**.NET Framework Remoting Overview**

.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.

**Advanced Remoting**

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(VS.85).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(VS.85).aspx)

Describes how to extend the [RealProxy](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.proxies.realproxy(VS.85).aspx) class to intercept calls at the moment of invocation.

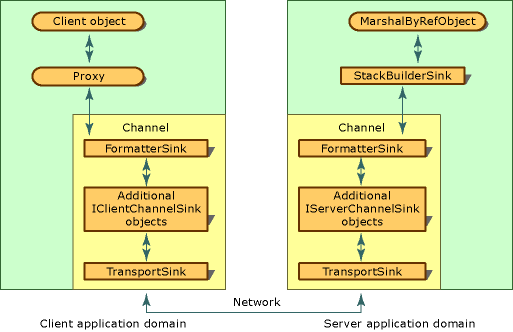
[Advanced Design Issues](http://msdn.microsoft.com/en-us/library/y1611dh0(VS.85).aspx)

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

**Sinks and Sink Chains**

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.

**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(VS.85).aspx) or [IServerChannelSinkfrlrfSystemRuntimeRemotingChannelsIServerChannelSinkClassTopic](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.channels.iserverchannelsink(VS.85).aspx). The first channel sink on the client must also implement [IMessageSinkfrlrfSystemRuntimeRemotingMessagingIMessageSinkClassTopic](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.messaging.imessagesink(VS.85).aspx). It typically implements [IClientFormatterSink](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.channels.iclientformattersink(VS.85).aspx) (which inherits from both **IMessageSink**, [IChannelSinkBase](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.channels.ichannelsinkbase(VS.85).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(VS.85).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.

|  |
| --- |
| **NoteNote** |
| 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(VS.85).aspx), [IClientFormatterSinkProviderfrlrfSystemRuntimeRemotingChannelsIClientFormatterSinkProviderClassTopic](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.channels.iclientformattersinkprovider(VS.85).aspx), or [IServerChannelSinkProvider](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.channels.iserverchannelsinkprovider(VS.85).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(VS.85).aspx) (or AsyncProcessMessage [frlrfSystemRuntimeRemotingMessagingIMessageSinkClassAsyncProcessMessageTopic](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.messaging.imessagesink.asyncprocessmessage(VS.85).aspx)). The formatter sink creates the transport header array and calls GetRequestStream 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(VS.85).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(VS.85).aspx).

Visual Basic

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl21VisualBasic');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl21VisualBasic');)

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

private IClientChannelSinkProvider CreateDefaultClientProviderChain(){

IClientChannelSinkProvider chain = new FirstClientFormatterSinkProvider();

IClientChannelSinkProvider sink = chain;

sink.Next = new SecondClientFormatterSinkProvider();

sink = sink.Next;

return chain;

}

|  |
| --- |
| **NoteNote** |
| 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[frlrfSystemRuntimeRemotingRemotingConfigurationClassConfigureTopic](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.remotingconfiguration.configure(VS.85).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(VS.85).aspx) and [SoapFormatterfrlrfSystemRuntimeSerializationFormattersSoapSoapFormatterClassTopic](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.formatters.soap.soapformatter(VS.85).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**[frlrfSystemRuntimeRemotingMessagingIMessageSinkClassAsyncProcessMessageTopic](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.messaging.imessagesink.asyncprocessmessage(VS.85).aspx) calls. At this stage the message has already been serialized and cannot be modified.

|  |
| --- |
| **NoteNote** |
| 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 CreateSinkreturns 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 StackBuilderSinkobject (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(VS.85).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(VS.85).aspx) but use the [BinaryClientFormatterSink](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.channels.binaryclientformattersink(VS.85).aspx) on the client.

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl34other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl34other');)

<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

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl35VisualBasic');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl35VisualBasic');)

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

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(VS.85).aspx).

