MSSQLTips.com
brought to you by Edgewood Solutions

Your daily source for SQL Server Tips

SQL Server Service Broker example on how to configure, send and receive messages

Written By: Jayakumar Krishnan -- 9/14/2009

Problem
SQL Server 2008 and 2005 offer the Service Broker feature. In this tip we will go through the different components of service broker and step by step on how to setup Service Broker for a single database.

Solution
What is service broker? Service Broker is a process of sending and receiving guaranteed, asynchronous messages by using extensions to the Transact-SQL Data Manipulation Language (DML). Messages can be sent to a queue in the same database as the sender, to another database in the same SQL Server instance, or to another SQL Server instance either on the same server or on a remote server.

In this article we will cover these items:

1. Configuring the Service Broker.
2. Sending and Receiving Messages.

Configuring the Service Broker

To configure the service broker to send and receive messages between databases (either in local/remote instances), it needs a few components in place to start the service. These are listed below.

1. Enable the Service Broker on the database
2. Create valid Message Types.
3. Create a Contract for the conversation.
4. Create Queues for the Communication.
5. Create Services for the Communication.

Note: For my test I've created a database named ServerBrokerTest, which I'm going to use in this article. I've used a single database to make this easy to understand. We can call it a single database conversation.

1. Enabling Service Broker

Service broker is a database level feature not an instance level. So it has to be enabled before starting the other configurations.

Use the following code to enable the Service Broker.

```
--Enabling service broker
USE master
ALTER DATABASE ServiceBrokerTest
SET ENABLE_BROKER;
```

Enabling service broker option can be verified with the Is_Broker_Enabled column (set to true [1]) in sys.databases catalog view.

2. Create Valid Message Types

Service Broker needs specific Message Types to send and receive messages, which have to be created before the communication starts. The initiator (Sender/Requestor) and the target (Receiver/Reply) have to use the same Message Type or else the communication will fail. The most common used Message Type is WELL_FORMED_XML.

The following example code is used to create a Message Type in the same database for the Initiator (Requestor/Sender) and the Target (Reply/Receiver). Use the following code to create the Message Types.

```
--Create Message Types for Request and Reply messages
USE ServiceBrokerTest

-- For Request
CREATE MESSAGE TYPE
[//SBTest/SBSample/RequestMessage]
VALIDATION=WELL_FORMED_XML;

-- For Reply
CREATE MESSAGE TYPE
[//SBTest/SBSample/ReplyMessage]
VALIDATION=WELL_FORMED_XML;
```

---

3. Create a Contract for the Conversation

Service Broker requires a contract to send and receive messages in a single or between multiple databases. The contract will ensurewhich Message Type is going to be used to send and receive messages between the Initiator (Requestor/Sender) and the Target (Reply/Receiver). Use the following code to create the contract.

```
--Create Contract for the Conversation
USE ServiceBrokerTest

CREATE CONTRACT [//SBTest/SBSample/SBContract]
(
[//SBTest/SBSample/RequestMessage]
SENT BY INITIATOR
,[//SBTest/SBSample/ReplyMessage]
SENT BY TARGET
);
```

---

4. Create Queues for the Communication

The Service Broker Queue is a Message Storage container which is used to keep the messages while sending and receiving. The below code creates two queues, one is the Initiator (Requester/Sender) and the other is Target (Receiver/Replier). Queues will be used by the Service Broker Services.

Use the following code is to create the Queues.

```
USE ServiceBrokerTest

--Create Queue for the Initiator
CREATE QUEUE SBInitiatorQueue;

--Create Queue for the Target
CREATE QUEUE SBTargetQueue;
```

---

5. Create Services for the Communication

The Service Broker Services route the Messages to the specific Queues. The Service and Queue are bonded to each other. When the Initiator or the Target send a Message, the service will route the messages to the appropriate Queues.

Use the following code to create the Service Broker Service.

```
--Create Service for the Target and the Initiator.
USE ServiceBrokerTest

--Create Service for the Initiator.
CREATE SERVICE [//SBTest/SBSample/SBInitiatorService]
ON QUEUE SBInitiatorQueue;

--Create Service for the Target.
CREATE SERVICE [//SBTest/SBSample/SBTargetService]
ON QUEUE SBTargetQueue
([//SBTest/SBSample/SBContract]);
```

Note: In the above code I've not specified the Contract name for the Initiator Service, but I specified for the Target Service, which means if no Contact name is specified on a Service then the Service can only initiate conversations but no other Services can use that service as a Target (Reply/Receiver).

Sending and Receiving Messages

In this section, I'll describe how to Send (Request - from the Initiator) and Reply (from the Target) and Receive a message between the Initiator and the Target. We can understand the conversation cycle between the Initiator and Target.

Sending Message to Target

The following code sends a request message from the Initiator to the Target. The code can be split into three parts.

