Disconnect the client from WCF service and notifiy the client before disconnecting.

A colleague of mine was asking me to create a Client-Server demo application using WCF. The following are the requirements:  
  
1>Service instance mode is “PerSession”.  
2>Session mode has to be “Required”.  
3>WCF Server will be able to terminate a session at any time. In other words, WCF Server can disconnect a WCF client.  
4>Before terminating the session, the WCF client will be notified.  
  
In order to meet the above requirement, I have used the WCF Callback for notifying the Client and disconnect the client by closing the incoming channel. Let’s see step by step of the implementation.  
  
Step 1> Create the WCF Contact:  
  
[ServiceContract(CallbackContract=typeof(INotifyClientCallback), SessionMode=SessionMode.Required)]   
public interface IService1  
{   
[OperationContract]  
bool StartingService();  
}  
public interface INotifyClientCallback  
{  
[OperationContract(IsOneWay = true)]  
void Disconnecting();  
}   
  
INotifyClientCallback interface for Callback.  
  
Step 2> Implementation of the Contact:  
  
[ServiceBehavior(InstanceContextMode = InstanceContextMode.PerSession)]  
public class Service1 : IService1  
{  
  
private static readonly Dictionary subscribers = new Dictionary();  
public static event EventHandler onClientAdded;  
  
///  
/// Returns the IP Address of the Client  
///  
///   
public string GetAddressAsString()  
{  
if (!OperationContext.Current.IncomingMessageProperties.ContainsKey(RemoteEndpointMessageProperty.Name))  
{  
return "127.0.0.1";  
}  
RemoteEndpointMessageProperty clientEndpoint =  
OperationContext.Current.IncomingMessageProperties[RemoteEndpointMessageProperty.Name] as RemoteEndpointMessageProperty;  
return clientEndpoint.Address;  
}  
  
  
public bool StartingService()  
{  
//Get the callback reference  
INotifyClientCallback callback = OperationContext.Current.GetCallbackChannel();  
  
string IPAddress = GetAddressAsString();  
lock (subscribers)  
{  
if (!subscribers.ContainsKey(IPAddress))  
{  
subscribers[IPAddress] = new CommunicationStore() { NotifyCallback = callback, IService = OperationContext.Current.InstanceContext };  
if (onClientAdded != null)  
{  
onClientAdded(IPAddress, null);  
}  
}  
}  
return true;  
}  
  
  
public static void Disconnect(string ipAddress)  
{  
if (subscribers.ContainsKey(ipAddress))  
{  
CommunicationStore com = subscribers[ipAddress];   
if (((ICommunicationObject)com.NotifyCallback).State == CommunicationState.Opened)  
{  
try  
{  
//fires the callback method  
com.NotifyCallback.Disconnecting();  
com.IService.IncomingChannels.FirstOrDefault().Close();  
}  
catch (Exception)  
{  
throw;  
}  
}  
}  
}   
  
}  
public class CommunicationStore  
{  
public InstanceContext IService { get; set; }  
public INotifyClientCallback NotifyCallback { get; set; }  
}  
  
Some important points of the service implementation:  
a>Use a static dictionary to keep the Client’s IP and callback channel. Before writing on the share object, lock the object.  
  
b>Gets the IP address of the client using the GetAddressAsString method. You can get the IP of the client from the incoming message. The following statement shows how can we get the IP adddress of the Client in WCF:  
  
RemoteEndpointMessageProperty clientEndpoint = OperationContext.Current.IncomingMessageProperties[RemoteEndpointMessageProperty.Name] as RemoteEndpointMessageProperty;   
String ipAddress = clientEndpoint.Address;  
  
If you are using the namepipe binding, you will not get the RemoteEndpointMessageProperty.  
  
c>When the client creates the proxy of the service, it will call StartingService method immediately. Inside the StartingService method, I am keeping the callback channel of the client and current instance into the dictionary.  
  
d>When the user of WCF service wants to disconnect a client, he/she will call the Disconnect method with the IP Address of the client.  
  
e>The Disconnect method uses the IP Address to get the callback channel of the client and associate service instance of the client from the dictionary. Eventually, it notifies the client by using callback channel and close the incoming channel.  
  
I am done with the service implementation.  
  
