How do I handle message failure in MSMQ bindings for WCF

Asked 16 years, 3 months ago Modified 12 years, 6 months ago

Viewed 14k times

17

I have create a WCF service and am utilising netMsmqBinding binding.

This is a simple service that passes a Dto to my service method and does not expect a response. The message is placed in an MSMQ, and once picked up inserted into a database.

What is the best method to make sure no data is being lost.

I have tried the 2 following methods:

1. Throw an exception

   This places the message in a dead letter queue for manual perusal. I can process this when my strvice starts

2. set the receiveRetryCount="3" on the binding

   After 3 tries - which happen instantanously, this seems to leave the message in queue, but fault my service. Restarting my service repeats this process.

Ideally I would like to do the follow:

Try process the message

- If this fails, wait 5 minutes for that message and try again.
- If that process fails 3 times, move the message to a dead letter queue.
- Restarting the service will push all messages from the dead letter queue back into the queue so that it can be processed.

Can I achieve this? If so how? Can you point me to any good articles on how best to utilize WCF and MSMQ for my given sceneria.

Any help would be much appreciated. Thanks!

Some additional information

I am using MSMQ 3.0 on Windows XP and Windows Server 2003. Unfortunately I can't use the built in poison message support targeted at MSMQ 4.0 and Vista/2008.

.net **wcf** **msmq**

Share

Improve this question

Follow

edited Jun 20, 2020 at 9:12

Community Bot

1 • 1

asked Sep 17, 2008 at 11:15

WebDude

6,205 • 6 • 35 • 44

4 Answers

Sorted by: Highest score (default)

14

I think with MSMQ (avaiable only on Vista) you might be able to to do like this:

```
<bindings>
    <netMsmqBinding>
        <binding name="PosionMessageHandling"
             receiveRetryCount="3"
             retryCycleDelay="00:05:00"
             maxRetryCycles="3"
             receiveErrorHandling="Move" />
    </netMsmqBinding>
</bindings>
```

WCF will immediately retry for ReceiveRetryCount times after the first call failure. After the batch has failed the message is moved to the retry queue. After a delay of RetryCycleDelay minute, the message moved from the retry queue to the endpoint queue and the batch is retried. This will be repeated MaxRetryCycle time. If all that fails the message is handled according to receiveErrorHandling which can be move (to poison queue), reject, drop or fault

By the way a good text about WCF and MSMQ is the chapther 9 of Progammig WCF book from Juval Lowy

Share Improve this answer

Follow

edited Nov 11, 2009 at 16:26

John Sibly

23k ● 7 ● 64 ● 82

answered Sep 17, 2008 at 12:29

λ

aogan

2,261 ● 1 ● 15 ● 24

This applies to MSMQ 4.0 not MSMQ 3.0 – Perhentian Nov 16, 2011 at 12:34

9

There's a sample in the SDK that might be useful in your case. Basically, what it does is attach an IErrorHandler implementation to your service that will catch the error when WCF declares the message to be "poison" (i.e. when all configured retries have been exhausted). What the sample does is move the message to another queue and then restart the ServiceHost associated with the message (since it will have faulted when the poison message was found).

It's not a very pretty sample, but it can be useful. There are a couple of limitations, though:

1- If you have multiple endpoints associated with your service (i.e. exposed through several queues), there's no way to know which queue the poison message arrived in.

If you only have a single queue, this won't be a problem. I haven't seen any official workaround for this, but I've experimented with one possible alternative which I've documented here: http://winterdom.com/weblog/2008/05/27/NetMSMQAndPoisonMessages.aspx

2- Once the problem message is moved to another queue, it becomes your responsibility, so it's up to you to move it back to the processing queue once the timeout is done (or attach a new service to that queue to handle it).

To be honest, in either case, you're looking at some "manual" work here that WCF just doesn't cover on it's own.

I've been recently working on a different project where I have a requirement to explicitly control how often retries happen, and my current solution was to create a set of retry queues and manually move messages between the retry queues and the main processing queue based on a set of timers and some heuristics, just using the raw System.Messaging stuff to handle the MSMQ queues. It seems to work pretty nicely, though there are a couple of gotchas if you go this way.

answered Sep 17, 2008 at 17:36

tomasr

13.8k ● 3 ● 41 ● 30

4

If you're using SQL-Server then you should use a distributed transaction, since both MSMQ and SQL-Server support it. What happens is you wrap your database write in a TransactionScope block and call scope.Complete() only if it succeeds. If it fails, then when your WCF method returns the message will be placed back into the queue to be tried again. Here's a trimmed version of code I use:

```
    [OperationBehavior(TransactionScopeRequired=true,
TransactionAutoComplete=true)]
    public void InsertRecord(RecordType record)
    {
        try
        {
            using (TransactionScope scope = new
TransactionScope(TransactionScopeOption.Required))
            {
                SqlConnection InsertConnection = new
SqlConnection(ConnectionString);
                InsertConnection.Open();

                // Insert statements go here

                InsertConnection.Close();

                // Vote to commit the transaction if t
                scope.Complete();
            }
        }
        catch (Exception ex)
        {
            logger.WarnException(string.Format("Distri
for {0}",

Transaction.Current.TransactionInformation.Distributed
                ex);
```

```
        }
    }
```

I test this by queueing up a large but known number of records, let WCF start lots of threads to handle many of them simultaneously (reaches 16 threads--16 messages off the queue at once), then kill the process in the middle of operations. When the program is restarted the messages are read back from the queue and processed again as if nothing happened, and at the conclusion of the test the database is consistent and has no missing records.

The Distributed Transaction Manager has an ambient presence, and when you create a new instance of TransactionScope it automatically searches for the current transaction within the scope of the method invokation--which should have been created already by WCF when it popped the message off the queue and invoked your method.

edited Sep 17, 2008 at 12:59

community wiki
2 revs
C. Lawrence Wenham

Hi Chris. I'm sure by attributing your operation behavior to TransactionScopeRequired=true negates the need to wrap your Sql Calls in a Transaction Scope as this is already being done. That being said, I'm not sure how your answer relates

to my question of MSMQ. – WebDude Sep 17, 2008 at 13:23

1

Unfortunately I'm stuck on Windows XP and Windows Server 2003 so that isn't an option for me. - (I will re-clarify that in my question as I found this solution after posting and realised i couldn't use it)

I found that one solution was to setup a custom handler which would move my message onto another queue or poison queue and restart my service. This seemed crazy to me. Imagine my Sql Server was down how often the service would be restarted.

SO what I've ended up doing is allowing the Line to fault and leave messages on the queue. I also log a fatal message to my system logging service that this has happened. Once our issue is resolved, I restart the service and all the messages start getting processed again.

I realised re-processing this message or any other will all fail, so why the need to move this message and the others to another queue. I may as well stop my service, and start it again when all is operating as expected.

aogan, you had the perfect answer for MSMQ 4.0, but unfortunately not for me

Share Improve this answer Follow

answered Sep 17, 2008 at 12:43

WebDude

6,205 6 35 44