Amazon Simple Queue Service
Developer Guide
Table of Contents

What is Amazon SQS? ................................................................. 1
What Can I Use Amazon SQS For? .................................................. 1
What Type of Queue Do I Need? .................................................... 2
What Are the Main Features of Amazon SQS? .............................. 2
What is the Basic Architecture of Amazon SQS? ....................... 3

Setting Up ................................................................................. 4
Step 1: Create an AWS Account .................................................. 4
Step 2: Create an IAM User ......................................................... 4
Step 3: Get Your Access Key ID and Secret Access Key ............... 5
Step 4: Get Ready to Use the Example Code ............................ 6
Next Steps .................................................................................. 6

Getting Started ......................................................................... 7
Prerequisites ............................................................................. 7
Step 1: Create a Queue .............................................................. 7
Step 2: Send a Message ............................................................ 8
Step 3: Receive and Delete Your Message .................................. 10
Step 4: Delete Your Queue ....................................................... 12
Next Steps ................................................................................ 13

Tutorials .................................................................................... 15
Creating a Queue ...................................................................... 15
   AWS Management Console .................................................. 16
   Java ...................................................................................... 17
   AWS CloudFormation .......................................................... 18
Creating a Queue with SSE ....................................................... 20
   AWS Management Console .................................................. 20
   Java ...................................................................................... 22
Configuring SSE for a Queue .................................................... 24
   AWS Management Console .................................................. 24
   Java ...................................................................................... 25
Listing All Queues .................................................................... 27
   AWS Management Console .................................................. 27
   Java ...................................................................................... 27
Adding Permissions to a Queue .................................................. 28
   AWS Management Console .................................................. 28
Sending a Message ................................................................... 29
   AWS Management Console .................................................. 29
   Java ...................................................................................... 31
Receiving and Deleting a Message ............................................. 32
   AWS Management Console .................................................. 32
   Java ...................................................................................... 35
Configuring a Dead-Letter Queue .............................................. 37
   AWS Management Console .................................................. 37
   Java ...................................................................................... 39
Purging a Queue ....................................................................... 40
   AWS Management Console .................................................. 40
Deleting a Queue ..................................................................... 41
   AWS Management Console .................................................. 41
   Java ...................................................................................... 42
Subscribing a Queue to a Topic ............................................... 42
   AWS Management Console .................................................. 42
Adding, Updating, and Removing Tags for a Queue ................. 44
   AWS Management Console .................................................. 44
   Java ...................................................................................... 44
How Queues Work .................................................................... 46
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Prerequisites</td>
<td>46</td>
</tr>
<tr>
<td>Standard Queues</td>
<td>47</td>
</tr>
<tr>
<td>Message Ordering</td>
<td>47</td>
</tr>
<tr>
<td>At-Least-Once Delivery</td>
<td>48</td>
</tr>
<tr>
<td>Consuming Messages Using Short Polling</td>
<td>48</td>
</tr>
<tr>
<td>Getting Started with Standard Queues</td>
<td>48</td>
</tr>
<tr>
<td>FIFO Queues</td>
<td>51</td>
</tr>
<tr>
<td>Message Ordering</td>
<td>51</td>
</tr>
<tr>
<td>FIFO Queue Logic</td>
<td>52</td>
</tr>
<tr>
<td>Exactly-Once Processing</td>
<td>53</td>
</tr>
<tr>
<td>Getting Started with FIFO Queues</td>
<td>53</td>
</tr>
<tr>
<td>Moving from a Standard Queue to a FIFO Queue</td>
<td>56</td>
</tr>
<tr>
<td>Compatibility</td>
<td>56</td>
</tr>
<tr>
<td>Queue and Message Identifiers</td>
<td>57</td>
</tr>
<tr>
<td>General Identifiers</td>
<td>57</td>
</tr>
<tr>
<td>Additional Identifiers for FIFO Queues</td>
<td>58</td>
</tr>
<tr>
<td>Resources Required to Process Messages</td>
<td>58</td>
</tr>
<tr>
<td>Visibility Timeout</td>
<td>59</td>
</tr>
<tr>
<td>Inflight Messages</td>
<td>60</td>
</tr>
<tr>
<td>Setting the Visibility Timeout</td>
<td>60</td>
</tr>
<tr>
<td>Changing the Visibility Timeout for a Message</td>
<td>60</td>
</tr>
<tr>
<td>Terminating the Visibility Timeout for a Message</td>
<td>60</td>
</tr>
<tr>
<td>Visibility Timeout API Actions</td>
<td>61</td>
</tr>
<tr>
<td>Dead-Letter Queues</td>
<td>61</td>
</tr>
<tr>
<td>How Do Dead-Letter Queues Work?</td>
<td>61</td>
</tr>
<tr>
<td>What are the Benefits of Dead-Letter Queues?</td>
<td>62</td>
</tr>
<tr>
<td>How Do Different Queue Types Handle Message Failure?</td>
<td>62</td>
</tr>
<tr>
<td>When Should I Use a Dead-Letter Queue?</td>
<td>63</td>
</tr>
<tr>
<td>Getting Started with Dead-Letter Queues</td>
<td>63</td>
</tr>
<tr>
<td>Troubleshooting Dead-Letter Queues</td>
<td>63</td>
</tr>
<tr>
<td>Message Lifecycle</td>
<td>64</td>
</tr>
<tr>
<td>Cost Allocation Tags</td>
<td>65</td>
</tr>
<tr>
<td>Overview</td>
<td>65</td>
</tr>
<tr>
<td>Getting Started with Tagging</td>
<td>65</td>
</tr>
<tr>
<td>Message Attributes</td>
<td>66</td>
</tr>
<tr>
<td>Message Attribute Items and Validation</td>
<td>66</td>
</tr>
<tr>
<td>Data Types</td>
<td>67</td>
</tr>
<tr>
<td>Using Message Attributes with the AWS Management Console</td>
<td>67</td>
</tr>
<tr>
<td>Using Message Attributes with the AWS SDKs</td>
<td>70</td>
</tr>
<tr>
<td>Using Message Attributes with the Amazon SQS Query API</td>
<td>71</td>
</tr>
<tr>
<td>MD5 Message-Digest Calculation</td>
<td>72</td>
</tr>
<tr>
<td>Long Polling</td>
<td>73</td>
</tr>
<tr>
<td>The Differences Between Short and Long Polling</td>
<td>74</td>
</tr>
<tr>
<td>Enabling Long Polling using the AWS Management Console</td>
<td>74</td>
</tr>
<tr>
<td>Enabling Long Polling Using the API</td>
<td>76</td>
</tr>
<tr>
<td>Enabling Long Polling Using the Query API</td>
<td>77</td>
</tr>
<tr>
<td>Delay Queues</td>
<td>77</td>
</tr>
<tr>
<td>Creating Delay Queues with the AWS Management Console</td>
<td>78</td>
</tr>
<tr>
<td>Creating Delay Queues with the Query API</td>
<td>80</td>
</tr>
<tr>
<td>Message Timers</td>
<td>81</td>
</tr>
<tr>
<td>Creating Message Timers Using the Console</td>
<td>81</td>
</tr>
<tr>
<td>Creating Message Timers Using the Query API</td>
<td>83</td>
</tr>
<tr>
<td>Managing Large Messages Using Amazon S3</td>
<td>84</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>85</td>
</tr>
<tr>
<td>Using the Extended Client Library for Java</td>
<td>85</td>
</tr>
<tr>
<td>Amazon SQS and JMS</td>
<td>88</td>
</tr>
<tr>
<td>Chapter</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Best Practices</td>
<td>109</td>
</tr>
<tr>
<td>General Recommendations</td>
<td>109</td>
</tr>
<tr>
<td>Processing Messages</td>
<td>109</td>
</tr>
<tr>
<td>Reducing Costs</td>
<td>110</td>
</tr>
<tr>
<td>Moving from a Standard Queue to a FIFO Queue</td>
<td>110</td>
</tr>
<tr>
<td>Recommendations for FIFO Queues</td>
<td>111</td>
</tr>
<tr>
<td>Using the Message Deduplication ID</td>
<td>111</td>
</tr>
<tr>
<td>Using the Message Group ID</td>
<td>112</td>
</tr>
<tr>
<td>Using the Receive Request Attempt ID</td>
<td>112</td>
</tr>
<tr>
<td>Limits</td>
<td>113</td>
</tr>
<tr>
<td>Limits Related to Queues</td>
<td>113</td>
</tr>
<tr>
<td>Limits Related to Messages</td>
<td>114</td>
</tr>
<tr>
<td>Limits Related to Policies</td>
<td>115</td>
</tr>
<tr>
<td>Monitoring and Logging</td>
<td>116</td>
</tr>
<tr>
<td>Monitoring Amazon SQS using CloudWatch</td>
<td>116</td>
</tr>
<tr>
<td>Common Monitoring Tasks</td>
<td>117</td>
</tr>
<tr>
<td>Access CloudWatch Metrics for Amazon SQS</td>
<td>117</td>
</tr>
<tr>
<td>Set CloudWatch Alarms for Amazon SQS Metrics</td>
<td>120</td>
</tr>
<tr>
<td>Available CloudWatch Metrics for Amazon SQS</td>
<td>123</td>
</tr>
<tr>
<td>Logging Amazon SQS API Actions Using CloudTrail</td>
<td>126</td>
</tr>
<tr>
<td>Amazon SQS Information in CloudTrail</td>
<td>126</td>
</tr>
<tr>
<td>Understanding Amazon SQS Log File Entries</td>
<td>127</td>
</tr>
<tr>
<td>Security</td>
<td>131</td>
</tr>
<tr>
<td>Authentication and Access Control</td>
<td>131</td>
</tr>
<tr>
<td>Authentication</td>
<td>131</td>
</tr>
<tr>
<td>Access Control</td>
<td>132</td>
</tr>
<tr>
<td>Overview of Managing Access</td>
<td>133</td>
</tr>
<tr>
<td>Using Identity-Based Policies (IAM) Policies for Amazon SQS</td>
<td>137</td>
</tr>
<tr>
<td>Creating Custom Policies Using the Amazon SQS Access Policy Language</td>
<td>144</td>
</tr>
<tr>
<td>Using Temporary Security Credentials</td>
<td>152</td>
</tr>
<tr>
<td>Amazon SQS API Permissions Reference</td>
<td>154</td>
</tr>
<tr>
<td>Server-Side Encryption</td>
<td>156</td>
</tr>
<tr>
<td>Benefits of Server-Side Encryption</td>
<td>156</td>
</tr>
<tr>
<td>What Does SSE for Amazon SQS Encrypt?</td>
<td>157</td>
</tr>
<tr>
<td>Key Terms</td>
<td>157</td>
</tr>
<tr>
<td>How Does the Data Key Reuse Period Work?</td>
<td>158</td>
</tr>
<tr>
<td>How Do I Estimate My AWS KMS Usage Costs?</td>
<td>159</td>
</tr>
<tr>
<td>What Permissions Do I Need to Use SSE?</td>
<td>160</td>
</tr>
<tr>
<td>Getting Started with SSE</td>
<td>161</td>
</tr>
<tr>
<td>Errors</td>
<td>161</td>
</tr>
<tr>
<td>Working with APIs</td>
<td>163</td>
</tr>
<tr>
<td>Making API Requests</td>
<td>163</td>
</tr>
<tr>
<td>Endpoints</td>
<td>164</td>
</tr>
<tr>
<td>Making Query Requests</td>
<td>164</td>
</tr>
<tr>
<td>Request Authentication</td>
<td>166</td>
</tr>
<tr>
<td>Responses</td>
<td>172</td>
</tr>
<tr>
<td>Shared Queues</td>
<td>173</td>
</tr>
<tr>
<td>Programming Languages</td>
<td>175</td>
</tr>
<tr>
<td>Batch API Actions</td>
<td>176</td>
</tr>
<tr>
<td>Maximum Message Size for SendMessageBatch</td>
<td>176</td>
</tr>
<tr>
<td>Client-Side Buffering and Request Batching</td>
<td>176</td>
</tr>
<tr>
<td>Increasing Throughput with Horizontal Scaling and Batching</td>
<td>179</td>
</tr>
</tbody>
</table>
What is Amazon Simple Queue Service?

Amazon Simple Queue Service (Amazon SQS) offers a reliable, highly-scalable hosted queue for storing messages as they travel between applications or microservices. It moves data between distributed application components and helps you decouple these components. Amazon SQS provides familiar middleware constructs such as dead-letter queues and poison-pill management. It also provides a generic web services API and can be accessed by any programming language that the AWS SDK supports. Amazon SQS supports both standard (p. 47) and FIFO queues (p. 51).

Topics
• What Can I Use Amazon SQS For? (p. 1)
• What Type of Queue Do I Need? (p. 2)
• What Are the Main Features of Amazon SQS? (p. 2)
• What is the Basic Architecture of Amazon SQS? (p. 3)

What Can I Use Amazon SQS For?

Use Amazon SQS when you need each unique message to be consumed only once and for cases such as the following:

• Decoupling the components of an application – You have a queue of work items and want to track the successful completion of each item independently. Amazon SQS tracks the ACK/FAIL results, so the application does not have to maintain a persistent checkpoint or cursor. After a configured visibility timeout, Amazon SQS deletes acknowledged messages and redelivers failed messages.

• Configuring individual message delay – You have a job queue and you need to schedule individual jobs with a delay. With standard queues, you can configure individual messages to have a delay of up to 15 minutes.

• Dynamically increasing concurrency or throughput at read time – You have a work queue and want to add more consumers until the backlog is cleared. Amazon SQS requires no pre-provisioning.

• Scaling transparently – You buffer requests and the load changes as a result of occasional load spikes or the natural growth of your business. Because Amazon SQS can process each buffered request
independently, Amazon SQS can scale transparently to handle the load without any provisioning instructions from you.

## What Type of Queue Do I Need?

<table>
<thead>
<tr>
<th>Standard Queue</th>
<th>FIFO Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available in all regions.</td>
<td>Available in the US East (N. Virginia), US East (Ohio), US West (Oregon), and EU (Ireland) regions.</td>
</tr>
<tr>
<td><strong>High Throughput</strong> – Standard queues can support a nearly unlimited number of transactions per second (TPS) per API action.</td>
<td><strong>First-In-First-Out Delivery</strong> – The order in which messages are sent and received is strictly preserved.</td>
</tr>
<tr>
<td><strong>At-Least-Once Delivery</strong> – A message is delivered at least once, but occasionally more than one copy of a message is delivered.</td>
<td><strong>Exactly-Once Processing</strong> – A message is delivered once and remains available until a consumer processes and deletes it. Duplicates are not introduced into the queue.</td>
</tr>
<tr>
<td><strong>Best-Effort Ordering</strong> – Occasionally, messages might be delivered in an order different from which they were sent.</td>
<td><strong>Limited Throughput</strong> – Without batching, FIFO queues can support up to 300 messages per second (300 send, receive, or delete operations per second). If you take advantage of the maximum batching (p. 176) of 10 messages per operation, FIFO queues can support up to 3,000 messages per second.</td>
</tr>
</tbody>
</table>

Send data between applications when the throughput is important, for example:

- Decouple live user requests from intensive background work: let users upload media while resizing or encoding it.
- Allocate tasks to multiple worker nodes: process a high number of credit card validation requests.
- Batch messages for future processing: schedule multiple entries to be added to a database.

Send data between applications when the order of events is important, for example:

- Ensure that user-entered commands are executed in the right order.
- Display the correct product price by sending price modifications in the right order.
- Prevent a student from enrolling in a course before registering for an account.

## What Are the Main Features of Amazon SQS?

Amazon SQS provides the following major features:

- **Redundant infrastructure** – Standard queues support at-least-once message delivery, while FIFO queues support exactly-once message processing. Amazon SQS provides highly-concurrent access to messages and high availability for producing and consuming messages.
What is the Basic Architecture of Amazon SQS?

There are three main actors in the overall system:

- The components of your distributed system
- Queues
- Messages in the queues

In the following diagram, your system has several components that send messages to the queue and receive messages from the queue. The diagram shows that a single queue, which has its messages (A-E), is redundantly saved across multiple Amazon SQS servers.

- **Multiple producers and consumers** – Multiple parts of your system can send or receive messages at the same time. Amazon SQS locks the message during processing, keeping other parts of your system from processing the message simultaneously.
- **Configurable settings per queue** – All of your queues don't have to be exactly alike. For example, you can optimize one queue for messages that require a longer processing time than others.
- **Variable message size** – Your messages can be up to 262,144 bytes (256 KB) in size. You can store the contents of larger messages using the Amazon Simple Storage Service (Amazon S3) or Amazon DynamoDB, with Amazon SQS holding a pointer to the Amazon S3 object. For more information, see Managing Amazon SQS Messages with Amazon S3. You can also split a large message into smaller ones.
- **Access control** – You control who can send messages to a queue, and who can receive messages from a queue.
- **Delay queues** – You can set a default delay on a queue, so that delivery of all enqueued messages is postponed for the specified duration. You can set the delay value when you create a queue with `CreateQueue`, and you can update the value with `SetQueueAttributes`. If you update the value, the new value affects only messages enqueued after the update.
Setting Up Amazon SQS

Before you can use Amazon SQS for the first time, you must complete the following steps.

**Step 1: Create an AWS Account**

To access any AWS service, you first need to create an AWS account, an Amazon.com account that can use AWS products. You can use your AWS account to view your activity and usage reports and to manage authentication and access.

To avoid using your AWS account root user for Amazon SQS operations, it is a best practice to create an IAM user for each person who needs administrative access to Amazon SQS.

**To set up a new account**

1. Open https://aws.amazon.com/, and then choose Create an AWS Account.
   
   **Note**
   
   This might be unavailable in your browser if you previously signed into the AWS Management Console. In that case, choose Sign In to the Console, and then choose Create a new AWS account.

2. Follow the online instructions.
   
   Part of the sign-up procedure involves receiving a phone call and entering a PIN using the phone keypad.

**Step 2: Create an IAM User**

**To create an IAM user for yourself and add the user to an Administrators group**

1. Use your AWS account email address and password to sign in to the AWS Management Console as the AWS account root user.
2. In the navigation pane of the console, choose Users, and then choose Add user.
3. For User name, type Administrator.
4. Select the check box next to AWS Management Console access, select Custom password, and then type the new user's password in the text box. You can optionally select Require password reset to force the user to select a new password the next time the user signs in.
5. Choose Next: Permissions.
6. On the Set permissions for user page, choose Add user to group.
7. Choose Create group.
8. In the Create group dialog box, type Administrators.
9. For Filter, choose Job function.
10. In the policy list, select the check box for AdministratorAccess. Then choose Create group.
11. Back in the list of groups, select the check box for your new group. Choose Refresh if necessary to see the group in the list.
12. Choose Next: Review to see the list of group memberships to be added to the new user. When you are ready to proceed, choose Create user.

You can use this same process to create more groups and users, and to give your users access to your AWS account resources. To learn about using policies to restrict users' permissions to specific AWS resources, go to Access Management and Example Policies.

Step 3: Get Your Access Key ID and Secret Access Key

To use Amazon SQS API actions (for example, using Java or through the AWS Command Line Interface), you need an access key ID and a secret access key.

Note
The access key ID and secret access key are specific to AWS Identity and Access Management. Don't confuse them with credentials for other AWS services, such as Amazon EC2 key pairs.

To get the access key ID and secret access key for an IAM user

Access keys consist of an access key ID and secret access key, which are used to sign programmatic requests that you make to AWS. If you don't have access keys, you can create them from the AWS Management Console. We recommend that you use IAM access keys instead of AWS account root user access keys. IAM lets you securely control access to AWS services and resources in your AWS account.

The only time that you can view or download the secret access keys is when you create the keys. You cannot recover them later. However, you can create new access keys at any time. You must also have permissions to perform the required IAM actions. For more information, see Delegating Permissions to Administer IAM Users, Groups, and Credentials in the IAM User Guide.

1. Open the IAM console.
2. In the navigation pane of the console, choose Users.
3. Choose your IAM user name (not the check box).
4. Choose the Security credentials tab and then choose Create access key.
5. To see the new access key, choose Show. Your credentials will look something like this:
   • Access key ID: AKIAIOSFODNN7EXAMPLE
   • Secret access key: wJalrXUtNfEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
6. To download the key pair, choose Download .csv file. Store the keys in a secure location.

Keep the keys confidential in order to protect your account, and never email them. Do not share them outside your organization, even if an inquiry appears to come from AWS or Amazon.com. No one who legitimately represents Amazon will ever ask you for your secret key.

Related topics
Step 4: Get Ready to Use the Example Code

This guide shows how to work with Amazon SQS using the AWS Management Console and using Java. If you want to use the example code, you must install the Java Standard Edition Development Kit and make some configuration changes to the example code.

You can write code in other programming languages. For more information, see the documentation of the AWS SDKs.

Note
You can explore Amazon SQS without writing code with tools such as the AWS Command Line Interface (AWS CLI) or Windows PowerShell. You can find AWS CLI examples in the Amazon SQS section of the AWS Command Line Interface Reference. You can find Windows PowerShell examples in the Amazon Simple Queue Service section of the AWS Tools for Windows PowerShell Reference.

Next Steps

Now that you're prepared for working with Amazon SQS, can get started (p. 7) with managing Amazon SQS queues and messages using the AWS Management Console. You can also try the more advanced Amazon SQS tutorials (p. 15).
Getting Started with Amazon SQS

This section helps you become more familiar with Amazon SQS by showing you how to manage queues and messages using the AWS Management Console.

Note
The Amazon Simple Queue Service Getting Started Guide has been retired. If you want to work with Amazon SQS programmatically, see the Amazon SQS Tutorials (p. 15) and Working with Amazon SQS APIs (p. 163) sections.

Prerequisites

Before you begin, complete the steps in Setting Up Amazon SQS (p. 4).

Step 1: Create a Queue

The first and most common Amazon SQS task is creating queues. The following example demonstrates how to create and configure a queue.

1. Sign in to the Amazon SQS console.
2. Choose Create New Queue.
3. On the Create New Queue page, ensure that you’re in the correct region and then type the Queue Name.

   Note
   The name of a FIFO queue must end with the .fifo suffix. FIFO queues are available in the US East (N. Virginia), US East (Ohio), US West (Oregon), and EU (Ireland) regions.

4. Standard is selected by default. Choose FIFO.
5. To create your queue with the default parameters, choose **Quick-Create Queue**.

Your new queue is created and selected in the queue list.

**Note**
When you create a queue, it can take a short time for the queue to propagate throughout Amazon SQS.

The **Queue Type** column helps you distinguish standard queues from FIFO queues at a glance. For a FIFO queue, the **Content-Based Deduplication** column displays whether you have enabled **exactly-once processing** (p. 53).

Your queue's **Name**, **URL**, and **ARN** are displayed on the **Details** tab.

**Step 2: Send a Message**

After you create your queue, you can send a message to it. The following example demonstrates sending a message to an existing queue.

1. From the queue list, select the queue that you've created.
2. From **Queue Actions**, select **Send a Message**.

The **Send a Message to QueueName** dialog box is displayed.

The following example shows the **Message Group ID** and **Message Deduplication ID** parameters specific to FIFO queues (**content-based deduplication (p. 53) is disabled**).

3. To send a message to a FIFO queue, type the **Message Body**, the **Message Group ID** `MyMessageGroupId1234567890`, and the **Message Deduplication ID** `MyMessageDeduplicationId1234567890`, and then choose **Send Message**. For more information, see **FIFO Queue Logic (p. 52)**.

   **Note**
   The message group ID is always required. However, if content-based deduplication is enabled, the message deduplication ID is optional.
Your message is sent and the **Send a Message to QueueName** dialog box is displayed, showing the attributes of the sent message.

The following example shows the **Sequence Number** attribute specific to FIFO queues.

4. Choose **Close**.

**Step 3: Receive and Delete Your Message**

After you send a message into a queue, you can consume it (retrieve it from the queue). When you request a message from a queue, you can’t specify which message to get. Instead, you specify the maximum number of messages (up to 10) that you want to get.

The following example demonstrates receiving and deleting a message.

1. From the queue list, select the queue that you have created.
2. From **Queue Actions**, select **View/Delete Messages**.
The View/Delete Messages in QueueName dialog box is displayed.

Note
The first time you take this action, an information screen is displayed. To hide the screen, check the Don't show this again checkbox.

3. Choose Start Polling for messages.

Amazon SQS begins to poll the messages in the queue. The dialog box displays a message from the queue. A progress bar at the bottom of the dialog box displays the status of the message's visibility timeout.

The following example shows the Message Group ID, Message Deduplication ID, and Sequence Number columns specific to FIFO queues.

4. Before the visibility timeout expires, select the message that you want to delete and then choose Delete 1 Message.
Step 4: Delete Your Queue

If you don't use an Amazon SQS queue (and don't foresee using it in the near future), it is a best practice to delete it from Amazon SQS. The following example demonstrates deleting a queue.

1. From the queue list, select the queue that you have created.

   ![Queue List]

2. From Queue Actions, select Delete Queue.

   ![Delete Queue]

5. Confirm that the message you want to delete is checked and choose Yes, Delete Checked Messages.

   The selected message is deleted.

   When the progress bar is filled in, the visibility timeout (p. 59) expires and the message becomes visible to consumers.


The Delete Messages dialog box is displayed.

![Delete Messages]

![Delete 1 Message]
3. Choose **Yes, Delete Queue**.

The queue is deleted.

---

**Next Steps**

Now that you've created a queue and learned how to send, receive, and delete messages and how to delete a queue, you might want to try the following:

- Enable server-side encryption for a new queue (p. 20) (or for an existing queue (p. 24)).
- Add permissions to a queue. (p. 28)
- Purge a queue. (p. 40)
- Configure a dead-letter queue. (p. 37)
- Subscribe a queue to an Amazon SNS topic. (p. 42)
- Add, update, or remove tags for a queue (p. 44).
- Learn more about Amazon SQS workflows and processes: Read How Queues Work (p. 46), Best Practices (p. 109), and Limits (p. 113). You can also explore the Amazon SQS Articles & Tutorials. If you ever have any questions, browse the Amazon SQS FAQs or participate in the Amazon SQS Developer Forums.
- Learn how to interact with Amazon SQS programmatically: Read Working with APIs (p. 163) and explore the Sample Code and Libraries and the developer centers:
  - Java
  - JavaScript
  - PHP
  - Python
  - Ruby
  - Windows & .NET
- Learn about keeping an eye on costs and resources: Start by reading the Monitoring and Logging (p. 116) section.
• Learn about protecting your data and access to it: Start by reading the Security (p. 131) section.
Amazon SQS Tutorials

This guide shows how to work with Amazon SQS using the AWS Management Console and using Java. If you want to use the example code, you must install the Java Standard Edition Development Kit and make some configuration changes to the example code.

