal remote method calls, or determine object identity, among other things. You can build your own proxy using the extensible [RealProxy](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.proxies.realproxy.aspx) class. To build a custom proxy involves deriving a class from the [RealProxy](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.proxies.realproxy.aspx)class and overriding the [RealProxy.Invoke](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.proxies.realproxy.invoke.aspx) method. The **new** (**New** in Visual Basic) statement can be intercepted by deriving from the [ProxyAttribute](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.proxies.proxyattribute.aspx) attribute and applying the attribute to a child of [ContextBoundObject](http://msdn.microsoft.com/en-us/library/system.contextboundobject.aspx). (Applying the proxy attribute to a child of [MarshalByRefObject](http://msdn.microsoft.com/en-us/library/system.marshalbyrefobject.aspx) is not supported.) When **new** is called, the derived [ProxyAttribute](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.proxies.proxyattribute.aspx) creates an instance of the custom proxy. An application can also create a custom proxy instance directly.

When the application code calls a method on a custom proxy, the overridden [RealProxy.Invoke](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.proxies.realproxy.invoke.aspx) method is called with an object that implements [IMessage](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.messaging.imessage.aspx). The [IMessage](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.messaging.imessage.aspx)implementation provides an [IDictionary](http://msdn.microsoft.com/en-us/library/system.collections.idictionary.aspx) implementation that provides name/value pairs of information about the method. For more information about particular entries in the dictionary, see the reference documentation for the [IMessage](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.messaging.imessage.aspx) interface and related interfaces ([IMethodCallMessage](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.messaging.imethodcallmessage.aspx) and [IMethodReturnMessage](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.messaging.imethodreturnmessage.aspx)).

A real object can be called from the overridden **Invoke** method by calling [RemotingServices.ExecuteMessage](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.remotingservices.executemessage.aspx), which dispatches the call to the real object.

You can also use the [EnterpriseServicesHelper.CreateConstructionReturnMessage](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.services.enterpriseserviceshelper.createconstructionreturnmessage.aspx) method to process an [IConstructionCallMessage](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.activation.iconstructioncallmessage.aspx) object and generate an [IConstructionReturnMessage](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.activation.iconstructionreturnmessage.aspx) object. You can also use the [RealProxy.InitializeServerObject](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.proxies.realproxy.initializeserverobject.aspx) method to create the backing object (the object represented by the proxy).

When using the derived [ProxyAttribute](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.proxies.proxyattribute.aspx), you can create a real object in the overridden **ProxyAttribute.CreateInstance** method and store it as a field in the custom proxy. The custom proxy can marshal the real object to obtain the [ObjRef](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.objref.aspx) object that contains the Uniform Resource Identifier (URI). The URI must be stored in the proxy, because the "\_\_Uri" entry in the collection returned by the [IMessage.Properties](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.messaging.imessage.properties.aspx) property must be set in the [IMessage](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.messaging.imessage.aspx) implementation to dispatch a call to a real object.

Of course, the message does not have to be dispatched on a real object; you can perform some task with it in **Invoke** and generate and return an object that implements the [IMethodReturnMessage](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.messaging.imethodreturnmessage.aspx)) interface.

To participate in marshaling, override [RealProxy.CreateObjRef](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.proxies.realproxy.createobjref.aspx) and provide a custom [ObjRef](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.objref.aspx)that extends [ObjRef](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.objref.aspx). If you want to add custom data to the custom [ObjRef](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.objref.aspx), override[ObjRef.GetObjectData](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.objref.getobjectdata.aspx). You add your custom data and delegate to the [ObjRef.GetObjectData](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.objref.getobjectdata.aspx) method to modify the object identity capabilities of the .NET remoting marshaling system.

During deserialization, the remoting system calls your overridden [GetRealObject](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.iobjectreference.getrealobject.aspx) on the custom [ObjRef](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.objref.aspx). Here, you should delegate to the base [GetRealObject](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.objref.getrealobject.aspx) method because the base handles the object identity and sets up remoting channels. The base also calls your overridden [ProxyAttribute.CreateProxy](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.proxies.proxyattribute.createproxy.aspx) method to allow you to set up your custom proxy.

