**Using Sessions**

In Windows Communication Foundation (WCF) applications, a *session* correlates a group of messages into a conversation. WCF sessions are different than the session object available in ASP.NET applications, support different behaviors, and are controlled in different ways. This topic describes the features that sessions enable in WCF applications and how to use them.

**Sessions in Windows Communication Foundation Applications**

When a service contract specifies that it requires a session, that contract is specifying that all calls (that is, the underlying message exchanges that support the calls) must be part of the same conversation. If a contract specifies that it allows sessions but does not require one, clients can connect and either establish a session or not establish a session. If the session ends and a message is sent through the same channel an exception is thrown.

WCF sessions have the following main conceptual features:

* They are explicitly initiated and terminated by the calling application (the WCF client).
* Messages delivered during a session are processed in the order in which they are received.
* Sessions correlate a group of messages into a conversation. Different types of correlation are possible. For instance, one session-based channel may correlate messages based on a shared network connection while another session-based channel may correlate messages based on a shared tag in the message body. The features that can be derived from the session depend on the nature of the correlation.
* There is no general data store associated with a WCF session.

If you are familiar with the [System.Web.SessionState.HttpSessionState](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.aspx) class in ASP.NET applications and the functionality it provides, you might notice the following differences between that kind of session and WCF sessions:

* ASP.NET sessions are always server-initiated.
* ASP.NET sessions are implicitly unordered.
* ASP.NET sessions provide a general data storage mechanism across requests.

This topic describes:

* The default execution behavior when using session-based bindings in the service model layer.
* The types of features that the WCF session-based, system-provided bindings provide.
* How to create a contract that declares a session requirement.
* How to understand and control the creation and termination of the session and the relationship of the session to the service instance.

**Default Execution Behavior Using Sessions**

A binding that attempts to initiate a session is called a *session-based* binding. Service contracts specify that they require, permit, or refuse session-based bindings by setting the [System.ServiceModel.ServiceContractAttribute.SessionMode](http://msdn.microsoft.com/en-us/library/system.servicemodel.servicecontractattribute.sessionmode.aspx) property on the service contract interface (or class) to one of the [System.ServiceModel.SessionMode](http://msdn.microsoft.com/en-us/library/system.servicemodel.sessionmode.aspx) enumeration values. By default, the value of this property is Allowed, which means that if a client uses a session-based binding with a WCF service implementation, the service establishes and uses the session provided.

When a WCF service accepts a client session, the following features are enabled by default:

1. All calls between a WCF client object are handled by the same service instance.
2. Different session-based bindings provide additional features.

**System-Provided Session Types**

A session-based binding supports the default association of a service instance with a particular session. However, different session-based bindings support different features in addition to enabling the session-based instancing control previously described.

WCF provides the following types of session-based application behavior:

* The [System.ServiceModel.Channels.SecurityBindingElement](http://msdn.microsoft.com/en-us/library/system.servicemodel.channels.securitybindingelement.aspx) supports security-based sessions, in which both ends of communication have agreed upon a specific secure conversation. For more information, see [Securing Services](http://msdn.microsoft.com/en-us/library/ms734769.aspx). For example, the [System.ServiceModel.WSHttpBinding](http://msdn.microsoft.com/en-us/library/system.servicemodel.wshttpbinding.aspx) binding, which contains support for both security sessions and reliable sessions, by default uses only a secure session that encrypts and digitally signs messages.
* The [System.ServiceModel.NetTcpBinding](http://msdn.microsoft.com/en-us/library/system.servicemodel.nettcpbinding.aspx) binding supports TCP/IP-based sessions to ensure that all messages are correlated by the connection at the socket level.
* The [System.ServiceModel.Channels.ReliableSessionBindingElement](http://msdn.microsoft.com/en-us/library/system.servicemodel.channels.reliablesessionbindingelement.aspx) element, which implements the WS-ReliableMessaging specification, provides support for reliable sessions in which messages can be configured to be delivered in order and exactly once, ensuring messages are received even when messages travel across multiple nodes during the conversation. For more information, see [Reliable Sessions](http://msdn.microsoft.com/en-us/library/ms730123.aspx).
* The [System.ServiceModel.NetMsmqBinding](http://msdn.microsoft.com/en-us/library/system.servicemodel.netmsmqbinding.aspx) binding provides MSMQ datagram sessions. For more information, see [Queues in Windows Communication Foundation](http://msdn.microsoft.com/en-us/library/ms731089.aspx).

Setting the **SessionMode** property does not specify the type of session the contract requires, only that it requires one.

**Creating a Contract That Requires a Session**

Creating a contract that requires a session states that the group of operations that the service contract declares must all be executed within the same session and that messages must be delivered in order. To assert the level of session support that a service contract requires, set the **System.ServiceModel.ServiceContractAttribute.SessionMode** property on your service contract interface or class to the value of the **System.ServiceModel.SessionMode** enumeration to specify whether the contract:

* Requires a session.
* Allows a client to establish a session.
* Prohibits a session.