You can write code in other programming languages. For more information, see the documentation of the AWS SDKs.

Note
You can explore Amazon SQS without writing code with tools such as the AWS Command Line Interface (AWS CLI) or Windows PowerShell. You can find AWS CLI examples in the Amazon SQS section of the AWS Command Line Interface Reference. You can find Windows PowerShell examples in the Amazon Simple Queue Service section of the AWS Tools for Windows PowerShell Reference.

Topics
- Tutorial: Creating an Amazon SQS Queue (p. 15)
- Tutorial: Creating an Amazon SQS Queue with Server-Side Encryption (p. 20)
- Tutorial: Configuring Server-Side Encryption (SSE) for an Existing Amazon SQS Queue (p. 24)
- Tutorial: Listing All Amazon SQS Queues in a Region (p. 27)
- Tutorial: Adding Permissions to an Amazon SQS Queue (p. 28)
- Tutorial: Sending a Message to an Amazon SQS Queue (p. 29)
- Tutorial: Receiving and Deleting a Message from an Amazon SQS Queue (p. 32)
- Tutorial: Configuring an Amazon SQS Dead-Letter Queue (p. 37)
- Tutorial: Purging Messages from an Amazon SQS Queue (p. 40)
- Tutorial: Deleting an Amazon SQS Queue (p. 41)
- Tutorial: Subscribing an Amazon SQS Queue to an Amazon SNS Topic (p. 42)
- Tutorial: Adding, Updating, and Removing Cost Allocation Tags for an Amazon SQS Queue (p. 44)

Tutorial: Creating an Amazon SQS Queue

The first and most common Amazon SQS task is creating queues. The following example demonstrates how to create and configure a queue.
AWS Management Console

1. Sign in to the Amazon SQS console.
2. Choose Create New Queue.
3. On the Create New Queue page, ensure that you’re in the correct region and then type the Queue Name.

   Note
   The name of a FIFO queue must end with the \texttt{.fifo} suffix. FIFO queues are available in the US East (N. Virginia), US East (Ohio), US West (Oregon), and EU (Ireland) regions.

4. Standard is selected by default. Choose FIFO.

5. Create your queue.
   - To create your queue with the default parameters, choose Quick-Create Queue.
   - To configure your queue's parameters, choose Configure Queue. When you finish configuring the parameters, choose Create Queue. For more information about creating a queue with SSE, see Tutorial: Creating an Amazon SQS Queue with Server-Side Encryption (p. 20).

The following example shows the Content-Based Deduplication parameter specific to FIFO queues.
Your new queue is created and selected in the queue list.

**Note**  
When you create a queue, it can take a short time for the queue to propagate throughout Amazon SQS.

The **Queue Type** column helps you distinguish standard queues from FIFO queues at a glance. For a FIFO queue, the **Content-Based Deduplication** column displays whether you have enabled exactly-once processing (p. 53).

Your queue’s **Name**, **URL**, and **ARN** are displayed on the **Details** tab.

**Java**

Before you begin working with the example code, specify your AWS credentials. For more information, see Set up AWS Credentials and Region for Development in the AWS SDK for Java Developer Guide.

**To create a standard queue**

1. Copy the example program (p. 48).

   The following section of the code creates the `MyQueue` queue:

   ```java
   // Create a queue
   System.out.println("Creating a new SQS queue called MyQueue.\n");
   CreateQueueRequest createQueueRequest = new
   CreateQueueRequest().withQueueName("MyQueue");
   ```
String myQueueUrl = sqs.createQueue(createQueueRequest).getQueueUrl();

2. Compile and run the example.

The queue is created.

To create a FIFO queue

1. Copy the example program (p. 53).

The following section of the code creates the MyFifoQueue.fifo queue:

```java
// Create a FIFO queue
System.out.println("Creating a new Amazon SQS FIFO queue called MyFifoQueue.fifo.\n");
Map<String, String> attributes = new HashMap<String, String>();
// A FIFO queue must have the FifoQueue attribute set to True
attributes.put("FifoQueue", "true");
// Generate a MessageDeduplicationId based on the content, if the user doesn't provide
// a MessageDeduplicationId
attributes.put("ContentBasedDeduplication", "true");
// The FIFO queue name must end with the .fifo suffix
CreateQueueRequest createQueueRequest = new
CreateQueueRequest("MyFifoQueue.fifo").withAttributes(attributes);
String myQueueUrl = sqs.createQueue(createQueueRequest).getQueueUrl();
```

2. Compile and run the example.

The queue is created.

AWS CloudFormation

You can use the AWS CloudFormation console and a JSON (or YAML) template to create an Amazon SQS queue. For more information, see Working with AWS CloudFormation Templates and the AWS::SQS::Queue Resource in the AWS CloudFormation User Guide.

1. Copy the following JSON code to a file named MyQueue.json. To create a standard queue, omit the FifoQueue and ContentBasedDeduplication properties. For more information on content-based deduplication, see Exactly-Once Processing (p. 53).

   **Note**
   
   The name of a FIFO queue must end with the .fifo suffix. FIFO queues are available in the US East (N. Virginia), US East (Ohio), US West (Oregon), and EU (Ireland) regions.

```json
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Resources": {
        "MyQueue": {
            "Properties": {
                "QueueName": "MyQueue.fifo",
                "FifoQueue": true,
                "ContentBasedDeduplication": true
            },
            "Type": "AWS::SQS::Queue"
        }
    },
    "Outputs": {
        "QueueName": {
            "Description": "The name of the queue",
            "Value": {
```
2. Sign in to the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation, and then choose Create Stack.

3. On the Select Template page, choose Upload a template to Amazon S3, choose your MyQueue.json file, and then choose Next.

4. On the Specify Details page, type MyQueue for Stack Name, and then choose Next.

5. On the Options page, choose Next.


AWS CloudFormation begins to create the MyQueue stack and displays the CREATE_IN_PROGRESS status. When the process is complete, AWS CloudFormation displays the CREATE_COMPLETE status.
7. (Optional) To display the name, URL, and ARN of the queue, choose the name of the stack and then on the next page expand the Outputs section.

![MyQueue section](image)

**Outputs**

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Description</th>
<th>Export Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueueURL</td>
<td></td>
<td>URL of the queue</td>
<td></td>
</tr>
<tr>
<td>QueueARN</td>
<td></td>
<td>ARN of the queue</td>
<td></td>
</tr>
</tbody>
</table>

**Tutorial: Creating an Amazon SQS Queue with Server-Side Encryption**

Server-side encryption (SSE) for Amazon SQS is available in the US East (N. Virginia), US East (Ohio), and US West (Oregon) regions. You can enable server-side encryption (SSE) for a queue to protect its data. For more information about using SSE, see Protecting Data Using Server-Side Encryption (SSE) and AWS KMS (p. 156).

**Important**
All requests to queues with SSE enabled must use HTTPS and Signature Version 4.

The following example demonstrates how to create an Amazon SQS queue with SSE enabled. Although the example uses a FIFO queue, SSE works with both standard and FIFO queues.

**AWS Management Console**

1. Sign in to the Amazon SQS console.
2. Choose Create New Queue.
3. On the Create New Queue page, ensure that you're in the correct region and then type the Queue Name.

**Note**
The name of a FIFO queue must end with the `.fifo` suffix. FIFO queues are available in the US East (N. Virginia), US East (Ohio), US West (Oregon), and EU (Ireland) regions.
4. **Standard** is selected by default. Choose **FIFO**.

What type of queue do you need?

- Standard
- FIFO

High Throughput: Standard queues have nearly unlimited transactions per second (TPS).

At-Least-Once Delivery: A message is delivered at least once, but occasionally more than one copy of a message is delivered.

Best-Effort Ordering: Occasionally, messages are delivered in an order different from which they were sent.

First-In-First-Out Delivery: The order in which messages are sent and received is strictly preserved.

Exactly-Once Processing: A message is guaranteed to be delivered at least once, but all duplicates of the message are removed.

Limited Throughput: 300 transactions per second (TPS).

Send data between applications when the throughput is important, for example:

- Decouple live user requests from intensive background work: let users upload media while reserving or encoding it.
- Allocate tasks to multiple worker nodes: process a high number of credit card validation requests.
- Batch messages for future processing: schedule multiple entries to be added to a database.

Send data between applications when the order of events is important, for example:

- Ensure that user-entered commands are executed in the right order.
- Display the correct product price by sending price modifications in the right order.
- Prevent a student from enrolling in a course before registering for an account.

5. Choose **Configure Queue**, and then choose **Use SSE**.

6. Specify the customer master key (CMK) ID. For more information, see Key Terms (p. 157).

For each CMK type, the **Description**, **Account**, and **Key ARN** of the CMK are displayed.

**Important**

If you aren't the owner of the CMK, or if you log in with an account that doesn't have the `kms:ListAliases` and `kms:DescribeKey` permissions, you won't be able to view information about the CMK on the Amazon SQS console.

Ask the owner of the CMK to grant you these permissions. For more information, see the **AWS KMS API Permissions: Actions and Resources Reference** in the **AWS Key Management Service Developer Guide**.

- The AWS-managed CMK for Amazon SQS is selected by default.

**Note**

Keep the following in mind:
• If you don’t specify a custom CMK, Amazon SQS uses the AWS-managed CMK for Amazon SQS. For instructions on creating custom CMKs, see Creating Keys in the AWS Key Management Service Developer Guide.

• The first time you use the AWS Management Console to specify the AWS-managed CMK for Amazon SQS for a queue, AWS KMS creates the AWS-managed CMK for Amazon SQS.
• Alternatively, the first time you use the SendMessage or SendMessageBatch API action on a queue with SSE enabled, AWS KMS creates the AWS-managed CMK for Amazon SQS.

• To use a custom CMK from your AWS account, select it from the list.

Note
For instructions on creating custom CMKs, see Creating Keys in the AWS Key Management Service Developer Guide.

• To use a custom CMK ARN from your AWS account or from another AWS account, select Enter an existing CMK ARN from the list and type or copy the CMK.

7. (Optional) For Data key reuse period, specify a value between 1 minute and 24 hours. The default is 5 minutes. For more information, see How Does the Data Key Reuse Period Work? (p. 158).

8. Choose Create Queue.

Your new queue is created with SSE. The encryption status, alias of the CMK, Description, Account, Key ARN, and the Data Key Reuse Period are displayed on the Encryption tab.

Server-side encryption (SSE) is enabled. SSE lets you protect the contents of messages in Amazon SQS queues using keys managed in the AWS Key Management Service (AWS KMS). Learn more.

To modify the SSE parameters, choose Queue Actions, Configure Queue.

Java

Before you begin working with the example code, specify your AWS credentials. For more information, see Set up AWS Credentials and Region for Development in the AWS SDK for Java Developer Guide.

Before you can use SSE, you must configure AWS KMS key policies to allow encryption of queues and encryption and decryption of messages. You must also ensure that the key policies of the customer master key (CMK) allow the necessary permissions. For more information, see What Permissions Do I Need to Use SSE? (p. 160).

1. Obtain the customer master key (CMK) ID. For more information, see Key Terms (p. 157).
Note
Keep the following in mind:

- If you don’t specify a custom CMK, Amazon SQS uses the AWS-managed CMK for Amazon SQS. For instructions on creating custom CMKs, see Creating Keys in the AWS Key Management Service Developer Guide.
- The first time you use the AWS Management Console to specify the AWS-managed CMK for Amazon SQS for a queue, AWS KMS creates the AWS-managed CMK for Amazon SQS.
- Alternatively, the first time you use the SendMessage or SendMessageBatch API action on a queue with SSE enabled, AWS KMS creates the AWS-managed CMK for Amazon SQS.

2. To enable server-side encryption, specify the CMK ID by setting the KmsMasterKeyId attribute of the CreateQueue or SetQueueAttributes action.

The following code example creates a new queue with SSE using the AWS-managed CMK for Amazon SQS:

```java
AmazonSQSClient client = new AmazonSQSClient(credentialsProvider);
CreateQueueRequest createRequest = new CreateQueueRequest("MyQueue");
Map<String, String> attributes = new HashMap<String, String>();
// Enable server-side encryption by specifying the alias ARN of the
// AWS-managed CMK for Amazon SQS.
String kmsMasterKeyAlias = "arn:aws:kms:us-east-2:123456789012:alias/aws/sqs";
attributes.put("KmsMasterKeyId", kmsMasterKeyAlias);
// (Optional) Specify the length of time, in seconds, for which Amazon SQS can reuse
attributes.put("KmsDataKeyReusePeriodSeconds", "60");
CreateQueueResult createResult = client.createQueue(createRequest);
```

The following code example creates a new queue with SSE using a custom CMK:

```java
AmazonSQSClient client = new AmazonSQSClient(credentialsProvider);
CreateQueueRequest createRequest = new CreateQueueRequest("MyQueue");
Map<String, String> attributes = new HashMap<String, String>();
// Enable server-side encryption by specifying the alias ARN of the custom CMK.
String kmsMasterKeyAlias = "arn:aws:kms:us-east-2:123456789012:alias/MyAlias";
attributes.put("KmsMasterKeyId", kmsMasterKeyAlias);
// (Optional) Specify the length of time, in seconds, for which Amazon SQS can reuse
// a data key to encrypt or decrypt messages before calling AWS KMS again.
attributes.put("KmsDataKeyReusePeriodSeconds", "864000");
CreateQueueResult createResult = client.createQueue(createRequest);
```

3. (Optional) Specify the length of time, in seconds, for which Amazon SQS can reuse a data key (p. 157) to encrypt or decrypt messages before calling AWS KMS again. Set the KmsDataKeyReusePeriodSeconds attribute of the CreateQueue or SetQueueAttributes action. Possible values may be between 60 seconds (1 minute) and 86,400 seconds (24 hours). If you don’t specify a value, the default value of 300 seconds (5 minutes) is used.

The first code example above sets the data key reuse time period to 60 seconds (1 minute). The second code example sets it to 86,400 seconds (24 hours). The following code example sets the data key reuse period to 60 seconds (1 minute):

```java
// (Optional) Specify the length of time, in seconds, for which Amazon SQS can reuse
// a data key to encrypt or decrypt messages before calling AWS KMS again.
```
attributes.put("KmsDataKeyReusePeriodSeconds", "60");

For information about how to retrieve the attributes of a queue, see Examples in the Amazon Simple Queue Service API Reference.

To retrieve the CMK ID or the data key reuse period for a particular queue, use the KmsMasterKeyId and KmsDataKeyReusePeriodSeconds attributes of the GetQueueAttributes action.

For information about how to switch a queue to a different CMK with the same alias, see Updating an Alias in the AWS Key Management Service Developer Guide.

Tutorial: Configuring Server-Side Encryption (SSE) for an Existing Amazon SQS Queue

Server-side encryption (SSE) for Amazon SQS is available in the US East (N. Virginia), US East (Ohio), and US West (Oregon) regions. You can enable SSE for a queue to protect its data. For more information about using SSE, see Protecting Data Using Server-Side Encryption (SSE) and AWS KMS (p. 156).

Important
All requests to queues with SSE enabled must use HTTPS and Signature Version 4. When you disable SSE, messages remain encrypted. You must receive and decrypt a message to view its contents.

The following example demonstrates enabling, disabling, and configuring SSE for an existing Amazon SQS queue.

AWS Management Console

1. Sign in to the Amazon SQS console.
2. From the queue list, select a queue.

3. From Queue Actions, select Configure Queue.

   The Configure QueueName dialog box is displayed.

4. To enable or disable SSE, use the Use SSE check box.
5. Specify the customer master key (CMK) ID. For more information, see Key Terms (p. 157).

   For each CMK type, the Description, Account, and Key ARN of the CMK are displayed.
Important
If you aren't the owner of the CMK, or if you log in with an account that doesn't have the `kms:ListAliases` and `kms:DescribeKey` permissions, you won't be able to view information about the CMK on the Amazon SQS console. Ask the owner of the CMK to grant you these permissions. For more information, see the AWS KMS API Permissions: Actions and Resources Reference in the AWS Key Management Service Developer Guide.

- To use the AWS-managed CMK for Amazon SQS, select it from the list.

![AWS KMS Customer Master Key (CMK)](default-awskms)

**Description**
Default master key that protects my SQS messages when no other key is defined

**Account**

**Key ARN**
am:aws:kms:us-east-1

**Note**
Keep the following in mind:

- If you don't specify a custom CMK, Amazon SQS uses the AWS-managed CMK for Amazon SQS. For instructions on creating custom CMKs, see Creating Keys in the AWS Key Management Service Developer Guide.

- The first time you use the AWS Management Console to specify the AWS-managed CMK for Amazon SQS for a queue, AWS KMS creates the AWS-managed CMK for Amazon SQS.

- Alternatively, the first time you use the `SendMessage` or `SendMessageBatch` API action on a queue with SSE enabled, AWS KMS creates the AWS-managed CMK for Amazon SQS.

- To use a custom CMK from your AWS account, select it from the list.

![AWS KMS Customer Master Key (CMK)](demo-key)

**Description**
A key for demonstrating the functionality of SSE in Amazon SQS

**Account**

**Key ARN**
am:aws:kms:us-east-1

**Note**
For instructions on creating custom CMKs, see Creating Keys in the AWS Key Management Service Developer Guide.

- To use a custom CMK ARN from your AWS account or from another AWS account, select Enter an existing CMK ARN from the list and type or copy the CMK.

![AWS KMS Customer Master Key (CMK)](arn:aws:kms:us-east-1)

6. (Optional) For Data key reuse period, specify a value between 1 minute and 24 hours. The default is 5 minutes. For more information, see How Does the Data Key Reuse Period Work? (p. 158).

7. Choose Save Changes.

Your changes are applied to the queue.

Java

Before you begin working with the example code, specify your AWS credentials. For more information, see Set up AWS Credentials and Region for Development in the AWS SDK for Java Developer Guide.
Before you can use SSE, you must configure AWS KMS key policies to allow encryption of queues and encryption and decryption of messages. You must also ensure that the key policies of the customer master key (CMK) allow the necessary permissions. For more information, see What Permissions Do I Need to Use SSE? (p. 160).

1. Obtain the customer master key (CMK) ID. For more information, see Key Terms (p. 157).

   **Note**
   Keep the following in mind:
   
   - If you don't specify a custom CMK, Amazon SQS uses the AWS-managed CMK for Amazon SQS. For instructions on creating custom CMKs, see Creating Keys in the AWS Key Management Service Developer Guide.
   - The first time you use the AWS Management Console to specify the AWS-managed CMK for Amazon SQS for a queue, AWS KMS creates the AWS-managed CMK for Amazon SQS.
   - Alternatively, the first time you use the SendMessage or SendMessageBatch API action on a queue with SSE enabled, AWS KMS creates the AWS-managed CMK for Amazon SQS.

2. To enable server-side encryption, specify the CMK ID by setting the KmsMasterKeyId attribute of the CreateQueue or SetQueueAttributes action.

   The following code example enables SSE for an existing queue using the AWS-managed CMK for Amazon SQS:

   ```java
   SetQueueAttributesRequest setAttributesRequest = new SetQueueAttributesRequest();
   setAttributesRequest.setQueueUrl(queueUrl);
   // Enable server-side encryption by specifying the alias ARN of the
   // AWS-managed CMK for Amazon SQS.
   String kmsMasterKeyAlias = "arn:aws:kms:us-east-2:123456789012:alias/aws/sqs";
   attributes.put("KmsMasterKeyId", kmsMasterKeyAlias);
   SetQueueAttributesResult setAttributesResult =
       client.setQueueAttributes(setAttributesRequest);
   ```

   To disable server-side encryption for an existing queue, set the KmsMasterKeyId attribute to an empty string using the SetQueueAttributes action.

   **Important**
   null is not a valid value for KmsMasterKeyId.

3. (Optional) Specify the length of time, in seconds, for which Amazon SQS can reuse a data key (p. 157) to encrypt or decrypt messages before calling AWS KMS. Set the KmsDataKeyReusePeriodSeconds attribute of the CreateQueue or SetQueueAttributes action. Possible values may be between 60 seconds (1 minute) and 86,400 seconds (24 hours). If you don't specify a value, the default value of 300 seconds (5 minutes) is used.

   The following code example sets the data key reuse period to 60 seconds (1 minute):

   ```java
   // (Optional) Specify the length of time, in seconds, for which Amazon SQS can reuse
   // a data key to encrypt or decrypt messages before calling AWS KMS again.
   attributes.put("KmsDataKeyReusePeriodSeconds", "60");
   ```

   For information about how to retrieve the attributes of a queue, see Examples in the Amazon Simple Queue Service API Reference.

   To retrieve the CMK ID or the data key reuse period for a particular queue, use the KmsMasterKeyId and KmsDataKeyReusePeriodSeconds attributes of the GetQueueAttributes action.
For information about how to switch a queue to a different CMK with the same alias, see Updating an Alias in the AWS Key Management Service Developer Guide.

Tutorial: Listing All Amazon SQS Queues in a Region

When you create a queue, it can take a short time for the queue to propagate throughout Amazon SQS. The following example demonstrates confirming your queue's existence by listing all queues in the current region.

AWS Management Console

1. Sign in to the Amazon SQS console.
2. Your queues in the current region are listed.

The **Queue Type** column helps you distinguish standard queues from FIFO queues at a glance. For a FIFO queue, the **Content-Based Deduplication** column displays whether you have enabled exactly-once processing (p. 53).

Your queue's **Name**, **URL**, and **ARN** are displayed on the **Details** tab.

Java

Before you begin working with the example code, specify your AWS credentials. For more information, see Set up AWS Credentials and Region for Development in the AWS SDK for Java Developer Guide.

**Note**
This action is identical for standard and FIFO queues.

1. Copy the standard queue example program (p. 48) or the FIFO queue example program (p. 53).
   
   The following section of the code list all queues in the current region:

   ```java
   // List queues
   System.out.println("Listing all queues in your account.\n");
   for (String queueUrl : sqs.listQueues().getQueueUrls()) {
       System.out.println(" QueueUrl: " + queueUrl);
   }
   System.out.println();
   ```

2. Compile and run the example.

   All queues in the current region created using API version 2012-11-05 are listed. The response include the following items:

   - The unique **queue URL**.
- The request ID that Amazon SQS assigned to your request.

Tutorial: Adding Permissions to an Amazon SQS Queue

You can specify to whom you allow (or explicitly deny) the ability to interact with your queue in specific ways by adding permissions to a queue. The following example demonstrates adding the permission for anyone to get a queue's URL.

**AWS Management Console**

1. Sign in to the Amazon SQS console.
2. From the queue list, select a queue.
3. From Queue Actions, select Add a Permission.
   
   ![Add a Permission dialog box](image)

   The Add a Permission dialog box is displayed.
4. In this example, you allow anyone to get the queue's URL:
1. Ensure that next to Effect, Allow is selected.
2. Next to Principal, check the Everybody box.
3. From the Actions drop-down list, select GetQueueUrl box.
4. Choose Add Permission.

The permission is added to the queue.

Your queue's policy Effect, Principals, Actions, and Conditions are displayed on your queue's Permissions tab.

---

**Tutorial: Sending a Message to an Amazon SQS Queue**

After you create your queue, you can send a message to it. The following example demonstrates sending a message to an existing queue.

**AWS Management Console**

1. Sign in to the Amazon SQS console.
2. From the queue list, select a queue.
3. From Queue Actions, select Send a Message.

The Send a Message to QueueName dialog box is displayed.

The following example shows the Message Group ID and Message Deduplication ID parameters specific to FIFO queues (content-based deduplication (p. 53) is disabled).

4. To send a message to a FIFO queue, type the Message Body, the Message Group ID MyMessageGroupId1234567890, and the Message Deduplication ID MyMessageDeduplicationId1234567890, and then choose Send Message. For more information, see FIFO Queue Logic (p. 52).

Note
The message group ID is always required. However, if content-based deduplication is enabled, the message deduplication ID is optional.
Your message is sent and the **Send a Message to QueueName** dialog box is displayed, showing the attributes of the sent message.

The following example shows the **Sequence Number** attribute specific to FIFO queues.

5. Choose **Close**.

**Java**

Before you begin working with the example code, specify your AWS credentials. For more information, see **Set up AWS Credentials and Region for Development** in the **AWS SDK for Java Developer Guide**.

**To send a message to a standard queue**

1. Copy the example program (p. 48).

The following section of the code sends the *This is my message text.* message to your queue:

```java
// Send a message
System.out.println("Sending a message to MyQueue.
");
sqs.sendMessage(new SendMessageRequest()
    .withQueueUrl(myQueueUrl)
    .withMessageBody("This is my message text.
");
```


2. Compile and run the example.

The message is sent to the queue. The response includes the following items:

- The **message ID** Amazon SQS assigns to the message.
- An MD5 digest of the message body, used to confirm that Amazon SQS received the message correctly (for more information, see [RFC1321](https://tools.ietf.org/html/rfc1321)).
- The **request ID** that Amazon SQS assigned to your request.

### To send a message to a FIFO queue

1. Copy the example program (p. 53).

The following section of the code sends the *This is my message text.* message to your queue:

```java
// Send a message
System.out.println("Sending a message to MyFifoQueue.fifo.\n");
SendMessageRequest sendMessageRequest = new SendMessageRequest(myQueueUrl, "This is my message text.");
// You must provide a non-empty MessageGroupId when sending messages to a FIFO queue
sendMessageRequest.setMessageGroupId("messageGroup1");
// Uncomment the following to provide the MessageDeduplicationId
//sendMessageRequest.setMessageDeduplicationId("1");
SendMessageResult sendMessageResult = sqs.sendMessage(sendMessageRequest);
String sequenceNumber = sendMessageResult.getSequenceNumber();
String messageId = sendMessageResult.getMessageId();
System.out.println("SendMessage succeed with messageId " + messageId + ", sequence number " + sequenceNumber + "\n");
```

2. Compile and run the example.

The message is sent to your queue.

