Transaction Handling

Miguel A. Castro
@miguelcastro67

pluralsight
hardcore developer training

The Need for Transactions

- **An operation call must maintain a consistent state**
  - Either the original state or a new one (in full)
  - Nothing in between
- **.NET supports transactional programming since ADO.NET 2.0**
- **WCF is a fully qualified transaction resource manager**
  - SQL Server
  - Oracle
- **With DTC, can fully contribute to distributed transactions**
  - Transactions that span multiple machines
  - From client, down to database
- **All transactional resources vote on transaction**
- **Votes must be unanimous to commit transaction**
  - Rollback will reverse everything involved

Transaction Support in WCF

- **Binding Switch**
  - **TransactionFlow** must be allowed on the binding
  - Consider this the transaction “master switch”
  - Refers to “flow” only but should be turned on for any usage

Transaction Support in WCF

- **Transaction Flow**
  - Allows, disallows, or mandates an existing transaction from entering a service operation *(from any proxy-caller)*
  - **TransactionFlow** attribute on operation contract
    - Allowed
    - NotAllowed (default)
    - Mandatory

Transaction Support in WCF

- **Transaction Scope**
  - Dictates whether or not a service operation will participate in a transaction
  - Which transaction depends on “flow” and/or presence of transaction
  - Set with **TransactionScope** property on **OperationBehavior** attribute
  - If “true”
    - If existing transaction allowed (and exists), operation will vote on it
    - If no existing transaction (or not allowed), WCF will start one

Demo Time

Transaction Voting

- **Takes place automatically by default**
  - **TransactionAutoComplete** property of **OperationBehavior** attribute
  - Value defaults to true
  - Operation votes to fail if exception
  - Can also fail using **Transaction.Current.Rollback()**
- **Can be turned off by setting to false**
  - Transaction will vote to fail by default
  - Can use **OperationContext.Current.SetTransactionComplete()**

Demo Time

Manual Transaction Programming

- **Sometimes need to handle transactions manually**
  - Suppress transaction from some code
  - Isolate various areas of code into separate transactions
- **Use System.Transactions.TransactionScope object manually**
  - Can start new transaction
  - Suppress any existing transaction
  - Join any existing transaction

Demo Time

Client Transactions

- **Remember, client can be anything (including a service)**
- **If client is service with WCF handling transactions**
  - Not much work to do
  - WCF will handle flowing (attributes apply) and voting
- **If client is using manual transaction programming**
  - Per-Call
    - Each call uses new service instance
    - Client can make one or more call, then close transaction
  - Per-Session
    - Client must close session (proxy) inside the transaction
    - Make one or more call (to same instance of course), then close proxy
    - Then close transaction

Demo Time