```
/*
============================================================================
Service Broker Sample 2: Asynchronous Triggers
============================================================================
 This script creates a set of objects that allow
 the easy implementation and usage of Asynchronous Triggers.
 Usage:
    1. Run this script on the database where you want the asynchronous trigger(s).

    2. Create a stored procedure which will receive the following two parameters:
        @inserted XML,
        @deleted XML
       This procedure will be responsible for parsing the inserted/deleted data
       and executing the actual trigger logic based on that data.

    3. Inside the actual table trigger, use the following code:
            DECLARE
                @inserted   XML,
                @deleted    XML;

            SELECT @inserted =
                ( SELECT * FROM inserted FOR XML PATH('row'), ROOT('inserted') );

            SELECT @deleted =
                ( SELECT * FROM deleted FOR XML PATH('row'), ROOT('deleted') );

            EXECUTE SB_AT_Fire_Trigger '{YourProcedureName}', @inserted, @deleted;

       But replace {YourProcedureName} with the name of the procedure you've
       created in step 2.

You may review script SB_AT_Sample for an AdventureWorks2008R2 example.
============================================================================
Copyright:  Eitan Blumin (C) 2013
Email:      eitan@madeira.co.il
Source:     www.madeira.co.il
Disclaimer:
    The author is not responsible for any damage this
    script or any of its variations may cause.
    Do not execute it or any variations of it on production
    environments without first verifying its validity
    on controlled testing and/or QA environments.
    You may use this script at your own risk and may change it
    to your liking, as long as you leave this disclaimer header
    fully intact and unchanged.
============================================================================
*/
-- Creation of the table to hold SB logs
IF OBJECT_ID('SB_AT_ServiceBrokerLogs') IS NULL
BEGIN
    CREATE TABLE SB_AT_ServiceBrokerLogs
    (
        LogID           BIGINT      IDENTITY(1,1)   NOT NULL,
        LogDate         DATETIME                    NOT NULL DEFAULT (GETDATE()),
        SPID            INT                         NOT NULL DEFAULT (@@SPID),
        ProgramName     NVARCHAR(255)               NOT NULL DEFAULT (APP_NAME()),
        HostName        NVARCHAR(255)               NOT NULL DEFAULT (HOST_NAME()),
        ErrorSeverity   INT                         NOT NULL DEFAULT (0),
        ErrorMessage    NVARCHAR(MAX)               NULL,
        ErrorLine       INT                         NULL,
        ErrorProc       SYSNAME                     NULL,
        QueueMessage    XML                         NULL,
        PRIMARY KEY NONCLUSTERED (LogID)
    );
    CREATE CLUSTERED INDEX IX_SB_AT_ServiceBrokerLogs ON SB_AT_ServiceBrokerLogs (LogDate
    ASC) WITH FILLFACTOR=100;
    PRINT 'Table SB_AT_ServiceBrokerLogs Created';
END
```

```
ELSE
    TRUNCATE TABLE SB_AT_ServiceBrokerLogs
GO
IF OBJECT_ID('SB_AT_HandleQueue') IS NOT NULL DROP PROCEDURE SB_AT_HandleQueue
RAISERROR(N'Creating SB_AT_HandleQueue...',0,0) WITH NOWAIT;
GO
-- This procedure is activated to handle each item in the Request queue
CREATE PROCEDURE SB_AT_HandleQueue
AS
    SET NOCOUNT ON;
    SET ARITHABORT ON
    DECLARE @msg XML
    DECLARE @DlgId UNIQUEIDENTIFIER