### Tutorial: Receiving and Deleting a Message from an Amazon SQS Queue

After you send a message into a queue, you can consume it from the queue. When you request a message from a queue, you can't specify which message to get. Instead, you specify the maximum number of messages (up to 10) that you want to get.

**Note**

Because Amazon SQS is a distributed system, a queue with very few messages might display an empty response to a receive request. In this case, you can rerun the request to get your message. Depending on your application's needs, you might have to use short or long polling (p. 73) to receive messages.

Amazon SQS doesn't automatically delete a message after receiving it for you, in case you don't successfully receive the message (for example, the consumers can fail or lose connectivity). To delete a message, you must send a separate request which acknowledges that you no longer need the message because you've successfully received and processed it.

The following example demonstrates receiving and deleting a message.
AWS Management Console

1. Sign in to the Amazon SQS console.
2. From the queue list, select a queue.

![Queue Selection](image)

3. From Queue Actions, select View/Delete Messages.

![Queue Actions](image)

The View/Delete Messages in QueueName dialog box is displayed.

Note
The first time you take this action, an information screen is displayed. To hide the screen, check the Don't show this again checkbox.

4. Choose Start Polling for messages.

![Polling Options](image)

Amazon SQS begins to poll the messages in the queue. The dialog box displays a message from the queue. A progress bar at the bottom of the dialog box displays the status of the message's visibility timeout.

The following example shows the Message Group ID, Message Deduplication ID, and Sequence Number columns specific to FIFO queues.
5. **Before** the visibility timeout expires, select the message that you want to delete and then choose **Delete 1 Message**.

The **Delete Messages** dialog box is displayed.
6. Confirm that the message you want to delete is checked and choose Yes, Delete Checked Messages.

The selected message is deleted.

When the progress bar is filled in, the visibility timeout (p. 59) expires and the message becomes visible to consumers.

7. Select Close.

Java

To specify the message to delete, provide the receipt handle that Amazon SQS returned when you received the message. You can delete only one message per action. To delete an entire queue, you must use the DeleteQueue action. (You can delete an entire queue even if the queue has messages in it.)

**Note**

If you don't have the receipt handle for the message, you can call the ReceiveMessage action to receive the message again. Each time you receive the message, you get a different receipt handle. Use the latest receipt handle when using the DeleteMessage action. Otherwise, your message might not be deleted from the queue.

Before you begin working with the example code, specify your AWS credentials. For more information, see Set up AWS Credentials and Region for Development in the AWS SDK for Java Developer Guide.

**To receive and delete a message from a standard queue**

1. Copy the example program (p. 48).

The following section of the code receives a message from your queue:

```java
System.out.println("Receiving messages from MyQueue.\n");
ReceiveMessageRequest receiveMessageRequest = new ReceiveMessageRequest(myQueueUrl);
List<Message> messages = sqs.receiveMessage(receiveMessageRequest).getMessages();
for (Message message : messages) {
    System.out.println("    MessageId:     " + message.getMessageId());
    System.out.println("    ReceiptHandle: " + message.getReceiptHandle());
    System.out.println("    MD5OfBody:     " + message.getMD5OfBody());
    System.out.println("    Body:          " + message.getBody());
    for (Entry<String, String> entry : message.getAttributes().entrySet()) {
        System.out.println("      Name:  " + entry.getKey());
        System.out.println("      Value: " + entry.getValue());
    }
}
System.out.println();
```

The following section of the code deletes the message:

```java
// Delete a message
```
System.out.println("Deleting a message.
");
String messageReceiptHandle = messages.get(0).getReceiptHandle();
sqs.deleteMessage(new DeleteMessageRequest()
    .withQueueUrl(myQueueUrl)
    .withReceiptHandle(messageReceiptHandle));

2. Compile and run the example.

The queue is polled and returns 0 or more messages. The example prints the following items:

- The message ID that you received when you sent the message to the queue.
- The receipt handle that you later use to delete the message.
- An MD5 digest of the message body (for more information, see RFC1321).
- The message body.
- The request ID that Amazon SQS assigned to your request

If no messages are received in this particular call, the response includes only the request ID.

The message is deleted. The response includes the request ID that Amazon SQS assigned to your request.

To receive and delete a message from a FIFO queue

1. Copy the example program (p. 53).

The following section of the code receives a message from your queue:

```java
// Receive messages
System.out.println("Receiving messages from MyFifoQueue.fifo.
");
ReceiveMessageRequest receiveMessageRequest = new ReceiveMessageRequest(myQueueUrl);
// Uncomment the following to provide the ReceiveRequestDeduplicationId
//receiveMessageRequest.setReceiveRequestAttemptId("1");
List<Message> messages = sqs.receiveMessage(receiveMessageRequest).getMessages();
for (Message message : messages) {
    System.out.println("  Message");
    System.out.println("    MessageId:     " + message.getMessageId());
    System.out.println("    ReceiptHandle: " + message.getReceiptHandle());
    System.out.println("    MD5OfBody:     " + message.getMD5OfBody());
    System.out.println("    Body:          " + message.getBody());
    for (Entry<String, String> entry : message.getAttributes().entrySet()) {
        System.out.println("    Attribute");
        System.out.println("      Name: " + entry.getKey());
        System.out.println("      Value: " + entry.getValue());
    }
}
System.out.println();
```

The following section of the code deletes the message:

```java
// Delete the message
System.out.println("Deleting the message.
");
String messageReceiptHandle = messages.get(0).getReceiptHandle();
sqs.deleteMessage(new DeleteMessageRequest(myQueueUrl, messageReceiptHandle));
```

2. Compile and run the example.

The message is received and deleted.
Tutorial: Configuring an Amazon SQS Dead-Letter Queue

A dead-letter queue is a queue that other (source) queues can target for messages that can't be processed (consumed) successfully. The following example demonstrates how to create a queue and to configure a dead-letter queue for it. For more information, see Using Amazon SQS Dead-Letter Queues (p. 61).

**Important**
The dead-letter queue of a FIFO queue must also be a FIFO queue. Similarly, the dead-letter queue of a standard queue must also be a standard queue.

**AWS Management Console**

1. Sign in to the Amazon SQS console.
2. Choose **Create New Queue**.
3. On the **Create New Queue** page, ensure that you're in the correct region and then type the **Queue Name**.

   **Note**
The name of a FIFO queue must end with the `.fifo` suffix. FIFO queues are available in the US East (N. Virginia), US East (Ohio), US West (Oregon), and EU (Ireland) regions.

4. **Standard** is selected by default. Choose **FIFO**.
5. Choose Configure Queue.
6. In this example, you enable the redrive policy for your new queue, set the MyDeadLetterQueue.fifo queue as the dead-letter queue, and set the number of maximum receives to 50.

1. To configure the dead-letter queue, choose Use Redrive Policy.
2. Enter the name of the existing Dead Letter Queue to which you want sources queues to send messages.
3. To configure the number of times that a message can be received before being sent to a dead-letter queue, set Maximum Receives to a value between 1 and 1,000.

   **Note**
   The Maximum Receives setting applies only to individual messages.

4. Choose Create Queue.
Your new dead-letter queue is created and selected in the queue list.

**Note**
When you create a queue, it can take a short time for the queue to propagate throughout Amazon SQS.

Your queue's **Maximum Receives** and **Dead Letter Queue** ARN are displayed on the **Redrive Policy** tab.

Java

Before you begin working with the example code, specify your AWS credentials. For more information, see **Set up AWS Credentials and Region for Development** in the **AWS SDK for Java Developer Guide**.

**To configure a dead-letter queue**

1. Copy the example program for a standard queue (p. 48) or a FIFO queue (p. 53).
2. Set a string that contains JSON-formatted parameters and values for the **RedrivePolicy** queue attribute:

   ```java
   ```

3. Use the **CreateQueue** or **SetQueueAttributesRequest** API action to set the **RedrivePolicy** queue attribute:

   ```java
   SetQueueAttributesRequest queueAttributes = new SetQueueAttributesRequest();
   Map<String,String> attributes = new HashMap<String,String>();
   attributes.put("RedrivePolicy", redrivePolicy);
   queueAttributes.setAttributes(attributes);
   queueAttributes.setQueueUrl(myQueueUrl);
   sqs.setQueueAttributes(queueAttributes);
   ```

4. Compile and run your program.
   The dead-letter queue is configured.

**Sample Request**

```
http://sqs.us-east-2.amazonaws.com/123456789012/MySourceQueue
?Action=SetQueueAttributes
&Attribute.1.Value=%7B%22maxReceiveCount%22%3A%225%22%2C+%22deadLetterTargetArn%22%3A%22arn
%3Aaws%3Asqs%3Aus-east-2%2C123456789012%3AMyDeadLetterQueue%22%7D
&Version=2012-11-05
&Attribute.1.Name=RedrivePolicy
```

**Note**
Queue names and queue URLs are case-sensitive.

**Sample Response**

```
<SetQueueAttributesResponse xmlns="http://queue.amazonaws.com/doc/2012-11-05/"
```
Tutorial: Purging Messages from an Amazon SQS Queue

If you don’t want to delete an Amazon SQS queue but need to delete all the messages from it, you can purge the queue. The following example demonstrates purging a queue.

**AWS Management Console**

1. Sign in to the Amazon SQS console.
2. From the queue list, select a queue.
3. From **Queue Actions**, select **Purge Queue**.
4. Choose **Yes, Purge Queue**.

All messages are purged from the queue.

The **Purge Queues** confirmation dialog box is displayed.
5. Choose **OK**.

### Tutorial: Deleting an Amazon SQS Queue

If you don’t use an Amazon SQS queue (and don’t foresee using it in the near future), it is a best practice to delete it from Amazon SQS. The following example demonstrates deleting a queue.

#### AWS Management Console

1. Sign in to the **Amazon SQS console**.
2. From the queue list, select a queue.
3. From **Queue Actions**, select **Delete Queue**.

   ![Queue Actions Screenshot](image)

   The **Delete Queues** dialog box is displayed.

4. Choose **Yes, Delete Queue**.

   The queue is deleted.
Java

Before you begin working with the example code, specify your AWS credentials. For more information, see Set up AWS Credentials and Region for Development in the AWS SDK for Java Developer Guide.

**Note**
This action is identical for standard and FIFO queues.

1. Copy the standard queue example program (p. 48) or the FIFO queue example program (p. 53).

   The following section of the code deletes the queue:

   ```java
   // Delete a queue
   System.out.println("Deleting the test queue.\n");
   sqs.deleteQueue(new DeleteQueueRequest(myQueueUrl));
   ```

2. Compile and run the example.

   The queue is deleted.

**Tutorial: Subscribing an Amazon SQS Queue to an Amazon SNS Topic**

You can subscribe one or more Amazon SQS queues to an Amazon SNS topic from a list of topics available for the selected queue. Amazon SQS manages the subscription and any necessary permissions. When you publish a message to a topic, Amazon SNS sends the message to every subscribed queue. For more information about Amazon SNS, see What is Amazon Simple Notification Service? in the Amazon Simple Notification Service Developer Guide.

The following example demonstrates subscribing an existing Amazon SQS queue to an existing Amazon SNS topic.

**Note**
Amazon SNS isn't currently compatible with FIFO queues.
When you subscribe an Amazon SQS queue to an Amazon SNS topic, Amazon SNS uses HTTPS to forward messages to Amazon SQS.

**AWS Management Console**

1. Sign in to the Amazon SQS console.
2. From the list of queues, choose the queue (or queues) to which you want to subscribe an Amazon SNS topic.

   ![Queue Selection](image)

3. From Queue Actions, select Subscribe Queue to SNS Topic (or Subscribe Queues to SNS Topic).
The Subscribe to a Topic dialog box is displayed.

4. From the Choose a Topic drop-down list, select an Amazon SNS topic to which you want to subscribe your queue (or queues), select the Topic Region (optional), and then choose Subscribe.

Note
Typing a different Topic ARN is useful when you want to subscribe a queue to an Amazon SNS topic from an AWS account other than the one you used to create your Amazon SQS queue. This is also useful if the Amazon SNS topic isn’t listed in the Choose a Topic drop-down list.

The Topic Subscription Result dialog box is displayed.

5. Review the list of Amazon SQS queues that are subscribed to the Amazon SNS topic and choose OK.

The queue is subscribed to the topic.

Note
To list your subscriptions, unsubscribe from topics, and delete topics, use the Amazon SNS console. For more information, see Clean Up in the Amazon Simple Notification Service Developer Guide.
To verify the results of the subscription, you can publish to the topic and then view the message that the topic sends to the queue. For more information, see Sending Amazon SNS Messages to Amazon SQS Queues in the Amazon Simple Notification Service Developer Guide.

Tutorial: Adding, Updating, and Removing Cost Allocation Tags for an Amazon SQS Queue

You can add cost allocation tags to your Amazon SQS queues to help organize and identify them. For a detailed overview of using Amazon SQS queue tags, see Tagging Your Amazon SQS Queues (p. 65).

AWS Management Console

The following steps assume that you already created an Amazon SQS queue (p. 15).

1. Sign in to the Amazon SQS console.
2. From the queue list, select a queue.
3. Choose the Tags tab.

The tags added to the queue are listed.

4. Choose Add/Edit Tags.
5. Modify queue tags:
   - To add a tag, choose Add New Tag, enter a Key and Value, and then choose Apply Changes.
   - To update a tag, change its Key and Value and then choose Apply Changes.
   - To remove a tag, choose ✗ next to a key-value pair and then choose Apply Changes.

The queue tag changes are applied.

Java

Before you begin working with the example code, specify your AWS credentials. For more information, see Set up AWS Credentials and Region for Development in the AWS SDK for Java Developer Guide.

To add, update, and remove tags from a queue

1. Copy the example program for a standard queue (p. 48) or a FIFO queue (p. 53).
2. To list the tags added to a queue, use the ListQueueTags API action:
ListQueueTagsRequest listQueueTagsRequest = new ListQueueTagsRequest(queueUrl);
ListQueueTagsResult listQueueTagsResult =
SQSClientFactory.newSQSClient().listQueueTags(listQueueTagsRequest);
System.out.println(String.format("ListQueueTags: \tTags for queue %s are %s.\n",
QUEUE_NAME, listQueueTagsResult.getTags()));

3. To add or update the values of the queue's tags using the tag's key, use the TagQueue API action:

Map<String, String> addedTags = new HashMap<>(){
    addedTags.put("Team", "Development");
    addedTags.put("Priority", "Beta");
    addedTags.put("Accounting ID", "456def");
    TagQueueRequest tagQueueRequest = new TagQueueRequest(queueUrl, addedTags);
    System.out.println(String.format("TagQueue: \t\tAdd tags %s to queue %s.\n", addedTags,
QUEUE_NAME));
    SQSClientFactory.newSQSClient().tagQueue(tagQueueRequest);

4. To remove a tag from the queue using the tag's key, use the UntagQueue API action:

List<String> tagKeys = Arrays.asList("Accounting ID");
UntagQueueRequest untagQueueRequest = new UntagQueueRequest(queueUrl, tagKeys);
System.out.println(String.format("UntagQueue: \t\tRemove tags %s from queue %s.\n", tagKeys,
QUEUE_NAME));
SQSClientFactory.newSQSClient().untagQueue(untagQueueRequest);

5. Compile and run your program.

The existing tags are listed, three are updated, and one tag is removed from the queue.
How Amazon SQS Queues Work

This section describes the types of Amazon SQS queues and their basic properties. It also describes the identifiers of queues and messages, and various queue and message management workflows.

Topics

• Basic Prerequisites (p. 46)
• Standard Queues (p. 47)
• FIFO (First-In-First-Out) Queues (p. 51)
• Queue and Message Identifiers (p. 57)
• Resources Required to Process Messages (p. 58)
• Visibility Timeout (p. 59)
• Using Amazon SQS Dead-Letter Queues (p. 61)
• Message Lifecycle (p. 64)
• Tagging Your Amazon SQS Queues (p. 65)
• Using Amazon SQS Message Attributes (p. 66)
• Amazon SQS Long Polling (p. 73)
• Amazon SQS Delay Queues (p. 77)
• Amazon SQS Message Timers (p. 81)
• Managing Large Amazon SQS Messages Using Amazon S3 (p. 84)
• Using JMS with Amazon SQS (p. 88)

Basic Prerequisites

The following basic prerequisites help you get started with Amazon SQS queues:

• You must assign a name to each of your queues. You can get a list of all your queues or a subset of your queues that share the same initial characters in their names. For example, you can get a list of all your queues whose names start with T3.
• A queue can be empty if you haven’t sent any messages to it or if you have deleted all the messages from it.
• You can delete a queue at any time, whether it’s empty or not. By default, a queue retains messages for four days. However, you can configure a queue to retain messages for up to 14 days after the message is sent.

Note
Unless your application specifically requires repeatedly creating queues and leaving them inactive or storing large amounts of data in your queue, consider using Amazon S3 for storing your data.

The following table lists the API actions you can use to work with queues.

<table>
<thead>
<tr>
<th>To do this...</th>
<th>Use this action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a queue</td>
<td>CreateQueue</td>
</tr>
<tr>
<td>Get the URL of an existing queue</td>
<td>GetQueueUrl</td>
</tr>
<tr>
<td>List your queues</td>
<td>ListQueues</td>
</tr>
<tr>
<td>Delete a queue</td>
<td>DeleteQueue</td>
</tr>
</tbody>
</table>

### Standard Queues

Amazon SQS offers standard as the default queue type. Standard queues can support a nearly unlimited number of transactions per second (TPS) per API action. Standard queues support at-least-once message delivery. However, occasionally (because of the highly distributed architecture that allows nearly unlimited throughput), more than one copy of a message might be delivered out of order. Standard queues provide best-effort ordering which ensures that messages are generally delivered in the same order as they’re sent.

You can use standard message queues in many scenarios, as long as your application can process messages that arrive more than once and out of order, for example:

• **Decouple live user requests from intensive background work** – Let users upload media while resizing or encoding it.
• **Allocate tasks to multiple worker nodes** – Process a high number of credit card validation requests.
• **Batch messages for future processing** – Schedule multiple entries to be added to a database.

For best practices of working with standard queues, see General Recommendations (p. 109).

**Topics**

• Message Ordering (p. 47)
• At-Least-Once Delivery (p. 48)
• Consuming Messages Using Short Polling (p. 48)
• Getting Started with Standard Queues (p. 48)

**Message Ordering**

A standard queue makes a best effort to preserve the order of messages, but more than one copy of a message might be delivered out of order. If your system requires that order be preserved, we recommend
At-Least-Once Delivery

Amazon SQS stores copies of your messages on multiple servers for redundancy and high availability. On rare occasions, one of the servers that stores a copy of a message might be unavailable when you receive or delete a message.

If this occurs, the copy of the message isn't deleted on that unavailable server, and you might get that message copy again when you receive messages. Design your applications to be idempotent (they should not be affected adversely when processing the same message more than once).

Consuming Messages Using Short Polling

The behavior of consuming messages from the queue depends on whether you use short (standard) polling, the default behavior, or long polling. For more information about long polling, see Amazon SQS Long Polling (p. 73).

When you consume messages from the queue using short polling, Amazon SQS samples a subset of the servers (based on a weighted random distribution) and returns messages from just these servers. Thus, a particular receive request might not return all of your messages. However, if you have a small number of messages in your queue (fewer than 1,000), one particular request might not return any of your messages, whereas a subsequent request returns them. If you keep consuming from your queues, Amazon SQS samples all of the servers, and you receive all your messages.

The following figure shows the short-polling behavior of messages returned after one of your system components makes a receive request. Amazon SQS samples several of the servers (in gray) and returns the messages from those servers (Message A, C, D, and B). Message E isn't returned to this particular request, but is returned to a subsequent request.

Getting Started with Standard Queues

The following example Java code creates a queue and sends, receives, and deletes a message.

```java
/*
 * Copyright 2010-2017 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 */
```
import java.util.List;
import java.util.Map.Entry;
import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Region;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.CreateQueueRequest;
import com.amazonaws.services.sqs.model.DeleteMessageRequest;
import com.amazonaws.services.sqs.model.DeleteQueueRequest;
import com.amazonaws.services.sqs.model.Message;
import com.amazonaws.services.sqs.model.ReceiveMessageRequest;
import com.amazonaws.services.sqs.model.SendMessageRequest;

/**
 * This sample demonstrates how to make basic requests to Amazon SQS using the
 * AWS SDK for Java.
 *
 * Prerequisites: You must have a valid Amazon Web Services developer account,
 * and be signed up to use Amazon SQS. For more information about Amazon SQS,
 * see http://aws.amazon.com/sqs
 *
 * Fill in your AWS access credentials in the provided credentials file
 * template, and be sure to move the file to the default location
 * (~/.aws/credentials) where the sample code loads the credentials from.
 *
 * IMPORTANT: To avoid accidental leakage of your credentials, DO NOT
 * keep the credentials file in your source directory.
 */
public class SimpleQueueServiceSample {

    public static void main(String[] args) throws Exception {

        /*
         * The ProfileCredentialsProvider returns your [default] credential profile
         * by reading from the credentials file located at ~/.aws/credentials.
         *
         * AWSCredentials credentials = null;
         * try {
         *     credentials = new ProfileCredentialsProvider().getCredentials();
         * } catch (Exception e) {
         *     throw new AmazonClientException(
         *         "Cannot load the credentials from the credential profiles file. " +
         *         "Please make sure that your credentials file is at the correct " +
         *         "location (~/.aws/credentials), and is in valid format.",
         *         e);
         * }
         * 
         * AmazonSQS sqs = AmazonSQSClientBuilder.standard()
         *     .withRegion(Regions.US_WEST_2)
         *     .build();
         */
System.out.println("===========================================");
System.out.println("Getting Started with Amazon SQS");
System.out.println("=========================================\n");

try {
    // Create a queue
    System.out.println("Creating a new SQS queue called MyQueue.\n");
    CreateQueueRequest createQueueRequest = new CreateQueueRequest("MyQueue");
    String myQueueUrl = sqs.createQueue(createQueueRequest).getQueueUrl();

    // List queues
    System.out.println("Listing all queues in your account.\n");
    for (String queueUrl : sqs.listQueues().getQueueUrls()) {
        System.out.println("  QueueUrl: " + queueUrl);
    }
    System.out.println();

    // Send a message
    System.out.println("Sending a message to MyQueue.\n");
    sqs.sendMessage(new SendMessageRequest(myQueueUrl, "This is my message text."));

    // Receive messages
    System.out.println("Receiving messages from MyQueue.\n");
    ReceiveMessageRequest receiveMessageRequest = new ReceiveMessageRequest(myQueueUrl);
    List<Message> messages = sqs.receiveMessage(receiveMessageRequest).getMessages();
    for (Message message : messages) {
        System.out.println("  Message");
        System.out.println("    MessageId: " + message.getMessageId());
        System.out.println("    ReceiptHandle: " + message.getReceiptHandle());
        System.out.println("    MD5OfBody: " + message.getMD5OfBody());
        System.out.println("    Body: " + message.getBody());
        for (Entry<String, String> entry : message.getAttributes().entrySet()) {
            System.out.println("  Attribute");
            System.out.println("    Name: " + entry.getKey());
            System.out.println("    Value: " + entry.getValue());
        }
    }
    System.out.println();

    // Delete a message
    System.out.println("Deleting a message.\n");
    String messageReceiptHandle = messages.get(0).getReceiptHandle();
    sqs.deleteMessage(new DeleteMessageRequest(myQueueUrl, messageReceiptHandle));

    // Delete a queue
    System.out.println("Deleting the test queue.\n");
    sqs.deleteQueue(new DeleteQueueRequest(myQueueUrl));
} catch (AmazonServiceException ase) {
    System.out.println("Caught an AmazonServiceException, which means your request made it " + "to Amazon SQS, but was rejected with an error response for some reason.");
    System.out.println("Error Message: " + ase.getMessage());
    System.out.println("HTTP Status Code: " + ase.getStatusCode());
    System.out.println("AWS Error Code: " + ase.getErrorCode());
    System.out.println("Error Type: " + ase.getErrorCode());
    System.out.println("Request ID: " + ase.getRequestId());
} catch (AmazonClientException ace) {
    System.out.println("Caught an AmazonClientException, which means the client encountered " + "a serious internal problem while trying to communicate with SQS, such as not " + "being able to access the network.");
FIFO (First-In-First-Out) Queues

FIFO queues are available in the US East (N. Virginia), US East (Ohio), US West (Oregon), and EU (Ireland) regions. In addition to having all the capabilities of the standard queue (p. 47), FIFO (First-In-First-Out) queues are designed to enhance messaging between applications when the order of operations and events is critical, or where duplicates can’t be tolerated. FIFO queues also provide exactly-once processing but have a limited number of transactions per second (TPS):

- Without batching, FIFO queues can support up to 300 messages per second (300 send, receive, or delete operations per second).
- If you take advantage of the maximum batching (p. 176) of 10 messages per operation, FIFO queues can support up to 3,000 messages per second.

FIFO queues are designed to enhance messaging between applications when the order of operations and events is critical, for example:

- Ensure that user-entered commands are executed in the right order.
- Display the correct product price by sending price modifications in the right order.
- Prevent a student from enrolling in a course before registering for an account.

**Note**
The name of a FIFO queue must end with the .fifo suffix. The suffix counts towards the 80-character queue name limit. To determine whether a queue is FIFO (p. 51), you can check whether the queue name ends with the suffix.

For best practices of working with FIFO queues, see Recommendations for FIFO (First-In-First-Out) Queues (p. 111) and General Recommendations (p. 109).

For information about compatibility of clients and services with FIFO queues, see Compatibility (p. 56).

**Topics**
- Message Ordering (p. 51)
- FIFO Queue Logic (p. 52)
- Exactly-Once Processing (p. 53)
- Getting Started with FIFO Queues (p. 53)
- Moving from a Standard Queue to a FIFO Queue (p. 56)
- Compatibility (p. 56)

**Message Ordering**

The FIFO queue improves upon and complements the standard queue (p. 47). The most important features of this queue type are FIFO (First-In-First-Out) delivery and exactly-once processing: The order in which messages are sent and received is strictly preserved and a message is delivered once and remains
available until a consumer processes and deletes it; duplicates are not introduced into the queue. In addition, FIFO queues support message groups that allow multiple ordered message groups within a single queue.