[ObjRef.IsFromThisAppDomain](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.objref.isfromthisappdomain.aspx) and [ObjRef.IsFromThisProcess](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.objref.isfromthisprocess.aspx) can be used to determine how you want to unmarshal.

Note that if you do not provide a custom [ObjRef](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.objref.aspx) during marshaling, the remoting system automatically marshals and unmarshals the object and the custom proxy is not used in the caller's application domain.

**Advanced Design Issues**

This page is specific to

**Microsoft Visual Studio 2008/.NET Framework 3.5**

Other versions are also available for the following:

[Microsoft Visual Studio 2003/.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/y1611dh0(VS.71).aspx)

[Microsoft Visual Studio 2005/.NET Framework 2.0](http://msdn.microsoft.com/en-us/library/y1611dh0(VS.80).aspx)

[.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/y1611dh0(VS.85).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).

Because the type information of a remote type is required by the client, it is often easiest to deploy the remote type assembly on the client. However, in many cases you do not want the client to have access to the implementation of the type. There are a number of ways to solve this problem:

* Declare an interface in an assembly that is shared by the server and the client.

|  |
| --- |
| **Note:** |
| The client must call [GetObject](http://msdn.microsoft.com/en-us/library/0315hz94.aspx) to instantiate the proxy. Using the operator **new** causes a compilation error because you cannot create an instance of an interface. |

* Generate a metadata assembly using the Soapsuds tool. For more information, see [Using Soapsuds.exe with Remoting](http://msdn.microsoft.com/en-us/library/xse48s01.aspx).
* Generate source code using the Soapsuds tool. For more information, see [Using Soapsuds.exe with Remoting](http://msdn.microsoft.com/en-us/library/xse48s01.aspx).

**Remoting Example: Channel Sink Provider**

This page is specific to

**Microsoft Visual Studio 2008/.NET Framework 3.5**

Other versions are also available for the following:

[Microsoft Visual Studio 2003/.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/2ckwbt8a(VS.71).aspx)

[Microsoft Visual Studio 2005/.NET Framework 2.0](http://msdn.microsoft.com/en-us/library/2ckwbt8a(VS.80).aspx)

[.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/2ckwbt8a(VS.85).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 code example builds a ChannelSinkPropertySetterProvider, which is a channel sink provider that can read values in the client application's configuration file and walk the sink stack looking for channel sinks that contain the properties it found in that file. This is useful to build a sink chain using the configuration file rather than creating the chain programmatically and passing [IDictionary](http://msdn.microsoft.com/en-us/library/system.collections.idictionary.aspx) objects to the constructors.

This provider does not insert any sink into the channel sink chain. It merely walks the sink chain looking for a set of properties and can easily be customized to take only those properties supported by a specific sink.

|  |
| --- |
| **Caution:** |
| .NET Remoting does not perform authentication or encryption by default. Therefore, it is recommended that you take all necessary steps to verify the identity of clients or servers before interacting with them remotely. Because .NET Remoting applications require **FullTrust** permissions to execute, if a unauthorized client were granted access on your server, the client could execute code as though it were fully-trusted. Always authenticate your endpoints and encrypt the communication streams, either by hosting your remoted types in Internet Information Services (IIS) or by building a custom channel sink pair to do this work. |
| **Note:** |
| This sample uses hard-coded passwords. It is never a good idea to hard-code passwords and is done here for illustrative purposes only. The best way to get a password from a user is to prompt the user. |

To compile and run this sample

1. Type the following commands at the command prompt:

Visual Basic

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

vbc /t:library /r:System.Runtime.Remoting.dll /out:PropSink.dll ChannelSinkPropertySetterProvider.vb

vbc /t:library /r:System.Runtime.Remoting.dll serviceclass.vb

vbc /r:System.Runtime.Remoting.dll /r:ServiceClass.dll /r:PropsSink.dll client.vb

vbc /r:System.Runtime.Remoting.dll /r:ServiceClass.dll server.vb

C#

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

csc /t:library /r:System.Runtime.Remoting.dll /out:PropSink.dll ChannelSinkPropertySetterProvider.cs

csc /t:library /r:System.Runtime.Remoting.dll serviceclass.cs

csc /r:System.Runtime.Remoting.dll /r:ServiceClass.dll /r:PropSink.dll client.cs

csc /r:System.Runtime.Remoting.dll /r:ServiceClass.dll server.cs

1. Open two command prompts pointing to the same directory. In one, type **server**. When that is running, type **client** at the other command prompt.