Step 3> Implementation of the callback and create the proxy of the service:  
  
public class Proxy  
{  
public static DuplexChannelFactory factory;  
static IService1 \_IService1;  
private static bool Connect()  
{  
  
try  
{  
WSDualHttpBinding binding = new WSDualHttpBinding();  
  
EndpointAddress address1 = new EndpointAddress(@"http://localhost:9008/Service1");  
  
EndpointAddress address = new EndpointAddress(@"net.tcp://localhost:9002/");  
NetTcpBinding nettcp = new NetTcpBinding();  
nettcp.ReliableSession.Enabled = true;  
nettcp.ReliableSession.InactivityTimeout = new TimeSpan(0, 0, 30);  
  
NetNamedPipeBinding namepipe = new NetNamedPipeBinding();  
EndpointAddress addressPipe = new EndpointAddress(@"net.pipe://localhost/");  
  
  
InstanceContext context = new InstanceContext(new NotifyCallback());  
  
binding.ClientBaseAddress = new Uri(@"http://localhost:9010/");  
  
factory = new DuplexChannelFactory(context, binding, address1);  
  
\_IService1 = factory.CreateChannel();  
\_IService1.StartingService();  
}  
catch (Exception ex)  
{  
  
}  
return true;  
}  
  
}   
  
public static IService1 ServiceProxy  
{  
get  
{  
if (\_IService1 == null)  
{  
Connect();  
}  
  
return \_IService1;  
}  
}  
}  
  
  
public class NotifyCallback : INotifyClientCallback  
{  
  
#region INotifyClientCallback Members  
  
public void Disconnecting()  
{  
MessageBox.Show("You are about to disconnect from the service");  
}  
  
#endregion  
}  
Points to ponder of the above code snippet:  
  
a>When you use the WCF callback, you have to use the DuplexChannelFactory rather then ChannelFactory to create the proxy.Othewise, you can generate the proxy.  
  
b>WCF callback supports the following binding: WSDualHttpBinding, NetTcpBinding and NetNamedPipeBinding. When you use the WSDualHttpBinding, don’t forget to set the ClientBaseAddress.  
  
c>Implement the callback interface.  
  
That’s it.  
  
If you need the source code of this demo, you can email me.  
  
Happy Programming!

Posted by [Atiquzzaman Setu](https://www.blogger.com/profile/00941869623763242782" \o "author profile)at [1:08 AM](http://setus.blogspot.in/2009/06/disconnect-client-from-wcf-service-and.html) [https://resources.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=6004394169581504268&postID=2183833244269278872)

[Email This](https://www.blogger.com/share-post.g?blogID=6004394169581504268&postID=2183833244269278872&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=6004394169581504268&postID=2183833244269278872&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=6004394169581504268&postID=2183833244269278872&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=6004394169581504268&postID=2183833244269278872&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=6004394169581504268&postID=2183833244269278872&target=pinterest)

2 comments:

**[http://lh3.googleusercontent.com/zFdxGE77vvD2w5xHy6jkVuElKv-U9_9qLkRYK8OnbDeJPtjSZ82UPq5w6hJ-SA=s35](https://www.blogger.com/profile/16267783769541904442)**

[**groovecontrol**](https://www.blogger.com/profile/16267783769541904442)**said...**

That's a very good article! Thanks for the help :)  
  
Turjo

[August 17, 2009 at 4:35 AM](http://setus.blogspot.com/2009/06/disconnect-client-from-wcf-service-and.html?showComment=1250508933286#c7187350504892500348)

**[http://lh3.googleusercontent.com/zFdxGE77vvD2w5xHy6jkVuElKv-U9_9qLkRYK8OnbDeJPtjSZ82UPq5w6hJ-SA=s35](https://www.blogger.com/profile/04545741660398391322)**

[**Priyanka**](https://www.blogger.com/profile/04545741660398391322)**said...**

RemoteEndpointMessageProperty clientEndpoint = OperationContext.Current.IncomingMessageProperties[RemoteEndpointMessageProperty.Name] as RemoteEndpointMessageProperty;   
String ipAddress = clientEndpoint.Address;  
  
  
ClientEndpoint.Address gives me blank. I am trying to run my service in Visual studio 2010 ultimate.

[August 20, 2010 at 1:59 AM](http://setus.blogspot.com/2009/06/disconnect-client-from-wcf-service-and.html?showComment=1282294789332#c3651667674682371230)

[Post a Comment](https://www.blogger.com/comment.g?blogID=6004394169581504268&postID=2183833244269278872)

How to reconnect to WCF service after exception occurred

There can be several reasons to raise exception when calling a WCF service. I am not going to discuss why the exception can be occurred. I will, however, discuss here how to reconnect to WCF service after exception.  
  