**FIFO Queue Logic**

**Key Terms**

The following key terms can help you better understand the functionality of FIFO queues. For detailed descriptions, see the *Amazon Simple Queue Service API Reference*.

**Message Deduplication ID**

The token used for deduplication of sent messages. If a message with a particular message deduplication ID is sent successfully, any messages sent with the same message deduplication ID are accepted successfully but aren't delivered during the 5-minute deduplication interval.

*Note*

Message deduplication applies to an entire queue, not to individual message groups.

**Message Group ID**

The tag that specifies that a message belongs to a specific message group. Messages that belong to the same message group are always processed one by one, in a strict order relative to the message group (however, messages that belong to different message groups might be processed out of order).

**Receive Request Attempt ID**

The token used for deduplication of ReceiveMessage calls.

**Sequence Number**

The large, non-consecutive number that Amazon SQS assigns to each message.

**Sending Messages**

If multiple messages are sent in succession to a FIFO queue, each with a distinct message deduplication ID, Amazon SQS stores the messages and acknowledges the transmission. Then, each message can be received and processed in the exact order in which the messages were transmitted.

In FIFO queues, messages are ordered based on message group ID. If multiple hosts (or different threads on the same host) send messages with the same message group ID to a FIFO queue, Amazon SQS stores the messages in the order in which they arrive for processing. To ensure that Amazon SQS preserves the order in which messages are sent and received, ensure that each producer uses a unique message group ID to send all its messages.

FIFO queue logic applies only per message group ID. Each message group ID represents a distinct ordered message group within an Amazon SQS queue. For each message group ID, all messages are sent and received in strict order. However, messages with different message group ID values might be sent and received out of order. You must associate a message group ID with a message. If you don't provide a message group ID, the action fails. If you require a single group of ordered messages, provide the same message group ID for messages sent to the FIFO queue.

**Receiving Messages**

You can't request to receive messages with a specific message group ID.
When receiving messages from a FIFO queue with multiple message group IDs, Amazon SQS first attempts to return as many messages with the same message group ID as possible. This allows other consumers to process messages with a different message group ID.

## Retrying Multiple Times

FIFO queues allow the producer or consumer to attempt multiple retries:

- If the producer detects a failed `SendMessage` action, it can retry sending as many times as necessary, using the same message deduplication ID. Assuming that the producer receives at least one acknowledgement before the deduplication interval expires, multiple retries neither affect the ordering of messages nor introduce duplicates.
- If the consumer detects a failed `ReceiveMessage` action, it can retry as many times as necessary, using the same receive request attempt ID. Assuming that the consumer receives at least one acknowledgement before the visibility timeout expires, multiple retries do not affect the ordering of messages.
- When you receive a message with a message group ID, no more messages for the same message group ID are returned unless you delete the message or it becomes visible.

## Exactly-Once Processing

Unlike standard queues, FIFO queues do not introduce duplicate messages. FIFO queues help you avoid sending duplicates to a queue. If you retry the `SendMessage` action within the 5-minute deduplication interval, Amazon SQS does not introduce any duplicates into the queue.

To configure deduplication, you must do one of the following:

- Enable content-based deduplication. This instructs Amazon SQS to use a SHA-256 hash to generate the message deduplication ID using the body of the message—but not the attributes of the message. For more information, see the documentation on the `CreateQueue`, `GetQueueAttributes`, and `SetQueueAttributes` actions in the *Amazon Simple Queue Service API Reference*.
- Explicitly provide the message deduplication ID (or view the sequence number) for the message. For more information, see the documentation on the `SendMessage`, `SendMessageBatch`, and `ReceiveMessage` actions in the *Amazon Simple Queue Service API Reference*.

## Getting Started with FIFO Queues

The following example Java code creates a queue and sends, receives, and deletes a message.

```java
package sqs.fifo.samples;
```
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Region;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClient;
import com.amazonaws.services.sqs.model.CreateQueueRequest;
import com.amazonaws.services.sqs.model.DeleteMessageRequest;
import com.amazonaws.services.sqs.model.DeleteQueueRequest;
import com.amazonaws.services.sqs.model.Message;
import com.amazonaws.services.sqs.model.ReceiveMessageRequest;
import com.amazonaws.services.sqs.model.SendMessageRequest;
import com.amazonaws.services.sqs.model.SendMessageResult;

public class SQSFIFOJavaClientSample {
    public static void main(String[] args) throws Exception {
        /*
         * The ProfileCredentialsProvider returns your [default]
         * credential profile by reading from the credentials file located at
         * (~/.aws/credentials).
         */
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider().getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException(
                    "Can't load the credentials from the credential profiles file. " +
                    "Please make sure that your credentials file is at the correct " +
                    "location (~/.aws/credentials), and is a in valid format. ",
                    e);
        }

        AmazonSQSClient sqs = new AmazonSQSClient(credentials);
        sqs.setEndpoint("https://sqs.us-east-2.amazonaws.com");

        System.out.println("=======================================================");
        System.out.println("Getting Started with Amazon SQS FIFO Queues");
        System.out.println("=======================================================");

        try {
            // Create a FIFO queue
            System.out.println("Creating a new Amazon SQS FIFO queue called MyFifoQueue.fifo.");
            Map<String, String> attributes = new HashMap<String, String>();
            // A FIFO queue must have the FifoQueue attribute set to True
            attributes.put("FifoQueue", "true");
            // Generate a MessageDeduplicationId based on the content, if the user doesn't
            // provide a MessageDeduplicationId
            attributes.put("ContentBasedDeduplication", "true");
            // The FIFO queue name must end with the .fifo suffix
            CreateQueueRequest createQueueRequest = new
                    CreateQueueRequest("MyFifoQueue.fifo").withAttributes(attributes);
            String myQueueUrl = sqs.createQueue(createQueueRequest).getQueueUrl();
            // List queues
            System.out.println("Listing all queues in your account.");
            for (String queueUrl : sqs.listQueues().getQueueUrls()) {
            }
        }
    }
}
System.out.println("  QueueUrl: " + queueUrl);
}
System.out.println();

// Send a message
System.out.println("Sending a message to MyFifoQueue.fifo.
");
SendMessageRequest sendMessageRequest = new SendMessageRequest(myQueueUrl, "This is my message text.");
// You must provide a non-empty MessageGroupId when sending messages to a FIFO queue
sendMessageRequest.setMessageGroupId("messageGroup1");
// Uncomment the following to provide the MessageDeduplicationId
//sendMessageRequest.setMessageDeduplicationId("1");
SendMessageResult sendMessageResult = sqs.sendMessage(sendMessageRequest);
String sequenceNumber = sendMessageResult.getSequenceNumber();
String messageId = sendMessageResult.getMessageId();
System.out.println("SendMessage succeed with messageId " + messageId + ", sequence number " + sequenceNumber + "\n");

// Receive messages
System.out.println("Receiving messages from MyFifoQueue.fifo.\n");
ReceiveMessageRequest receiveMessageRequest = new ReceiveMessageRequest(myQueueUrl);
// Uncomment the following to provide the ReceiveRequestDeduplicationId
//receiveMessageRequest.setReceiveRequestAttemptId("1");
List<Message> messages = sqs.receiveMessage(receiveMessageRequest).getMessages();
for (Message message : messages) {
    System.out.println("  Message");
    System.out.println("    MessageId:     " + message.getMessageId());
    System.out.println("    ReceiptHandle: " + message.getReceiptHandle());
    System.out.println("    MD5OfBody:     " + message.getMD5OfBody());
    System.out.println("    Body:          " + message.getBody());
    for (Entry<String, String> entry : message.getAttributes().entrySet()) {
        System.out.println("      Attribute");
        System.out.println("        Name:  " + entry.getKey());
        System.out.println("        Value: " + entry.getValue());
    }
}
System.out.println();

// Delete the message
System.out.println("Deleting the message.\n");
String messageReceiptHandle = messages.get(0).getReceiptHandle();
sqs.deleteMessage(new DeleteMessageRequest(myQueueUrl, messageReceiptHandle));

// Delete the queue
System.out.println("Deleting the queue.\n");
sqs.deleteQueue(new DeleteQueueRequest(myQueueUrl));
}

} catch (AmazonServiceException ase) {
    System.out.println("Caught an AmazonServiceException, which means your request made it " + "to Amazon SQS, but was rejected with an error response for some reason." + "Error Message: " + ase.getMessage());
    System.out.println("HTTP Status Code: " + ase.getStatusCode());
    System.out.println("AWS Error Code: " + ase.getErrorCode());
    System.out.println("Error Type: " + ase.getErrorType());
    System.out.println("Request ID: " + ase.getRequestId());
} catch (AmazonClientException ace) {
    System.out.println("Caught an AmazonClientException, which means the client encountered " + "a serious internal problem while trying to communicate with SQS, such as not " + "being able to access the network." + "Error Message: " + ace.getMessage());
Moving from a Standard Queue to a FIFO Queue

If you have an existing application that uses standard queues and you want to take advantage of the ordering or exactly-once processing features of FIFO queues, you need to configure the queue and your application correctly.

Note
You can't convert an existing standard queue into a FIFO queue. To make the move, you must either create a new FIFO queue for your application or delete your existing standard queue and recreate it as a FIFO queue.

Moving Checklist

Use the following checklist to ensure that your application works correctly with a FIFO queue.

- FIFO queues are limited to 300 transactions per second (TPS). If your application generates a high throughput of messages, consider using a standard queue instead.
- FIFO queues don't support per-message delays, only per-queue delays. If your application sets the same value of the `DelaySeconds` parameter on each message, you must modify your application to remove the per-message delay and set `DelaySeconds` on the entire queue instead.
- Every message sent to a FIFO queue requires a message group ID. If you don't need multiple ordered message groups, specify the same message group ID for all your messages.
- Before sending messages to a FIFO queue, confirm the following:
  - If your application can send messages with identical message bodies, you can modify your application to provide a unique message deduplication ID for each sent message.
  - If your application sends messages with unique message bodies, you can enable content-based deduplication.
- You don't have to make any code changes to your consumer. However, if it takes a long time to process messages and your visibility timeout is set to a high value, consider adding a receive request attempt ID to each `ReceiveMessage` action. This allows you to retry receive attempts in case of networking failures and prevents queues from pausing due to failed receive attempts.

For more information, see the Amazon Simple Queue Service API Reference.

Compatibility

Clients

The Amazon SQS Buffered Asynchronous Client doesn't currently support FIFO queues.

Services

If your application uses multiple AWS services, or a mix of AWS and external services, it is important to understand which service functionality doesn't support FIFO queues.

Some AWS or external services that send notifications to Amazon SQS might not be compatible with FIFO queues, despite allowing you to set a FIFO queue as a target.

The following features of AWS services aren't currently compatible with FIFO queues:
Queue and Message Identifiers

General Identifiers

Queue Name and URL

When you create a new queue, you must specify a queue name that is unique within the scope of all your queues. Amazon SQS assigns each queue you create an identifier called a queue URL that includes the queue name and other Amazon SQS components. Whenever you want to perform an action on a queue, you provide its queue URL.

The name of a FIFO queue must end with the .fifo suffix. The suffix counts towards the 80-character queue name limit. To determine whether a queue is FIFO, you can check whether the queue name ends with the suffix.

The following is the queue URL for a queue named MyQueue owned by a user with the AWS account number 123456789012.

http://sqs.us-east-2.amazonaws.com/123456789012/MyQueue

Important
In your system, always store the entire queue URL exactly as Amazon SQS returns it to you when you create the queue (for example, http://sqs.us-east-2.amazonaws.com/123456789012/queue2). Don't build the queue URL from its separate components each time you need to specify the queue URL in a request because Amazon SQS can change the components that make up the queue URL.

You can also get the queue URL for a queue by listing your queues. For more information, see ListQueues.

Message ID

Each message receives a system-assigned message ID that Amazon SQS returns to you in the SendMessage response. This identifier is useful for identifying messages. (However, to delete a message you need the message's receipt handle.) The maximum length of a message ID is 100 characters.

Receipt Handle

Every time you receive a message from a queue, you receive a receipt handle for that message. This handle is associated with the action of receiving the message, not with the message itself. To delete the message or to change the message visibility, you must provide the receipt handle (not the message
ID). Thus, you must always receive a message before you can delete it (you can't put a message into the queue and then recall it). The maximum length of a receipt handle is 1024 characters.

**Important**
If you receive a message more than once, each time you receive it, you get a different receipt handle. You must provide the most recently received receipt handle when you request to delete the message (otherwise, the message might not be deleted).

The following is an example of a receipt handle.

```
MbZj6wDwli+JvwwJаАV+3dcjkJ2YW2vA3+STFF1jTM8tJ7g6HRG6PYSasuWXPB+CW
LjiFjgXUv1uJ1gUPA8WV66FU/WeR4mq20KpEGYWbnRgJVAyeMjeU52BdtcQ+QE
auM2c08Rv37aIW2iJKq3MoMFx1YvV11A2x/KSbkJ0=
```

### Additional Identifiers for FIFO Queues

For more information about the following identifiers, see *Exactly-Once Processing* (p. 53) and the Amazon Simple Queue Service API Reference.

#### Message Deduplication ID

The token used for deduplication of sent messages. If a message with a particular message deduplication ID is sent successfully, any messages sent with the same message deduplication ID are accepted successfully but aren't delivered during the 5-minute deduplication interval.

#### Message Group ID

The tag that specifies that a message belongs to a specific message group. Messages that belong to the same message group are always processed one by one, in a strict order relative to the message group (however, messages that belong to different message groups might be processed out of order).

#### Sequence Number

The large, non-consecutive number that Amazon SQS assigns to each message.

### Resources Required to Process Messages

To help you estimate the resources you need to process queued messages, Amazon SQS can determine the approximate number of delayed, visible, and not visible messages in a queue. For more information about visibility, see *Visibility Timeout* (p. 59).

**Note**

For standard queues, the result is approximate because of the distributed architecture of Amazon SQS. In most cases, the count should be close to the actual number of messages in the queue.

For FIFO queues, the result is exact.

The following table lists the API action to use.

<table>
<thead>
<tr>
<th>To do this...</th>
<th>Use this action</th>
<th>Use this AttributeName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get the approximate number of messages in the queue.</td>
<td>GetQueueAttributeName</td>
<td>ApproximateNumberOfMessages</td>
</tr>
</tbody>
</table>
## Visibility Timeout

When a consumer receives and processes a message from a queue, the message remains in the queue. Amazon SQS doesn’t automatically delete the message. Because Amazon SQS is a distributed system, there’s no guarantee that the consumer actually receives the message (for example, due to a connectivity issue, or due to an issue in the consumer application). Thus, the consumer must delete the message from the queue after receiving and processing it.

Immediately after the message is received, it remains in the queue. To prevent other consumers from processing the message again, Amazon SQS sets a visibility timeout, a period of time during which Amazon SQS prevents other consumers from receiving and processing the message.

### Note
For standard queues, the visibility timeout isn’t a guarantee against receiving a message twice. For more information, see [At-Least-Once Delivery](#).

FIFO queues allow the producer or consumer to attempt multiple retries:

- If the producer detects a failed `SendMessage` action, it can retry sending as many times as necessary, using the same message deduplication ID. Assuming that the producer receives at least one acknowledgement before the deduplication interval expires, multiple retries neither affect the ordering of messages nor introduce duplicates.
- If the consumer detects a failed `ReceiveMessage` action, it can retry as many times as necessary, using the same receive request attempt ID. Assuming that the consumer receives at least one acknowledgement before the visibility timeout expires, multiple retries do not affect the ordering of messages.
- When you receive a message with a message group ID, no more messages for the same message group ID are returned unless you delete the message or it becomes visible.

### Topics
- Inflight Messages (p. 60)
- Setting the Visibility Timeout (p. 60)
- Changing the Visibility Timeout for a Message (p. 60)
- Terminating the Visibility Timeout for a Message (p. 60)
- Visibility Timeout API Actions (p. 61)

---

<table>
<thead>
<tr>
<th>To do this...</th>
<th>Use this action</th>
<th>Use this AttributeName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get the approximate number of messages that are pending to be added to the queue.</td>
<td>GetQueueAttributes</td>
<td>ApproximateNumberOfMessagesDelayed</td>
</tr>
<tr>
<td>Get the approximate number of messages in the queue that are not visible (messages in flight).</td>
<td>GetQueueAttributes</td>
<td>ApproximateNumberOfMessagesNotVisible</td>
</tr>
</tbody>
</table>
Inflight Messages

A message is considered to be in flight after it's received from a queue by a consumer, but not yet deleted from the queue.

For standard queues, there can be a maximum of 120,000 inflight messages per queue. If you reach this limit, Amazon SQS returns the OverLimit error message. To avoid reaching the limit, you should delete messages from the queue after they're processed. You can also increase the number of queues you use to process your messages.

For FIFO queues, there can be a maximum of 20,000 inflight messages per queue. If you reach this limit, Amazon SQS returns no error messages.

Setting the Visibility Timeout

The visibility timeout begins when Amazon SQS returns a message. During this time, the consumer processes and deletes the message. However, if the consumer fails before deleting the message and your system doesn't call the DeleteMessage action for that message before the visibility timeout expires, the message becomes visible to other consumers and the message is received again. If a message must be received only once, your consumer should delete it within the duration of the visibility timeout.

Every Amazon SQS queue has the default visibility timeout setting of 30 seconds. You can change this setting for the entire queue. Typically, you should set the visibility timeout to the average time it takes to process and delete a message from the queue. When receiving messages, you can also set a special visibility timeout for the returned messages without changing the overall queue timeout.

If you don't know how long it takes to process a message, specify the initial visibility timeout (for example, 2 minutes) and the period of time after which you can check whether the message is processed (for example, 1 minute). If the message isn't processed, extend the visibility timeout (for example, to 3 minutes).

Changing the Visibility Timeout for a Message

When you receive a message from a queue and begin to process it, the visibility timeout for the queue may be insufficient (for example, you might need to process and delete a message). You can shorten or extend a message's visibility by specifying a new timeout value using the ChangeMessageVisibility API action.

For example, if the default timeout for a queue is 60 seconds, 15 seconds have elapsed since you received the message, and you send a ChangeMessageVisibility call with VisibilityTimeout set to 10 seconds, the 10 seconds begin to count from the time that you make the ChangeMessageVisibility call. Thus, any attempt to change the visibility timeout or to delete that message 10 seconds after you initially change the visibility timeout (a total of 25 seconds) might result in an error.

**Note**

The new timeout period takes effect from the time you call the ChangeMessageVisibility API action. In addition, the new timeout period applies only to the particular receipt of the message. ChangeMessageVisibility does not affect the timeout of later receipts of the message or later queues.

Terminating the Visibility Timeout for a Message

When you receive a message from a queue, you might find that you actually don't want to process and delete that message. Amazon SQS allows you to terminate the visibility timeout for a specific message. This makes the message immediately visible to other components in the system and available for processing.
To terminate a message's visibility timeout after calling `ReceiveMessage`, call `ChangeMessageVisibility` with `VisibilityTimeout` set to 0 seconds.

Visibility Timeout API Actions

The following table lists the API actions you can use to manipulate the visibility timeout. Use each action's `VisibilityTimeout` parameter to set or get the value.

<table>
<thead>
<tr>
<th>Task</th>
<th>API Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set the visibility timeout for a queue</td>
<td><code>SetQueueAttributes</code></td>
</tr>
<tr>
<td>View the visibility timeout for a queue</td>
<td><code>GetQueueAttributes</code></td>
</tr>
<tr>
<td>Set the visibility timeout for received messages without affecting the visibility timeout of the entire queue</td>
<td><code>ReceiveMessage</code></td>
</tr>
<tr>
<td>Extend or terminate a message's visibility timeout</td>
<td><code>ChangeMessageVisibility</code></td>
</tr>
<tr>
<td>Extend or terminate the visibility timeout for up to 10 messages</td>
<td><code>ChangeMessageVisibilityBatch</code></td>
</tr>
</tbody>
</table>

Using Amazon SQS Dead-Letter Queues

Amazon SQS supports **dead-letter queues**. A dead-letter queue is a queue that other (source) queues can target for messages that can't be processed (consumed) successfully. You can set aside and isolate these messages in the dead-letter queue to determine why their processing doesn't succeed.

**Topics**
- How Do Dead-Letter Queues Work? (p. 61)
- What are the Benefits of Dead-Letter Queues? (p. 62)
- How Do Different Queue Types Handle Message Failure? (p. 62)
- When Should I Use a Dead-Letter Queue? (p. 63)
- Getting Started with Dead-Letter Queues (p. 63)
- Troubleshooting Dead-Letter Queues (p. 63)

How Do Dead-Letter Queues Work?

Sometimes, messages can't be processed because of a variety of possible issues, such as erroneous conditions within the producer or consumer application. For example, if a user places an order within a certain number of minutes of creating an account, the producer might pass a message with an empty string instead of a customer identifier.

Occasionally, producers and consumers might fail to interpret aspects of the protocol that they use to communicate, causing message corruption or loss. Also, the consumer's hardware errors might corrupt message payload.

The **redrive policy** specifies the **source queue**, the **dead-letter queue**, and the conditions under which Amazon SQS moves messages from the former to the latter if the consumer of the source queue fails to process a message a specified number of times. For example, if the source queue has a redrive policy with `maxReceiveCount` set to 5, and the consumer of the source queue receives a message 5 times without ever processing and deleting it, Amazon SQS moves the message to the dead-letter queue.
To specify a dead-letter queue, you can use the AWS Management Console or an API action (p. 37). You must do this for each queue that sends messages to a dead-letter queue. Multiple queues can target a single dead-letter queue. For more information, see Tutorial: Configuring an Amazon SQS Dead-Letter Queue (p. 37) and the RedrivePolicy attribute of the CreateQueue or SetQueueAttributes API action.

Important
The dead-letter queue of a FIFO queue must also be a FIFO queue. Similarly, the dead-letter queue of a standard queue must also be a standard queue. You must use the same AWS account to create the dead-letter queue and the other queues that send messages to the dead-letter queue. Also, dead-letter queues must reside in the same region as the other queues that use the dead-letter queue. For example, if you create a queue in the US East (Ohio) region and you want to use a dead-letter queue with that queue, the second queue must also be in the US East (Ohio) region.

What are the Benefits of Dead-Letter Queues?

The main task of a dead-letter queue is handling message failure. A dead-letter queue lets you set aside and isolate messages that can't be processed correctly to determine why their processing didn't succeed. Setting up a dead-letter queue allows you to do the following:

- Configure an alarm for any messages delivered to a dead-letter queue.
- Examine logs for exceptions that might have caused messages to be delivered to a dead-letter queue.
- Analyze the contents of messages delivered to a dead-letter queue to diagnose software or the producer's or consumer's hardware issues.
- Determine whether you have given your consumer sufficient time to process messages.

How Do Different Queue Types Handle Message Failure?

Standard Queues

Standard queues (p. 47) keep processing messages until the expiration of the retention period. This ensures continuous processing of messages, which minimizes the chances of your queue being blocked by messages that can't be processed. It also ensures fast recovery for your queue.

In a system that processes thousands of messages, having a large number of messages that the consumer repeatedly fails to acknowledge and delete might increase costs and place extra load on the hardware. Instead of trying to process failing messages until they expire, it is better to move them to a dead-letter queue after a few processing attempts.

Note
Standard queues allow a high number of in-flight messages. If the majority of your messages can't be consumed and aren't sent to a dead-letter queue, your rate of processing valid messages can slow down. Thus, to maintain the efficiency of your queue, you must ensure that your application handles message processing correctly.

FIFO Queues

FIFO queues (p. 51) ensure exactly-once processing by consuming messages in sequence from a message group. Thus, although the consumer can continue to retrieve ordered messages from another message group, the first message group remains unavailable until the message blocking the queue is processed successfully.
When Should I Use a Dead-Letter Queue?

Do use dead-letter queues with standard queues. You should always take advantage of dead-letter queues when your applications don't depend on ordering. Dead-letter queues can help you troubleshoot incorrect message transmission operations.

Note
Even when you use dead-letter queues, you should continue to monitor your queues and retry sending messages that fail for transient reasons.

Do use dead-letter queues to decrease the number of messages and to reduce the possibility of exposing your system to poison-pill messages (messages that can be received but can't be processed).

Don't use a dead-letter queue with standard queues when you want to be able to keep retrying the transmission of a message indefinitely. For example, don't use a dead-letter queue if your program must wait for a dependent process to become active or available.

Don't use a dead-letter queue with a FIFO queue if you don't want to break the exact order of messages or operations. For example, don't use a dead-letter queue with instructions in an Edit Decision List (EDL) for a video editing suite, where changing the order of edits changes the context of subsequent edits.

Getting Started with Dead-Letter Queues

For information about how to create a dead-letter queue using the AWS Management Console or using the query API action, see the Tutorial: Configuring an Amazon SQS Dead-Letter Queue (p. 37) tutorial.

You can configure an Amazon SQS queue as a dead-letter queue using the following API actions.

<table>
<thead>
<tr>
<th>Task</th>
<th>API Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configure a dead-letter queue for a new queue.</td>
<td>CreateQueue</td>
</tr>
<tr>
<td>Configure a dead-letter queue for an existing queue.</td>
<td>SetQueueAttributes</td>
</tr>
<tr>
<td>Determine whether a queue uses a dead-letter queue.</td>
<td>GetQueueAttributes</td>
</tr>
</tbody>
</table>

Troubleshooting Dead-Letter Queues

In some cases, Amazon SQS dead-letter queues might not always behave as expected. This section gives an overview of common issues and shows how to resolve them.

Viewing Messages using the AWS Management Console Might Cause Messages to be Moved to a Dead-Letter Queue

Amazon SQS counts viewing a message in the AWS Management Console against the corresponding queue's redrive policy. Thus, if you view a message in the AWS Management Console the number of times specified in the corresponding queue's redrive policy, the message is moved to the corresponding queue's dead-letter queue.
To adjust this behavior, you can do one of the following:

- Increase the **Maximum Receives** setting for the corresponding queue's redrive policy.
- Avoid viewing the corresponding queue's messages in the AWS Management Console.