1. Determine the Services and contract.
2. Prepare the Message
3. Send the Message.

Sending a message is a single transaction, which includes all 3 items above.

```
--Sending a Request Message to the Target
USE ServiceBrokerTest

DECLARE @InitDlgHandle UNIQUEIDENTIFIER
DECLARE @RequestMessage VARCHAR(1000)

BEGIN TRAN

--Determine the Initiator Service, Target Service and the Contract
BEGIN DIALOG @InitDlgHandle
FROM SERVICE
[//SBTest/SBSample/SBInitiatorService]
TO SERVICE
'//SBTest/SBSample/SBTargetService'
ON CONTRACT
[//SBTest/SBSample/SBContract]
WITH ENCRYPTION=OFF;

--Prepare the Message
SELECT @RequestMessage = N'<RequestMessage> Send a Message to Target </RequestMessage>';

--Send the Message
```

```
SEND ON CONVERSATION @InitDlgHandle
MESSAGE TYPE
[//SBTest/SBSample/RequestMessage]
(@RequestMessage);
SELECT @RequestMessage AS SentRequestMessage;

COMMIT TRAN
```

Note: TO SERVICE needs to be specified in the single quotes because it is case sensitive, service broker uses a byte-by-byte comparison with the Target service name. The above code will give the below result set.

```
SentRequestMessage
<RequestMessage> Send a Message to Target </RequestMessage>
```

Receiving and Sending Message to Initiator

The following code receives a message from the Initiator and sends a Reply message to it.

```
--Receiving a Message and sending a Reply from the Target
USE ServiceBrokerTest

DECLARE @TargetDlgHandle UNIQUEIDENTIFIER
DECLARE @ReplyMessage VARCHAR(1000)
DECLARE @ReplyMessageName Sysname

BEGIN TRAN;

--Receive message from Initiator
RECEIVE TOP(1)
@TargetDlgHandle=Conversation_Handle
,@ReplyMessage=Message_Body
,@ReplyMessageName=Message_Type_Name
FROM SBTargetQueue;

SELECT @ReplyMessage AS ReceivedRequestMessage;

-- Confirm and Send a reply
IF @ReplyMessageName=N'//SBTest/SBSample/RequestMessage'
BEGIN
DECLARE @RplyMsg VARCHAR(1000)
SELECT @RplyMsg =N'<RplyMsg> Send a Message to Initiator</RplyMsg>';

SEND ON CONVERSATION @TargetDlgHandle
MESSAGE TYPE
[//SBTest/SBSample/ReplyMessage]
(@RplyMsg);
END CONVERSATION @TargetDlgHandle;
END

SELECT @RplyMsg AS SentReplyMessage;

COMMIT TRAN;
```

The above will give two result sets as below, 1 is received message from the Initiator and 2 is the sent message to the Initiator from the Target.

```
ReceivedRequestMessage
<RequestMessage> Send a Message to Target </RequestMessage>
```

```
SentReplyMessage
<RplyMsg> Send a Message to Initiator</RplyMsg>
```

Receiving a Reply Message from the Target

The below code receives a reply message from the Target.

```
--Receiving Reply Message from the Target.
USE ServiceBrokerTest

DECLARE @InitiatorReplyDlgHandle UNIQUEIDENTIFIER
DECLARE @ReplyReceivedMessage VARCHAR(1000)

BEGIN TRAN;

RECEIVE TOP(1)
@InitiatorReplyDlgHandle=Conversation_Handle
,@ReplyReceivedMessage=Message_Body
FROM SBInitiatorQueue;

END CONVERSATION @InitiatorReplyDlgHandle;

SELECT @ReplyReceivedMessage AS ReceivedRepliedMessage;

COMMIT TRAN;
```

Below is the output of the above transaction, which confirms that the reply message received from the Target.

```
ReceivedRepliedMessage
<RplyMsg> Send a Message to Initiator</RplyMsg>
```

That is pretty much all there is to setting up Service Broker and sending and receiving some simple messages.

Some Useful Catalog Views

The following catalog views are helpful to check the usage of the Service Broker Objects.

```
--Checking the usage of the Messages, Contracts and Queues using System views.
USE ServiceBrokerTest

SELECT * FROM sys.service_contract_message_usages
SELECT * FROM sys.service_contract_usages
SELECT * FROM sys.service_queue_usages
```

I hope this article has taught you how to configure the Service Broker in single database and start the conversation cycle between the Initiator and Target. Conversation between databases also has the same steps, but the initiator and the target will be two different databases and the services and queues have to be created in both of the databases.

Next Steps

Review these other tips related to Service Broker

- Service Broker Troubleshooting
- SQL Server 2005 Service Broker Infrastructure Objects
- What exactly is SQL Server 2005 Service Broker?