Contents

There are several timeouts which can affect your application. Learning to **understand and troubleshoot timeouts** will help to quickly diagnose and fix problems.

If you've received a "timeout" exception, the first question to ask is "Where is it coming from?" In a n-tier application the timeout might come from the database server, the communications infrastructure (WCF), or even IIS. In a 2-tier application not hosted in ASP.NET, the timeout exception will originate from the database.

Any exception you receive will be an *EntityServerException* or one of its subtypes.

Database / transaction timeouts

You might receive a database-related timeout during either a query or save.

Query

- You receive an *EntityServerException* with a *FailureType* of *Data* and *OperationType* of *Query*. The message itself may be less than helpful, it might say something like "An error occurred while executing the command definition...", or maybe it will say "Timeout expired". In a 2-tier application you can drill into the inner exceptions to see the underlying database exception and message. In SQL Server that will be a *System.Data.SqlClient.SqlException* with a message which begins with "Timeout expired." In an n-tier application, you'll need to check the *RemoteExceptionName* and *RemoteExceptionDetails* to find that information.

  To fix this problem, modify the query's *CommandTimeout*. Every query contains a *CommandTimeout* property which you can set to override the provider's default query timeout value. With SQL Server, that default timeout is 30 seconds.

- If the *CommandTimeout* is sufficient but the *transaction timeout* is not, you'll get an *EntityServerException* with a message stating either "The transaction has aborted" or "Transaction timeout". In some cases the error message will state "The transaction associated with the current connection has completed but has not been disposed." The *FailureType* is *Other* in all cases.

  By default, queries will use *TransactionSettings* with a *UseTransactionScope* option set to true, which indicates that the query should be wrapped in a TransactionScope. When a query is wrapped in a *TransactionScope* then the *Timeout* value on the *TransactionSettings* determines the allowed time for the overall transaction. The *TransactionSettings* can be passed with the QueryStrategy or set on the *EntityManager.DefaultQueryStrategy*. The default timeout value is 1 minute. If your query does not require a *TransactionScope* you can set *UseTransactionScope* to false.

Saves

- You receive an *EntityManagerSaveException* with a *FailureType* of *Data* and an *OperationType* of *Save*. You'll need to drill into the details of the exception to see the underlying *System.TimeoutException* with the "Transaction Timeout" message.

  You cannot set the command timeout for individual inserts, updates and deletes performed as part of the save processing. Here only the *TransactionSettings.Timeout* can be used to alter the processing timeout. You can set the desired timeout on the SaveOptions passed from the client, or on the *EntityServer* with the *TransactionSettings.Default* singleton. The default timeout is 1 minute.

- You receive an *EntityManagerSaveException* with a *FailureType* of *Other* and an *OperationType* of *Save*. The error message states "The transaction associated with the current connection has completed but has not been disposed."

  This will occur when the outer *TransactionScope* times out before the inner database transaction completes. To resolve the problem set a larger timeout value on the SaveOptions passed from the client, or on the *EntityServer* with the *TransactionSettings.Default* singleton. The default timeout is 1 minute.

Communication timeouts

Most communication-related timeout error messages will state something along the lines of "The HTTP request to 'http://yourserver/EntityServer.svc' has exceeded the allotted timeout." If you drill into the exception details you'll usually see a *System.TimeoutException* wrapping a *System.Net.WebException*.

You might get a timeout for any type of request to the *EntityServer*, and they can occur regardless of whether the request was synchronous or asynchronous.

The default *SendTimeout* on the client is 1 minute, and it's this timeout that is most often the cause of communication timeouts. You can override the default value in two ways - via either configuration or code. See the advanced configuration topic for more information.

There are additional timeout settings available on both client and server, but they don't often need to be adjusted from their default values. If you are saving large amounts of data then you might find that in addition to the client's *SendTimeout* you also need to adjust the server's *ReceiveTimeout*.

In your 2-tier application you obviously don't need to be concerned with communications exceptions.

IIS

Even when your WCF timeouts seem sufficient, you might receive an exception stating that "The operation has timed out.". The problem might be in another setting applicable only when hosting the *EntityServer* under IIS, the *executionTimeout*. The default value is 110 seconds, and can be modified via the web.config:

```
<system.web>
  <httpRuntime executionTimeout="110"/>
</system.web>
```