# Scheduling Jobs in SQL Server Express

By **Mladen Prajdić** on **27 August 2008** | 18 Comments | Tags: Backup/Restore, Service Broker

Article Series Navigation:

---

As we all know SQL Server 2005 Express is a very powerful free edition of SQL Server 2005. However it does not contain SQL Server Agent service. Because of this scheduling jobs is not possible. So if we want to do this we have to install a free or commercial 3rd party product. This usually isn't allowed due to the security policies of many hosting companies and thus presents a problem. Maybe we want to schedule daily backups, database reindexing, statistics updating, etc. This is why I wanted to have a solution based only on SQL Server 2005 Express and not dependent on the hosting company. And of course there is one based on our old friend the Service Broker.

## New terminology

To achieve scheduling we will use **SQL Server Service Broker**. If you're not familiar with this great addition to the storage engine go read my **previous** **three** **articles** about it. There you'll get familiarized with the terminology and database objects used in this article. Done? OK, let's move on.

So you're familiar with services, queues, activation procedures, messages, contracts, conversations, etc... The new member we have to take a look at is the

**Conversation Timer:**

```
BEGIN CONVERSATION TIMER ( conversation_handle )
   TIMEOUT = timeoutInSeconds
[ ; ]
```

When the conversation timer is set it waits the number of seconds specified in the timeout and then it sends the **http://schemas.microsoft.com/SQL/ServiceBroker/DialogTimer** message to the local queue that is the part of a conversation. It never sends the message to the remote queue. As the DialogTimer message comes to the queue, the activation stored procedure associated with the queue fires, receives the message from the queue and executes whatever logic we have programmed it to.

Don't mistake the conversation timer for the conversation lifetime! Each part of the conversation can have a different conversation timer set while the conversation lifetime is the time from the beginning to the end of the conversation.

## 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 two tables:

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

**- ScheduledJobsErrors** stores possible errors when manipulating scheduled jobs

```
CREATE TABLE ScheduledJobs
(
    ID INT IDENTITY(1,1),
    ScheduledSql nvarchar(max) NOT NULL,
    FirstRunOn datetime NOT NULL,
    LastRunOn datetime,
    LastRunOK BIT NOT NULL DEFAULT (0),
    IsRepeatable BIT NOT NULL DEFAULT (0),
    IsEnabled BIT NOT NULL DEFAULT (0),
    ConversationHandle uniqueidentifier NULL

)

CREATE TABLE ScheduledJobsErrors
(
    Id BIGINT IDENTITY(1, 1) PRIMARY KEY,
    ErrorLine INT,
    ErrorNumber INT,
    ErrorMessage NVARCHAR(MAX),
    ErrorSeverity INT,
    ErrorState INT,
    ScheduledJobId INT,
    ErrorDate DATETIME NOT NULL DEFAULT GETUTCDATE()
)
```

## 2. Needed stored procedures

For our simple scheduling we need three stored procedures. Only the pieces of code are shown here so look at the accompanying script for full code.

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.

**- usp_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.

```
INSERT INTO ScheduledJobs(ScheduledSql, FirstRunOn, IsRepeatable,
ConversationHandle)
VALUES (@ScheduledSql, @FirstRunOn, @IsRepeatable, NULL)
```

```
SELECT @ScheduledJobId = SCOPE_IDENTITY()

...

BEGIN DIALOG CONVERSATION @ConversationHandle
    FROM SERVICE   [//ScheduledJobService]

    TO SERVICE      '//ScheduledJobService',
                    'CURRENT DATABASE'
    ON CONTRACT     [//ScheduledJobContract]

    WITH ENCRYPTION = OFF;

BEGIN CONVERSATION TIMER (@ConversationHandle)
TIMEOUT = @TimeoutInSeconds;
```

**- usp_RemoveScheduledJob** 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.

```
IF EXISTS (SELECT *
        FROM sys.conversation_endpoints
        WHERE conversation_handle = @ConversationHandle)
    END CONVERSATION @ConversationHandle

DELETE ScheduledJobs WHERE Id = @ScheduledJobId
```