This channel sink provider supports the **writeToConsole** attribute to indicate whether you want a console dump of the provider's activities in the client console. For this example, the attribute is set to **true**.

This application runs on a single computer or across a network. To run this application over a network, replace **"localhost"** in the client configuration with the name of the remote computer.

**ChannelSinkPropertySetterProvider**

Visual Basic

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

Imports System

Imports System.Collections

Imports System.IO

Imports System.Runtime.Remoting

Imports System.Runtime.Remoting.Channels

'This class implements a client-side channel sink provider that

'walks the channel sink chain, looking for channel sink

'properties that match those specified in the configuration file. If it

'finds them, the provider sets them to the values specified in the

'configuration file. This is a simple helper provider that returns no

'channel itself. Instead, it merely returns the next channel sink it can find, or null.

Public Class ChannelSinkPropertySetterProvider

Implements IClientChannelSinkProvider

Private \_next As IClientChannelSinkProvider = Nothing

Private \_channelSinkProperties As IDictionary = Nothing

Private \_providerData As ICollection = Nothing

' Set the writeToConsole attribute on this provider element in the

' configuration file to "true"; otherwise, information is not written to the console.

Private \_consoleDump As Boolean = False

'Default constructor.

Public Sub New()

Console.WriteLine("Default constructor called.")

End Sub

'Constructor with properties. If writeToConsole attribute is "true",

'this constructor will dump all custom configuration properties set

'in the configuration file.

Public Sub New(ByVal properties As IDictionary, ByVal providerData As ICollection)

\_channelSinkProperties = properties

' Sets the private console dump property for this provider.

If (properties.Contains("writeToConsole")) Then

\_consoleDump = Boolean.Parse(properties.Item("writeToConsole").ToString)

End If

\_providerData = providerData

If (\_consoleDump = True) Then

Console.WriteLine("ChannelSinkPropertySetterProvider custom constructor called.")

Dim sinkData As SinkProviderData

For Each sinkData In providerData

Console.WriteLine("SinkProvider element: " & sinkData.Name)

Dim prop As DictionaryEntry

For Each prop In sinkData.Properties

Console.WriteLine("Prop name: " & prop.Key.ToString() & " value: " & prop.Value.ToString())

Next

Dim child As Object

For Each child In sinkData.Children

Console.WriteLine("Child: " & child.GetType().Name)

Next

Next

Dim entry As DictionaryEntry

For Each entry In properties

Console.WriteLine("channel sink properties: " & entry.Key.ToString() & ", " & entry.Value.ToString())

Next

Console.WriteLine()

End If

End Sub

'Called by the channel. Normally, this method takes any other sinks

'created by other providers in the chain, links them together, and

'then returns its own sink to the channel. In this case, this

'provider merely sets matching properties on each channel sink in the

'chain, and then returns the \*\*next\*\* channel sink to the channel or

'returns null, indicating to the channel that it is the end of the

'custom channel sink chain.

Public Function CreateSink(ByVal channel As System.Runtime.Remoting.Channels.IChannelSender, ByVal url As String, ByVal remoteChannelData As Object) As System.Runtime.Remoting.Channels.IClientChannelSink Implements System.Runtime.Remoting.Channels.IClientChannelSinkProvider.CreateSink

If (\_consoleDump) Then

Console.WriteLine("CreateSink is called.")

Console.WriteLine("By " & channel.ChannelName)

End If

Dim nextSink As IClientChannelSink = Nothing

If (\_next IsNot Nothing) Then

nextSink = \_next.CreateSink(channel, url, remoteChannelData)

If (nextSink Is Nothing) Then

If (\_consoleDump) Then

Console.WriteLine("Next sink is null!")

Return Nothing

End If

End If

WalkSinkChain(nextSink)

End If

Return nextSink

End Function

' This call walks the sink chain, setting properties as it goes.

' The channelSinkProperties are the SinkProviderData dictionaries

' that contain the name of the subnode in the configuration file, and

' a dictionary entry of attribute/value entries on that element.