Setting the **SessionMode** property does not, however, specify the type of session-based behavior the contract requires. It instructs WCF to confirm at runtime that the configured binding (which creates the communication channel) for the service does, does not, or can establish a session when implementing a service. Again, the binding can satisfy that requirement with any type of session-based behavior it chooses—security, transport, reliable, or some combination. The exact behavior depends on the **System.ServiceModel.SessionMode** value selected. If the configured binding of the service does not conform to the value of **SessionMode**, an exception is thrown. Bindings and the channels they create that support sessions are said to be session-based.

The following service contract specifies that all operations in the ICalculatorSession must be exchanged within a session. None of the operations returns a value to the caller except the Equals method. However, the Equals method takes no parameters and, therefore, can only return a non-zero value inside a session in which data has already been passed to the other operations. This contract requires a session to function properly. Without a session associated with a specific client, the service instance has no way of knowing what previous data this client has sent.

Visual Basic

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

<ServiceContract(Namespace:="http://Microsoft.ServiceModel.Samples", SessionMode:=SessionMode.Required)> \_

Public Interface ICalculatorSession

<OperationContract(IsOneWay:=True)> \_

Sub Clear()

<OperationContract(IsOneWay:=True)> \_

Sub AddTo(ByVal n As Double)

<OperationContract(IsOneWay:=True)> \_

Sub SubtractFrom(ByVal n As Double)

<OperationContract(IsOneWay:=True)> \_

Sub MultiplyBy(ByVal n As Double)

<OperationContract(IsOneWay:=True)> \_

Sub DivideBy(ByVal n As Double)

<OperationContract()> \_

Function Equal() As Double

End Interface

C#

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

[ServiceContract(Namespace="http://Microsoft.ServiceModel.Samples", SessionMode=SessionMode.Required)]

public interface ICalculatorSession

{

[OperationContract(IsOneWay=true)]

void Clear();

[OperationContract(IsOneWay = true)]

void AddTo(double n);

[OperationContract(IsOneWay = true)]

void SubtractFrom(double n);

[OperationContract(IsOneWay = true)]

void MultiplyBy(double n);

[OperationContract(IsOneWay = true)]

void DivideBy(double n);

[OperationContract]

double Equals();

}

If a service allows a session, then a session is established and used if the client initiates one; otherwise, no session is established.

**Sessions and Service Instances**

If you use the default instancing behavior in WCF, all calls between a WCF client object are handled by the same service instance. Therefore, at the application level, you can think of a session as enabling application behavior similar to local call behavior. For example, when you create a local object:

* A constructor is called.
* All subsequent calls made to the WCF client object reference are processed by the same object instance.
* A destructor is called when the object reference is destroyed.

Sessions enable a similar behavior between clients and services as long as the default service instance behavior is used. If a service contract requires or supports sessions, one or more contract operations can be marked as initiating or terminating a session by setting the [IsInitiating](http://msdn.microsoft.com/en-us/library/system.servicemodel.operationcontractattribute.isinitiating.aspx) and [IsTerminating](http://msdn.microsoft.com/en-us/library/system.servicemodel.operationcontractattribute.isterminating.aspx) properties.

*Initiating operations* are those that must be called as the first operation of a new session. Non-initiating operations can be called only after at least one initiating operation has been called. You can therefore create a kind of session constructor for your service by declaring initiating operations designed to take input from clients appropriate to the beginning of the service instance. (The state is associated with the session, however, and not the service object.)

*Terminating operations*, conversely, are those that must be called as the last message in an existing session. In the default case, WCF recycles the service object and its context after the session with which the service was associated is closed. You can, therefore, create a kind of destructor by declaring terminating operations designed to perform a function appropriate to the end of the service instance.

|  |
| --- |
| **Description: ms733040.note(en-us,VS.100).gifNote:** |
| Although the default behavior bears a resemblance to local constructors and destructors, it is only a resemblance. Any WCF service operation can be an initiating or terminating operation, or both at the same time. In addition, in the default case, initiating operations can be called any number of times in any order; no additional sessions are created once the session is established and associated with an instance unless you explicitly control the lifetime of the service instance (by manipulating the [System.ServiceModel.InstanceContext](http://msdn.microsoft.com/en-us/library/system.servicemodel.instancecontext.aspx) object). Finally, the state is associated with the session and not the service object. |

For example, the ICalculatorSession contract used in the preceding example requires that the WCF client object first call the Clear operation prior to any other operation and that the session with this WCF client object should terminate when it calls the Equals operation. The following code example shows a contract that enforces these requirements. Clear must be called first to initiate a session, and that session ends when Equals is called.