If any exception occurs while calling WCF service, the proxy of the WCF service goes to invalid state. Invalid means you cannot use that proxy any more. You have to recreate the proxy before calling any operation of the service.  
  
I will use ChannelFactory here in order to create the proxy of the WCF service. Let the service contact is IService. The following code snippet shows how a proxy would be created using ChannelFactory:  
  
private static ChannelFactory \_Factory;  
private static IService \_Proxy;  
public static Timer \_reconnectTimer;  
public static bool CreateProxy()  
{  
try  
{  
  
//get the factory instance  
\_Factory = new ChannelFactory("ServiceEndpoint");  
  
\_Proxy = \_jerichoFactory.CreateChannel();  
\_Proxy.IsServiceLive();  
//add the faulted event  
((ICommunicationObject)\_Proxy).Faulted += new EventHandler(Proxy\_Faulted);  
  
if (\_reconnectTimer != null)  
{  
\_reconnectTimer.Stop();  
}   
}  
catch (Exception)  
{  
return false;  
}  
  
return true;  
}  
  
In the above code, I have subscribed the Faulted event of the proxy. Make sure you have cast the proxy into ICommunicationObject. Faulted event is fired whenever any exception occurres during the service call. Notice, I have called IsServiceLive() method after creating the channel. IsServiceLive() method simply returns true to make sure that the proxy can connect to the service. The following code snippet is the implementation of Faulted event.  
  
static void Proxy\_Faulted(object sender, EventArgs e)  
{  
IChannel channel = sender as IChannel;  
if (channel != null)  
{  
channel.Abort();  
channel.Close();  
}  
//Disable the keep alive timer now that the channel is faulted   
//The proxy channel should no longer be used   
AbortProxy();  
//fire the OnProxyFaulted event  
if (OnProxyFaulted != null)  
{  
OnProxyFaulted(null, null);  
}  
if (\_reconnectTimer == null)  
{  
\_reconnectTimer = new Timer(2000 \* 60);  
\_reconnectTimer.Elapsed += new ElapsedEventHandler(ReconnectTimer\_Elapsed);  
}  
  
//Enable the try again timer and attempt to reconnect   
\_reconnectTimer.Start();  
}  
  
static public void AbortProxy()  
{  
if (\_Factory != null)  
{  
\_Factory.Abort();  
\_Factory.Close();  
\_Factory = null;  
}  
}  
  
In the implementation of the Faulted event, I have canceled and aborted the channel of the proxy. After that, I have created a time which will be fired in every two seconds. Let’s see the implementation of the timer.  
  
static void ReconnectTimer\_Elapsed(object sender, ElapsedEventArgs e)  
{  
if (CreateProxy())  
{   
\_reconnectTimer.Stop();  
}  
}  
  
This is very straightforward implementation of the timer. If it can create the proxy and connect to the service, it will stop the timer. This is how you can reconnect to the WCF service after exception.  
  
I could have created the proxy in the Faulted event. But it is better to use a timer to poll the service after a certain time.  
  
Happy programming!

Posted by [Atiquzzaman Setu](https://www.blogger.com/profile/00941869623763242782" \o "author profile)at [4:24 AM](http://setus.blogspot.in/2009/06/how-to-reconnect-to-wcf-service-after.html) [https://resources.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=6004394169581504268&postID=7783121904391212179)

[Email This](https://www.blogger.com/share-post.g?blogID=6004394169581504268&postID=7783121904391212179&target=email)[BlogThis!](https://www.blogger.com/share-post.g?blogID=6004394169581504268&postID=7783121904391212179&target=blog)[Share to Twitter](https://www.blogger.com/share-post.g?blogID=6004394169581504268&postID=7783121904391212179&target=twitter)[Share to Facebook](https://www.blogger.com/share-post.g?blogID=6004394169581504268&postID=7783121904391212179&target=facebook)[Share to Pinterest](https://www.blogger.com/share-post.g?blogID=6004394169581504268&postID=7783121904391212179&target=pinterest)

3 comments:

**[http://lh3.googleusercontent.com/zFdxGE77vvD2w5xHy6jkVuElKv-U9_9qLkRYK8OnbDeJPtjSZ82UPq5w6hJ-SA=s35-w35](https://www.blogger.com/profile/15391971300694571961)**

[**dMorgan**](https://www.blogger.com/profile/15391971300694571961)**said...**

Nice, this is what I was looking for.  
Just one question: what is "\_jerichoFactory"? I didn't see it declared or explained anywhere

[July 28, 2010 at 6:22 AM](http://setus.blogspot.com/2009/06/how-to-reconnect-to-wcf-service-after.html?showComment=1280323356276#c3500187176189996515)

**[http://1.bp.blogspot.com/_z0QE86-d-3U/SjnsQuMrnAI/AAAAAAAAABY/iJ_XkkXQKbg/S45-s35/SDC10775.JPG](https://www.blogger.com/profile/00941869623763242782)**

[**Atiquzzaman Setu**](https://www.blogger.com/profile/00941869623763242782)**said...**

Thank you for reading my blog.  
  