Private Sub WalkSinkChain(ByVal thisSink As IClientChannelSink)

If (thisSink Is Nothing) Then

Return

End If

If (\_consoleDump) Then

Console.WriteLine("Walking the sink chain to find sink properties... ")

End If

While (thisSink IsNot Nothing)

If (\_consoleDump) Then

Console.WriteLine(New String("\_", 80))

Console.WriteLine("Next sink is : " & CType(thisSink, Object).GetType().Name)

DumpSinkProperties(thisSink)

End If

SetSinkProperties(thisSink)

thisSink = thisSink.NextChannelSink

End While

Return

End Sub

Private Sub DumpSinkProperties(ByVal sink As IClientChannelSink)

If (sink.Properties Is Nothing) Then

Console.WriteLine("There are no properties available on the " & CType(sink, Object).GetType().Name + " channelsink.")

Return

End If

Dim entry As DictionaryEntry

For Each entry In sink.Properties

Console.Write("ChannelSink property: " & entry.Key.ToString() & " value: ")

If (entry.Value Is Nothing) Then

Console.WriteLine("No value.")

Else

Console.WriteLine(entry.Value.ToString())

End If

Next

End Sub

' This method sets properties on the sink.

' The algorithm is that in the absence of instance attribute/value

' entries, the provider element template attribute/value entries will

' be set. This is a simple implementation that does not care about the

' element name underneath the provider element.

Private Sub SetSinkProperties(ByVal sink As IClientChannelSink)

If (sink.Properties Is Nothing) Then

Console.WriteLine("There are no properties available on the " & CType(sink.Properties, Object).GetType().Name & " channelsink.")

Return

End If

Dim entry As DictionaryEntry

For Each entry In sink.Properties

If (\_channelSinkProperties.Contains(entry.Key)) Then

If (\_consoleDump) Then

Console.WriteLine("Setting sink property template on " & CType(sink, Object).GetType().Name & "." & entry.Key.ToString())

End If

sink.Properties.Item(entry.Key) = \_channelSinkProperties.Item(entry.Key)

End If

Next

Dim provider As SinkProviderData

For Each provider In \_providerData

Dim configEntry As DictionaryEntry

For Each configEntry In provider.Properties

If (sink.Properties.Contains(configEntry.Key)) Then

If (\_consoleDump) Then

Console.WriteLine("Setting Instance override on " & CType(sink, Object).GetType().Name & "." & configEntry.Key)

End If

End If

sink.Properties.Item(configEntry.Key) = configEntry.Value

Next

Next

If (\_consoleDump) Then

DumpSinkProperties(sink)

End If

End Sub

Public Property [Next]() As System.Runtime.Remoting.Channels.IClientChannelSinkProvider Implements System.Runtime.Remoting.Channels.IClientChannelSinkProvider.Next

Get

Return \_next

End Get

Set(ByVal value As System.Runtime.Remoting.Channels.IClientChannelSinkProvider)

\_next = value

End Set

End Property

' This can be called in the constructor in case this provider is

' intended to build its own channel sink provider chain. Without

' providing such a chain, this provider must be specified in a

' configuration file with other providers.

Private Function CreateDefaultClientProviderChain() As IClientChannelSinkProvider

Dim chain As IClientChannelSinkProvider = New SoapClientFormatterSinkProvider()

Dim sink As IClientChannelSinkProvider = chain

sink.Next = New BinaryClientFormatterSinkProvider()

sink = sink.Next

Return chain

End Function

End Class

C#

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

using System;

using System.Collections;

using System.IO;

using System.Reflection;

using System.Runtime.Remoting;

using System.Runtime.Remoting.Channels;

using System.Runtime.Remoting.Channels.Http;

using System.Runtime.Remoting.Messaging;

using System.Runtime.Remoting.MetadataServices;

// This class implements a client-side channel sink provider that

// walks the channel sink chain, looking for channel sink

// properties that match those specified in the configuration file. If it

// finds them, the provider sets them to the values specified in the

// configuration file. This is a simple helper provider that returns no

// channel itself. Instead, it merely returns the next channel sink it can find, or null.

