This page is specific to

Microsoft Visual Studio 2005/.NET Framework 2.0

.NET Framework Developer's Guide

**Object Pooling**

The COM+ Object Pooling service enables you to avoid the overhead of creating each object from scratch. When an object is activated, it is pulled from the pool. When the object is deactivated, it is placed back into the pool to await the next request.

You can configure object pooling by applying the [ObjectPoolingAttribute](http://msdn.microsoft.com/en-us/library/system.enterpriseservices.objectpoolingattribute(VS.80).aspx) attribute to a class that derives from the [System.EnterpriseServices.ServicedComponent](http://msdn.microsoft.com/en-us/library/system.enterpriseservices.servicedcomponent(VS.80).aspx) class.

The procedure for applying the [ObjectPoolingAttribute](http://msdn.microsoft.com/en-us/library/system.enterpriseservices.objectpoolingattribute(VS.80).aspx) and setting its properties is described in [How to: Create a Pooled Object and Set Its Size and Time-out Limits](http://msdn.microsoft.com/en-us/library/sbbb7sce(VS.80).aspx).

Object pooling lets you control the number of connections you use, as opposed to connection pooling, where you control the maximum number reached. Following are important differences between object pooling and connection pooling:

* Creation. When using connection pooling, creation is on the same thread, so if there is nothing in the pool, a connection is created on your behalf. With object pooling, the pool can create a new object. However, if you have already reached your maximum, it instead gives you the next available object. This is crucial behavior when it takes a long time to create an object, but do not use it for very long.
* Enforcement of minimums and maximums. This is not done in connection pooling. The maximum value in object pooling is very important when trying to scale your application. You might need to multiplex thousands of requests to just a few objects. (TPC/C benchmarks rely on this.)

COM+ object pooling is practically identical to .NET Framework managed SQL Client connection pooling. For example, creation is on a different thread and minimums and maximums are enforced.

|  |
| --- |
| **NoteNote** |
| Application domains affect the behavior of object pooling. In Windows 2000, when application activation is set to Library and you have multiple application domains, pooled objects are all created in the default application domain and shared between multiple clients. In the same situation, using Windows XP and Windows Server 2003, there is one object pool per application domain. Using either operating system with multiple application domains and your application activation set to server, out-of-process clients use the object pool in the default application domain. |
| **NoteNote** |
| In general, you do not have to call DisposeObject from the client when using serviced components. However, it is necessary when you are using the COM+ Object Pooling service while the Just-in-Time (JIT) Activation service is not enabled. In this case, to verify that it is safe to return the object to the pool, COM+ must be notified when you are finished with the object. In general, if you intend to make only one call at a time on a pooled object, it is a good idea to enable JIT activation with object pooling. If you intend to get a reference and make multiple calls on it, using object pooling without JIT activation may result in better performance. |

This page is specific to

Microsoft Visual Studio 2008/.NET Framework 3.5

.NET Framework Developer's Guide

**How to: Create a Pooled Object and Set Its Size and Time-out Limits**

[Note: This topic is pre-release documentation and is subject to change in future releases. Blank topics are included as placeholders.]

For a class that derives from the [System.EnterpriseServices.ServicedComponent](http://msdn.microsoft.com/en-us/library/system.enterpriseservices.servicedcomponent(VS.90).aspx) class, COM+ Object Pooling can be used to avoid the overhead of instantiating objects from scratch. Instead objects get pulled from a pool when activated. For more information, see [Object Pooling](http://msdn.microsoft.com/en-us/library/ff7105zk(VS.90).aspx).

**To create a pooled object and set its size and time-out limits**

1. Define a class that derives from the [System.EnterpriseServices.ServicedComponent](http://msdn.microsoft.com/en-us/library/system.enterpriseservices.servicedcomponent(VS.90).aspx) class, and apply the [ObjectPoolingAttribute](http://msdn.microsoft.com/en-us/library/system.enterpriseservices.objectpoolingattribute(VS.90).aspx) attribute to the class. For example, the following code defines a class named TestObjectPooling and sets the [MinPoolSize](http://msdn.microsoft.com/en-us/library/system.enterpriseservices.objectpoolingattribute.minpoolsize(VS.90).aspx), [MaxPoolSize](http://msdn.microsoft.com/en-us/library/system.enterpriseservices.objectpoolingattribute.maxpoolsize(VS.90).aspx) and [CreationTimeout](http://msdn.microsoft.com/en-us/library/system.enterpriseservices.objectpoolingattribute.creationtimeout(VS.90).aspx) properties for the class.

Visual Basic

<ObjectPooling(MinPoolSize := 2, MaxPoolSize := 5, \_

CreationTimeout := 20000)> \_

Public Class TestObjectPooling

Inherits ServicedComponent

End Class

C#

[ObjectPooling(Enabled=true, MinPoolSize=2, MaxPoolSize=5, CreationTimeout=20000)]

public class TestObjectPooling : ServicedComponent

{

}

1. Override the [Activate()](http://msdn.microsoft.com/en-us/library/system.enterpriseservices.servicedcomponent.activate(VS.90).aspx) , [Deactivate()](http://msdn.microsoft.com/en-us/library/system.enterpriseservices.servicedcomponent.deactivate(VS.90).aspx), and [CanBePooled()](http://msdn.microsoft.com/en-us/library/system.enterpriseservices.servicedcomponent.canbepooled(VS.90).aspx) methods of the [System.EnterpriseServices.ServicedComponent](http://msdn.microsoft.com/en-us/library/system.enterpriseservices.servicedcomponent(VS.90).aspx) class.
2. Test the pooled object, in a client application:
   1. Create an instance of the pooled object class and call the methods on the pooled object. For example, the following code creates an instance of the TestObjectPooling class and calls a Perform method.

Visual Basic

Public Class App

Overloads Public Shared Sub Main(args() As String)

Dim order As New TestObjectPooling()

order.Perform()

C#

public class App

{

public static int Main(string[] args)

{

TestObjectPooling order = new TestObjectPooling();

order.Perform();

* 1. Call the DisposeObject method to return the object to the pool.

Visual Basic

ServicedComponent.DisposeObject (order)

C#

ServicedComponent.DisposeObject (order);

http://i.msdn.microsoft.com/Global/Images/clear.gifExample

Visual Basic

<ObjectPooling(MinPoolSize := 2, MaxPoolSize := 5, \_

CreationTimeout := 20000)> \_

Public Class TestObjectPooling

Inherits ServicedComponent

Public Sub Perform ()

' Method contents go here.

End Sub

Protected Overrides Sub Activate()

' Called when removed from the pool.

End Sub

Protected Overrides Sub Deactivate()

' Called before deactivating or placing back in pool.

End Sub

Protected Overrides Function CanBePooled() As Boolean

' Called after Deactivate. Indicate your vote here.

Return True

End Function

End Class

C#

[ObjectPooling(Enabled=true, MinPoolSize=2, MaxPoolSize=5, CreationTimeout=20000)]

public class TestObjectPooling : ServicedComponent

{

public void Perform ()

{

// Method contents go here.

}

protected override void Activate()

{

// Called when removed from the pool.

}

protected override void Deactivate()

{

// Called before deactivating or placing back in pool.

}

protected override bool CanBePooled()

{

// Called after Deactivate. Indicate your vote here.

return true;

}

}