"\_jerichoFactory" is typo here. It should be "\_Factory" instead of "\_jerichoFactory".

[July 28, 2010 at 10:27 PM](http://setus.blogspot.com/2009/06/how-to-reconnect-to-wcf-service-after.html?showComment=1280381263524#c3526127236078980405)

**[http://lh3.googleusercontent.com/zFdxGE77vvD2w5xHy6jkVuElKv-U9_9qLkRYK8OnbDeJPtjSZ82UPq5w6hJ-SA=s35-w35](https://www.blogger.com/profile/17096025936969502692)**

[**Thichca2**](https://www.blogger.com/profile/17096025936969502692)**said...**

Hi,   
In .net remoting I do not have to re-create my object again if my network is disconnected.  
Why would it need in WCF?  
It is really inconvenient if it is just a network glitch and your app just need to try again

[February 8, 2012 at 7:31 AM](http://setus.blogspot.com/2009/06/how-to-reconnect-to-wcf-service-after.html?showComment=1328715096748#c3668610611688386241)

[Post a Comment](https://www.blogger.com/comment.g?blogID=6004394169581504268&postID=7783121904391212179)

How to count the number of active session in WCF

There are three ways we can manage instance of WCF service. In other words, WCF supports three types of instance activation. These three types of activation are:  
1> Per-call session – It allocates a new service instance per client request.  
2> Sessionful session – It allocates a new service instance per client connection.  
3> Singleton session – All clients share the same service instance.  
  
Here, our goal is to count the number of active WCF service instance and set the limit of maximum concurrent client when the activation mode is Sessionful.  
Let’s create a WCF Service. We will start by creating a WCF Contact.  
  
ServiceContract(SessionMode=SessionMode.Required)]   
public interface ISessionCount  
{   
[OperationContract]  
string GetData(string s);  
[OperationContract]  
bool StartingService();  
[OperationContract]  
bool StopingService();  
}  
Here, the above code snippet is a WCF Contact where the SessionMode is Required. It means the client has to use the session mode.  
  
[ServiceBehavior(InstanceContextMode=InstanceContextMode.PerSession)]  
public class SessionCount : ISessionCount, IDisposable  
{  
//static int Counter;  
static object Counter = new object();  
static SessionCount()  
{  
Counter = 0;  
}  
public SessionCount()  
{   
  
}  
  
public string GetData(string s)  
{  
return "This is from Service :" + s;  
}  
  
public bool StartingService()  
{  
  
lock (Counter)  
{  
int localCount = Convert.ToInt32(Counter);  
Counter = localCount + 1;  
if (localCount > 2)  
{  
this.Dispose();  
return false;  
}  
  
}  
return true;  
}  
  
public bool StopingService()  
{  
return true;  
}  
  
#region IDisposable Members  
  
public void Dispose()  
{  
lock (Counter)  
{  
Counter = Convert.ToInt32(Counter) - 1;  
}  
}  
  
#endregion   
}  
  
The above code snippet is the implementation of the service contact. Let’s examine the implementation point by point.  
  
1> Implement the IDisposable Interface. When the service instance will be disposed, the Dispose method will be fired.  
2> Declare a static Counter object. Instead of primitive type(Int, Long), I am using Object here because I have to lock the object before accessing this. We have to lock the object because of thread related issue.  
3> Initialize the counter object in the type constructor. Type constructor is fired once in the application domain regardless of the number of new instance created.  
4> Assume the maximum concurrent client of the WCF service can be 2.  
5> When a client will create a proxy of the WCF service, it will call the start Service.  
6> In the start service, we will increase the counter and check whether it exceeds the maximum concurrent user limet or not. If it exceeds then dispose the current instance return false.  
7> When the client will be shutdown, it will call the stop service method which eventually will dispose the service instacne. However, if the client close the proxy, the service instace will be disposed. The responsibility of the dispose method is to decrease the Counter.  
  