**- usp_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** which we'd have to again handle separately. We want to keep things simple.

```
RECEIVE TOP(1)
        @ConversationHandle = conversation_handle,
        @message_type_name = message_type_name
FROM ScheduledJobQueue

...


SELECT  @ScheduledJobId = ID,
       @ScheduledSql = ScheduledSql,
       @IsRepeatable = IsRepeatable

FROM    ScheduledJobs
WHERE   ConversationHandle = @ConversationHandle AND IsEnabled = 1

...

-- run our job
EXEC (@ScheduledSql)
```

## 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.

```
CREATE CONTRACT [//ScheduledJobContract]

    ([http://schemas.microsoft.com/SQL/ServiceBroker/DialogTimer] SENT BY
INITIATOR)

CREATE QUEUE ScheduledJobQueue
    WITH STATUS = ON,
    ACTIVATION (
        PROCEDURE_NAME = usp_RunScheduledJob,
        MAX_QUEUE_READERS = 20, -- we expect max 20 jobs to start
simultaneously

        EXECUTE AS 'dbo' );

CREATE SERVICE [//ScheduledJobService]
    AUTHORIZATION dbo
    ON QUEUE ScheduledJobQueue ([//ScheduledJobContract])
```

## 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 usp_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 usp_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 usp_RemoveScheduledJob 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.

## 5. How to Schedule Jobs - Example

Here is an example with three scheduled jobs: a daily backup job of our test database, a faulty script and a one time update of statistics. All are run 30 seconds after you add them with the usp_AddScheduledJob stored procedure.

```
GO
DECLARE @ScheduledSql nvarchar(max), @RunOn datetime, @IsRepeatable BIT

SELECT  @ScheduledSql = N'DECLARE @backupTime DATETIME, @backupFile
NVARCHAR(512);
    SELECT @backupTime = GETDATE(),
           @backupFile = ''C:\TestScheduledJobs_'' +
                replace(replace(CONVERT(NVARCHAR(25), @backupTime, 120),
                '' '', ''_''), '':'', ''_'') + N''.bak'';
           BACKUP DATABASE TestScheduledJobs TO DISK = @backupFile;',
        @RunOn = dateadd(s, 30, getdate()),
        @IsRepeatable = 0

EXEC usp_AddScheduledJob @ScheduledSql, @RunOn, @IsRepeatable
GO

DECLARE @ScheduledSql nvarchar(max), @RunOn datetime, @IsRepeatable BIT

SELECT  @ScheduledSql = N'select 1, where 1=1',
        @RunOn = dateadd(s, 30, getdate()),
        @IsRepeatable = 1

EXEC usp_AddScheduledJob @ScheduledSql, @RunOn, @IsRepeatable
GO

DECLARE @ScheduledSql nvarchar(max), @RunOn datetime, @IsRepeatable BIT

SELECT  @ScheduledSql = N'EXEC sp_updatestats;',
        @RunOn = dateadd(s, 30, getdate()),
        @IsRepeatable = 0

EXEC usp_AddScheduledJob @ScheduledSql, @RunOn, @IsRepeatable
GO
```

## 6. Monitoring

We can monitor our scheduled jobs' conversations currently being processed and their success by running these queries:

```
-- show the currently active conversations.

-- Look at dialog_timer column (in UTC time) to see when will the job be
run next
SELECT * FROM sys.conversation_endpoints
-- shows the number of currently executing activation procedures
```

```
SELECT * FROM sys.dm_broker_activated_tasks

-- see how many unreceived messages are still in the queue.
-- should be 0 when no jobs are running
SELECT * FROM ScheduledJobQueue with (nolock)

-- view our scheduled jobs' statuses
SELECT * FROM ScheduledJobs  with (nolock)
-- view any scheduled jobs errors that might have happend

SELECT * FROM ScheduledJobsErrors  with (nolock)
```

# Conclusion

Here we've seen how to build a simple job scheduler that can schedule jobs once or daily at the time **specified in the database we run this in**. The complete code can be found in this file.