**Configure SQL Server Service Broker for sending stored procedure data**

<http://searchsqlserver.techtarget.com/tip/0,289483,sid87_gci1310106,00.html>

Denny Cherry  
04.16.2008

|  |
| --- |
| In SQL Server 2005, Microsoft introduced an exciting new feature called the Service Broker, and with it came a new concept for many database administrators. The concept, out-of-process messaging, is an application that developers have been using for many years via products such as Microsoft Message Queue (MSMQ). To set the scene, a SQL Server command is run and the data from that command needs to be processed. However, it doesn't need to be processed right away, as the user isn't expecting any output from the command. This is where message queuing comes into play.  Normally, when users fire a stored procedure in SQL Server, they must wait for the stored procedure to complete the processing of data before the next action can be completed. By using Service Broker, they can queue the actual processing of the data for a later date. In this case, the user will run a stored procedure, but instead of the data processing, we take the input data and package it as a message and send that message to a queue. We configure the queue to read the message automatically and process the data within the message.  Messages can be sent within the same database, from database to database on the same server or from one server to another server including servers across the internet. Think of the process as kind of like email. When I send an email to my friend, he receives it and will read it (processing the message) and later do something based on the content. Service Broker works in much the same way.  **Four steps for setting up Service Broker to send and receive messages:**  There are several components of Service Broker that must be configured before you can actually send and receive messages. For our example here, we will put messages into a single queue for automatic processing. The processing of these messages can be automatic or on-demand with a Windows application or service reading the messages from the queue.  Like all other objects, Service Broker names must be unique within the database. However, more care should be taken when setting up your Service Broker object names if you are going to be sending messages from server to server. It's usually recommended that some combination of system name and function name be put into the Service Broker object names. This ensures that the object names are unique across your enterprise. It will come in handy when you start sending messages between systems already using the Service Broker internally and you want to avoid object-naming problems. Try using a UNC style name to ensure uniqueness. For this example, we will start our object names with tcp://SearchSQLServer/.   1. The first object you want to set up in Service Broker is Message Type, which tells SQL Server about basic information within the message. Service Broker messages can be data of any type: text, binary, xml, numeric, etc. It's recommended that XML is used, because it allows for the sending of all the other data types within it. We use the CREATE MESSAGE TYPE command to create a message type. There are four validation options available. If you want to enforce valid XML formatting, select the WELL\_FORMED\_XML or the VALID\_XML WITH SCHEMA COLLECTION (to use an XSD to enforce your XML schema).   CREATE MESSAGE TYPE [tcp://SearchSQLServer/SampleMessageType] AUTHORIZATION  dbo VALIDATION = NONE   1. The next object we'll need to set up is the contract. The contract tells SQL Server which message types relate to each other. Personally, I like to keep things simple and use a single message type for all communication in a specific process. However, the Microsoft SQL Server Books OnLine examples show using one message type for the sending system and a second for the receiving system.   CREATE CONTRACT [tcp://SearchSQLServer/SampleContract] AUTHORIZATION dbo ( [tcp://SearchSQLServer/SampleMessageType] SENT BY ANY )   1. Now we're on to the actual queue in Service Broker. The queue is kind of like a table. This is where the messages are stored while they wait to be processed. Unlike a table, you cannot define the schema of the queue. For this reason, it's recommended that XML be used to send the data because then you can define the schema within the XML.   CREATE QUEUE [tcp://SearchSQLServer/SampleQueue] AUTHORIZATION dbo  The next Service Broker object to configure is the service. The service is used by Service Broker to deliver messages to the correct queue within the database and to bind the contract to the queue message.  