This is straight forward way we can protect the maximum concurrent client in WCF service.

### How to send/receive large data in WCF

Sometimes, we need to send or receive large data in WCF. For example, if we want to send a large xml file from WCF-client to WCF-service then we cannot send this file using default configuration of binding. We need to do some tweaks to send large data. Let us see some of the properties of binding which can help us to increase the size of the send/receive message:

**MaxBufferSize** - gets or sets a value that specificies the maximum size, in bytes, of the buffer used to messages in memory.

**MaxReceivedMessageSize** – gets or sets the maximum size of the received message that is processed by binding. The default value is 65,536 bytes

If we change the default value of these two properties and set the maximum value supported by WCF, we can increase the size of sending/receiving message.

The following code snippet shows how you can set these properties.  
NetTcpBinding netTcpBinding = new NetTcpBinding();  
netTcpBinding.MaxBufferSize = 65536;//set max value supported by WCF  
netTcpBinding.MaxReceivedMessageSize = 204003200;//set max value

These settings have to be in both client and server side. By setting the above properties, we can send/receive around 5MB.

There is a problem if we follow the above approach. To understand the problem, we need to know how WCF works while sending/receiving data.

By default, when the client and the service exchange messages, these messages are buffered on the receiving end and delivered once the entire message is received. This is true whether it is the client sending a message to the service or the service returning a message to the client. When the client calls the service, the service is invoked when the message is received in its entirety. The client is unblocked when the returned message with the results of the invocation is received in its entirety. Therefore, for sending/receiving large data, this can create a huge problem. To handle such cases, WCF enables the receiving side (be it the client or the service) to start processing the data in the message while the message is still being received by the channel. Such processing is called streaming transfer mode.

We can change the transfer mode of the service by changing the following property.

**TransferMode** – specificies whether messages are buffered or streamed or a request, response.

The following code snippet shows how you can set this property:  
NetTcpBinding netTcpBinding = new NetTcpBinding();  
netTcpBinding.TransferMode = TransferMode.Streamed;

However, setting the TransferMode as a Streamed has some limitations. If we use streamed transfer mode then there are the following constraints in the operation contract:  
· Parameter can be only stream type.  
· Return type has to be stream type.  
· Number of parameters have to be less than or equal to one.  
· Only the BasicHttpBinding, NetTcpBinding, and NetNamedPipeBinding support streaming.

Setting the TransferMode as a Streamed is good choice when you will send/receive large file.

Hope this help!

# TIP: Closing your WCF Connections properly

Posted by [**SM Altaf**](http://www.codeguru.com/member.php/SM+Altaf/) on **March 9th, 2009**

Top of Form

* 0.25

0.50

0.75

1.00

1.25

1.50

1.75

2.00

2.25

2.50

2.75

3.00

3.25

3.50

3.75

4.00

4.25

4.50

4.75

5.00

* Vote!

Bottom of Form

* [inShare](javascript:void(0);)

In a [previous tip](http://www.codeguru.com/csharp/.net/net_general/tipstricks/article.php/c15907/), you looked at how you could use ChannelFactory or Proxy Classes to talk to a WCF service. Now you can have a look at how to properly close of these connections.

If you've used Visual Studio to generate a WCF proxy class for one of your WCF services, go on and have a look at it now. In it, have a look at the generated client class:

1. public partial class StockServiceClient : System.ServiceModel.ClientBase<StockService.IStockService>, StockService.IStockService {
2. ...

The client that you use in your main code inherits from ClientBase<>. Looking at [ClientBase on MSDN](http://msdn.microsoft.com/en-us/library/ms576141.aspx), you see that it implements IDisposable. As with most other classes implementing IDisposable, it is tempting to use the C# using statement for your WCF clients.

