Configuring Service Broker for Asynchronous Processing

In my last article, I talked about the benefits of implementing asynchronous processing using Service Broker in SQL Server over the other methods that exist for decoupled processing of long tasks. In this article, we'll go through all the components that need to be configured for a basic Service Broker configuration in a single database, and the important considerations for the conversation management between broker services. To get started, we'll need to create a database and enable the database for Service Broker usage:

```
CREATE DATABASE AsyncProcessingDemo;
GO

IF (SELECT is_broker_enabled FROM sys.databases WHERE name =
N'AsyncProcessingDemo') = 0
BEGIN
  ALTER DATABASE AsyncProcessingDemo SET ENABLE_BROKER;
END
GO

USE AsyncProcessingDemo;
GO
```

Configuring broker components

The basic objects that need to be created in the database are the message types for the messages, a contract that defines how the messages will be sent between the services, a queue and the initiator service, and a queue and the target service. Many examples online for service broker show complex object naming for the message types, contracts and services for Service Broker. However, there isn't a requirement for the names to be complex, and simple object names can be used for any of the objects.

For the messages, we will need to create a message type for the request, which will be called `AsyncRequest`, and a message type for the result, which will be called `AsyncResult`. Both will use XML that will be validated as correctly formed by the broker services to send and receive the data required by the services.

```
-- Create the message types
CREATE MESSAGE TYPE [AsyncRequest] VALIDATION = WELL_FORMED_XML;
CREATE MESSAGE TYPE [AsyncResult]  VALIDATION = WELL_FORMED_XML;
```

The contract specifies that the `AsyncRequest` will be sent by the initiating service to the target service and that the target service will return a `AsyncResult` message back to the initiating service. The contract can also specify multiple message types for the initiator and target, or that a specific message type can be sent by any service, if the specific processing requires it.

```
-- Create the contract
CREATE CONTRACT [AsyncContract]
```

```
(
  [AsyncRequest] SENT BY INITIATOR,
  [AsyncResult]  SENT BY TARGET
);
```

For each of the services, a queue must be created to provide storage of the messages received by the service. The target service where the request will be sent needs to be created specifying the `AsyncContract` to allow messages to be sent to the service. In this case the service is named `ProcessingService` and will be created on the `ProcessingQueue` within the database. The initiating service does not require a contract to be specified, which makes it only able to receive messages in response to a conversation that was initiated from it.

```
-- Create the processing queue and service - specify the contract to allow
sending to the service
CREATE QUEUE ProcessingQueue;
CREATE SERVICE [ProcessingService] ON QUEUE ProcessingQueue
([AsyncContract]);

-- Create the request queue and service
CREATE QUEUE RequestQueue;
CREATE SERVICE [RequestService] ON QUEUE RequestQueue;
```

Sending a Message for Processing

As I explained in the previous article, I prefer to implement a wrapper stored procedure for sending a new message to a broker service, so that it can be modified once to scale performance if required. This procedure is a simple wrapper around creating a new conversation and sending the message to the `ProcessingService`.

```
-- Create the wrapper procedure for sending messages
CREATE PROCEDURE dbo.SendBrokerMessage
        @FromService SYSNAME,
        @ToService   SYSNAME,
        @Contract    SYSNAME,
        @MessageType SYSNAME,
        @MessageBody XML
AS
BEGIN
  SET NOCOUNT ON;

  DECLARE @conversation_handle UNIQUEIDENTIFIER;

  BEGIN TRANSACTION;

  BEGIN DIALOG CONVERSATION @conversation_handle
    FROM SERVICE @FromService
    TO SERVICE @ToService
    ON CONTRACT @Contract
    WITH ENCRYPTION = OFF;

  SEND ON CONVERSATION @conversation_handle
    MESSAGE TYPE @MessageType(@MessageBody);
```

```
  COMMIT TRANSACTION;
END
GO
```

Using the wrapper stored procedure we can now send a test message to the `ProcessingService` to validate that we have set up the broker services correctly.

```
-- Send a request
EXECUTE dbo.SendBrokerMessage
  @FromService = N'RequestService',
  @ToService   = N'ProcessingService',
  @Contract    = N'AsyncContract',
  @MessageType = N'AsyncRequest',
  @MessageBody =
N'<AsyncRequest><AccountNumber>12345</AccountNumber></AsyncRequest>';

-- Check for message on processing queue
SELECT CAST(message_body AS XML) FROM ProcessingQueue;
GO
```