    DECLARE @Info nvarchar(max)
    DECLARE @ErrorsCount int
    SET @ErrorsCount = 0

    -- Set whether to log verbose status messages before and after each operation
    DECLARE @Verbose BIT = 1

    -- Allow 10 retries in case of service broker errors
    WHILE @ErrorsCount < 10
    BEGIN

        BEGIN TRANSACTION
        BEGIN TRY
            -- Make sure queue is active
            IF EXISTS (SELECT NULL FROM sys.service_queues
                       WHERE NAME = 'SB_AT_Request_Queue'
                       AND is_receive_enabled = 0)
                ALTER QUEUE SB_AT_Request_Queue WITH STATUS = ON;

            -- handle one message at a time
            WAITFOR
            (
                RECEIVE TOP(1)
                    @msg        = convert(xml,message_body),
                    @DlgId      = conversation_handle
                FROM dbo.SB_AT_Request_Queue
            );

            -- exit when waiting has been timed out
            IF @@ROWCOUNT = 0
            BEGIN
                IF @@TRANCOUNT > 0
                    ROLLBACK TRANSACTION;
                BREAK;
            END

            -- Retreive data from xml message
            DECLARE
                @ProcedureName  VARCHAR(1000),
                @inserted       XML,
                @deleted        XML

            SELECT
                @ProcedureName      = x.value('(/Request/ProcedureName)[1]','VARCHAR(1000)'),
                @inserted           = x.query('/Request/inserted/inserted'),
                @deleted            = x.query('/Request/deleted/deleted')
            FROM @msg.nodes('/Request') AS T(x);

            -- Log operation start
            IF @Verbose = 1
                INSERT INTO SB_AT_ServiceBrokerLogs(ErrorSeverity,ErrorMessage,QueueMessage)
                VALUES(0,'Starting Process',@msg);

            -- Encapsulate execution in TRY..CATCH
            -- to catch errors in the specific request
```

```
            BEGIN TRY

                -- Execute Request
                EXEC @ProcedureName @inserted, @deleted;

            END TRY
            BEGIN CATCH

                -- log operation fail
                INSERT INTO
                SB_AT_ServiceBrokerLogs(ErrorSeverity,ErrorMessage,ErrorLine,ErrorProc,QueueMe
                ssage)
                VALUES(ERROR_SEVERITY(),ERROR_MESSAGE(),ERROR_LINE(),ERROR_PROCEDURE(),@msg);

            END CATCH

            -- commit
            IF @@TRANCOUNT > 0
                COMMIT TRANSACTION;

            -- Log operation end
            IF @Verbose = 1
                INSERT INTO SB_AT_ServiceBrokerLogs(ErrorSeverity,ErrorMessage,QueueMessage)
                VALUES(0,'Finished Process',@msg);

                -- reset xml message
                SET @msg = NULL;
        END TRY
        BEGIN CATCH

            -- rollback transaction
            -- this will also rollback the extraction of the message from the queue
            IF @@TRANCOUNT > 0
                ROLLBACK TRANSACTION;

            -- log operation fail
            INSERT INTO
            SB_AT_ServiceBrokerLogs(ErrorSeverity,ErrorMessage,ErrorLine,ErrorProc,QueueMessag
            e)
            VALUES(ERROR_SEVERITY(),ERROR_MESSAGE(),ERROR_LINE(),ERROR_PROCEDURE(),@msg);

            -- increase error counter
            SET @ErrorsCount = @ErrorsCount + 1;

            -- wait 5 seconds before retrying
            WAITFOR DELAY '00:00:05'
        END CATCH

    END
GO