The **NumberOfMessagesSent** and **NumberOfMessagesReceived** for a Dead-Letter Queue Don't Match

If you send a message to a dead-letter queue manually, it is captured by the **NumberOfMessagesSent** metric. However, a message is sent to a dead-letter queue as a result of a failed processing attempt, it isn't captured by this metric. Thus, it is possible for the values of **NumberOfMessagesSent** and **NumberOfMessagesReceived** to be different.

---

**Message Lifecycle**

The following diagram describes the lifecycle of an Amazon SQS message, from creation to deletion. In this example, a queue already exists.

1. Component 1 sends Message A to the queue
2. Component 2 retrieves Message A from the queue and the visibility timeout period starts
3. Component 2 processes Message A and then deletes it from the queue during the visibility timeout period

Message Lifecycle

1. Component 1 sends Message A to a queue, and the message is distributed across the Amazon SQS servers redundantly.
2. When Component 2 is ready to process a message, it consumes messages from the queue, and Message A is returned. While Message A is being processed, it remains in the queue and isn't returned to subsequent receive requests for the duration of the visibility timeout.
Tagging Your Amazon SQS Queues

To organize and identify your Amazon SQS queues for cost allocation, you can add metadata tags that identify a queue's purpose, owner, or environment. This is especially useful when you have many queues.

You can use cost allocation tags to organize your AWS bill to reflect your own cost structure. To do this, sign up to get your AWS account bill to include tag keys and values. For more information, see Setting Up a Monthly Cost Allocation Report in the AWS Billing and Cost Management User Guide.

Overview

Each tag consists of a key-value pair that you define. For example, you can easily identify your production and testing queues if you tag your queues as follows:

<table>
<thead>
<tr>
<th>Queue</th>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MyQueueA</td>
<td>QueueType</td>
<td>Production</td>
</tr>
<tr>
<td>MyQueueB</td>
<td>QueueType</td>
<td>Testing</td>
</tr>
</tbody>
</table>

**Note**

When you use queue tags, keep the following guidelines in mind:

- We don't recommend adding more than 50 tags to a queue.
- Tags don't have any semantic meaning. Amazon SQS interprets tags as character strings.
- Tags are case-sensitive.
- A new tag with a key identical to that of an existing tag overwrites the existing tag.
- Tagging API actions are limited to 5 TPS per AWS account. If your application requires a higher throughput, file a technical support request.

For a full list of tag restrictions, see Limits Related to Queues (p. 113).

You can't add tags to a queue when you create it. However, you can add, update, or remove tags for existing queues at any time using the AWS Management Console or the Amazon SQS API.

**Getting Started with Tagging**

For information on how to manage Amazon SQS queue tags using the AWS Management Console or API actions, see the Adding, Updating, and Removing Tags from an Amazon SQS Queue (p. 44) tutorial.
You can list, add, update, or remove tags for an Amazon SQS queue using the following API actions:

<table>
<thead>
<tr>
<th>Task</th>
<th>API Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add tags to a queue or update the tags added to a queue</td>
<td>TagQueue</td>
</tr>
<tr>
<td>Remove tags from a queue</td>
<td>UntagQueue</td>
</tr>
<tr>
<td>List the tags added to a queue</td>
<td>ListQueueTags</td>
</tr>
</tbody>
</table>

Using Amazon SQS Message Attributes

Amazon SQS provides support for message attributes. Message attributes allow you to provide structured metadata items (such as timestamps, geospatial data, signatures, and identifiers) about the message. Message attributes are optional and separate from, but sent along with, the message body. This information can be used by the consumer of the message to help decide how to handle the message without having to first process the message body. Each message can have up to 10 attributes. To specify message attributes, you can use the AWS Management Console, AWS software development kits (SDKs), or query API.

Topics

- Message Attribute Items and Validation (p. 66)
- Message Attribute Data Types and Validation (p. 67)
- Using Message Attributes with the AWS Management Console (p. 67)
- Using Message Attributes with the AWS SDKs (p. 70)
- Using Message Attributes with the Amazon SQS Query API (p. 71)
- MD5 Message-Digest Calculation (p. 72)

Message Attribute Items and Validation

Each message attribute consists of the following items:

- **Name** – The message attribute name can contain the following characters: A-Z, a-z, 0-9, underscore(_), hyphen(-), and period (.). The name must not start or end with a period, and it should not have successive periods. The name is case-sensitive and must be unique among all attribute names for the message. The name can be up to 256 characters long. The name can't start with AWS or Amazon. (or any variations in casing) because these prefixes are reserved for use by Amazon Web Services.

- **Type** – The supported message attribute data types are String, Number, and Binary. You can also provide custom information about the type. The data type has the same restrictions on the content as the message body. The data type is case-sensitive, and it can be up to 256 bytes long. For more information, see the Message Attribute Data Types and Validation (p. 67) section.

- **Value** – The user-specified message attribute value. For string data types, the value attribute has the same restrictions on the content as the message body. For more information, see SendMessage.

Name, type, and value must not be empty or null. In addition, the message body should not be empty or null. All parts of the message attribute, including name, type, and value, are included in the message size restriction, which is currently 256 KB (262,144 bytes).
Message Attribute Data Types and Validation

Message attribute data types identify how the message attribute values are handled by Amazon SQS. For example, if the type is a number, Amazon SQS validates that it's a number.

Amazon SQS supports the logical data types **Binary**, **Number**, and **String** with optional custom type labels in the format `.custom-type`.

- **Binary** – Binary type attributes can store any binary data, for example, compressed data, encrypted data, or images.
- **Number** – Numbers are positive or negative integers or floating point numbers. Numbers have sufficient range and precision to encompass most of the possible values that integers, floats, and doubles typically support. A number can have up to 38 digits of precision, and it can be between $10^{-128}$ and $10^{+126}$. Leading and trailing zeroes are trimmed.
- **String** – Strings are Unicode with UTF-8 binary encoding. For more information, see ASCII Printable Characters.

You can append a custom type label to any supported data type to create custom data types. This capability is similar to type traits in programming languages. For example, if you have an application that needs to know which type of number is being sent in the message, you can create custom types similar to the following: **Number.byte**, **Number.short**, **Number.int**, and **Number.float**. Another example using the binary data type is to use **Binary.gif** and **Binary.png** to distinguish among different image file types in a message or batch of messages. The appended data is optional and opaque to Amazon SQS, which means that the appended data isn't interpreted, validated, or used by Amazon SQS. The Custom Type extension has the same restrictions on allowed characters as the message body.

Using Message Attributes with the AWS Management Console

You can use the AWS Management Console to configure message attributes. In the Amazon SQS console, select a queue, choose the **Queue Actions** drop-down list, and then select **Send a Message**. The console expects the user to input a Base-64-encoded value for sending a Binary type.
On the **Message Attributes** tab, enter a name, select the type, and enter a value for the message attribute. Optionally, you can also append custom information to the type. For example, the following screen shows the **Number** type selected with **byte** added for customization. For more information about custom data for the supported data types, see the Message Attribute Data Types and Validation (p. 67) section.

To add an attribute, choose **Add Attribute**. The attribute information appears in the **Name**, **Type**, and **Values** list.
You can also use the console to view information about the message attributes for received messages. In the console, select a queue, and from the **Queue Actions** drop-down list select **View/Delete Messages**. In the list of messages, choose **Message Details** to view the information. For example, you can see the message attribute size and MD5 message digest.
Using Message Attributes with the AWS SDKs

The AWS SDKs provide APIs in several languages for using message attributes with Amazon SQS. This section includes some Java examples that show how to work with message attributes. These examples can be integrated with the SimpleQueueServiceSample.java sample from the SDK for Java. MessageBody and MessageAttributes checksums are automatically calculated and compared with the data Amazon SQS returns by the latest SDK for Java. For more information about the SDK for Java, see Getting Started with the AWS SDK for Java.

The following three Java examples show how to use the MessageAttributeValue method to set the String, Number, and Binary parameters for the message attributes:

**String**

```java
Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
messageAttributes.put("attributeName", new MessageAttributeValue().withDataType("String").withStringValue("string-value-attribute-value"));
```

**Number**

```java
Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
messageAttributes.put("attributeName", new MessageAttributeValue().withDataType("Number").withStringValue("230.000000000000000001"));
```

**Binary**

```java
Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
messageAttributes.put("attributeName", new MessageAttributeValue().withDataType("Binary").withBinaryValue(ByteBuffer.wrap(new byte[10])));
```

The following three examples show how to use the optional custom type for the message attributes:

**String—Custom**

```java
Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
messageAttributes.put("AccountId", new MessageAttributeValue().withDataType("String.AccountId").withStringValue("000123456"));
```

**Number—Custom**

```java
Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
messageAttributes.put("AccountId", new MessageAttributeValue().withDataType("Number.AccountId").withStringValue("000123456"));
```

**Note**

Because the Type is a number, the result in the ReceiveMessage call is 123456.

**Binary—Custom**

```java
Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
messageAttributes.put("PhoneIcon", new MessageAttributeValue().withDataType("Binary.JPEG").withBinaryValue(ByteBuffer.wrap(new byte[10])));```
To send a message using one of the previous message attribute examples, your code should look similar to the following:

```java
SendMessageRequest request = new SendMessageRequest();
request.withMessageBody("A test message body.");
request.withQueueUrl("MyQueueUrlStringHere");
request.withMessageAttributes(messageAttributes);
sqs.sendMessage(request);
```

### Using Message Attributes with the Amazon SQS Query API

To specify message attributes with the query API, you call the `SendMessage`, `SendMessageBatch`, or `ReceiveMessage` actions.

**Note**

How you structure the `AUTHPARAMS` depends on how you sign your API request. For information about `AUTHPARAMS` in Signature Version 4, see [Examples of Signed Signature Version 4 Requests](#).

A query API request for this example looks similar to the following:

```plaintext
http://sqs.us-east-2.amazonaws.com/123456789012/MyQueue
...?
Action=SendMessage
&MessageBody=This+is+a+test+message
&MessageAttribute.1.Name=test_attribute_name_1
&MessageAttribute.1.Value.StringValue=test_attribute_value_1
&MessageAttribute.1.Value.DataType=String
&MessageAttribute.2.Name=test_attribute_name_2
&MessageAttribute.2.Value.StringValue=test_attribute_value_2
&MessageAttribute.2.Value.DataType=String
&Version=2012-11-05
&Expires=2014-05-05T22%3A52%3A43PST
&AUTHPARAMS
```

**Note**

Queue names and queue URLs are case-sensitive.

The query API response should look similar to the following:

```xml
HTTP/1.1 200 OK
...
<SendMessageResponse>
  <SendMessageResult>
    <MD5OfMessageBody>
      fafb00f5732ab283681e124bf8747ed1
    </MD5OfMessageBody>
    <MD5OfMessageAttributes>
      3ae8f24a165a8cedc005670c81a27295
    </MD5OfMessageAttributes>
    <MessageId>
      5fea7756-0ea4-451a-a703-a558b933e274
    </MessageId>
  </SendMessageResult>
</ResponseMetadata>
```
When using `SendMessageBatch`, the message attributes need to be specified on each individual message in the batch.

A query API request for this example looks similar to the following:

```
http://sqs.us-east-2.amazonaws.com/123456789012/MyQueue
...?Action=SendMessageBatch
&SendMessageBatchRequestEntry.1.Id=test_msg_001
&SendMessageBatchRequestEntry.1.MessageBody=test%20message%20body%201
&SendMessageBatchRequestEntry.2.Id=test_msg_002
&SendMessageBatchRequestEntry.2.MessageBody=test%20message%20body%202
&SendMessageBatchRequestEntry.2.DelaySeconds=60
&SendMessageBatchRequestEntry.2.MessageAttribute.1.Name=test_attribute_name_1
&SendMessageBatchRequestEntry.2.MessageAttribute.1.Value.StringValue=test_attribute_value_1
&SendMessageBatchRequestEntry.2.MessageAttribute.1.Value.DataType=String
&Version=2012-11-05
&Expires=2014-05-05T22%3A52%3A43PST
&AUTHPARAMS
```

The query API response should look similar to the following:

```
HTTP/1.1 200 OK
...
<SendMessageBatchResult>
<SendMessageBatchResultEntry>
  <Id>test_msg_001</Id>
  <MessageId>0a5231c7-8bff-4955-be2e-8dc7c50a25fa</MessageId>
  <MD5OfMessageBody>0e024d309850c78cba5eabbeff7cae71</MD5OfMessageBody>
</SendMessageBatchResultEntry>
<SendMessageBatchResultEntry>
  <Id>test_msg_002</Id>
  <MessageId>15ee1ed3-87e7-40c1-bdaa-2e49968ea7e9</MessageId>
  <MD5OfMessageBody>7fb8146a82f95e0af155278f406862c2</MD5OfMessageBody>
  <MD5OfMessageAttributes>295c5fa15a51aae6884d1d1d99ca50</MD5OfMessageAttributes>
</SendMessageBatchResultEntry>
</SendMessageBatchResult>
```

### MD5 Message-Digest Calculation

If you want to calculate the MD5 message digest for Amazon SQS message attributes and you're either using the query API or one of the AWS SDKs that does not support MD5 message digest for Amazon SQS message attributes, then you must use the following information about the algorithm to calculate the MD5 message digest of the message attributes.

**Note**

Currently the AWS SDK for Java supports MD5 message digest for Amazon SQS message attributes. This is available in the `MessageMD5ChecksumHandler` class. If you're using the SDK for Java, then you do not need to use the following information.

The high-level steps of the algorithm to calculate the MD5 message digest for Amazon SQS message attributes are:

1. Sort all message attributes by name in ascending order.
2. Encode the individual parts of each attribute (name, type, and value) into a buffer.
3. Compute the message digest of the entire buffer.

To encode a single Amazon SQS message attribute:
1. Encode the name (length of name [4 bytes] + UTF-8 bytes of the name).
2. Encode the type (length of type [4 bytes] + UTF-8 bytes of the type).
3. Encode the transport type (string or binary) of the value [1 byte].
   a. For the string transport type, encode 1.
   b. For the binary transport type, encode 2.

Note
The string and number logical data types use the string transport type. The binary logical data type uses the binary transport type.

4. Encode the attribute value.
   a. For a string transport type, encode the attribute value (length [4 bytes] + the UTF-8 bytes of the value).
   b. For a binary transport type, encode the attribute value (length [4 bytes] + use the raw bytes directly).

The following diagram shows the encoding of the MD5 message digest for a single message attribute:

Amazon SQS Long Polling

Long polling helps reduce your cost of using Amazon SQS by reducing the number of empty responses (when there are no messages available to return in reply to a ReceiveMessage request sent to an Amazon SQS queue) and eliminating false empty responses (when messages are available in the queue but aren’t included in the response):

- Long polling reduces the number of empty responses by allowing Amazon SQS to wait until a message is available in the queue before sending a response. Unless the connection times out, the response to the ReceiveMessage request contains at least one of the available messages, up to the maximum number of messages specified in the ReceiveMessage action.
• Long polling eliminates false empty responses by querying all (rather than a limited number) of the servers.
• Long polling returns messages as soon any message becomes available.

Topics
• The Differences Between Short and Long Polling (p. 74)
• Enabling Long Polling using the AWS Management Console (p. 74)
• Enabling Long Polling Using the API (p. 76)
• Enabling Long Polling Using the Query API (p. 77)

The Differences Between Short and Long Polling
Amazon SQS uses short polling by default, querying only a subset of the servers (based on a weighted random distribution) to determine whether any messages are available for inclusion in the response.

Short polling occurs when the WaitTimeSeconds parameter of a ReceiveMessage call is set to 0 in one of two ways:
• The ReceiveMessage call sets WaitTimeSeconds to 0.
• The ReceiveMessage call doesn't set WaitTimeSeconds and the queue attribute ReceiveMessageWaitTimeSeconds is set to 0.

Note
For the WaitTimeSeconds parameter of ReceiveMessage, a value set between 1 and 20 has priority over any value set for the queue attribute ReceiveMessageWaitTimeSeconds.

Enabling Long Polling using the AWS Management Console
You can enable long polling using the AWS Management Console by setting a Receive Message Wait Time to a value greater than 0.

To enable long polling with the AWS Management Console for a new queue
1. Sign in to the Amazon SQS console.
2. Select Create New Queue.
3. In the Create New Queue dialog box, type the Queue Name.
4. For **Receive Message Wait Time**, type a positive integer value, from 1 to 20 seconds.

5. Choose **Create Queue**.
You can use the AWS Management Console to change the **Receive Message Wait Time** setting for an existing queue.

**To set a new Receive Message Wait Time value for an existing queue**

1. Select a queue.
2. From the **Queue Actions** drop-down list, select **Configure Queue**.
3. For **Receive Message Wait Time**, type a positive integer value.
4. Choose **Save Changes**.

## Enabling Long Polling Using the API

The following table lists the API actions to use.

<table>
<thead>
<tr>
<th>Use this action</th>
<th>Use...</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReceiveMessage</td>
<td>WaitTimeSeconds parameter</td>
</tr>
<tr>
<td>CreateQueue</td>
<td>ReceiveMessageWaitTimeSeconds attribute</td>
</tr>
<tr>
<td>SetQueueAttributes</td>
<td>ReceiveMessageWaitTimeSeconds attribute</td>
</tr>
</tbody>
</table>
Important
If you decide to implement long polling with multiple queues, we recommend using one thread for each queue instead of trying to use a single thread for polling all of the queues. When you use one thread for each queue, your application can process the messages in each of the queues as they become available. A single thread for multiple queues might cause your application to become blocked from processing available messages in the other queues while waiting (up to 20 seconds) for a queue that doesn't have any available messages.

In most cases, when using long polling, set the timeout value to a maximum of 20 seconds. If the 20-second maximum doesn't work for your application, set a shorter timeout for long polling (the minimum is 1 second). If you don't use an AWS SDK to access Amazon SQS, or if you configure an AWS SDK to have a shorter timeout, you may need to modify your Amazon SQS client to allow for longer requests or to use a shorter timeout for long polling.

Enabling Long Polling Using the Query API

The following example enables long polling by calling the `ReceiveMessage` action with the `WaitTimeSeconds` parameter set to 10 seconds.

```
http://sqs.us-east-2.amazonaws.com/123456789012/testQueue/?Action=ReceiveMessage
&WaitTimeSeconds=10
&MaxNumberOfMessages=5
&VisibilityTimeout=15
&AttributeName=All;
&Version=2012-11-05
&Expires=2013-10-25T22%3A52%3A43PST
&AUTHPARAMS
```

The following example shows another way to enable long polling. Here, the `ReceiveMessageWaitTimeSeconds` attribute for the `SetQueueAttributes` action is set to 20 seconds.

```
http://sqs.us-east-2.amazonaws.com/123456789012/testQueue/?Action=SetQueueAttributes
&Attribute.Name=ReceiveMessageWaitTimeSeconds
&Attribute.Value=20
&Version=2012-11-05
&Expires=2013-10-25T22%3A52%3A43PST
&AUTHPARAMS
```

Amazon SQS Delay Queues

Topics
- Creating Delay Queues with the AWS Management Console (p. 78)
- Creating Delay Queues with the Query API (p. 80)

Delay queues let you postpone the delivery of new messages in a queue for the specified number of seconds. If you create a delay queue, any message that you send to that queue is invisible to consumers for the duration of the delay period. You can use the `CreateQueue` action to create a delay queue by setting the `DelaySeconds` attribute to any value between 0 and 900 (15 minutes). You can also
change an existing queue into a delay queue using the `SetQueueAttributes` action to set the queue's `DelaySeconds` attribute.

**Note**
For standard queues, the per-queue delay setting *isn't retroactive*: If you change the `DelaySeconds` attribute, it doesn't affect the delay of messages already in the queue.

For FIFO queues, the per-queue delay setting *is retroactive*: If you change the `DelaySeconds` attribute, it affects the delay of messages already in the queue.

Delay queues are similar to visibility timeouts because both features make messages unavailable to consumers for a specific period of time. The difference between delay queues and visibility timeouts is that for delay queues a message is hidden when it's first added to queue, whereas for visibility timeouts a message is hidden only after a message is consumed from the queue. The following figure illustrates the relationship between delay queues and visibility timeouts.

A message is considered to be *in flight* after it's received from a queue by a consumer, but not yet deleted from the queue.

For standard queues, there can be a maximum of 120,000 inflight messages per queue. If you reach this limit, Amazon SQS returns the `OverLimit` error message. To avoid reaching the limit, you should delete messages from the queue after they're processed. You can also increase the number of queues you use to process your messages.

For FIFO queues, there can be a maximum of 20,000 inflight messages per queue. If you reach this limit, Amazon SQS returns no error messages.

To set delay seconds on individual messages, rather than for an entire queue, use message timers. If you send a message with a message timer, Amazon SQS uses the message timer's delay seconds value instead of the delay queue's delay seconds value. For more information, see Amazon SQS Message Timers (p. 81).

Creating Delay Queues with the AWS Management Console

You can create a delay queue using the AWS Management Console by setting a `Delivery Delay` to a value greater than 0.

**To create a delay queue with the AWS Management Console**

1. Sign in to the Amazon SQS console.
2. Choose Create New Queue.
3. In the **Create New Queue** dialog box, type your **Queue Name**.

4. For **Delivery Delay**, type a positive integer value.

5. Choose **Create Queue**.
You can use the AWS Management Console to change the Delivery Delay setting for an existing queue by selecting the Configure Queue action with an existing queue selected.

**To set a new delivery delay value for an existing queue**

1. Select an existing queue and then from the Queue Actions drop-down box select Configure Queue.

2. Change the Delivery Delay value to a positive integer.

3. Choose Save Changes.

**Creating Delay Queues with the Query API**

The following Query API example calls the CreateQueue action to create a delay queue that hides each message from consumers for the first 45 seconds that the message is in the queue.

How you structure the AUTHPARAMS depends on how you're signing your API request. For information about AUTHPARAMS in Signature Version 4, see Examples of Signed Signature Version 4 Requests.

```
```
Amazon SQS Message Timers

Amazon SQS message timers allow you to specify an initial invisibility period for a message that you add to a queue. For example, if you send a message with the `DelaySeconds` parameter set to 45, the message isn't visible to consumers for the first 45 seconds during which the message stays in the queue. The default value for `DelaySeconds` is 0.

**Note**
FIFO queues don't support timers on individual messages.
A message is considered to be in flight after it's received from a queue by a consumer, but not yet deleted from the queue.

For standard queues, there can be a maximum of 120,000 inflight messages per queue. If you reach this limit, Amazon SQS returns the `OverLimit` error message. To avoid reaching the limit, you should delete messages from the queue after they're processed. You can also increase the number of queues you use to process your messages.

To set a delay period that applies to all messages in a queue, use delay queues (p. 77). A message timer setting for an individual message overrides any `DelaySeconds` value that applies to the entire delay queue.

**Topics**
- Creating Message Timers Using the Console (p. 81)
- Creating Message Timers Using the Query API (p. 83)

Creating Message Timers Using the Console

To send a message with a message timer using the AWS Management Console

1. Sign in to the Amazon SQS console.
2. Select a queue.
3. From the **Queue Actions** drop-down list, select **Send a Message**.

   **Note**
   The **Queue Actions** drop-down list is available only if a queue is selected.

4. In the **Send a Message to MyQueue** dialog box, type a message.

5. In the **Delay delivery of this message by** text box, enter a delay value (for example, 30).
Creating Message Timers Using the Query API

The following Query API example applies a 45-second initial visibility delay for a single message sent with SendMessage.
How you structure the AUTHPARAMS depends on how you're signing your API request. For information about AUTHPARAMS in Signature Version 4, see Examples of Signed Signature Version 4 Requests.

```text
http://sqs.us-east-2.amazonaws.com/123456789012/testQueue/
?Action=SendMessage
&MessageBody=This+is+a+test+message
&DelaySeconds=45
&Version=2012-11-05
&Expires=2015-12-18T22%3A52%3A43PST
&AUTHPARAMS
```

**Note**

Queue names and queue URLs are case-sensitive.

You can also use the Query API `SendMessageBatch` action to send up to 10 messages with message timers. You can assign a different `DelaySeconds` value to each message or assign no value at all. If you do not set a value for `DelaySeconds`, the message might still be subject to a delay if you're adding the message to a delay queue. For more information about delay queues, see Amazon SQS Delay Queues (p. 77). The following example uses `SendMessageBatch` to send three messages: one message without a message timer and two messages with different values for `DelaySeconds`.

```text
http://sqs.us-east-2.amazonaws.com/123456789012/testQueue/
?Action=SendMessageBatch
&SendMessageBatchRequestEntry.1.Id=test_msg_no_message_timer
&SendMessageBatchRequestEntry.1.MessageBody=test%20message%20body%201
&SendMessageBatchRequestEntry.2.Id=test_msg_delay_45_seconds
&SendMessageBatchRequestEntry.2.MessageBody=test%20message%20body%202
&SendMessageBatchRequestEntry.2.DelaySeconds=45
&SendMessageBatchRequestEntry.3.Id=test_msg_delay_2_minutes
&SendMessageBatchRequestEntry.3.MessageBody=test%20message%20body%203
&SendMessageBatchRequestEntry.3.DelaySeconds=120
&Version=2012-11-05
&Expires=2015-12-18T22%3A52%3A43PST
&AUTHPARAMS
```

Managing Large Amazon SQS Messages Using Amazon S3

You can manage Amazon SQS messages with Amazon S3. This is especially useful for storing and consuming messages with a message size of up to 2 GB. To manage Amazon SQS messages with Amazon S3, use the Amazon SQS Extended Client Library for Java. Specifically, you use this library to:

- Specify whether messages are always stored in Amazon S3 or only when a message's size exceeds 256 KB.
- Send a message that references a single message object stored in an Amazon S3 bucket.
- Get the corresponding message object from an Amazon S3 bucket.
- Delete the corresponding message object from an Amazon S3 bucket.

**Note**

You can use the Amazon SQS Extended Client Library for Java to manage Amazon SQS messages using Amazon S3. However, you can't do this using the AWS CLI, the Amazon SQS console, the Amazon SQS HTTP API, or any of the AWS SDKs—except for the SDK for Java.
Prerequisites

To manage Amazon SQS messages with Amazon S3, you need the following:

- **AWS SDK for Java** – There are two different ways to include the SDK for Java in your project. You can either download and install the SDK for Java, or if you use Maven to obtain the Amazon SQS Extended Client Library for Java, then the SDK for Java is included as a dependency. The SDK for Java and Amazon SQS Extended Client Library for Java require the J2SE Development Kit 7.0 or later. For information about downloading the SDK for Java, see SDK for Java. For more information about using Maven, see the note following this list.

- **Amazon SQS Extended Client Library for Java** – If you do not use Maven, then you must add the package file, `amazon-sqs-java-extended-client-lib.jar`, to the Java build class path. For information about downloading the library, see Amazon SQS Extended Client Library for Java.

- **Amazon S3 bucket** – You must create a new Amazon S3 bucket or use an existing bucket to store messages. We recommend that you create a new bucket for this purpose. To control bucket space and charges to your AWS account, you should also set a lifecycle configuration rule on the bucket to permanently delete message objects after a certain period of time following their creation date. For instructions, see Managing Lifecycle Configuration or the example (p. 85) following this section.

  **Note**
  The Amazon SQS Extended Client Library for Java includes support for Maven as follows:

  ```xml
  <dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>amazon-sqs-java-extended-client-lib</artifactId>
  <version>1.0.1</version>
  </dependency>
  ```

Using the Amazon SQS Extended Client Library for Java

After you have met the prerequisites (p. 85), use the following Java code example to get started managing Amazon SQS messages with Amazon S3.

This example creates an Amazon S3 bucket with a random name and adds a lifecycle rule to permanently delete objects after 14 days. It then creates a queue and sends to the queue a random message that is over 256 KB in size. The message is stored in the Amazon S3 bucket. The example then consumes the message and prints out information about the consumed message. The example then deletes the message, queue, and bucket.

