**[An Introduction to Asynchronous Processing with Service Broker](http://sqlperformance.com/2014/03/sql-performance/intro-to-service-broker" \o "An Introduction to Asynchronous Processing with Service Broker)**

 Posted by [Jonathan Kehayias](http://sqlperformance.com/author/jonathansqlskills-com) on March 18, 2014  [Add comments](http://sqlperformance.com/2014/03/sql-performance/intro-to-service-broker#respond)

Mar 182014

FROM: http://sqlperformance.com/2014/03/sql-performance/intro-to-service-broker

I love modifying SQL Server code to improve performance, but there are occasionally scenarios where even after tuning the code, indexes and design a user task from the application takes longer to complete than the expected end-user experience. When this happens the user interface either has to wait for the process to complete or we have to come up with an alternate way of handling the task. The asynchronous processing provided by Service Broker is a good fit for many of these scenarios and allows background processing of the long running task to be performed separately from the user interface, allowing the user to continue working immediately without waiting for the task to actually be performed. Over my next few articles, I hope to create a series on how you can leverage Service Broker with the appropriate explanations and code examples along the way to make it easier to leverage Service Broker's capabilities without implementation problems.

**Methods of Performing Asynchronous Processing**

There are a number of ways to deal with a long running, but already tuned process. The application code can also be rewritten to use a BackgroundWorker, the background ThreadPool, or a manually written Thread based solution in .NET that performs the operation asynchronously. However, this allows for an unlimited number of these long running processes to be submitted by the application, unless additional coding work is done to track and limit the number of active processes. This means that the application will have a potential performance impact, or under load will hit a limit and return to the previous waiting we were trying to prevent originally.

I've also seen these type of processes turned into SQL Agent jobs tied to a table that is used to store the information to process. Then the job is either scheduled to run periodically, or is started by the application using sp\_start\_job when a change is stored for processing. However, this only allows for a serial execution of the long running processes, since SQL Agent doesn't allow a job to be run multiple times concurrently. The job would also have to be designed to handle scenarios where multiple rows enter the processing table so that the correct order of processing occurs and subsequent submissions are processed separately.

Leveraging Service Broker for asynchronous processing in SQL Server actually addresses the limitations with the previously mentioned methods for handling the asynchronous processing. The broker implementation allows new tasks to be queued for asynchronous processing in the background, and also allows for parallel processing of the tasks that have been queued up to a configured limit. However, unlike the application tier having to wait when the limit is hit, the broker solution simply queues the new message being received and allows it to be processed when one of the current processing tasks completes — this allows the application to continue without waiting.

**Single Database Service Broker Configuration**

While Service Broker configurations can become complex, for simple asynchronous processing you only need to know the basic concepts to build single database configuration. A single database configuration only requires:

1. Creating two message types
   * One for requesting the asynchronous processing
   * One for the return message when the processing completes
2. A contract using the message types
   * Defines which message type is sent by the initiator service and which message type is returned by the target service
3. A queue, service, and activation procedure for the target
   * The queue provides the storage of messages sent to the target service by the initiator service
   * The activation procedure automates the processing of messages from the queue
     + Returns a completed message to the initiator service when it completes processing a requested task
     + Handles the http://schemas.microsoft.com/SQL/ServiceBroker/EndDialog and http://schemas.microsoft.com/SQL/ServiceBroker/Error system message types
4. A queue, service, and activation procedure for the initiator
   * The queue provides the storage of messages sent to the service
   * The activation procedure is optional but automates the processing of messages from the queue
     + Processes the completed message to the target service and ends the conversation
     + Handles the http://schemas.microsoft.com/SQL/ServiceBroker/EndDialog and http://schemas.microsoft.com/SQL/ServiceBroker/Error system message types

In addition to these basic components, I prefer to use a wrapper stored procedure for creating a conversation and sending messages between broker services to keep the code clean, and make it easier to scale as necessary by implementing [conversation reuse](http://rusanu.com/2007/04/25/reusing-conversations/) or the [150 conversation trick](http://msdn.microsoft.com/en-us/library/dd576261.aspx) explained in the SQLCAT team whitepaper. For many of the simple asynchronous processing configurations, these performance tuning techniques might not need to be implemented.  However, by using a wrapper stored procedure, it becomes much easier to change a single point in code, instead of changing every procedure that sends a message in the future, should it become necessary.

If you haven't given Service Broker a look, it might provide an alternate method of performing decoupled processing asynchronously to solve a number of possible scenarios. In [my next post](http://www.sqlperformance.com/2014/03/sql-performance/configuring-service-broker) we'll walk through the source code for an example implementation and explain where specific changes would need to be made to leverage the code for asynchronous processing.

**Share this post:**

* [Facebook](http://sqlperformance.com/2014/03/sql-performance/intro-to-service-broker?share=facebook&nb=1)
* [Twitter](http://sqlperformance.com/2014/03/sql-performance/intro-to-service-broker?share=twitter&nb=1)
* [Google](http://sqlperformance.com/2014/03/sql-performance/intro-to-service-broker?share=google-plus-1&nb=1)
* [LinkedIn15](http://sqlperformance.com/2014/03/sql-performance/intro-to-service-broker?share=linkedin&nb=1)

 Tagged with: [asynchronous](http://sqlperformance.com/tag/asynchronous), [service broker](http://sqlperformance.com/tag/service-broker)

**3 Responses to “An Introduction to Asynchronous Processing with Service Broker”**

1. [**New Article: Asynchronous Processing with Service Broker - Jonathan Kehayias**](http://www.sqlskills.com/blogs/jonathan/new-article-asynchronous-processing-service-broker/) says:

[March 18, 2014 at 4:19 PM](http://sqlperformance.com/2014/03/sql-performance/intro-to-service-broker#comment-75115)

[…] An Introduction to Asynchronous Processing with Service Broker […]

1. **Rick Willemain** says:

[March 19, 2014 at 6:15 PM](http://sqlperformance.com/2014/03/sql-performance/intro-to-service-broker#comment-75441)

Very happy to see this topic / service broker usage presented !  
Thank you !!

[Reply](http://sqlperformance.com/2014/03/sql-performance/intro-to-service-broker?replytocom=75441#respond)

1. [**(SFTW) SQL Server Links 21/03/14 • John Sansom**](http://www.johnsansom.com/sftw-sql-server-links-210314/) says:

[March 21, 2014 at 12:18 PM](http://sqlperformance.com/2014/03/sql-performance/intro-to-service-broker#comment-75949)

[…] An Introduction to Asynchronous Processing with Service Broker – Jonathan Kehayias (Blog|Twitter) […]