DECLARE @SQL nvarchar(max)

-- Enable service broker
IF EXISTS (SELECT * FROM sys.databases WHERE database_id = DB_ID() AND is_broker_enabled = 0)
BEGIN
    SET @SQL = 'ALTER DATABASE [' + DB_NAME() + '] SET NEW_BROKER WITH ROLLBACK IMMEDIATE';
    EXEC(@SQL);
    PRINT 'Enabled Service Broker for DB ' + DB_NAME();
END
GO
-- Drop existing objects

IF EXISTS (SELECT NULL FROM sys.services WHERE NAME = '//SB_AT/ProcessReceivingService')
    DROP SERVICE [//SB_AT/ProcessReceivingService];

IF EXISTS (SELECT NULL FROM sys.services WHERE NAME = '//SB_AT/ProcessStartingService')
    DROP SERVICE [//SB_AT/ProcessStartingService];
```

```
IF EXISTS (SELECT NULL FROM sys.service_queues WHERE NAME = 'SB_AT_Request_Queue')
    DROP QUEUE dbo.SB_AT_Request_Queue;

IF EXISTS (SELECT NULL FROM sys.service_queues WHERE NAME = 'SB_AT_Response_Queue')
    DROP QUEUE dbo.SB_AT_Response_Queue;

IF EXISTS (SELECT NULL FROM sys.service_contracts WHERE NAME = '//SB_AT/Contract')
    DROP CONTRACT [//SB_AT/Contract];

IF EXISTS (SELECT NULL FROM sys.service_message_types WHERE name='//SB_AT/Message')
    DROP MESSAGE TYPE [//SB_AT/Message];
GO
-- Create service broker objects

RAISERROR(N'Creating Message Type...',0,0) WITH NOWAIT;
CREATE MESSAGE TYPE [//SB_AT/Message]
    VALIDATION = WELL_FORMED_XML;

RAISERROR(N'Creating Contract...',0,0) WITH NOWAIT;
CREATE CONTRACT [//SB_AT/Contract]
    ([//SB_AT/Message] SENT BY ANY);

RAISERROR(N'Creating Response Queue...',0,0) WITH NOWAIT;
CREATE QUEUE dbo.SB_AT_Response_Queue
    WITH STATUS=ON;          -- This queue doesn't have activation because it needs to be
    emptied manually

RAISERROR(N'Creating Request Queue...',0,0) WITH NOWAIT;
CREATE QUEUE dbo.SB_AT_Request_Queue
    WITH STATUS=ON,
    ACTIVATION (
        PROCEDURE_NAME = SB_AT_HandleQueue,      -- sproc to run when queue receives message
        MAX_QUEUE_READERS = 10,                  -- max concurrent instances
        EXECUTE AS SELF
        );

RAISERROR(N'Creating Recieving Service...',0,0) WITH NOWAIT;
CREATE SERVICE [//SB_AT/ProcessReceivingService]
    AUTHORIZATION dbo
ON QUEUE dbo.SB_AT_Request_Queue ([//SB_AT/Contract]);

RAISERROR(N'Creating Sending Service...',0,0) WITH NOWAIT;
CREATE SERVICE [//SB_AT/ProcessStartingService]
    AUTHORIZATION dbo
ON QUEUE dbo.SB_AT_Response_Queue ([//SB_AT/Contract]);
GO
IF OBJECT_ID('SB_AT_Fire_Trigger') IS NOT NULL DROP PROCEDURE SB_AT_Fire_Trigger;
RAISERROR(N'Creating SB_AT_Fire_Trigger...',0,0) WITH NOWAIT;
GO
-- This procedure sends items to the queue for asynchronous triggers
CREATE PROCEDURE SB_AT_Fire_Trigger
    @ProcedureName  VARCHAR(1000),
    @inserted       XML = NULL,
    @deleted        XML = NULL
AS
    SET NOCOUNT ON;

    DECLARE @msg XML

    -- build the XML message
    SET @msg = (SELECT
                    ProcedureName   = @ProcedureName,
                    inserted        = @inserted,
                    deleted         = @deleted
                FOR XML PATH('Request'))

    DECLARE @DlgId UNIQUEIDENTIFIER
```

```
    BEGIN DIALOG @DlgId
        FROM SERVICE [//SB_AT/ProcessStartingService]
        TO SERVICE '//SB_AT/ProcessReceivingService',
        'CURRENT DATABASE'
        ON CONTRACT [//SB_AT/Contract]
    WITH ENCRYPTION = OFF;

    -- send the message
    SEND ON CONVERSATION @DlgId
    MESSAGE TYPE [//SB_AT/Message] (@msg);

    PRINT N'Started SB_AT process on dialogId ' +
    ISNULL(convert(varchar(100),@DlgId),'(null)');

GO
```