To change this client to use a [TcpChannel](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.channels.tcp.tcpchannel(VS.85).aspx) object with the [SoapClientFormatterSinkfrlrfSystemRuntimeRemotingChannelsSoapClientFormatterSinkClassTopic](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.channels.soapclientformattersink(VS.85).aspx) object, you must change only the namespaces and the [RegisterChannel](http://msdn.microsoft.com/en-us/library/2eeyt4ba(VS.85).aspx)call, as shown in the following code:

Visual Basic

[[](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl41VisualBasic');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl41VisualBasic');)

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

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

**Extending RealProxy**

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(VS.85).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(VS.85).aspx)class and overriding the [RealProxy.Invoke](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.proxies.realproxy.invoke(VS.85).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(VS.85).aspx) attribute and applying the attribute to a child of [ContextBoundObject](http://msdn.microsoft.com/en-us/library/system.contextboundobject(VS.85).aspx). (Applying the proxy attribute to a child of [MarshalByRefObject](http://msdn.microsoft.com/en-us/library/system.marshalbyrefobject(VS.85).aspx) is not supported.) When **new** is called, the derived [ProxyAttribute](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.proxies.proxyattribute(VS.85).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(VS.85).aspx) method is called with an object that implements [IMessage](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.messaging.imessage(VS.85).aspx). The [IMessage](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.messaging.imessage(VS.85).aspx)implementation provides an [IDictionary](http://msdn.microsoft.com/en-us/library/system.collections.idictionary(VS.85).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(VS.85).aspx) interface and related interfaces ([IMethodCallMessage](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.messaging.imethodcallmessage(VS.85).aspx) and [IMethodReturnMessage](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.messaging.imethodreturnmessage(VS.85).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(VS.85).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(VS.85).aspx) method to process an [IConstructionCallMessage](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.activation.iconstructioncallmessage(VS.85).aspx) object and generate an IConstructionReturnMessage object. You can also use the [RealProxy.InitializeServerObject](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.proxies.realproxy.initializeserverobject(VS.85).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(VS.85).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(VS.85).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(VS.85).aspx) property must be set in the [IMessage](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.messaging.imessage(VS.85).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(VS.85).aspx)) interface.

To participate in marshaling, override [RealProxy.CreateObjRef](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.proxies.realproxy.createobjref(VS.85).aspx) and provide a custom [ObjRef](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.objref(VS.85).aspx)that extends [ObjRef](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.objref(VS.85).aspx). If you want to add custom data to the custom [ObjRef](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.objref(VS.85).aspx), override[ObjRef.GetObjectData](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.objref.getobjectdata(VS.85).aspx). You add your custom data and delegate to the [ObjRef.GetObjectData](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.objref.getobjectdata(VS.85).aspx) method to modify the object identity capabilities of the .NET remoting marshaling system.

During deserialization, the remoting system calls your overridden [IObjectReference.GetRealObject](http://msdn.microsoft.com/en-us/library/system.runtime.serialization.iobjectreference.getrealobject(VS.85).aspx) on the custom [ObjRef](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.objref(VS.85).aspx). Here, you should delegate to the base GetRealObject[frlrfSystemRuntimeRemotingObjRefClassGetRealObjectTopic](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.objref.getrealobject(VS.85).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(VS.85).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(VS.85).aspx) and [ObjRef.IsFromThisProcess](http://msdn.microsoft.com/en-us/library/system.runtime.remoting.objref.isfromthisprocess(VS.85).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(VS.85).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**

Because the type information of a remote type is required by the client, it is often easiest to simply deploy the remote type assembly on the client. However, in many cases you will 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 will be shared by the server and the client.

|  |
| --- |
| **NoteNote** |
| The client will have to call GetObject to instantiate the proxy. Using the operator new will cause 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 [Using Soapsuds.exe with Remoting](http://msdn.microsoft.com/en-us/library/xse48s01(VS.85).aspx).
* Generate source code using the Soapsuds tool. For more information see Using [Using Soapsuds.exe with Remoting](http://msdn.microsoft.com/en-us/library/xse48s01(VS.85).aspx).