public class ChannelSinkPropertySetterProvider : IClientChannelSinkProvider

{

private IClientChannelSinkProvider \_next = null;

private IDictionary \_channelSinkProperties = null;

private ICollection \_providerData = null;

// Set the writeToConsole attribute on this provider element in the

// configuration file to "true"; otherwise, information is not written to the console.

private bool \_consoleDump = true;

// Default constructor.

public ChannelSinkPropertySetterProvider()

{

Console.WriteLine("Default constructor called.");

}

// Constructor with properties. If writeToConsole attribute is "true",

// this constructor will dump all custom configuration properties set

// in the configuration file.

public ChannelSinkPropertySetterProvider(IDictionary properties, ICollection providerData)

{

\_channelSinkProperties = properties;

// Sets the private console dump property for this provider.

if (properties["writeToConsole"] != null)

\_consoleDump = Boolean.Parse(properties["writeToConsole"].ToString());

\_providerData = providerData;

if (\_consoleDump)

{

Console.WriteLine("ChannelSinkPropertySetterProvider custom constructor called.");

foreach (SinkProviderData sinkData in providerData)

{

Console.WriteLine("SinkProvider element: " + sinkData.Name);

foreach (DictionaryEntry prop in sinkData.Properties)

{

Console.WriteLine("Prop name: " + prop.Key.ToString() + " value: " + prop.Value.ToString());

}

foreach (object child in sinkData.Children)

{

Console.WriteLine("Child: " + child.GetType().Name);

}

}

foreach (DictionaryEntry entry in properties)

{

Console.WriteLine("channel sink properties: " + entry.Key.ToString() + ", " + entry.Value.ToString());

}

Console.WriteLine();

}

}

// Called by the channel. Normally, this method takes any other sinks

// created by other providers in the chain, links them together, and

// then returns its own sink to the channel. In this case, this

// provider merely sets matching properties on each channel sink in the

// chain, and then returns the \*\*next\*\* channel sink to the channel or

// returns null, indicating to the channel that it is the end of the

// custom channel sink chain.

public IClientChannelSink CreateSink(IChannelSender channel, string url, object remoteChannelData)

{

if (\_consoleDump)

{

Console.WriteLine("CreateSink is called.");

Console.WriteLine("By " + channel.GetType().Name);

}

IClientChannelSink nextSink = null;

if (\_next != null)

{

nextSink = \_next.CreateSink(channel, url, remoteChannelData);

if (nextSink == null)

{

if (\_consoleDump)

Console.WriteLine("Next sink is null!");

return null;

}

WalkSinkChain(nextSink);

}

return nextSink;

}

// This call walks the sink chain, setting properties as it goes.

// The channelSinkProperties are the SinkProviderData dictionaries

// that contain the name of the subnode in the configuration file, and

// a dictionary entry of attribute/value entries on that element.

private void WalkSinkChain(IClientChannelSink thisSink)

{

if (thisSink == null)

return;

if (\_consoleDump)

Console.WriteLine("\r\n\tWalking the sink chain to find sink properties... \r\n");

while (thisSink != null)

{

if (\_consoleDump)

{

Console.WriteLine(new String('\_', 80));

Console.WriteLine("Next sink is : " + thisSink.GetType().Name);

DumpSinkProperties(thisSink);

}

SetSinkProperties(thisSink);

thisSink = thisSink.NextChannelSink;

}

return;

}

private void DumpSinkProperties(IClientChannelSink sink)

{

if (sink.Properties == null)

{

Console.WriteLine("There are no properties available on the " + sink.GetType().Name + " channelsink.");

return;

}

foreach (DictionaryEntry entry in sink.Properties)

{

Console.Write("ChannelSink property: " + entry.Key.ToString() + " value: ");

if (entry.Value == null)

Console.WriteLine("No value.");

else

Console.WriteLine(entry.Value.ToString());

}

}

// This method sets properties on the sink.