1. //It's tempting, but don't do this:
2. using(StockService.StockServiceClient client = new StockService.StockServiceClient(
3. "StockBasicHttpEndpoint", stockServiceUrl))
4. {
5. client.GetStockIdByName("MSFT");
6. }

**http://www.codeguru.com/images/pdfImage.jpg**

**Brief: OpenStack is Now Ready for Business**

[**Download Now**](http://o1.qnsr.com/cgi/r?WT.qs_dlk=WKqGnArIZ7sAAFcYB-4AAAAZ;;n=203;c=1395297;s=9479;x=7936;f=201308291424140;u=j;z=TIMESTAMP;k=http://assetform.codeguru.com/controller?asset=225939910&srvid=95910&vkey=14094310&io=226651&BLUID=2017021922031231656690001&qset=CONTACTFORM_HQB&formHQB=y&domain=www.codeguru.com&cfurl=http%3A%2F%2Fwww.codeguru.com%2Fcsharp%2Fcsharp%2Fnet30%2Farticle.php%2Fc15941%2FTIP-Closing-your-WCF-Connections-properly.htm)

And as you know, the using block will automatically call client.Dispose() when it's done. There's a problem with it though — with ClientBase, the Dispose() method makes a call to Close(). When the ClientBase's Close() method is called, a web service call actually goes out to the WCF service, informing it that the connection session is no longer required. This appears counter-intuitive, but it was done by design. The problem with this mechanism is that when the Close() method is called, an exception can be thrown. After all, it involves yet another network call to a web service. It's for this reason that the using statement isn't recommended with WCF clients.

Instead, you should attempt to Close() it in a try-catch block, with the catch block performing an Abort()

1. //Call your web service as usual.
2. StockService.StockServiceClient client = new StockService.StockServiceClient(
3. "StockBasicHttpEndpoint", stockServiceUrl);
5. string StockId = client.GetStockIdByName("MSFT");
7. //Done with the service, let's close it.
8. try
9. {
10. if (client.State != System.ServiceModel.CommunicationState.Faulted)
11. {
12. client.Close();
13. }
14. }
15. catch (Exception ex)
16. {
17. client.Abort();
18. }

The same thing applies to your ChannelFactory, if you use one — and the code to close it is exactly the same.

1. //Create channel from factory
2. ChannelFactory channel =
3. new ChannelFactory<IService1>("bindingName");
4. IService1 client = channel.CreateChannel();
5. client.DoWork();
6. //Done with the service, let's close it.
7. try
8. {
9. if (client.State != System.ServiceModel.CommunicationState.Faulted)
10. {
11. client.Close();
12. }
13. }
14. catch (Exception ex)
15. {
16. client.Abort();
17. }

So the points made here, very briefly, are

* Don't use 'using' with your WCF client classes
* .Close() the client in a try-catch block and .Abort() it if an exception occurs.

## About the Author

#### SM Altaf

Mendhak is a web developer and a Microsoft MVP who works with ASP.NET and PHP among the usual array[] of web technologies. He is also rumored to be a three eyed frog, but the evidence is lacking. He can be contacted via his website, [www.mendhak.com](http://www.mendhak.com/). 

## Comments

#### Is this still true in 4.5

Posted by Brian Wagg on 02/02/2016 01:40pm

Is this still true in .NET 4.5? I've read a few places that WCF will call dispose for you if you are using perCall. But it doesn't go into details as to whether it is calling dispose on your client proxy or on other objects it may have had to create when communicating with the server. Any insight would be appreciated. An example of where it is stated: http://www.codeguru.com/csharp/csharp/net30/article.php/c15941/TIP-Closing-your-WCF-Connections-properly.htm

[Reply](http://www.codeguru.com/csharp/csharp/net30/article.php/c15941/TIP-Closing-your-WCF-Connections-properly.htm#54077822)

#### I don't think so either!

Posted by Mark on 04/17/2012 07:19am

You are better off aborting the object in the 'Finally' part of the try-ctach. In your example you won't be able to abort unless exception is thrown. What if no exception is thrown but the transaction lasts forever? You then have to realy on yout timeout quota, which effectively (in the case of a large app) may end up in multiple sessions lingering and waiting to be disposed off. Please remove this post - it is nto right and may be confusing for some readers. Regards.

[Reply](http://www.codeguru.com/csharp/csharp/net30/article.php/c15941/TIP-Closing-your-WCF-Connections-properly.htm#1246223)

#### No, I don't think so.

Posted by JerryNixon on 03/09/2011 03:12pm

1. With your TIP the method call (cal named it DoWork()) may fail, causing the flow to exit the method and bypass the possibility of the Abort() call in your catch { client.Abort(); }. 2. WIth your TIP the method call may pass, but the state may be faulted, but Abort() cannot be called because an exception is required. It seems to me that you should: 1. Move the method call into your try { } block. 2. Remove the (State != Faulted) if statement. Are you sure about your suggestion here?

[Reply](http://www.codeguru.com/csharp/csharp/net30/article.php/c15941/TIP-Closing-your-WCF-Connections-properly.htm#51459910)