```java
import java.util.Arrays;
import java.util.Iterator;
import java.util.List;
import java.util.UUID;
import com.amazon.sqs.javamessaging.AmazonSQSExtendedClient;
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClient;
import org.joda.time.DateTime;
import org.joda.time.format.DateTimeFormat;
import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.regions.Region;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
```
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.BucketLifecycleConfiguration;
import com.amazonaws.services.s3.model.ListVersionsRequest;
import com.amazonaws.services.s3.model.ObjectListing;
import com.amazonaws.services.s3.model.S3ObjectSummary;
import com.amazonaws.services.s3.model.S3VersionSummary;
import com.amazonaws.services.s3.model.VersionListing;
import com.amazonaws.services.sqs.model.CreateQueueRequest;
import com.amazonaws.services.sqs.model.DeleteMessageRequest;
import com.amazonaws.services.sqs.model.DeleteQueueRequest;
import com.amazonaws.services.sqs.model.Message;
import com.amazonaws.services.sqs.model.ReceiveMessageRequest;
import com.amazonaws.services.sqs.model.SendMessageRequest;
import com.amazon.sqs.javamessaging.ExtendedClientConfiguration;

public class SQSExtendedClientExample {
    private static final String s3BucketName = UUID.randomUUID() + "-" +
            DateTimeFormat.forPattern("yyMMdd-hhmmss").print(new Date());

    public static void main(String[] args) {
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException("Cannot load the AWS credentials from the expected AWS credential profiles file. " +
                    "Make sure that your credentials file is at the correct " +
                    "location (/home/$USER/.aws/credentials) and is in a valid format.", e);
        }

        AmazonS3 s3 = new AmazonS3Client(credentials);
        Region s3Region = Region.getRegion(Regions.US_WEST_2);
        s3.setRegion(s3Region);

        // Set the Amazon S3 bucket name, and set a lifecycle rule on the bucket to
        // permanently delete objects a certain number of days after
        // each object's creation date.
        // Then create the bucket, and enable message objects to be stored in the bucket.
        BucketLifecycleConfiguration.Rule expirationRule = new
                BucketLifecycleConfiguration.Rule();
        expirationRule.withExpirationInDays(14).withStatus("Enabled");
        BucketLifecycleConfiguration lifecycleConfig = new
                BucketLifecycleConfiguration().withRules(expirationRule);
        s3.createBucket(s3BucketName);
        s3.setBucketLifecycleConfiguration(s3BucketName, lifecycleConfig);
        System.out.println("Bucket created and configured.");

        // Set the SQS extended client configuration with large payload support enabled.
        ExtendedClientConfiguration extendedClientConfig = new ExtendedClientConfiguration()
                .withLargePayloadSupportEnabled(s3, s3BucketName);

        AmazonSQS sqsExtended = new AmazonSQSExtendedClient(new AmazonSQSClient(credentials),
                extendedClientConfig);
        Region sqsRegion = Region.getRegion(Regions.US_WEST_2);
        sqsExtended.setRegion(sqsRegion);

        // Create a long string of characters for the message object to be stored in the
        // bucket.
        int stringLength = 300000;
        char[] chars = new char[stringLength];
        Arrays.fill(chars, 'x');
        String myLongString = new String(chars);
    }
}
// Create a message queue for this example.
String QueueName = "QueueName" + UUID.randomUUID().toString();
CreateQueueRequest createQueueRequest = new CreateQueueRequest(QueueName);
String myQueueUrl = sqsExtended.createQueue(createQueueRequest).getQueueUrl();
System.out.println("Queue created.");

// Send the message.
SendMessageRequest myMessageRequest = new SendMessageRequest(myQueueUrl, myLongString);
sqsExtended.sendMessage(myMessageRequest);
System.out.println("Sent the message.");

// Receive messages, and then print general information about them.
ReceiveMessageRequest receiveMessageRequest = new ReceiveMessageRequest(myQueueUrl);
List<Message> messages = sqsExtended.receiveMessage(receiveMessageRequest).getMessages();
for (Message message : messages) {
    System.out.println("Message received:");
    System.out.println("  ID: " + message.getMessageId());
    System.out.println("  Receipt handle: " + message.getReceiptHandle());
    System.out.println("  Message body (first 5 characters): " + message.getBody().substring(0, 5));
}

// Delete the message, the queue, and the bucket.
String messageReceiptHandle = messages.get(0).getReceiptHandle();
sqsExtended.deleteMessage(new DeleteMessageRequest(myQueueUrl, messageReceiptHandle));
System.out.println("Deleted the message.");
sqsExtended.deleteQueue(new DeleteQueueRequest(myQueueUrl));
System.out.println("Deleted the queue.");
deleteBucketAndAllContents(s3);
System.out.println("Deleted the bucket.");

private static void deleteBucketAndAllContents(AmazonS3 client) {
    ObjectListing objectListing = client.listObjects(s3BucketName);
    while (true) {
        for (Iterator<?> iterator = objectListing.getObjectSummaries().iterator();
            iterator.hasNext(); ) {
            S3ObjectSummary objectSummary = (S3ObjectSummary) iterator.next();
            client.deleteObject(s3BucketName, objectSummary.getKey());
        }
        if (objectListing.isTruncated()) {
            objectListing = client.listNextBatchOfObjects(objectListing);
        } else {
            break;
        }
    }
    VersionListing list = client.listVersions(new ListVersionsRequest().withBucketName(s3BucketName));
    for (Iterator<?> iterator = list.getVersionSummaries().iterator();
        iterator.hasNext(); ) {
        S3VersionSummary s = (S3VersionSummary) iterator.next();
        client.deleteVersion(s3BucketName, s.getKey(), s.getVersionId());
    }
    client.deleteBucket(s3BucketName);
Using JMS with Amazon SQS

The Amazon SQS Java Messaging Library is a JMS interface for Amazon SQS that lets you take advantage of Amazon SQS in applications that already use JMS. The interface lets you use Amazon SQS as the JMS provider with minimal code changes. Together with the AWS SDK for Java, the Amazon SQS Java Messaging Library lets you create JMS connections and sessions, as well as producers and consumers that send and receive messages to and from Amazon SQS queues.

The library supports sending and receiving messages to a queue (the JMS point-to-point model) according to the JMS 1.1 specification. The library supports sending text, byte, or object messages synchronously to Amazon SQS queues. The library also supports receiving objects synchronously or asynchronously.

For information about features of the Amazon SQS Java Messaging Library that support the JMS 1.1 specification, see Supported JMS 1.1 Implementations (p. 107) and the Amazon SQS FAQs.

Topics
- Prerequisites (p. 88)
- Getting Started with the Amazon SQS Java Messaging Library (p. 89)
- Using the Amazon SQS Java Message Service (JMS) Client with Other Amazon SQS Clients (p. 94)
- Code Examples (p. 95)
- Supported JMS 1.1 Implementations (p. 107)

Prerequisites

Before you begin, you must have the following prerequisites:

- **SDK for Java**

  There are two ways to include the SDK for Java in your project:
  - Download and install the SDK for Java.
  - Use Maven to get the Amazon SQS Java Messaging Library. (The SDK for Java is included as a dependency. The SDK for Java and Amazon SQS Java Messaging Library require J2SE Development Kit 7.0 or later.)

    For information about downloading the SDK for Java, see SDK for Java.

- **Amazon SQS Java Messaging Library**

  If you do not use Maven, you must add the package file `amazon-sqs-java-messaging-lib.jar` to the Java build class path. For information about downloading the library, see Amazon SQS Java Messaging Library.

    **Note**
    The Amazon SQS Java Messaging Library includes support for Maven and the Spring Framework.
    For code samples that use Maven, the Spring Framework, and the Amazon SQS Java Messaging Library, see Code Examples (p. 95).
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>amazon-sqs-java-messaging-lib</artifactId>
  <version>1.0.4</version>
  <type>jar</type>
</dependency>

- **Amazon SQS Queue**

  Create a queue using the AWS Management Console for Amazon SQS, the `CreateQueue` API, or the wrapped Amazon SQS client included in the Amazon SQS Java Messaging Library.
  
  - For information about creating a queue with Amazon SQS using either the AWS Management Console or the `CreateQueue` API, see Creating a Queue (p. 15).
  - For information about using the Amazon SQS Java Messaging Library, see Getting Started with the Amazon SQS Java Messaging Library (p. 89).

**Getting Started with the Amazon SQS Java Messaging Library**

To get started using JMS with Amazon SQS, use the code examples in this section. The following sections show how to create a JMS connection and a session, and how to send and receive a message.

The wrapped Amazon SQS client object included in the Amazon SQS Java Messaging Library checks if an Amazon SQS queue exists. If the queue does not exist, the client creates it.

**Creating a JMS Connection**

1. Create a connection factory and call the `createConnection` method against the factory.

   **Note**
   The `EnvironmentVariableCredentialsProvider` class in the following example assumes that the `AWS_ACCESS_KEY_ID` (or `AWS_ACCESS_KEY`) and `AWS_SECRET_KEY` (or `AWS_SECRET_ACCESS_KEY`) environment variables are set.
   For more information about providing the required credentials to the factory, see `Interface AWSCredentialsProvider`.

   ```java
   // Create the connection factory using the environment variable credential provider.
   // Connections this factory creates can talk to the queues in us-east-2 region.
   SQSConnectionFactory connectionFactory = new SQSConnectionFactory(  
     new ProviderConfiguration(),  
     AmazonSQSClientBuilder.standard()  
       .withRegion(Regions.US_EAST_2)  
       .withCredentials(new EnvironmentVariableCredentialsProvider())  
   );
   // Create the connection.
   SQSConnection connection = connectionFactory.createConnection();
   ```

   The `SQSConnection` class extends `javax.jms.Connection`. Together with the JMS standard connection methods, `SQSConnection` offers additional methods, such as `getAmazonSQSClient` and `getWrappedAmazonSQSClient`. Both methods let you perform administrative operations not included in the JMS specification, such as creating new queues. However, the `getWrappedAmazonSQSClient` method also provides a wrapped version of the Amazon SQS client used by the current connection. The wrapper transforms every exception from the client into an
JMSException, allowing it to be more easily used by existing code that expects JMSException occurrences.

2. You can use the client objects returned from `getAmazonSQSClient` and `getWrappedAmazonSQSClient` to perform administrative operations not included in the JMS specification (for example, you can create an Amazon SQS queue).

   If you have existing code that expects JMS exceptions, then you should use `getWrappedAmazonSQSClient`:
   - If you use `getWrappedAmazonSQSClient`, the returned client object transforms all exceptions into JMS exceptions.
   - If you use `getAmazonSQSClient`, the exceptions are all Amazon SQS exceptions.

## Creating an Amazon SQS Queue

The wrapped client object checks if an Amazon SQS queue exists.

If a queue does not exist, the client creates it. If the queue does exist, the function does not return anything. For more information, see the "Create the queue if needed" section in the `TextMessageSender.java (p. 97)` example.

### To create a standard queue

```java
// Get the wrapped client
AmazonSQSMessagingClientWrapper client = connection.getWrappedAmazonSQSClient();

// Create an SQS queue named TestQueue, if it does not already exist
if (!client.queueExists("TestQueue")) {
    client.createQueue("TestQueue");
}
```

### To create a FIFO queue

```java
// Get the wrapped client
AmazonSQSMessagingClientWrapper client = connection.getWrappedAmazonSQSClient();

// Create an Amazon SQS FIFO queue named TestQueue.fifo, if it does not already exist
if (!client.queueExists("TestQueue.fifo")) {
    Map<String, String> attributes = new HashMap<String, String>();
    attributes.put("FifoQueue", "true");
    attributes.put("ContentBasedDeduplication", "true");
    client.createQueue(new CreateQueueRequest().withQueueName("TestQueue.fifo").withAttributes(attributes));
}
```

**Note**

The name of a FIFO queue must end with the `.fifo` suffix.

For more information on the `ContentBasedDeduplication` attribute, see Exactly-Once Processing (p. 53).

## Sending Messages Synchronously

1. When the connection and the underlying Amazon SQS queue are ready, create a nontransacted JMS session with AUTO_ACKNOWLEDGE mode.

   ```java
   // Create the nontransacted session with AUTO_ACKNOWLEDGE mode
   ```
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

2. To send a text message to the queue, create a JMS queue identity and a message producer.

// Create a queue identity and specify the queue name to the session
Queue queue = session.createQueue("TestQueue");
// Create a producer for the 'TestQueue'
MessageProducer producer = session.createProducer(queue);

3. Create a text message and send it to the queue.
   • To send a message to a standard queue, you don't need to set any additional parameters.

   // Create the text message
   TextMessage message = session.createTextMessage("Hello World!");
   // Send the message
   producer.send(message);
   System.out.println("JMS Message " + message.getJMSMessageID());

   • To send a message to a FIFO queue, you must set the message group ID. You can also set a
     message deduplication ID. For more information, see Key Terms (p. 52).

   // Create the text message
   TextMessage message = session.createTextMessage("Hello World!");
   // Set the message group ID
   message.setStringProperty("JMSXGroupID", "Default");
   // You can also set a custom message deduplication ID
   message.setStringProperty("JMS_SQS_DeduplicationId", "hello");
   // Here, it's not needed because content-based deduplication is enabled for the queue
   // Send the message
   producer.send(message);
   System.out.println("JMS Message " + message.getJMSMessageID());
   System.out.println("JMS Message Sequence Number " +
                      message.getStringProperty("JMS_SQS_SequenceNumber"));

Receiving Messages Synchronously

1. To receive messages, create a consumer for the same queue and invoke the start method.

   You can call the start method on the connection at any time. However, the consumer does not
   begin to receive messages until you call it.

   // Create a consumer for the 'TestQueue'
   MessageConsumer consumer = session.createConsumer(queue);
   // Start receiving incoming messages
   connection.start();

2. Call the receive method on the consumer with a timeout set to 1 second, and then print the
   contents of the received message.

   • After receiving a message from a standard queue, you can access the contents of the message.

   // Receive a message from 'TestQueue' and wait up to 1 second
   Message receivedMessage = consumer.receive(1000);
After receiving a message from a FIFO queue, you can access the contents of the message and other, FIFO-specific message attributes, such as the message group ID, message deduplication ID, and sequence number. For more information, see Key Terms (p. 52).

```java
// Receive a message from 'TestQueue' and wait up to 1 second
Message receivedMessage = consumer.receive(1000);

// Cast the received message as TextMessage and display the text
if (receivedMessage != null) {
    System.out.println("Received: " + ((TextMessage) receivedMessage).getText());
    System.out.println("Group id: " + receivedMessage.getStringProperty("JMSXGroupID"));
    System.out.println("Message deduplication id: " + receivedMessage.getStringProperty("JMS_SQS_DeduplicationId"));
    System.out.println("Message sequence number: " + receivedMessage.getStringProperty("JMS_SQS_SequenceNumber"));
}
```

3. Close the connection and the session.

```java
// Close the connection (and the session).
connection.close();
```

The output looks similar to the following:

```
JMS Message ID:8example-588b-44e5-bbcf-d816example2
Received: Hello World!
```

**Note**
You can use the Spring Framework to initialize these objects.
For additional information, see SpringExampleConfiguration.xml, SpringExample.java, and the other helper classes in ExampleConfiguration.java and ExampleCommon.java in the Code Examples (p. 95) section.

For complete examples of sending and receiving objects, see TextMessageSender.java (p. 97) and SyncMessageReceiver.java (p. 98).

### Receiving Messages Asynchronously

In the example in Getting Started with the Amazon SQS Java Messaging Library (p. 89), a message is sent to TestQueue and received synchronously.

The following example shows how to receive the messages asynchronously through a listener.

1. Implement the MessageListener interface.

```java
class MyListener implements MessageListener {
    @Override
    public void onMessage(Message message) {
        try {
            // Cast the received message as TextMessage and print the text to screen.
```
The `onMessage` method of the `MessageListener` interface is called when you receive a message. In this listener implementation, the text stored in the message is printed.

2. Instead of explicitly calling the `receive` method on the consumer, set the message listener of the consumer to an instance of the `MyListener` implementation. The main thread waits for one second.

```java
// Create a consumer for the 'TestQueue'.
MessageConsumer consumer = session.createConsumer(queue);

// Instantiate and set the message listener for the consumer.
consumer.setMessageListener(new MyListener());

// Start receiving incoming messages.
connection.start();

// Wait for 1 second. The listener `onMessage()` method is invoked when a message is received.
Thread.sleep(1000);
```

The rest of the steps are identical to the ones in the Getting Started with the Amazon SQS Java Messaging Library (p. 89) example. For a complete example of an asynchronous consumer, see `AsyncMessageReceiver.java` in Code Examples (p. 95).

The output for this example looks similar to the following:

```
JMS Message ID:8example-588b-44e5-bbcf-d816example2
Received: Hello World!
```

### Using Client Acknowledge Mode

The example in Getting Started with the Amazon SQS Java Messaging Library (p. 89) uses `AUTO_ACKNOWLEDGE` mode where every received message is acknowledged automatically (and therefore deleted from the underlying Amazon SQS queue).

1. To explicitly acknowledge the messages after they're processed, you must create the session with `CLIENT_ACKNOWLEDGE` mode.

```java
// Create the non-transacted session with `CLIENT_ACKNOWLEDGE` mode.
Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
```

2. When the message is received, display it and then explicitly acknowledge it.

```java
// Cast the received message as `TextMessage` and print the text to screen. Also acknowledge the message.
if (receivedMessage != null) {
    System.out.println("Received: " + ((TextMessage) receivedMessage).getText());
    receivedMessage.acknowledge();
    System.out.println("Acknowledged: " + message.getJMSMessageID());
}
```
Note
In this mode, when a message is acknowledged, all messages received before this message are implicitly acknowledged as well. For example, if 10 messages are received, and only the 10th message is acknowledged (in the order the messages are received), then all of the previous nine messages are also acknowledged.

The rest of the steps are identical to the ones in the Getting Started with the Amazon SQS Java Messaging Library (p. 89) example. For a complete example of a synchronous consumer with client acknowledge mode, see SyncMessageReceiverClientAcknowledge.java in Code Examples (p. 95).

The output for this example looks similar to the following:

| JMS Message ID:4example-aa0e-403f-b6df-5e02example5 |
| Received: Hello World! |
| Acknowledged: ID:4example-aa0e-403f-b6df-5e02example5 |

Using Unordered Acknowledge Mode

When using CLIENT_ACKNOWLEDGE mode, all messages received before an explicitly-acknowledged message are acknowledged automatically. For more information, see Using Client Acknowledge Mode (p. 93).

The Amazon SQS Java Messaging Library provides another acknowledgement mode. When using UNORDERED_ACKNOWLEDGE mode, all received messages must be individually and explicitly acknowledged by the client, regardless of their reception order. To do this, create a session with UNORDERED_ACKNOWLEDGE mode.

```java
// Create the non-transacted session with UNORDERED_ACKNOWLEDGE mode.
Session session = connection.createSession(false, SQSSession.UNORDERED_ACKNOWLEDGE);
```

The remaining steps are identical to the ones in the Using Client Acknowledge Mode (p. 93) example. For a complete example of a synchronous consumer with UNORDERED_ACKNOWLEDGE mode, see SyncMessageReceiverUnorderedAcknowledge.java.

In this example, the output looks similar to the following:

| JMS Message ID:dexample-73ad-4adb-bc6c-4357example7 |
| Received: Hello World! |
| Acknowledged: ID:dexample-73ad-4adb-bc6c-4357example7 |

Using the Amazon SQS Java Message Service (JMS) Client with Other Amazon SQS Clients

Using the Amazon SQS Java Message Service (JMS) Client with the AWS SDK limits Amazon SQS message size to 256 KB. However, you can create a JMS provider using any Amazon SQS client. For example, you can use the JMS Client with the Amazon SQS Extended Client Library for Java to send an Amazon SQS message that contains a reference to a message payload (up to 2 GB) in Amazon S3. For more information, see Managing Large Amazon SQS Messages Using Amazon S3 (p. 84).

The following Java code example creates the JMS provider for the Extended Client Library:

```java
AmazonS3 s3 = new AmazonS3Client(creds);
```
Region s3Region = Region.getRegion(Regions.US_WEST_2);
s3.setRegion(s3Region);

// Set the Amazon S3 bucket name, and set a lifecycle rule on the bucket to
// permanently delete objects a certain number of days after each object's creation date.
// Next, create the bucket, and enable message objects to be stored in the bucket.
BucketLifecycleConfiguration.Rule expirationRule = new BucketLifecycleConfiguration.Rule();
expirationRule.withExpirationInDays(14).withStatus("Enabled");
BucketLifecycleConfiguration lifecycleConfig = new
BucketLifecycleConfiguration().withRules(expirationRule);
s3.createBucket(s3BucketName);
s3.setBucketLifecycleConfiguration(s3BucketName, lifecycleConfig);
System.out.println("Bucket created and configured.");

// Set the SQS extended client configuration with large payload support enabled.
ExtendedClientConfiguration extendedClientConfig = new ExtendedClientConfiguration()
     .withLargePayloadSupportEnabled(s3, s3BucketName);
AmazonSQS sqsExtended = new AmazonSQSExtendedClient(new AmazonSQSClient(credentials),
     extendedClientConfig);
Region sqsRegion = Region.getRegion(Regions.US_WEST_2);
sqsExtended.setRegion(sqsRegion);

The following Java code example creates the connection factory:

// Create the connection factory using the environment variable credential provider.
// Pass the configured Amazon SQS Extended Client to the JMS connection factory.
SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
     new ProviderConfiguration(),
     sqsExtended
);

// Create the connection.
SQSConnection connection = connectionFactory.createConnection();

**Code Examples**

The following code examples show how to use JMS with Amazon SQS standard queues. For more
information about working with FIFO queues, see To create a FIFO queue (p. 90), Sending Messages
Synchronously (p. 90), and Receiving Messages Synchronously (p. 91). (Receiving messages
synchronously is the same for standard and FIFO queues. However, messages in FIFO queues contain
more attributes.)

**ExampleConfiguration.java**

The following Java code example sets the default queue name, the region, and the credentials to be used
with the other Java examples.

```java
public class ExampleConfiguration {
    public static final String DEFAULT_QUEUE_NAME = "SQSJMSClientExampleQueue";
    public static final Region DEFAULT_REGION = Region.getRegion(Regions.US_EAST_2);
    private static String getParameter( String args[], int i ) {
        if( i + 1 >= args.length ) {
            throw new IllegalArgumentException( "Missing parameter for " + args[i] );
        }
        return args[i+1];
    }
}
```
/**
 * Parse the command line and return the resulting config. If the config parsing fails
 * print the error and the usage message and then call System.exit
 *
 * @param app the app to use when printing the usage string
 * @param args the command line arguments
 * @return the parsed config
 */

private ExampleConfiguration(String args[]) {
    for( int i = 0; i < args.length; ++i ) {
        String arg = args[i];
        if( arg.equals( "--queue" ) ) {
            setQueueName(getParameter(args, i));
            i++;
        } else if( arg.equals( "--region" ) ) {
            String regionName = getParameter(args, i);
            try {
                setRegion(Region.getRegion(Regions.fromName(regionName)));
            } catch( IllegalArgumentException e ) {
                throw new IllegalArgumentException("Unrecognized region " +
                    regionName);
            }
            i++;
        } else if( arg.equals( "--credentials" ) ) {
            String credsFile = getParameter(args, i);
            try {
                setCredentialsProvider( new PropertiesFileCredentialsProvider(credsFile) );
            } catch (AmazonClientException e) {
                throw new IllegalArgumentException("Error reading credentials from " +
                    credsFile, e );
            }
            i++;
        } else {
            throw new IllegalArgumentException("Unrecognized option " + arg);
        }
    }
}

private String queueName = DEFAULT_QUEUE_NAME;
private Region region = DEFAULT_REGION;
private AWSCredentialsProvider credentialsProvider = new
DefaultAWSCredentialsProviderChain();

public String getQueueName() {
    return queueName;
}

public void setQueueName(String queueName) {
    this.queueName = queueName;
}
public Region getRegion() {
    return region;
}

public void setRegion(Region region) {
    this.region = region;
}

public AWSCredentialsProvider getCredentialsProvider() {
    return credentialsProvider;
}

public void setCredentialsProvider(AWSCredentialsProvider credentialsProvider) {
    // Make sure they're usable first
    credentialsProvider.getCredentials();
    this.credentialProvider = credentialsProvider;
}

TextMessageSender.java
The following Java code example creates a text message producer.