// The algorithm is that in the absence of instance attribute/value

// entries, the provider element template attribute/value entries will

// be set. This is a simple implementation that does not care about the

// element name underneath the provider element.

private void SetSinkProperties(IClientChannelSink sink)

{

if (sink.Properties == null)

{

Console.WriteLine("There are no properties available on the " + sink.GetType().Name + " channelsink.");

return;

}

foreach (DictionaryEntry entry in sink.Properties)

{

if (\_channelSinkProperties.Contains(entry.Key))

{

if (\_consoleDump)

Console.WriteLine("Setting sink property template on " + sink.GetType().Name + "." + entry.Key.ToString());

sink.Properties[entry.Key] = \_channelSinkProperties[entry.Key];

}

}

foreach (SinkProviderData provider in \_providerData)

{

foreach (DictionaryEntry configEntry in provider.Properties)

{

if (sink.Properties.Contains(configEntry.Key))

if (\_consoleDump)

Console.WriteLine("Setting Instance override on " + sink.GetType().Name + "." + configEntry.Key);

sink.Properties[configEntry.Key] = configEntry.Value;

}

}

if (\_consoleDump)

DumpSinkProperties(sink);

}

public IClientChannelSinkProvider Next

{

get

{

return \_next;

}

set

{

\_next = value;

}

}

// This can be called in the constructor in case this provider is

// intended to build its own channel sink provider chain. Without

// providing such a chain, this provider must be specified in a

// configuration file with other providers.

private IClientChannelSinkProvider CreateDefaultClientProviderChain()

{

IClientChannelSinkProvider chain = new SoapClientFormatterSinkProvider();

IClientChannelSinkProvider sink = chain;

sink.Next = new BinaryClientFormatterSinkProvider();

sink = sink.Next;

return chain;

}

}

**Client**

Visual Basic

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

Imports System

Imports System.Runtime.Remoting

Imports [Shared]

Public Class Client

Shared Sub Main()

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

Dim service As ServiceClass = New ServiceClass()

Console.WriteLine(service.GetServerTime())

End Sub

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl10_code');" \o "Copy Code)

using System;

using System.Runtime.Remoting;

public class Client

{

public static void Main(string[] Args)

{

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

ServiceClass service = new ServiceClass();

Console.WriteLine(service.GetServerTime());

}

}

**Server**

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl11_code');" \o "Copy Code)

Imports System

Imports System.Runtime.Remoting

Class Server

Shared Sub Main()

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

Console.WriteLine("Press enter to stop this process.")

Console.ReadLine()

End Sub

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl12_code');" \o "Copy Code)

using System;

using System.Diagnostics;

using System.Runtime.Remoting;

using System.Runtime.Remoting.Channels;

using System.Runtime.Remoting.Channels.Tcp;

public class ServerProcess

{

public static void Main(string[] Args)

{

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

Console.WriteLine("Press enter to stop this process.");

Console.ReadLine();

}

}

**ServiceClass**

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl13_code');" \o "Copy Code)

Imports System

Imports System.Runtime.Remoting

Public Class ServiceClass

Inherits MarshalByRefObject

Private starttime As DateTime

Public Sub New()

Console.WriteLine("A ServiceClass has been created.")

starttime = DateTime.Now

End Sub

Protected Overrides Sub Finalize()

Console.WriteLine("This object is being collected after " & (New TimeSpan(DateTime.Now.Ticks - starttime.Ticks)).ToString() & " seconds.")

End Sub

Public Function GetServerTime() As DateTime

Console.WriteLine("Time requested by a client.")

Return DateTime.Now

End Function

End Class

C#

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

using System;

using System.Runtime.Remoting;

public class ServiceClass : MarshalByRefObject

{

private DateTime starttime;

public ServiceClass()

{

Console.WriteLine("A ServiceClass has been created.");

starttime = DateTime.Now;

}

~ServiceClass()

{

Console.WriteLine("This object is being collected after " + (new TimeSpan(DateTime.Now.Ticks - starttime.Ticks)).ToString() + " seconds.");

}

public DateTime GetServerTime()

{

Console.WriteLine("Time requested by a client.");

return DateTime.Now;

}

}

**Client.exe.config**

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

<configuration>

<system.runtime.remoting>

<application>

<client>

<wellknown

type="ServiceClass, ServiceClass"

url="http://localhost:8080/RemoteObject"

/>

</client>

<channels>

<channel ref="http">

<clientProviders>

<formatter ref="soap"/>

<provider ref="propsetter" username="bob" writeToConsole="true">

<myElement allowAutoRedirect="true"/>

<endpoint preauthenticate="true"/>

<endpoint url="example.com:9000" password="xyz" />

<endpoint url="example.com:9001" password="123" />

<endpoint timeout="10000"/>

<endpoint url="example.com:\*" username="bob2" password="qwerty" domain="hello" />