Processing Messages

While we could manually process the messages from the `ProcessingQueue`, we'll probably want the messages to be processed automatically as they are sent to the `ProcessingService`. To do this a activation stored procedure needs to be created that we'll test and then later bind to the queue to automate the processing upon queue activation. To process a message we need to `RECEIVE` the message from the queue within a transaction, along with the message type and conversation handle for the message. The message type ensures that the appropriate logic is applied to the message being processed, and the conversation handle allows a response to be sent back to the initiating service when the message has been processed.

The `RECEIVE` command allows a single message or multiple messages within the same conversation handle or group to be processed in a single transaction. To process multiple messages, a table variable must be used, or to do single message processing, a local variable can be used. The activation procedure below retrieves a single message from the queue, checks the message type to determine if it is an `AsyncRequest` message, and then performs the long running process based on the message information received. If it doesn't receive a message within the loop, it will wait up to 5000ms, or 5 seconds, for another message to enter the queue before exiting the loop and terminating its execution. After processing a message, it builds an `AsyncResult` message and sends it back to the initiator on the same conversation handle that the message was received from. The procedure also checks the message type to determine if an `EndDialog` or `Error` message has been received to clean up the conversation by ending it.

```
-- Create processing procedure for processing queue
CREATE PROCEDURE dbo.ProcessingQueueActivation
AS
BEGIN
  SET NOCOUNT ON;

  DECLARE @conversation_handle UNIQUEIDENTIFIER;
```

```
  DECLARE @message_body XML;
  DECLARE @message_type_name sysname;

  WHILE (1=1)
  BEGIN
    BEGIN TRANSACTION;

    WAITFOR
    (
      RECEIVE TOP (1)
        @conversation_handle = conversation_handle,
        @message_body = CAST(message_body AS XML),
        @message_type_name = message_type_name
      FROM ProcessingQueue
    ), TIMEOUT 5000;

    IF (@@ROWCOUNT = 0)
    BEGIN
      ROLLBACK TRANSACTION;
      BREAK;
    END

    IF @message_type_name = N'AsyncRequest'
    BEGIN
      -- Handle complex long processing here
      -- For demonstration we'll pull the account number and send a reply
back only

      DECLARE @AccountNumber INT =
@message_body.value('(AsyncRequest/AccountNumber)[1]', 'INT');

      -- Build reply message and send back
      DECLARE @reply_message_body XML = N'
        ' + CAST(@AccountNumber AS NVARCHAR(11)) + '
      ';

      SEND ON CONVERSATION @conversation_handle
        MESSAGE TYPE [AsyncResult] (@reply_message_body);
    END

    -- If end dialog message, end the dialog
    ELSE IF @message_type_name =
N'http://schemas.microsoft.com/SQL/ServiceBroker/EndDialog'
    BEGIN
      END CONVERSATION @conversation_handle;
    END

    -- If error message, log and end conversation
    ELSE IF @message_type_name =
N'http://schemas.microsoft.com/SQL/ServiceBroker/Error'
    BEGIN
      -- Log the error code and perform any required handling here
      -- End the conversation for the error
      END CONVERSATION @conversation_handle;
    END

    COMMIT TRANSACTION;
```

```
  END
END
GO
```

