**Contents**



This topic covers how an ***EntityManager*** can lose its connection to the server and **reconnect** later.

# Connected by default

By default, an *EntityManager* automatically and immediately tries to connect to the server upon construction; ordinarily you do not have to initiate a connection explicitly.

# Becoming disconnected

Various circumstances may cause an *EntityManager* to become disconnected.

1. You deliberately create an offline manager.
2. You decide to take the manager offline by calling its *Disconnect* method.
3. The application loses its connection to the server as it began a server operation or in the midst of a server operation.

In all three cases, the *EntityManager* enters a disconnected state, a fact easily determined by accessing its *IsConnected* property. You can also listen on the *ConnectionStateChanged* event.

The *IsConnected* property reports whether the manager "believes" it is connected to the *EntityServer*. It may "think" it is connected when, in fact, it could not reach the server if it tried.

# Losing the connection

The application decided to disconnect in examples #1 and #2. In the 3rd example, the application loses the connection involuntarily and without warning.

The *EntityManager* responds by setting itself in the disconnected state and then prepares to throw an *EntityServerConnectionException*. You can (and should) be ready to intercept and handle that exception by attaching a handler to the manager's *EntityServerError* event.

See the the topic devoted to *handling the EntityServerError event*.

Your handler next decides how to proceed. Rather than shut down, you may choose to tell the rest of the application about the problem and continue running in a well-defined offline mode.

# Anticipating connectivity problems

It is wise to listen to the .NET event that announces the gain and loss of network connectivity. The requisite event differs from one client platform to the next but there always is such an event.

However, do not depend upon it to reflect the true and complete state of affairs. A flakey connection could break in the middle of an attempt to connect. The network connection may be good but the middle tier server could be down. The *EntityManager* will disconnect and prepare to throw an exception whatever the cause. You can glean the details from the exception in the body of the *EntityServerErrorEventArgs*.

Your application could lose contact with the server at any time. Plan for it. Remember to handle the *EntityServerError* event if you want your application to survive a disconnection or if you prefer to shutdown gracefully.

# Restore the connection

An *EntityManager* will not reconnect spontaneously. Once offline, it stays offline until you reconnect explicitly. A DevForce application with a well-stocked entity cache can operate offline effectively for a long time, albeit with limited functionality.

Your application must decide when to try to reconnect. It may decide to try again if a .NET event signals restored network access, or a retry timer fires, or in response to a user action.

Whatever the trigger, your code must call the *EntityManager*'s *Connect* or *ConnectAsync* methods.

As always, code defensively and be prepared for connectivity failures.