**Scheduling Jobs in SQL Server Express**

To achieve scheduling we will use **SQL Server Service Broker.**

**Service Broker:**

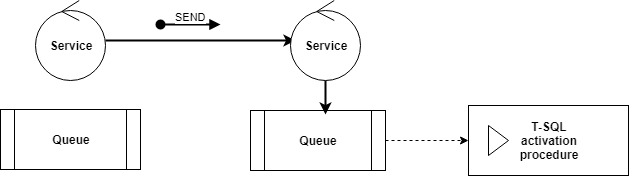
SQL Server Service Broker provide native support for messaging and queuing in the SQL Server Database Engine and Azure SQL Managed Instance. Developers can easily create sophisticated applications that use the Database Engine components to communicate between disparate databases, and build distributed and reliable applications.

## When to use Service Broker

Use Service Broker components to implement native in-database asynchronous message processing functionalities. Application developers who use Service Broker can distribute data workloads across several databases without programming complex communication and messaging internals. Service Broker reduces development and test work because Service Broker handles the communication paths in the context of a conversation. It also improves performance. For example, front-end databases supporting Web sites can record information and send process intensive tasks to queue in back-end databases. Service Broker ensures that all tasks are managed in the context of transactions to assure reliability and technical consistency.

## Overview

Service Broker is a message delivery framework that enables you to create native in-database service-oriented applications. Unlike classic query processing functionalities that constantly read data from the tables and process them during the query lifecycle, in service-oriented application you have database services that are exchanging the messages. Every service has a queue where the messages are placed until they are processed.



The messages in the queues can be fetched using the Transact-SQL RECEIVE command or by the activation procedure that will be called whenever the message arrives in the queue.

## How it works

Let's see how this scheduling infrastructure is built from start in simple bullet points:

1. Create the needed tables for our scheduled jobs information

2. Create the needed stored procedures that handle scheduled jobs

3. Create the needed contract, queue and service

### ****1. Needed tables****

We need 3 tables:

**-JobSchedule stores information job**

**- ScheduledJobs**stores information about our scheduled jobs

**- ScheduledJobsLog** stores possible errors/Jobs Run details when manipulating scheduled jobs

### ****2. Needed stored procedures****

For our simple scheduling we need 4 stored procedures.

First two expose the scheduling functionality we use. The third one isn't supposed to be used directly but it can be if it is needed. And forth will use to handle service broker errors.

**- PS\_AddScheduledJob** adds a row for our job to the ScheduledJobs table, starts a new conversation on it and set a timer on it. Adding and conversation starting is done in a transaction since we want this to be an atomic operation.

- **PS\_DeleteScheduledJob** performs cleanup. It accepts the id of the scheduled job we wish to remove. It ends the conversation that the inputted scheduled job lives on, and it deletes the row from the ScheduledJobs table. Removing the job and ending the conversation is also done in a transaction as an atomic operation.

**- PS\_RunScheduledJob** is the activation stored procedure on the queue and it receives the dialog timer messages put there by our conversation timer from the queue. Depending on the IsRepeatable setting it either sets the daily interval or ends the conversation. After that it runs our scheduled job and updates the ScheduledJobs table with the status of the finished scheduled job. This stored procedure isn't transactional since any errors are stored in the error table and we don't want to return the DialogTimer message back to the queue, which would cause problems with looping and [**poison messages**](http://weblogs.sqlteam.com/mladenp/archive/2007/07/25/60269.aspx) which we'd have to again handle separately. We want to keep things simple.

- **rsp\_MaintainScheduledJob** is tohandleall errors of service broker like if poison in generated then it will restart service broker, if queue is inactive then it will activate that queue. If on job is not run and it should run daily then it reschedule that particular job on respective time.

### ****3. Needed Service Broker objects****

For everything to work we need to make a simple setup used by the Service Broker:

**- [//ScheduledJobContract]** is the contract that allows only sending of the "http://schemas.microsoft.com/SQL/ServiceBroker/DialogTimer" message type.

**- ScheduledJobQueue** is the queue we use to post our DialogTimer messages to and run the usp\_RunScheduledJob activation procedure that runs the scheduled job.

**- [//ScheduledJobService]** is a service set on top of the ScheduledJobQueue and bound by the [//ScheduledJobContract] contract.

### ****4.Tying it all together****

Now that we have created all our objects let's see how they all work together.

First we have to have a valid SQL statement that we'll run as a scheduled job either daily or only once. We can add it to or remove it from the ScheduldJobs table by using our PS\_AddScheduledJob stored procedure. This procedure starts a new conversation and links it to our scheduled job. After that it sets the conversation timer to elapse at the date and time we want our job to run.

At this point we have our scheduled job lying nicely in a table and a timer that will run it at our time. When the scheduled time comes the dialog timer fires and service broker puts a DialogTimer message into the ScheduledJobQueue. The queue has an activation stored procedure PS\_RunScheduledJob associated with it which runs every time a new message arrives to the queue.

This activation stored procedure then receives our DialogTimer message from the queue, uses the conversation handle that comes with the message and looks up the job associated with that conversation handle. If our job is a run only once type it ends the conversation else it resets the timer to fire again in 24 hours. After that it runs our job. When the job finishes (either succeeds or fails) the status is written back to the ScheduledJobs table. And that's it.

We can also manually remove the job at any time with the PS\_DeleteScheduledJob stored procedure that ends the conversation and its timer from our job and then deletes a row from the ScheduledJobs table.

The whole infrastructure is quite simple and low maintenance.

Reference

<https://www.sqlteam.com/articles/scheduling-jobs-in-sql-server-express>

<https://www.sqlteam.com/articles/scheduling-jobs-in-sql-server-express-2>

https://docs.microsoft.com/en-us/sql/database-engine/configure-windows/sql-server-service-broker?view=sql-server-ver15