Sessions, Instancing, and Concurrency

.NET Framework (current version)

A *session* is a correlation of all messages sent between two endpoints. *Instancing* refers to controlling the lifetime of user-defined service objects and their related InstanceContext objects. *Concurrency* is the term given to the control of the number of threads executing in an InstanceContext at the same time.

This topic describes these settings, how to use them, and the various interactions between them.

Sessions

When a service contract sets the ServiceContractAttribute.SessionMode property to SessionMode.Required, that contract is saying 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. If the session ends and a message is sent over the same session-based channel an exception is thrown.

WCF sessions have the following main conceptual features:

- They are explicitly initiated and terminated by the calling application.
- Messages delivered during a session are processed in the order in which they are received.
- Sessions correlate a group of messages into a conversation. The meaning of that correlation is an abstraction. 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 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.

Client applications and service applications interact with sessions in different ways. Client applications initiate sessions and then receive and process the messages sent within the session. Service applications can use sessions as an extensibility point to add additional behavior. This is done by working directly with the InstanceContext or implementing a custom instance context provider.

Instancing

The instancing behavior (set by using the ServiceBehaviorAttribute.InstanceContextMode property) controls how the InstanceContext is created in response to incoming messages. By default, each InstanceContext is associated with one user-defined service object, so (in the default case) setting the InstanceContextMode property also controls the instancing of user-defined service objects. The InstanceContextMode enumeration defines the instancing modes.

The following instancing modes are available:

- PerCall: A new InstanceContext (and therefore service object) is created for each client request.
- PerSession: A new InstanceContext (and therefore service object) is created for each new client session and maintained for the lifetime of that session (this requires a binding that supports sessions).
- Single: A single InstanceContext (and therefore service object) handles all client requests for the lifetime of the application.

The following code example shows the default InstanceContextMode value, PerSession being explicitly set on a service class.

```
[ServiceBehavior(InstanceContextMode=InstanceContextMode.PerSession)]
public class CalculatorService : ICalculatorInstance
{
    ...
}
```

And while the ServiceBehaviorAttribute.InstanceContextMode property controls how often the InstanceContext is released, the OperationBehaviorAttribute.ReleaseInstanceMode and ServiceBehaviorAttribute.ReleaseServiceInstanceOnTransactionComplete properties control when the service object is released.

Well-Known Singleton Services

One variation on single instance service objects is sometimes useful: you can create a service object yourself and create the service host using that object. To do so, you must also set the ServiceBehaviorAttribute.InstanceContextMode property to Single or an exception is thrown when the service host is opened.

Use the ServiceHost.ServiceHost(Object, Uri[]) constructor to create such a service. It provides an alternative to implementing a custom System.ServiceModel.Dispatcher.IInstanceContextInitializer when you wish to provide a specific object instance for use by a singleton service. You can use this overload when your service implementation type is difficult to construct (for example, if it does not implement a default parameterless public constructor).

Note that when an object is provided to this constructor, some features related to the Windows Communication Foundation (WCF) instancing behavior work differently. For example, calling InstanceContext.ReleaseServiceInstance has no effect when a singleton object instance is provided. Similarly, any other instance-release mechanism is ignored. The ServiceHost always behaves as if the OperationBehaviorAttribute.ReleaseInstanceMode property is set to ReleaseInstanceMode.None for all operations.

Sharing InstanceContext Objects

You can also control which sessionful channel or call is associated with which InstanceContext object by performing that association yourself.

Concurrency

Concurrency is the control of the number of threads active in an InstanceContext at any one time. This is controlled by using the ServiceBehaviorAttribute.ConcurrencyMode with the ConcurrencyMode enumeration.

The following three concurrency modes are available:

- Single: Each instance context is allowed to have a maximum of one thread processing messages in the instance context at a time. Other threads wishing to use the same instance context must block until the original thread exits the instance context.
- Multiple: Each service instance can have multiple threads processing messages concurrently. The service implementation must be thread-safe to use this concurrency mode.
- Reentrant: Each service instance processes one message at a time, but accepts re-entrant operation calls. The service only accepts these calls when it is calling out through a WCF client object.

Note

Understanding and developing code that safely uses more than one thread can be difficult to write successfully. Before using Multiple or Reentrant values, ensure that your service is properly designed for these modes. For more information, seeConcurrencyMode.

The use of concurrency is related to the instancing mode. In PerCallinstancing, concurrency is not relevant, because each message is processed by a new InstanceContext and, therefore, never more than one thread is active in the InstanceContext.

The following code example demonstrates setting the ConcurrencyMode property to Multiple.

```
[ServiceBehavior(ConcurrencyMode=ConcurrencyMode.Multiple, InstanceContextMode =
InstanceContextMode.Single)]
public class CalculatorService : ICalculatorConcurrency
{
    ...
}
```

Sessions Interact with InstanceContext Settings

Sessions and InstanceContext interact depending upon the combination of the value of the SessionMode enumeration in a contract and the ServiceBehaviorAttribute.InstanceContextMode property on the service implementation, which controls the association between channels and specific service objects.

The following table shows the result of an incoming channel either supporting sessions or not supporting sessions given a service's combination of the values of the ServiceContractAttribute.SessionMode property and the ServiceBehaviorAttribute.InstanceContextMode property.

InstanceContextMode	**Required**	**Allowed**	**NotAllowed**

value			
PerCall	- Behavior with sessionful channel: A session and InstanceContext for each call. - Behavior with sessionless channel: An exception is thrown.	- Behavior with sessionful channel: A session and InstanceContext for each call. - Behavior with sessionless channel: An InstanceContext for each call.	- Behavior with sessionful channel: An exception is thrown. - Behavior with sessionless channel: An InstanceContext for each call.
PerSession	- Behavior with sessionful channel: A session and InstanceContext for each channel. - Behavior with sessionless channel: An exception is thrown.	- Behavior with sessionful channel: A session and InstanceContext for each channel. - Behavior with sessionless channel: An InstanceContext for each call.	- Behavior with sessionful channel: An exception is thrown. - Behavior with sessionless channel: An InstanceContext for each call.
Single	- Behavior with sessionful channel: A session and one InstanceContext for all calls. - Behavior with sessionless channel: An exception is thrown.	- Behavior with sessionful channel: A session and InstanceContext for the created or user-specified singleton. - Behavior with sessionless channel: An InstanceContext for the created or user-specified singleton.	- Behavior with sessionful channel: An exception is thrown. - Behavior with sessionless channel: An InstanceContext for each created singleton or for the user-specified singleton.

See Also

Using Sessions
How to: Create a Service That Requires Sessions
How to: Control Service Instancing
Concurrency
Instancing
Session