```java
public class TextMessageSender {
    public static void main(String args[]) throws JMSException {
        ExampleConfiguration config = ExampleConfiguration.parseConfig("TextMessageSender", args);
        ExampleCommon.setupLogging();

        // Create the connection factory based on the config
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
            new ProviderConfiguration(),
            AmazonSQSClientBuilder.standard()
                .withRegion(config.getRegion().getName())
                .withCredentials(config.getCredentialsProvider())
        );

        // Create the connection
        SQSConnection connection = connectionFactory.createConnection();

        // Create the queue if needed
        ExampleCommon.ensureQueueExists(connection, config.getQueueName());

        // Create the session
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        MessageProducer producer = session.createProducer( session.createQueue( config.getQueueName() ) );

        sendMessages(session, producer);

        // Close the connection. This closes the session automatically
        connection.close();
        System.out.println( "Connection closed" );
    }

    private static void sendMessages( Session session, MessageProducer producer ) {
        BufferedReader inputReader = new BufferedReader(
            new InputStreamReader( System.in, Charset.defaultCharset() )
        );

        try {
            String input;
            while( true ) {
```
System.out.print( "Enter message to send (leave empty to exit): " );
input = inputReader.readLine();
if( input == null || input.equals("" ) ) break;

TextMessage message = session.createTextMessage(input);
producer.send(message);
System.out.println( "Send message " + message.getJMSMessageID() );
}
} catch (EOFException e) {
    // Just return on EOF
} catch (IOException e) {
    System.err.println( "Failed reading input: " + e.getMessage() );
} catch (JMSException e) {
    System.err.println( "Failed sending message: " + e.getMessage() );
e.printStackTrace();
}
}

SyncMessageReceiver.java

The following Java code example creates a synchronous message consumer.

```java
public class SyncMessageReceiver {
    public static void main(String args[]) throws JMSException {
        ExampleConfiguration config = ExampleConfiguration.parseConfig("SyncMessageReceiver", args);

        ExampleCommon.setupLogging();

        // Create the connection factory based on the config
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
            new ProviderConfiguration(),
            AmazonSQSClientBuilder.standard()
                .withRegion(config.getRegion().getName())
                .withCredentials(config.getCredentialsProvider())
            );

        // Create the connection
        SQSConnection connection = connectionFactory.createConnection();

        // Create the queue if needed
        ExampleCommon.ensureQueueExists(connection, config.getQueueName());

        // Create the session
        Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
        MessageConsumer consumer = session.createConsumer( session.createQueue( config.getQueueName() ) );

        connection.start();
        receiveMessages(session, consumer);

        // Close the connection. This closes the session automatically
        connection.close();
    }

    private static void receiveMessages( Session session, MessageConsumer consumer ) {
        try {
            while( true ) {
                System.out.println( "Waiting for messages" );
                // Wait 1 minute for a message
                Message message = consumer.receive(TimeUnit.MINUTES.toMillis(1));
```
The following Java code example creates an asynchronous message consumer.

```java
public class AsyncMessageReceiver {
    public static void main(String[] args) throws JMSException, InterruptedException {
        ExampleConfiguration config = ExampleConfiguration.parseConfig("AsyncMessageReceiver", args);

        ExampleCommon.setupLogging();

        // Create the connection factory based on the config
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(new ProviderConfiguration(),
                AmazonSQSClientBuilder.standard()
                .withRegion(config.getRegion().getName())
                .withCredentials(config.getCredentialsProvider())
        );

        // Create the connection
        SQSConnection connection = connectionFactory.createConnection();

        // Create the queue if needed
        ExampleCommon.ensureQueueExists(connection, config.getQueueName());

        // Create the session
        Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
        MessageConsumer consumer = session.createConsumer( session.createQueue( config.getQueueName() ) );

        ReceiverCallback callback = new ReceiverCallback();
        consumer.setMessageListener( callback );

        // No messages are processed until this is called
        connection.start();

        callback.waitForOneMinuteOfSilence();
        System.out.println( "Returning after one minute of silence" );

        // Close the connection. This closes the session automatically
        connection.close();
        System.out.println( "Connection closed" );
    }

    private static class ReceiverCallback implements MessageListener {
        // Used to listen for message silence
        private volatile long timeOfLastMessage = System.nanoTime();
    }
}
```
public void waitForOneMinuteOfSilence() throws InterruptedException {
    for(;;) {
        long timeSinceLastMessage = System.nanoTime() - timeOfLastMessage;
        long remainingTillOneMinuteOfSilence =
            TimeUnit.MINUTES.toNanos(1) - timeSinceLastMessage;
        if( remainingTillOneMinuteOfSilence < 0 ) {
            break;
        }
        TimeUnit.NANOSECONDS.sleep(remainingTillOneMinuteOfSilence);
    }
}

@override
public void onMessage(Message message) {
    try {
        ExampleCommon.handleMessage(message);
        message.acknowledge();
        System.out.println( "Acknowledged message " + message.getJMSMessageID() );
        timeOfLastMessage = System.nanoTime();
    } catch (JMSException e) {
        System.err.println( "Error processing message: " + e.getMessage() );
        e.printStackTrace();
    }
}
}

SyncMessageReceiverClientAcknowledge.java

The following Java code example creates a synchronous consumer with client acknowledge mode.

/*
 * An example class to demonstrate the behavior of CLIENT_ACKNOWLEDGE mode for received messages. This example
 * complements the example given in SyncMessageReceiverUnorderedAcknowledge for UNORDERED_ACKNOWLEDGE mode.
 * First, a session, a message producer, and a message consumer are created. Then, two messages are sent. Next, two messages
 * are received but only the second one is acknowledged. After waiting for the visibility time out period, an attempt to
 * receive another message is made. It's shown that no message is returned for this attempt since in CLIENT_ACKNOWLEDGE mode,
 * as expected, all the messages prior to the acknowledged messages are also acknowledged.
 * This ISN'T the behavior for UNORDERED_ACKNOWLEDGE mode. Please see SyncMessageReceiverUnorderedAcknowledge
 * for an example.
 */
public class SyncMessageReceiverClientAcknowledge {

    // Visibility time-out for the queue. It must match to the one set for the queue for this example to work.
    private static final long TIME_OUT_SECONDS = 1;

    public static void main(String args[]) throws JMSException, InterruptedException {
        // Create the configuration for the example
        ExampleConfiguration config = ExampleConfiguration.parseConfig("SyncMessageReceiverClientAcknowledge", args);

        // Setup logging for the example
        ExampleCommon.setupLogging();
    }
}
// Create the connection factory based on the config
SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
    new ProviderConfiguration(),
    AmazonSQSClientBuilder.standard()
        .withRegion(config.getRegion().getName())
        .withCredentials(config.getCredentialsProvider())
);

// Create the connection
SQSConnection connection = connectionFactory.createConnection();

// Create the queue if needed
ExampleCommon.ensureQueueExists(connection, config.getQueueName());

// Create the session with client acknowledge mode
Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);

// Create the producer and consume
MessageProducer producer = session.createProducer(session.createQueue(config.getQueueName()));
MessageConsumer consumer = session.createConsumer(session.createQueue(config.getQueueName()));

// Open the connection
connection.start();

// Send two text messages
sendMessage(producer, session, "Message 1");
sendMessage(producer, session, "Message 2");

// Receive a message and don’t acknowledge it
receiveMessage(consumer, false);

// Receive another message and acknowledge it
receiveMessage(consumer, true);

// Wait for the visibility time out, so that unacknowledged messages reappear in the queue
System.out.println("Waiting for visibility timeout...");
Thread.sleep(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));

// Attempt to receive another message and acknowledge it. This results in receiving no messages since
// we have acknowledged the second message. Although we did not explicitly acknowledge the first message,
// in the CLIENT_ACKNOWLEDGE mode, all the messages received prior to the explicitly acknowledged message
// are also acknowledged. Therefore, we have implicitly acknowledged the first message.
receiveMessage(consumer, true);

// Close the connection. This closes the session automatically
connection.close();
System.out.println("Connection closed.");
}

/**
 * Sends a message through the producer.
 * @param producer Message producer
 * @param session Session
 * @param messageText Text for the message to be sent
 * @throws JMSException
 */
private static void sendMessage(MessageProducer producer, Session session, String messageText) throws JMSException {

// Create the connection factory based on the config
SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
    new ProviderConfiguration(),
    AmazonSQSClientBuilder.standard()
        .withRegion(config.getRegion().getName())
        .withCredentials(config.getCredentialsProvider())
);

// Create the connection
SQSConnection connection = connectionFactory.createConnection();

// Create the queue if needed
ExampleCommon.ensureQueueExists(connection, config.getQueueName());

// Create the session with client acknowledge mode
Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);

// Create the producer and consume
MessageProducer producer = session.createProducer(session.createQueue(config.getQueueName()));
MessageConsumer consumer = session.createConsumer(session.createQueue(config.getQueueName()));

// Open the connection
connection.start();

// Send two text messages
sendMessage(producer, session, "Message 1");
sendMessage(producer, session, "Message 2");

// Receive a message and don’t acknowledge it
receiveMessage(consumer, false);

// Receive another message and acknowledge it
receiveMessage(consumer, true);

// Wait for the visibility time out, so that unacknowledged messages reappear in the queue
System.out.println("Waiting for visibility timeout...");
Thread.sleep(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));

// Attempt to receive another message and acknowledge it. This results in receiving no messages since
// we have acknowledged the second message. Although we did not explicitly acknowledge the first message,
// in the CLIENT_ACKNOWLEDGE mode, all the messages received prior to the explicitly acknowledged message
// are also acknowledged. Therefore, we have implicitly acknowledged the first message.
receiveMessage(consumer, true);

// Close the connection. This closes the session automatically
connection.close();
System.out.println("Connection closed.");
}
// Create a text message and send it
producer.send(session.createTextMessage(messageText));
}

/**
 * Receives a message through the consumer synchronously with the default timeout
 * (TIME_OUT_SECONDS).
 * If a message is received, the message is printed. If no message is received, "Queue
 * is empty!" is
 * printed.
 * @param consumer Message consumer
 * @param acknowledge If true and a message is received, the received message is
 * acknowledged.
 * @throws JMSException
 */
private static void receiveMessage(MessageConsumer consumer, boolean acknowledge)
throws JMSException {
    // Receive a message
    Message message = consumer.receive(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));
    if (message == null) {
        System.out.println("Queue is empty!");
    } else {
        // Since this queue has only text messages, cast the message object and print
        the
text
        System.out.println("Received: " + ((TextMessage) message).getText());
        // Acknowledge the message if asked
        if (acknowledge) message.acknowledge();
    }
}

SyncMessageReceiverUnorderedAcknowledge.java

The following Java code example creates a synchronous consumer with unordered acknowledge mode.

/**
 * An example class to demonstrate the behavior of UNORDERED_ACKNOWLEDGE mode for received
 * messages. This example
 * complements the example given in @link SyncMessageReceiverClientAcknowledge} for
 * CLIENT_ACKNOWLEDGE mode.
 * *
 * First, a session, a message producer, and a message consumer are created. Then, two
 * messages are sent. Next, two messages
 * are received but only the second one is acknowledged. After waiting for the visibility
 * time out period, an attempt to
 * receive another message is made. It's shown that the first message received in the prior
 * attempt is returned again
 * for the second attempt. In UNORDERED_ACKNOWLEDGE mode, all the messages must be
 * explicitly acknowledged no matter what
 * the order they're received.
 * *
 * This ISN'T the behavior for CLIENT_ACKNOWLEDGE mode. Please see @link
 * SyncMessageReceiverClientAcknowledge
 * for an example.
 */
public class SyncMessageReceiverUnorderedAcknowledge {

    // Visibility time-out for the queue. It must match to the one set for the queue for
    this example to work.
    private static final long TIME_OUT_SECONDS = 1;
public static void main(String args[]) throws JMSException, InterruptedException {
    // Create the configuration for the example
    ExampleConfiguration config =
        ExampleConfiguration.parseConfig("SyncMessageReceiverUnorderedAcknowledge", args);

    // Setup logging for the example
    ExampleCommon.setupLogging();

    // Create the connection factory based on the config
    SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
        new ProviderConfiguration(),
        AmazonSQSClientBuilder.standard()
            .withRegion(config.getRegion().getName())
            .withCredentials(config.getCredentialsProvider())
    );

    // Create the connection
    SQSConnection connection = connectionFactory.createConnection();

    // Create the queue if needed
    ExampleCommon.ensureQueueExists(connection, config.getQueueName());

    // Create the session with unordered acknowledge mode
    Session session = connection.createSession(false, SQSSession.UNORDERED_ACKNOWLEDGE);

    // Create the producer and consume
    MessageProducer producer =
        session.createProducer(session.createQueue(config.getQueueName()));
    MessageConsumer consumer =
        session.createConsumer(session.createQueue(config.getQueueName()));

    // Open the connection
    connection.start();

    // Send two text messages
    sendMessage(producer, session, "Message 1");
    sendMessage(producer, session, "Message 2");

    // Receive a message and don't acknowledge it
    receiveMessage(consumer, false);

    // Receive another message and acknowledge it
    receiveMessage(consumer, true);

    // Wait for the visibility time out, so that unacknowledged messages reappear in the queue
    System.out.println("Waiting for visibility timeout...");
    Thread.sleep(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));

    // Attempt to receive another message and acknowledge it. This results in receiving the first message since
    // we have acknowledged only the second message. In the UNORDERED_ACKNOWLEDGE mode,
    // all the messages must
    // be explicitly acknowledged.
    receiveMessage(consumer, true);

    // Close the connection. This closes the session automatically
    connection.close();
    System.out.println("Connection closed.");
}

/**
 * Sends a message through the producer.
 * @param producer Message producer
 */
private static void sendMessage(MessageProducer producer, Session session, String
messageText) throws JMSException {
    // Create a text message and send it
    producer.send(session.createTextMessage(messageText));
}

/**
* Receives a message through the consumer synchronously with the default timeout
*(TIME_OUT_SECONDS).
* If a message is received, the message is printed. If no message is received, "Queue
* is empty!" is
* printed.
*
* @param consumer Message consumer
* @param acknowledge If true and a message is received, the received message is
* acknowledged.
* @throws JMSException
*/
private static void receiveMessage(MessageConsumer consumer, boolean acknowledge)
throws JMSException {
    // Receive a message
    Message message = consumer.receive(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));
    if (message == null) {
        System.out.println("Queue is empty!");
    } else {
        // Since this queue has only text messages, cast the message object and print
        the text
        System.out.println("Received: "+((TextMessage) message).getText());
        // Acknowledge the message if asked
        if (acknowledge) message.acknowledge();
    }
}

SpringExampleConfiguration.xml

The following XML code example is a bean configuration file for SpringExample.java (p. 105).

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:util="http://www.springframework.org/schema/util"
    xmlns:p="http://www.springframework.org/schema/p"
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/util
    http://www.springframework.org/schema/util/spring-util-3.0.xsd"
    xmlns:service="http://www.amazonaws.org/schema/servicex"
    xmlns:aws="http://www.amazonaws.org/schema/aws"
    xmlns:aws-ecs="http://www.amazonaws-ecs.org/schema/aws-ecs"
>
    <bean id="CredentialsProviderBean"
        class="com.amazonaws.auth.DefaultAWSCredentialsProviderChain"/>

    <bean id="ClientBuilder" class="com.amazonaws.services.sqs.AmazonSQSClientBuilder"
        factory-method="standard">
        <property name="region" value="us-east-2"/>
        <property name="credentials" ref="CredentialsProviderBean"/>
    </bean>
```
SpringExample.java

The following Java code example uses the bean configuration file to initialize your objects.

```java
public class SpringExample {
    public static void main(String[] args) throws JMSException {
        if (args.length != 1 || !args[0].endsWith(".xml")) {
            System.err.println("Usage: " + SpringExample.class.getName() + " <spring config.xml>");
            System.exit(1);
        }
        File springFile = new File(args[0]);
        if (!springFile.exists() || !springFile.canRead()) {
            System.err.println("File " + args[0] + " does not exist or isn’t readable.");
            System.exit(2);
        }
        ExampleCommon.setupLogging();
        FileSystemXmlApplicationContext context = new FileSystemXmlApplicationContext("file://" +
            springFile.getAbsolutePath());
        Connection connection;
        try {
            connection = context.getBean(Connection.class);
        } catch (NoSuchBeanDefinitionException e) {
            System.err.println("Can’t find the JMS connection to use: " +
                e.getMessage());
            System.exit(3);
        }
        String queueName;
        try {
            queueName = context.getBean("QueueName", String.class);
        } catch (NoSuchBeanDefinitionException e) {
            System.err.println("Can’t find the name of the queue to use: " +
                e.getMessage());
            System.exit(3);
        }
    }
}
```
ExampleCommon.java

The following Java code example checks if an Amazon SQS queue exists and then creates one if it does not. It also includes example logging code.

```java
public class ExampleCommon {
    /**
     * A utility function to check the queue exists and create it if needed. For most
     * use cases this is usually done by an administrator before the application is run.
     */
    public static void ensureQueueExists(SQSConnection connection, String queueName) throws JMSException {
        AmazonSQSMessagingClientWrapper client = connection.getWrappedAmazonSQSClient();

        /**
         * In most cases, you can do this with just a createQueue call, but GetQueueUrl
         * (called by queueExists) is a faster operation for the common case where the
         * queue already exists. Also many users and roles have permission to call GetQueueUrl
         * but do not have permission to call CreateQueue.
         */
        if(!client.queueExists(queueName)) {
            client.createQueue(queueName);
        }
    }
}
```
Supported JMS 1.1 Implementations

The Amazon SQS Java Messaging Library supports the following JMS 1.1 implementations. For more information about the supported features and capabilities of the Amazon SQS Java Messaging Library, see the Amazon SQS FAQ.

Supported Common Interfaces

- Connection
- ConnectionFactory
- Destination
- Session
- MessageConsumer
- MessageProducer

Supported Message Types

- ByteMessage
- ObjectMessage
- TextMessage

Supported Message Acknowledgment Modes

- AUTO_ACKNOWLEDGE
- CLIENT_ACKNOWLEDGE
- DUPS_OK_ACKNOWLEDGE
• **UNORDERED_ACKNOWLEDGE**

**Note**
The **UNORDERED_ACKNOWLEDGE** mode isn't part of the JMS 1.1 specification. This mode helps Amazon SQS allow a JMS client to explicitly acknowledge a message.

**JMS-Defined Headers and Reserved Properties**

**For Sending Messages**

When you send messages, you can set the following headers and properties for each message:

- JMSXGroupID (required for FIFO queues, not allowed for standard queues)
- JMS_SQS_DeduplicationId (optional for FIFO queues, not allowed for standard queues)

After you send messages, Amazon SQS sets the following headers and properties for each message:

- JMSMessageID
- JMS_SQS_SequenceNumber (only for FIFO queues)

**For Receiving Messages**

When you receive messages, Amazon SQS sets the following headers and properties for each message:

- JMSDestination
- JMSMessageID
- JMSRedelivered
- JMSXDeliveryCount
- JMSXGroupID (only for FIFO queues)
- JMS_SQS_DeduplicationId (only for FIFO queues)
- JMS_SQS_SequenceNumber (only for FIFO queues)
Best Practices for Amazon SQS

These best practices can help you make the most of Amazon SQS.

Topics
- General Recommendations (p. 109)
- Recommendations for FIFO (First-In-First-Out) Queues (p. 111)

General Recommendations

The following guidelines can help you reduce costs and process messages efficiently using Amazon SQS.

Processing Messages

- To ensure that there is sufficient time to process a message, you should use one of the following strategies:
  - If you know (or can reasonably estimate) how long it takes to process a message, extend the message's visibility timeout to the maximum time it takes to process and delete the message. For more information, see Configuring the Visibility Timeout (p. 60) and Changing a Message's Visibility Timeout (p. 60).
  - If you don't know how long it takes to process a message, specify the initial visibility timeout (for example, 2 minutes) and the period of time after which you can check whether the message is processed (for example, 1 minute). If the message isn't processed, extend the visibility timeout (for example, to 3 minutes).

  Note
  If you need to extend the visibility timeout for longer than 12 hours, consider using Amazon Simple Workflow Service.

- To handle request errors, you should use one of the following strategies:
  - If you use an AWS SDK, you already have automatic retry and backoff logic at your disposal. For more information, see Error Retries and Exponential Backoff in AWS in the Amazon Web Services General Reference.
  - If you do not use the AWS SDK features for retry and backoff, allow a pause (for example, 200 ms) before retrying the ReceiveMessage action after receiving no messages, a timeout, or an error.
message from Amazon SQS. For subsequent use of `ReceiveMessage` that gives the same results, allow a longer pause (for example, 400 ms).

- To capture all messages that can't be processed, and to ensure the correctness of CloudWatch metrics, you should configure a dead-letter queue (p. 61).
- The redrive policy redirects messages to a dead-letter queue after the source queue fails to process a message a specified number of times.
- Using a dead-letter queue decreases the number of messages and reduces the possibility of exposing you to poison pill messages (messages that are received but can't be processed).
- Including a poison pill message in a queue can distort the `ApproximateAgeOfOldestMessage` (p. 123) CloudWatch metric by giving an incorrect age of the poison pill message. Configuring a dead-letter queue helps avoid false alarms when using this metric.
- To avoid inconsistent message processing by standard queues, avoid setting the number of maximum receives to 1 when you configure a dead-letter queue.

**Important**
In some unlikely scenarios, if you set the number of maximum receives to 1, any time a `ReceiveMessage` call fails, a message might be moved to a dead-letter queue without being received.

### Reducing Costs

- To reduce costs, batch your message actions:
  - To send, receive, and delete messages, and to change the message visibility timeout for multiple messages with a single action, use the Amazon SQS batch API actions (p. 176).
  - To combine client-side buffering with request batching, use long polling together with the buffered asynchronous client (p. 176) included with the AWS SDK for Java.

**Note**
The Amazon SQS Buffered Asynchronous Client doesn't currently support FIFO queues.

- To take advantage of additional potential reduced cost or near-instantaneous response, use one of the following polling modes:
  - Long polling lets you consume messages from your Amazon SQS queue as soon as they become available.
    - To reduce the cost of using Amazon SQS and to decrease the number of empty receives to an empty queue (responses to the `ReceiveMessage` action which return no messages), enable long polling. For more information, see Amazon SQS Long Polling (p. 73).
    - To increase efficiency when polling for multiple threads with multiple receives, decrease the number of threads.
    - Long polling is preferable over short polling in most cases.
  - Short polling returns responses immediately, even if the polled Amazon SQS queue is empty.
    - To satisfy the requirements of an application that expects immediate responses to the `ReceiveMessage` request, use short polling.
    - Short polling is billed the same as long polling.

### Moving from a Standard Queue to a FIFO Queue

- If you’re not setting the `DelaySeconds` parameter on each message, you can move to a FIFO queue by providing a message group ID for every sent message. For more information, see Moving from a Standard Queue to a FIFO Queue (p. 56).
Recommendations for FIFO (First-In-First-Out) Queues

The following guidelines can help you use the message deduplication ID and message group ID optimally. For more information, see the `SendMessage` and `SendMessageBatch` actions in the Amazon Simple Queue Service API Reference.

Using the Message Deduplication ID

The message deduplication ID is the token used for deduplication of sent messages. If a message with a particular message deduplication ID is sent successfully, any messages sent with the same message deduplication ID are accepted successfully but aren't delivered during the 5-minute deduplication interval.

**Note**
Message deduplication applies to an entire queue, not to individual message groups.

- If you have a single producer and a single consumer and the messages are unique because an application-specific message ID is included in the body of the message, you should follow these guidelines:
  - Enable content-based deduplication for the queue (each of your messages has a unique body). The producer can omit the message deduplication ID.
  - Although the consumer isn't required to provide a receive request attempt ID for each request, it's a best practice because it allows fail-retry sequences to execute faster.
  - You can retry send or receive requests because they don't interfere with the ordering of messages in FIFO queues.

- The producer should provide message deduplication ID values for each message send in the following scenarios:
  - Messages sent with identical message bodies that Amazon SQS must treat as unique.
  - Messages sent with identical content but different message attributes that Amazon SQS must treat as unique.
  - Messages sent with different content (for example, retry counts included in the message body) that Amazon SQS must treat as duplicates.

- The deduplication process in FIFO queues is time-sensitive. When designing your application, you should ensure that both the producer and the consumer can recover in case of a client or network outage.
  - The producer must be aware of the deduplication interval of the queue. Amazon SQS has a minimum deduplication interval of 5 minutes. Retrying `SendMessage` requests after the deduplication interval expires can introduce duplicate messages into the queue. For example, a mobile device in a car sends messages whose order is important. If the car loses cellular connectivity for a period of time before receiving an acknowledgement, retrying the request after regaining cellular connectivity can create a duplicate.
  - The consumer must have a visibility timeout that minimizes the risk of being unable to process messages before the visibility timeout expires. You can extend the visibility timeout while the messages are being processed by calling the `ChangeMessageVisibility` action. However, if the visibility timeout expires, another consumer can immediately begin to process the messages, causing a message to be processed multiple times. To avoid this scenario, configure a dead-letter queue (p. 61).
Using the Message Group ID

The message group ID is the tag that specifies that a message belongs to a specific message group. Messages that belong to the same message group are always processed one by one, in a strict order relative to the message group (however, messages that belong to different message groups might be processed out of order).

- To interleave multiple ordered message groups within a single FIFO queue, you should use message group ID values (for example, session data for multiple users). In this scenario, multiple consumers can process the queue, but the session data of each user is processed in a FIFO manner.

  **Note**
  When messages that belong to a particular message group ID are invisible, no other consumer can process messages with the same message group ID.

- To avoid processing duplicate messages in a system with multiple producers and consumers where throughput and latency are more important than ordering, the producer should generate a unique message group ID for each message.

  **Note**
  In this scenario, duplicates are eliminated. However, the ordering of message can't be guaranteed.
  Any scenario with multiple producers and consumers increases the risk of inadvertently delivering a duplicate message if a worker does not process the message within the visibility timeout and the message becomes available to another worker.

Using the Receive Request Attempt ID

The receive request attempt ID is the large, non-consecutive number that Amazon SQS assigns to each message.