CREATE SERVICE [tcp://SearchSQLServer/SampleService] AUTHORIZATION dbo ON QUEUE [tcp://SearchSQLServer/SampleQueue] ( [tcp://SearchSQLServer/SampleContract] )   1. The last object is the route and it's optional. The route is only required when you are sending messages from one database server to another. The route is tied to a local copy of the server that is receiving the message. If the server receiving the messages is mirrored to another server using database mirroring, then you'll want to include the MIRROR\_ADDRESS parameter. If you want the route to delete itself automatically, you can set the LIFETIME flag.   CREATE ROUTE [tcp://SearchSQLServer/SampleRoute] AUTHORIZATION dbo WITH SERVICE\_NAME = '[tcp://SearchSQLServer/SampleService]', BROKER\_INSTANCE = 'AB2F3EB9-6662-4AAF-8682-A9A48C3BDD3B', ADDRESS = 'TCP://RemoteServer:8888', MIRROR\_ADDRESS = 'TCP://MirrorServer:8888'  The BROKER\_INSTANCE parameter is the value of the service\_broker\_guid column from the sys.databases view of the database located on the remote server..  In order to send a message from one queue to another, you must use two commands. The first is the BEGIN DIALOG command, followed by the SEND command. Use the BEGIN DIALOG command to create a conversation between the sending and receiving services. If you are using an already existing conversation, then you don't need to use the BEGIN DIALOG command. After creating the new conversation using the BEGIN DIALOG command, use the SEND command to send the message on that conversation using the conversation handle that is returned from the BEGIN DIALOG command.  DECLARE/font> @dialog\_handle UNIQUEIDENTIFIER, @XMLData XML ;  SET @XMLData = (SELECT \* FROM sys.tables FOR XML AUTO)  BEGIN DIALOG @dialog\_handle FROM SERVICE [tcp://SearchSQLServer/SampleService] TO SERVICE 'tcp://SearchSQLServer/SampleService' ON CONTRACT [tcp://SearchSQLServer/SampleContract];  SEND ON CONVERSATION @dialog\_handle MESSAGE TYPE [tcp://SearchSQLServer/SampleMessageType] (@XMLData);  In order to view and process the message, the RECEIVE command is used. The RECEIVE command is much like SELECT, in which the data is returned. When using the RECEIVE command, however, the message is received only once. After the message has been received and if it is the last message in the conversation – most people only put a single message in a conversation – use the END CONVERSATION command to close it. The RECEIVE command can be used with the WAITFOR command and a WHILE loop to process all messages in the queue in a single run of the procedure.  DECLARE @dialog\_handle UNIQUEIDENTIFIER, @XMLData XML ;  RECEIVE TOP (1) @dialog\_handle = conversation\_handle, @XMLData = CAST(message\_body AS XML) FROM [tcp://SearchSQLServer/SampleQueue]  END CONVERSATION @dialog\_handle  SELECT @XMLData  When you end the conversation, a message is actually sent from the receiving queue to the sending queue which informs the sending queue, service and SQL Server that the conversation is closed. This end conversation message must then be processed by the sending queue in order to remove the data from the queue. It is recommended that a basic procedure be used on the sending queue to automatically clear out these messages.  CREATE PROCEDURE usp\_ProcessAck AS DECLARE @xml AS XML DECLARE @dialog\_handle as uniqueidentifier WHILE 1=1 SET @dialog\_handle = NULL  WAITFOR ( RECEIVE TOP (1) @dialog\_handle = conversation\_handle, @xml = cast(message\_body as xml) FROM [tcp://SearchSQLServer/SampleQueue]), TIMEOUT 1000  IF @dialog\_handle IS NULL break  END CONVERSATION @dialog\_handle END GO  After you create this procedure, use the ALTER QUEUE command to set up the queue to run the procedure automatically when messages arrive by using the ACTIVATION commands of the QUEUE. You can set the number of parallel executions of the procedure by setting the MAX\_QUEUE\_READERS greater than 1. In systems that are very high load, or when the processing takes some length of time, additional queue readers can speed up the processing of data by processing the data from the queue in parallel.  ALTER QUEUE [tcp://SearchSQLServer/SampleQueue] WITH ACTIVATION (STATUS=ON, PROCEDURE\_NAME = dbo.usp\_ProcessAck, MAX\_QUEUE\_READERS = 2, EXECUTE AS SELF)  SQL Server Service Broker may be complex to set up and there isn't much documentation about it on the Internet. However, once Service Broker is configured and working correctly, it can provide a rock solid interdatabase or intradatabase communications platform for sending data quickly and en masse from one system to another. |