Visual Basic

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

<ServiceContract(Namespace:="http://Microsoft.ServiceModel.Samples", SessionMode:=SessionMode.Required)> \_

Public Interface ICalculatorSession

<OperationContract(IsOneWay:=True, IsInitiating:=True, IsTerminating:=False)> \_

Sub Clear()

<OperationContract(IsOneWay:=True, IsInitiating:=False, IsTerminating:=False)> \_

Sub AddTo(ByVal n As Double)

<OperationContract(IsOneWay:=True, IsInitiating:=False, IsTerminating:=False)> \_

Sub SubtractFrom(ByVal n As Double)

<OperationContract(IsOneWay:=True, IsInitiating:=False, IsTerminating:=False)> \_

Sub MultiplyBy(ByVal n As Double)

<OperationContract(IsOneWay:=True, IsInitiating:=False, IsTerminating:=False)> \_

Sub DivideBy(ByVal n As Double)

<OperationContract(IsInitiating:=False, IsTerminating:=True)> \_

Function Equal() As Double

End Interface

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl20_code');" \o "Copy Code)

[ServiceContract(Namespace="http://Microsoft.ServiceModel.Samples", SessionMode=SessionMode.Required)]

public interface ICalculatorSession

{

[OperationContract(IsOneWay=true, IsInitiating=true, IsTerminating=false)]

void Clear();

[OperationContract(IsOneWay = true, IsInitiating = false, IsTerminating = false)]

void AddTo(double n);

[OperationContract(IsOneWay = true, IsInitiating = false, IsTerminating = false)]

void SubtractFrom(double n);

[OperationContract(IsOneWay = true, IsInitiating = false, IsTerminating = false)]

void MultiplyBy(double n);

[OperationContract(IsOneWay = true, IsInitiating = false, IsTerminating = false)]

void DivideBy(double n);

[OperationContract(IsInitiating = false, IsTerminating = true)]

double Equals();

}

Services do not start sessions with clients. In WCF client applications, a direct relationship exists between the lifetime of the session-based channel and the lifetime of the session itself. As such, clients create new sessions by creating new session-based channels and tear down existing sessions by closing session-based channels gracefully. A client starts a session with a service endpoint by calling one of the following:

* [System.ServiceModel.ICommunicationObject.Open](http://msdn.microsoft.com/en-us/library/ms195524.aspx) on the channel returned by a call to [System.ServiceModel.ChannelFactory.CreateChannel](http://msdn.microsoft.com/en-us/library/ms575250.aspx).
* [System.ServiceModel.ClientBase.Open](http://msdn.microsoft.com/en-us/library/ms575279.aspx) on the WCF client object generated by the [ServiceModel Metadata Utility Tool (Svcutil.exe)](http://msdn.microsoft.com/en-us/library/aa347733.aspx).
* An initiating operation on either type of WCF client object (by default, all operations are initiating). When the first operation is called, the WCF client object automatically opens the channel and initiates a session.

Typically a client ends a session with a service endpoint by calling one of the following:

* [System.ServiceModel.ICommunicationObject.Close](http://msdn.microsoft.com/en-us/library/ms195520.aspx) on the channel returned by a call to **System.ServiceModel.ChannelFactory.CreateChannel**.
* [System.ServiceModel.ClientBase.Close](http://msdn.microsoft.com/en-us/library/ms575273.aspx) on the WCF client object generated by Svcutil.exe.
* A terminating operation on either type of WCF client object (by default, no operations are terminating; the contract must explicitly specify a terminating operation). When the first operation is called, the WCF client object automatically opens the channel and initiates a session.

For examples, see [How to: Create a Service That Requires Sessions](http://msdn.microsoft.com/en-us/library/ms733058.aspx) as well as the [Default Service Behavior](http://msdn.microsoft.com/en-us/library/ms751438.aspx) and [Instancing](http://msdn.microsoft.com/en-us/library/ms752230.aspx) samples.

For more information about clients and sessions, see [Accessing Services Using a Client](http://msdn.microsoft.com/en-us/library/ms733912.aspx).

**Sessions Interact with InstanceContext Settings**

There is an interaction between the **SessionMode** enumeration in a contract and the [System.ServiceModel.ServiceBehaviorAttribute.InstanceContextMode](http://msdn.microsoft.com/en-us/library/system.servicemodel.servicebehaviorattribute.instancecontextmode.aspx) property, which controls the association between channels and specific service objects. For more information, see [Sessions, Instancing, and Concurrency](http://msdn.microsoft.com/en-us/library/ms731193.aspx).

**Sharing InstanceContext Objects**

You can also control which session-based channel or call is associated with which **InstanceContext** object by performing that association yourself. For a complete example, see [InstanceContextSharing](http://msdn.microsoft.com/en-us/library/aa354514.aspx).

**Sessions and Streaming**

When you have a large amount of data to transfer, the streaming transfer mode in WCF is a feasible alternative to the default behavior of buffering and processing messages in memory in their entirety. You may get unexpected behavior when streaming calls with a session-based binding. All streaming calls are made through a single channel (the datagram channel) that does not support sessions even if the binding being used is configured to use sessions. If multiple clients make streaming calls to the same service object over a session-based binding, and the service object's concurrency mode is set to single and its instance context mode is set to **PerSession**, all calls must go through the datagram channel and so only one call is processed at a time. One or more clients may then time out. You can work around this issue by either setting the service object's **InstanceContextMode** to **PerCall** or Concurrency to multiple.

|  |
| --- |
| **Description: ms733040.note(en-us,VS.100).gifNote:** |
| MaxConcurrentSessions have no effect in this case because there is only one "session" available. |