</provider>

</clientProviders>

</channel>

</channels>

</application>

<channelSinkProviders>

<clientProviders>

<provider

id="propsetter"

type="ChannelSinkPropertySetterProvider, PropSink"

/>

</clientProviders>

</channelSinkProviders>

<debug loadTypes="true" />

</system.runtime.remoting>

</configuration>

**Server.exe.config**

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

<configuration>

<system.runtime.remoting>

<application>

<service>

<wellknown mode="SingleCall"

type="ServiceClass, ServiceClass"

objectUri="RemoteObject"

/>

</service>

<channels>

<channel port="8080" ref="http" />

</channels>

</application>

</system.runtime.remoting>

</configuration>

**Remoting Example: Hosting in Internet Information Services (IIS)**

This page is specific to

**Microsoft Visual Studio 2008/.NET Framework 3.5**

Other versions are also available for the following:

[Microsoft Visual Studio 2003/.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/c2swb8ah(VS.71).aspx)

[Microsoft Visual Studio 2005/.NET Framework 2.0](http://msdn.microsoft.com/en-us/library/c2swb8ah(VS.80).aspx)

[.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/c2swb8ah(VS.85).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 sample implements a basic Web service with a few complications. The [BinaryFormatter](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.formatters.binary.binaryformatter.aspx) is used because the payload is more compact and the system takes less time to serialize and deserialize the stream. In addition, if Internet Information Services (IIS) is using Windows Integrated Authentication (also known as NTLM authentication), the server authenticates the client and then returns to the client the identity that IIS was able to authenticate. Finally, you can help protect your Web service by changing the URL in the client configuration file to use "https" as the protocol scheme and configuring IIS to require Secure Sockets Layer (SSL) encryption for that virtual directory (the sample does not demonstrate this process).

|  |
| --- |
| **Caution:** |
| .NET Framework remoting does not do authentication or encryption by default. Therefore, it is recommended that you take all necessary steps to make certain of the identity of clients or servers before interacting with them remotely. Because .NET Framework remoting applications require **FullTrust** permissions to execute, if an unauthorized client were granted access on your server, the client could execute code as though it were fully trusted. Always authenticate your endpoints and encrypt the communication streams, either by hosting your remoted types in IIS or by building a custom channel sink pair to do this work. |

To compile and run this sample

1. Save all the files in a directory named RemoteIIS.
2. Compile the entire sample by typing the following commands at the command prompt:

Visual Basic

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

vbc /t:library ServiceClass.vb

vbc /r:System.Runtime.Remoting.dll /r:ServiceClass.dll Client.vb

C#

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

csc /t:library ServiceClass.cs

csc /r:System.Runtime.Remoting.dll /r:ServiceClass.dll Client.cs

1. Create a \bin subdirectory and copy ServiceClass.dll into that directory.
2. Create an application in IIS. Make the application alias "HttpBinary" and set the source directory to the "RemoteIIS" directory.
3. Set the authentication method for this virtual directory to Integrated Windows Authentication (formerly NTLM authentication). If anonymous access is selected, HttpContext.Current.User.Identity.Name will be **null** and GetServerString will return "\*\*\*unavailable\*\*\*" for the user alias. To prevent this from occurring, deselect anonymous access.
4. Make sure that IIS is started; at the command prompt in the "RemoteIIS" directory, type client.

This application runs on a single computer or across a network. 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.

**ServiceClass**

Visual Basic

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

Imports System

Imports System.Runtime.Remoting

Imports System.Web

Public Interface IService

Function GetServerTime() As DateTime

Function GetServerString() As String

End Interface

Public Class ServiceClass

Inherits MarshalByRefObject

Implements IService

Private InstanceHash As Integer

Public Sub New()

InstanceHash = Me.GetHashCode()

End Sub

Public Function GetServerTime() As Date Implements IService.GetServerTime

Return DateTime.Now

End Function

Public Function GetServerString() As String Implements IService.GetServerString

' Use the HttpContext to acquire what IIS thinks the client's identity is.

