

In [Chapter 9](#), "Error Handling and Concurrency Control," I will present a solution that demonstrates an administrator-friendly technique for providing pessimistic locking in a disconnected scenario.

Reliability

Reliability is often discussed in terms of fault tolerance and robustness, but it is also looked at in terms of recoverability. Hardware can provide you with some fault tolerance, but in this book, I will only discuss software aspects of reliability. In an effort to avoid confusion due to the myriad terms that are used in the industry, I'll discuss reliability in terms of

- Using preventative measures
- Actively taking care of a problem
- Using recovery/fail over when a problem occurs

Using Preventative Measures

As I'm sure you know, instead of letting a problem occur before taking care of it, you can take precautions to make it less probable that a problem will occur at all. For example, most of us would rather change the oil in our car's engine every now and then instead of replacing the engine when it crashes because of bad oil or lack of oil. A typical example of a preventative measure in a technical sense is to use the new feature in COM+ 1.5 to request recycling of your server processes, say, once a day.

Actively Taking Care of a Problem

You can also think of good error handling with automatic retry mechanisms as a way of achieving fault tolerance. It's no use that the user has to click a Retry button to make a new try because there was a locking conflict; the program should instead retry several times on its own. Another proposal is to use message queuing often. If the message has been written to the queue, many retries can occur.

Another way to see it is that the system will be able to cope, for example, with incorrect input, and will just tell the client that the input wasn't acceptable, without going to an unexpected state.

Note

I will address such reliability aspects in [Chapter 3](#), where I discuss assertions and design by contract.

Using Recovery/Fail Over When a Problem Occurs

.NET Component Services will help a great deal when problems occur. For example, if there is a serious exception, the process will be nailed down and a fresh process will automatically be started. Meanwhile, one way to get safe recoverability is to use a transaction log, as SQL Server 2000 does. Another way is to expect fail over to do the job; that is, when one machine faults, another machine takes over. A typical product that helps with fail over support is Microsoft Cluster Services in Windows 2000.

Note