The `RequestQueue` will also need to process the messages that are sent to it, so an additional procedure for processing the `AsyncResult` messages returned by the ProcessingQueueActivation procedure needs to be created. Since we know that the AsnycResult message means that all of the processing work has completed, the conversation can be ended once we process that message, which will send a EndDialog message to the ProcessingService, which will then be processed by it's activation procedure to end the conversation cleaning everything up and avoiding the fire and forget problems that happen when conversations are ended properly.

```
-- Create procedure for processing replies to the request queue
CREATE PROCEDURE dbo.RequestQueueActivation
AS
BEGIN
  SET NOCOUNT ON;

  DECLARE @conversation_handle UNIQUEIDENTIFIER;
  DECLARE @message_body XML;
  DECLARE @message_type_name sysname;

  WHILE (1=1)
  BEGIN
    BEGIN TRANSACTION;

    WAITFOR
    (
      RECEIVE TOP (1)
        @conversation_handle = conversation_handle,
        @message_body = CAST(message_body AS XML),
        @message_type_name = message_type_name
      FROM RequestQueue
    ), TIMEOUT 5000;

    IF (@@ROWCOUNT = 0)
    BEGIN
      ROLLBACK TRANSACTION;
      BREAK;
    END

    IF @message_type_name = N'AsyncResult'
    BEGIN
      -- If necessary handle the reply message here
      DECLARE @AccountNumber INT =
@message_body.value('(AsyncResult/AccountNumber)[1]', 'INT');

      -- Since this is all the work being done, end the conversation to send
the EndDialog message
      END CONVERSATION @conversation_handle;
    END

    -- If end dialog message, end the dialog
    ELSE IF @message_type_name =
N'http://schemas.microsoft.com/SQL/ServiceBroker/EndDialog'
```

```
    BEGIN
       END CONVERSATION @conversation_handle;
    END

    -- If error message, log and end conversation
    ELSE IF @message_type_name =
N'http://schemas.microsoft.com/SQL/ServiceBroker/Error'
    BEGIN
       END CONVERSATION @conversation_handle;
    END

    COMMIT TRANSACTION;
  END
END
GO
```

Testing the Procedures

Prior to automating the queue processing for our services, it is important to test the activation procedures to ensure that they process the messages appropriately, and to prevent a queue from being disabled should an error occur that isn't handled properly. Since there is already a message on the `ProcessingQueue` the `ProcessingQueueActivation` procedure can be executed to process that message. Keep in mind that the `WAITFOR` will cause the procedure to take 5 seconds to terminate, even though the message is processed immediately from the queue. After processing the message, we can verify the procedure worked correctly by querying the `RequestQueue` to see if an `AsyncResult` message exists, and then we can verify that the `RequestQueueActivation` procedure functions correctly by executing it.

```
-- Process the message from the processing queue
EXECUTE dbo.ProcessingQueueActivation;
GO

-- Check for reply message on request queue
SELECT CAST(message_body AS XML) FROM RequestQueue;
GO

-- Process the message from the request queue
EXECUTE dbo.RequestQueueActivation;
GO
```

Automating the Processing

At this point, all of the components are complete to fully automate our processing. The only thing remaining is to bind the activation procedures to their appropriate queues, and then send another test message to validate it gets processed and nothing remains in the queues afterwards.

```
-- Alter the processing queue to specify internal activation
ALTER QUEUE ProcessingQueue
    WITH ACTIVATION
    (
      STATUS = ON,
      PROCEDURE_NAME = dbo.ProcessingQueueActivation,
      MAX_QUEUE_READERS = 10,
```

```
      EXECUTE AS SELF
    );
GO

-- Alter the request queue to specify internal activation
ALTER QUEUE RequestQueue
    WITH ACTIVATION
    (
      STATUS = ON,
      PROCEDURE_NAME = dbo.RequestQueueActivation,
      MAX_QUEUE_READERS = 10,
      EXECUTE AS SELF
    );
GO

-- Test automated activation
-- Send a request

EXECUTE dbo.SendBrokerMessage
        @FromService = N'RequestService',
        @ToService   = N'ProcessingService',
        @Contract    = N'AsyncContract',
        @MessageType = N'AsyncRequest',
        @MessageBody =
N'<AsyncRequest><AccountNumber>12345</AccountNumber></AsyncRequest>';

-- Check for message on processing queue
-- nothing is there because it was automatically processed
SELECT CAST(message_body AS XML) FROM ProcessingQueue;
GO

-- Check for reply message on request queue
-- nothing is there because it was automatically processed
SELECT CAST(message_body AS XML) FROM RequestQueue;
GO
```

Summary

The basic components for automated asynchronous processing in SQL Server Service Broker can be configured in a single database setup to allow for decoupled processing of long running tasks. This can be a powerful tool for improving application performance, from an end user's experience, by decoupling the processing from the end user's interactions with the application.