Dim temp As String = HttpContext.Current.User.Identity.Name

If (temp Is Nothing Or temp.Equals(String.Empty)) Then

temp = "\*\*unavailable\*\*"

End If

Return "Hi there. You are being served by instance number: " \_

& InstanceHash.ToString() \_

& ". Your alias is: " \_

& temp

End Function

End Class

C#

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

using System;

using System.Runtime.Remoting;

using System.Runtime.Remoting.Channels;

using System.Threading;

using System.Web;

public interface IService

{

DateTime GetServerTime();

string GetServerString();

}

// IService exists to demonstrate the possibility of publishing only the interface.

public class ServiceClass : MarshalByRefObject, IService

{

private int InstanceHash;

public ServiceClass()

{

InstanceHash = this.GetHashCode();

}

public DateTime GetServerTime()

{

return DateTime.Now;

}

public string GetServerString()

{

// Use the HttpContext to acquire what IIS thinks the client's identity is.

string temp = HttpContext.Current.User.Identity.Name;

if (temp == null || temp.Equals(string.Empty))

temp = "\*\*unavailable\*\*";

return "Hi there. You are being served by instance number: "

+ InstanceHash.ToString()

+ ". Your alias is: "

+ temp;

}

}

**Web.config**

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

<configuration>

<system.runtime.remoting>

<application>

<service>

<wellknown

mode="SingleCall" objectUri="SAService.rem"

type="ServiceClass, ServiceClass"/>

</service>

<channels>

<channel ref="http"/>

</channels>

</application>

</system.runtime.remoting>

</configuration>

**Client**

Visual Basic

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

Imports System

Imports System.Collections

Imports System.Net

Imports System.Runtime.Remoting

Imports System.Runtime.Remoting.Channels

Imports System.Security.Principal

Public Class Client

Public Shared Sub Main()

' Tells the system about the remote object and customizes the HttpChannel

' to use the binary formatter (which understands that base64 encoding is needed).

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

' New proxy for the ServiceClass.

' If you publish only the IService interface, you must use Activator.GetObject.

Dim service As ServiceClass = New ServiceClass()

' Programmatically customizes the properties given to the channel. This sample uses the

' application configuration file.

Dim Props As IDictionary = ChannelServices.GetChannelSinkProperties(service)

Props.Item("credentials") = CredentialCache.DefaultCredentials

' Reports the client identity name.

Console.WriteLine("ConsoleIdentity: " & WindowsIdentity.GetCurrent().Name)

' Writes what the server returned.

Console.WriteLine("The server says : " & service.GetServerString())

Console.WriteLine("Server time is: " & service.GetServerTime())

End Sub

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl10_code');" \o "Copy Code)

using System;

using System.Collections;

using System.Net;

using System.Runtime.Remoting;

using System.Runtime.Remoting.Channels;

using System.Security.Principal;

class Client

{

static void Main(string[] args)

{

// Tells the system about the remote object and customizes the HttpChannel

// to use the binary formatter (which understands that base64 encoding is needed).

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

// New proxy for the ServiceClass.

// If you publish only the IService interface, you must use Activator.GetObject.

ServiceClass service = new ServiceClass();

// Programmatically customizes the properties given to the channel. This sample uses the

// application configuration file.

IDictionary Props = ChannelServices.GetChannelSinkProperties(service);

Props["credentials"] = CredentialCache.DefaultCredentials;

// Reports the client identity name.

Console.WriteLine("ConsoleIdentity: " + WindowsIdentity.GetCurrent().Name);

// Writes what the server returned.

Console.WriteLine("The server says : " + service.GetServerString());

Console.WriteLine("Server time is: " + service.GetServerTime());

}

}

**Client.exe.config**

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl11_code');" \o "Copy Code)

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

<configuration>

<system.runtime.remoting>

<application>

<channels>

<channel ref="http" useDefaultCredentials="true" port="0">

<clientProviders>

<formatter

ref="binary"

/>

</clientProviders>

</channel>

</channels>

<client>

<wellknown

url="http://localhost:80/HttpBinary/SAService.rem"

type="ServiceClass, ServiceClass"

/>

</client>

</application>

</system.runtime.remoting>

</configuration>