During a long-lasting network outage that causes connectivity issues between your SDK and Amazon SQS, it's a best practice to provide the receive request attempt ID and to retry with the same receive request attempt ID if the SDK operation fails.
Amazon SQS Limits

This topic lists limits within Amazon Simple Queue Service (Amazon SQS).

Topics
- Limits Related to Queues (p. 113)
- Limits Related to Messages (p. 114)
- Limits Related to Policies (p. 115)

Limits Related to Queues

The following table lists limits related to queues.

<table>
<thead>
<tr>
<th>Limit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inflight messages per queue</td>
<td>For standard queues, there can be a maximum of 120,000 inflight messages per queue. If you reach this limit, Amazon SQS returns the OverLimit error message. To avoid reaching the limit, you should delete messages from the queue after they're processed. You can also increase the number of queues you use to process your messages. For FIFO queues, there can be a maximum of 20,000 inflight messages per queue. If you reach this limit, Amazon SQS returns no error messages.</td>
</tr>
<tr>
<td>Queue name</td>
<td>A queue name can have up to 80 characters. The following characters are accepted: alphanumeric characters, hyphens (-), and underscores (_). Note Queue names are case-sensitive (for example, Test-queue and test-queue are different queues).</td>
</tr>
</tbody>
</table>
Limits Related to Messages

The following table lists limits related to messages.

<table>
<thead>
<tr>
<th>Limit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message attributes</td>
<td>A message can contain up to 10 metadata attributes.</td>
</tr>
<tr>
<td>Message batch</td>
<td>A single message batch request can include a maximum of 10 messages. For more information, see Advanced Configuration (p. 177) in the Amazon SQS Batch API Actions (p. 176) section.</td>
</tr>
</tbody>
</table>
| Message content   | A message can include only XML, JSON, and unformatted text. The following Unicode characters are allowed: 
#x9 | #xA | #xD | #x20 to #xD7FF | #xE000 to #xFFFD | #x10000 to #x10FFFF
Any characters not included in this list are rejected. For more information, see the W3C specification for characters. |
| Message retention | By default, a message is retained for 4 days. The minimum is 60 seconds (1 minute). The maximum is 1,209,600 seconds (14 days).     |
| Message throughput| Standard queues can support a nearly unlimited number of transactions per second (TPS) per API action.                              |
Limits Related to Policies

The following table lists limits related to policies.

<table>
<thead>
<tr>
<th>Name</th>
<th>Maximum Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bytes</td>
<td>8192</td>
</tr>
<tr>
<td>Conditions</td>
<td>10</td>
</tr>
<tr>
<td>Principals</td>
<td>50</td>
</tr>
<tr>
<td>Statements</td>
<td>20</td>
</tr>
</tbody>
</table>
Monitoring and Logging Amazon SQS Queues

This section provides information about monitoring and logging Amazon SQS queues.

Topics
- Monitoring Amazon SQS using CloudWatch (p. 116)
- Logging Amazon SQS API Actions Using AWS CloudTrail (p. 126)

Monitoring Amazon SQS using CloudWatch

Amazon SQS and Amazon CloudWatch are integrated so you can use CloudWatch to view and analyze metrics for your Amazon SQS queues. You can view and analyze your queues' metrics from the Amazon SQS console, the CloudWatch console, the command line, or programmatically.

CloudWatch metrics for your Amazon SQS queues are automatically collected and pushed to CloudWatch every five minutes. These metrics are gathered on all queues that meet the CloudWatch guidelines for being active. CloudWatch considers a queue to be active for up to six hours if it contains any messages or if any API action accesses it.

Note
There is no charge for the Amazon SQS metrics reported in CloudWatch. They're provided as part of the Amazon SQS service. Detailed monitoring (or one-minute metrics) is currently unavailable for Amazon SQS. CloudWatch metrics are supported for both standard and FIFO queues.

Topics
- Common Monitoring Tasks (p. 117)
- Access CloudWatch Metrics for Amazon SQS (p. 117)
- Set CloudWatch Alarms for Amazon SQS Metrics (p. 120)
Common Monitoring Tasks

Use the following decision matrix to determine which set of instructions to follow to complete your desired task.

<table>
<thead>
<tr>
<th>Task</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quickly display a default view of CloudWatch metrics over time for up to 10 queues at once.</td>
<td>Access Metrics Using the Amazon SQS Console (p. 117)</td>
</tr>
<tr>
<td>Further customize the default views of CloudWatch metrics.</td>
<td>Access Metrics Using the CloudWatch Console (p. 119)</td>
</tr>
<tr>
<td>Set alarms when metrics meet or exceed specified conditions.</td>
<td>Set CloudWatch Alarms for Amazon SQS Metrics (p. 120)</td>
</tr>
<tr>
<td>Create complex dashboards that display metrics for multiple Amazon SQS queues together.</td>
<td>Access Metrics Using the AWS CLI (p. 120)</td>
</tr>
<tr>
<td>Access CloudWatch metrics from the command line or programmatically.</td>
<td>Access Metrics Using the CloudWatch CLI (p. 120)</td>
</tr>
<tr>
<td></td>
<td>Access Metrics Using the CloudWatch API (p. 120)</td>
</tr>
</tbody>
</table>

Access CloudWatch Metrics for Amazon SQS

You can access metrics for Amazon SQS using the Amazon SQS console, the CloudWatch console, the AWS CLI, CloudWatch's own CLI, or programmatically using the CloudWatch API. The following procedures show you how to access the metrics using these different options.

Access Metrics Using the Amazon SQS Console

1. Sign in to the Amazon SQS console.
2. In the list of queues, choose (check) the boxes for the queues that you want to access metrics for.
   You can show metrics for up to 10 queues.
3. Choose the Monitoring tab.
4. To understand what a particular graph represents, hover over the information icon next to the desired graph, or see Available CloudWatch Metrics for Amazon SQS (p. 123).

5. To change the time range for all of the graphs at the same time, for Time Range, choose the desired time range (for example, Last Hour).

6. To view additional statistics for an individual graph, choose the graph. After the graph displays in a larger dialog box, for Statistic, choose the desired statistic (for example, Sum). For a list of supported statistics, see Available CloudWatch Metrics for Amazon SQS (p. 123).
Access CloudWatch Metrics for Amazon SQS

7. To change the time range and time interval that an individual graph displays (for example, to show a time range of the last 24 hours instead of the last 5 minutes, or to show a time period of every hour instead of every 5 minutes), with the graph's dialog box still displayed, for **Time Range**, choose the desired time range (for example, **Last 24 Hours**). For **Period**, choose the desired time period within the specified time range (for example, **1 Hour**). When you're finished looking at the graph, choose **Close**.

8. To work with additional CloudWatch features, on the **Monitoring** tab, choose **View all CloudWatch metrics**, and then follow the instructions in the **Access Metrics Using the CloudWatch Console** (p. 119) procedure.

**Access Metrics Using the CloudWatch Console**


2. In the navigation pane, choose **Metrics**.

3. Select the **SQS** metric namespace.

4. Select the **Queue Metrics** metric dimension.
You can now examine your Amazon SQS metrics:

- To sort the metrics, use the column heading.
- To graph a metric, select the check box next to the metric.
- To filter by metric, choose the metric name and then choose **Add to search**.

For more information and additional options, see [Graph Metrics](#) and [Using Amazon CloudWatch Dashboards](#).

**Access Metrics Using the AWS CLI**

Run the `get-metric-statistics` command. For more information, see [Get Statistics for a Metric](#).

**Access Metrics Using the CloudWatch CLI**

Run the `mon-get-stats` command.

**Access Metrics Using the CloudWatch API**

Call the `GetMetricStatistics` operation. For more information, see [Get Statistics for a Metric](#).

**Set CloudWatch Alarms for Amazon SQS Metrics**

CloudWatch allows you to trigger alarms when a threshold is met for a metric. For example, you can set an alarm for the `NumberOfMessagesSent` metric so that when the number of messages exceeds a
specified limit over a specified time period, then an email notification can be sent to inform you of the event.

To set an alarm (CloudWatch console)

1. Sign in to the AWS Management Console and open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose Alarms, and then choose Create Alarm. The Create Alarm dialog box displays.
3. On the Select Metric page, choose Browse Metrics, SQS:

4. For SQS > Queue Metrics, choose (check) the box that you want to set an alarm for the combination of QueueName and Metric Name. (For a list of available metrics, see Available CloudWatch Metrics for Amazon SQS (p. 123)). For example, choosing (checking) the box for MyQueue, NumberOfMessagesSent sets an alarm based on the number of messages sent to the MyQueue queue.
5. Choose Next. The Define Alarm page displays.

6. For Alarm Threshold, fill in the Name and Description boxes. For is, for, Period, and Statistic, specify the conditions for the alarm. For example, let’s say you chose (checked) the box for MyQueue, NumberOfMessagesSent on the Select Metric page, and you want to alarm when more than 100 messages are sent in any hour to the MyQueue queue. You’d then set the following:

   • Set is to > 100.
   • Set for to 1.
   • Set Period to 1 Hour.
   • Set Statistic to Sum.
7. For **Actions** and **Whenever this alarm**, choose **State is ALARM**. For **Send notification to**, if you want CloudWatch to send you an email when the alarm state is reached, either select an existing Amazon SNS topic or choose **New list**. If you choose **New list**, you can set the name and list comma-separated email addresses for a new topic. This list is saved; it appears for future alarms.

   **Note**
   If you choose **New list** to create a new Amazon SNS topic, the email addresses must be verified before they receive any notifications. Emails are sent only when the alarm enters an alarm state. If this alarm state change happens before the email addresses are verified, they won't receive a notification.

8. Choose **Create Alarm**. CloudWatch creates the alarm and then displays the alarms list.

For more information, see [Creating Amazon CloudWatch Alarms](#).

### Available CloudWatch Metrics for Amazon SQS

Amazon SQS sends the following metrics to CloudWatch.

**Note**
For standard queues, the result is approximate because of the distributed architecture of Amazon SQS. In most cases, the count should be close to the actual number of messages in the queue.
For FIFO queues, the result is exact.
# Amazon SQS Metrics

The AWS/SQS namespace includes the following metrics.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApproximateAgeOfOldestMessage</td>
<td>The approximate age of the oldest non-deleted message in the queue.</td>
</tr>
<tr>
<td></td>
<td>Units: Seconds</td>
</tr>
<tr>
<td></td>
<td>Valid Statistics: Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console)</td>
</tr>
<tr>
<td>ApproximateNumberOfMessagesDelayed</td>
<td>The number of messages in the queue that are delayed and not available for reading immediately. This can happen when the queue is configured as a delay queue or when a message has been sent with a delay parameter.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Valid Statistics: Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console)</td>
</tr>
<tr>
<td>ApproximateNumberOfMessagesNotVisible</td>
<td>The number of messages that are &quot;in flight.&quot; Messages are considered in flight if they have been sent to a client but have not yet been deleted or have not yet reached the end of their visibility window.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Valid Statistics: Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console)</td>
</tr>
<tr>
<td>ApproximateNumberOfMessagesVisible</td>
<td>The number of messages available for retrieval from the queue.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Valid Statistics: Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console)</td>
</tr>
<tr>
<td>NumberOfEmpty Receives</td>
<td>The number of ReceiveMessage API calls that did not return a message.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Valid Statistics: Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console)</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NumberOfMessagesDeleted</td>
<td>The number of messages deleted from the queue.</td>
</tr>
<tr>
<td></td>
<td><strong>Units:</strong> Count</td>
</tr>
<tr>
<td></td>
<td><strong>Valid Statistics:</strong> Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console)</td>
</tr>
<tr>
<td></td>
<td>Amazon SQS emits the NumberOfMessagesDeleted metric for every successful deletion operation that uses a valid receipt handle, including duplicate deletions. The following scenarios might cause the value of the NumberOfMessagesDeleted metric to be higher than expected:</td>
</tr>
<tr>
<td></td>
<td>• Calling the DeleteMessage action on different receipt handles that belong to the same message: If the message is not processed before the visibility timeout expires, the message becomes available to other consumers that can process it and delete it again, increasing the value of the NumberOfMessagesDeleted metric.</td>
</tr>
<tr>
<td></td>
<td>• Calling the DeleteMessage action on the same receipt handle: If the message is processed and deleted but you call the DeleteMessage action again using the same receipt handle, a success status is returned, increasing the value of the NumberOfMessagesDeleted metric.</td>
</tr>
<tr>
<td>NumberOfMessagesReceived</td>
<td>The number of messages returned by calls to the ReceiveMessage API action.</td>
</tr>
<tr>
<td></td>
<td><strong>Units:</strong> Count</td>
</tr>
<tr>
<td></td>
<td><strong>Valid Statistics:</strong> Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console)</td>
</tr>
<tr>
<td>NumberOfMessagesSent</td>
<td>The number of messages added to a queue.</td>
</tr>
<tr>
<td></td>
<td><strong>Units:</strong> Count</td>
</tr>
<tr>
<td></td>
<td><strong>Valid Statistics:</strong> Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console)</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SentMessageSize</td>
<td>The size of messages added to a queue.</td>
</tr>
<tr>
<td></td>
<td>Units: Bytes</td>
</tr>
<tr>
<td></td>
<td>Valid Statistics: Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console)</td>
</tr>
<tr>
<td></td>
<td>Note that SentMessageSize does not display as an available metric in the CloudWatch console until at least one message is sent to the corresponding queue.</td>
</tr>
</tbody>
</table>

Dimensions for Amazon SQS Metrics

The only dimension that Amazon SQS sends to CloudWatch is QueueName. This means that all available statistics are filtered by QueueName.

Logging Amazon SQS API Actions Using AWS CloudTrail

Amazon SQS is integrated with CloudTrail, a service that captures API calls made by or on behalf of Amazon SQS in your AWS account and delivers the log files to the specified Amazon S3 bucket. CloudTrail captures API calls made from the Amazon SQS console or from the Amazon SQS API. You can use the information collected by CloudTrail to determine which requests are made to Amazon SQS, the source IP address from which the request is made, who made the request, when it is made, and so on. To learn more about CloudTrail, including how to configure and enable it, see the AWS CloudTrail User Guide.

CloudTrail is supported for both standard and FIFO queues.

Amazon SQS Information in CloudTrail

When CloudTrail logging is enabled in your AWS account, API calls made to Amazon SQS actions are tracked in log files. Amazon SQS records are written together with other AWS service records in a log file. CloudTrail determines when to create and write to a new file based on a time period and file size.

The following actions are supported:

- AddPermission
- CreateQueue
- DeleteQueue
- PurgeQueue
- RemovePermission
- SetQueueAttributes

Every log entry contains information about who generated the request. The user identity information in the log helps you determine whether the request was made with root or IAM user credentials,
with temporary security credentials for a role or federated user, or by another AWS service. For more information, see the `userIdentity` field in the CloudTrail Event Reference.

You can store your log files in your bucket for as long as you want, but you can also define Amazon S3 lifecycle rules to archive or delete log files automatically. By default, your log files are encrypted using Amazon S3 server-side encryption (SSE).

You can choose to have CloudTrail publish Amazon SNS notifications when new log files are delivered if you want to take quick action upon log file delivery. For more information, see Configuring Amazon SNS Notifications for CloudTrail.

You can also aggregate Amazon SQS log files from multiple AWS regions and multiple AWS accounts into a single Amazon S3 bucket. For more information, see Receiving CloudTrail Log Files from Multiple Regions.

Understanding Amazon SQS Log File Entries

CloudTrail log files contain one or more log entries where each entry is made up of multiple JSON-formatted events. A log entry represents a single request from any source and includes information about the requested action, any parameters, the date and time of the action, and so on. The log entries are not guaranteed to be in any particular order. That is, they're not an ordered stack trace of the public API calls.

AddPermission

The following example shows a CloudTrail log entry for AddPermission:

```json
{
  "Records": [
    {
      "eventVersion": "1.01",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:user/Alice",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
      },
      "eventTime": "2014-07-16T00:44:19Z",
      "eventSource": "sqs.amazonaws.com",
      "eventName": "AddPermission",
      "awsRegion": "us-east-2",
      "sourceIPAddress": "192.0.2.0",
      "userAgent": "Mozilla/5.0 (X11; Linux x86_64; rv:24.0) Gecko/20100101 Firefox/24.0",
      "requestParameters": {
        "actions": [
          "SendMessage"
        ],
        "aWSAccountIds": [
          "123456789012"
        ],
        "label": "label",
        "queueUrl": "http://test-sqs.amazon.com/123456789012/hello1"
      },
      "responseElements": null,
      "requestID": "334c4ccdb9bb-50fa-abdb-80f274981d60",
      "eventID": "0552b000-09a3-47d6-a810-c5f9fd2534fe"
    }
  ]
}
```
CreateQueue

The following example shows a CloudTrail log entry for CreateQueue:

```
{  
    "Records": [  
        {  
            "eventVersion": "1.01",
            "userIdentity": {  
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::123456789012:user/Alice",
                "accountId": "123456789012",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },  
            "eventTime": "2014-07-16T00:42:42Z",
            "eventSource": "sqs.amazonaws.com",
            "eventName": "CreateQueue",
            "awsRegion": "us-east-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "Mozilla/5.0 (X11; Linux x86_64; rv:24.0) Gecko/20100101 Firefox/24.0",
            "requestParameters": {  
                "queueName": "hello1"
            },  
            "responseElements": {  
                "queueUrl": "http://test-sqs.amazon.com/123456789012/hello1"
            },  
            "requestID": "49ebbdb7-5cd3-5323-8a00-f1889011fee9",
            "eventID": "68f4e71c-4f2f-4625-8378-130ac89660b1"
        }
    ]
}
```

DeleteQueue

The following example shows a CloudTrail log entry for DeleteQueue:

```
{  
    "Records": [  
        {  
            "eventVersion": "1.01",
            "userIdentity": {  
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::123456789012:user/Alice",
                "accountId": "123456789012",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },  
            "eventTime": "2014-07-16T00:44:47Z",
            "eventSource": "sqs.amazonaws.com",
            "eventName": "DeleteQueue",
            "awsRegion": "us-east-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "Mozilla/5.0 (X11; Linux x86_64; rv:24.0) Gecko/20100101 Firefox/24.0",
            "requestParameters": {  
                "queueUrl": "http://test-sqs.amazon.com/123456789012/hello1"
            },  
            "responseElements": null,
            "requestID": "e4c0cc05-4faa-51d5-aab2-803a8294388d",
            "eventID": "aflbb158-6443-4b4d-abfd-1b867280d964"
        }
    ]
}
```
RemovePermission

The following example shows a CloudTrail log entry for RemovePermission:

```
{
  "Records": [
    {
      "eventVersion": "1.01",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:user/Alice",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
      },
      "eventTime": "2014-07-16T00:44:36Z",
      "eventSource": "sqs.amazonaws.com",
      "eventName": "RemovePermission",
      "awsRegion": "us-east-2",
      "sourceIPAddress": "192.0.2.0",
      "userAgent": "Mozilla/5.0 (X11; Linux x86_64; rv:24.0) Gecko/20100101 Firefox/24.0",
      "requestParameters": {
        "label": "label",
        "queueUrl": "http://test-sqs.amazon.com/123456789012/hello1"
      },
      "responseElements": null,
      "requestID": "48178821-9c2b-5be0-88bf-c41e5118162a",
      "eventID": "fed8a623-3fe9-4e64-9543-586d9e500159"
    }
  ]
}
```

SetQueueAttributes

The following example shows a CloudTrail log entry for SetQueueAttributes:

```
{
  "Records": [
    {
      "eventVersion": "1.01",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:user/Alice",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
      },
      "eventTime": "2014-07-16T00:43:15Z",
      "eventSource": "sqs.amazonaws.com",
      "eventName": "SetQueueAttributes",
      "awsRegion": "us-east-2",
      "sourceIPAddress": "192.0.2.0",
      "userAgent": "Mozilla/5.0 (X11; Linux x86_64; rv:24.0) Gecko/20100101 Firefox/24.0",
      "requestParameters": {
        "attributes": {
          "VisibilityTimeout": "100"
        }
      },
      "responseElements": null,
      "requestID": "48178821-9c2b-5be0-88bf-c41e5118162a",
      "eventID": "fed8a623-3fe9-4e64-9543-586d9e500159"
    }
  ]
}
```
Amazon Simple Queue Service Developer Guide
Understanding Amazon SQS Log File Entries

},
  "queueUrl": "http://test-sqs.amazon.com/123456789012/hello1"
},
"responseElements": null,
"requestID": "7f15d706-f3d7-5221-b9ca-9b393f349b79",
"eventID": "8b66b2dc-2661-49b1-b328-94317815088b"
}
]
Amazon SQS Security

This section provides information about Amazon SQS security, authentication and access control, and the access policy language.

Topics
- Authentication and Access Control for Amazon SQS (p. 131)
- Protecting Data Using Server-Side Encryption (SSE) and AWS KMS (p. 156)

Authentication and Access Control for Amazon SQS

Access to Amazon SQS requires credentials that AWS can use to authenticate your requests. These credentials must have permissions to access AWS resources, such as an Amazon SQS queues and messages. The following sections provide details on how you can use AWS Identity and Access Management (IAM) and Amazon SQS to help secure your resources by controlling access to them.

Topics
- Authentication (p. 131)
- Access Control (p. 132)

Authentication

You can access AWS as any of the following types of identities:

- **AWS account root user** – When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account root user and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

- **IAM user** – An IAM user is an identity within your AWS account that has specific custom permissions (for example, permissions to create a queue in Amazon SQS). You can use an IAM user name and password to sign in to secure AWS webpages like the AWS Management Console, AWS Discussion Forums, or the AWS Support Center.
In addition to a user name and password, you can also generate access keys for each user. You can use these keys when you access AWS services programmatically, either through one of the several SDKs or by using the AWS Command Line Interface (CLI). The SDK and CLI tools use the access keys to cryptographically sign your request. If you don’t use AWS tools, you must sign the request yourself. Amazon SQS supports Signature Version 4, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 Signing Process in the AWS General Reference.

- **IAM role** – An IAM role is an IAM identity that you can create in your account that has specific permissions. It is similar to an IAM user, but it is not associated with a specific person. An IAM role enables you to obtain temporary access keys that can be used to access AWS services and resources. IAM roles with temporary credentials are useful in the following situations:

  - **Federated user access** – Instead of creating an IAM user, you can use existing user identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as federated users. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated Users and Roles in the IAM User Guide.

  - **AWS service access** – You can use an IAM role in your account to grant an AWS service permissions to access your account’s resources. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data from that bucket into an Amazon Redshift cluster. For more information, see Creating a Role to Delegate Permissions to an AWS Service in the IAM User Guide.

  - **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using Roles for Applications on Amazon EC2 in the IAM User Guide.

**Access Control**

Amazon SQS has its own resource-based permissions system that uses policies written in the same language used for AWS Identity and Access Management (IAM) policies. This means that you can achieve similar things with Amazon SQS policies and IAM policies.

**Note**

It is important to understand that all AWS accounts can delegate their permissions to users under their accounts. Cross-account access allows you to share access to your AWS resources without having to manage additional users. For information about using cross-account access, see Enabling Cross-Account Access in the IAM User Guide. Currently, Amazon SQS supports only a limited subset of the condition keys available in IAM. For more information, see Amazon SQS API Permissions: Actions and Resource Reference (p. 154).

The following sections describe how to manage permissions for Amazon SQS. We recommend that you read the overview first.
Overview of Managing Access Permissions to Your Amazon Simple Queue Service Resource

Every AWS resource is owned by an AWS account, and permissions to create or access a resource are governed by permissions policies. An account administrator can attach permissions policies to IAM identities (users, groups, and roles), and some services (such as Amazon SQS) also support attaching permissions policies to resources.

Note
An account administrator (or administrator user) is a user with administrative privileges. For more information, see IAM Best Practices in the IAM User Guide.

When granting permissions, you specify what users get permissions, the resource they get permissions for, and the specific actions that you want to allow on the resource.

Topics
- Amazon Simple Queue Service Resource and Operations (p. 133)
- Understanding Resource Ownership (p. 134)
- Managing Access to Resources (p. 134)
- Specifying Policy Elements: Actions, Effects, Resources, and Principals (p. 136)
- Specifying Conditions in a Policy (p. 137)

Amazon Simple Queue Service Resource and Operations

In Amazon SQS, the only resource is the queue. In a policy, use an Amazon Resource Name (ARN) to identify the resource that the policy applies to. The following resource has a unique ARN associated with it:

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>ARN Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queue</td>
<td>arn:aws:sqs:region:account_id:queue_name</td>
</tr>
</tbody>
</table>

The following are examples of the ARN format for queues:

- An ARN for a queue named my_queue in the US East (Ohio) region, belonging to AWS Account 123456789012:

  arn:aws:sqs:us-east-2:123456789012:my_queue

- An ARN for a queue named my_queue in each of the different regions that Amazon SQS supports:

  arn:aws:sqs:*:123456789012:my_queue
• An ARN that uses * or ? as a wildcard for the queue name. In the following examples, the ARN matches all queues prefixed with `my_prefix_`:

```
arn:aws:sqs::*:123456789012:my_prefix_*
```

You can get the ARN value for an existing queue by calling the `GetQueueAttributes` action. The value of the `QueueArn` attribute is the ARN of the queue. For more information about ARNs, see IAM ARNs in the IAM User Guide.

Amazon SQS provides a set of API operations that work with the queue resource. For more information, see Amazon SQS API Permissions: Actions and Resource Reference (p. 154).

Understanding Resource Ownership

The AWS account owns the resources that are created in the account, regardless of who created the resources. Specifically, the resource owner is the AWS account of the principal entity (that is, the root account, an IAM user, or an IAM role) that authenticates the resource creation request. The following examples illustrate how this works:

• If you use the root account credentials of your AWS account to create an Amazon SQS queue, your AWS account is the owner of the resource (in Amazon SQS, the resource is the Amazon SQS queue).
• If you create an IAM user in your AWS account and grant permissions to create a queue to the user, the user can create the queue. However, your AWS account (to which the user belongs) owns the queue resource.
• If you create an IAM role in your AWS account with permissions to create an Amazon SQS queue, anyone who can assume the role can create a queue. Your AWS account (to which the role belongs) owns the queue resource.

Managing Access to Resources

A permissions policy describes the permissions granted to accounts. The following section explains the available options for creating permissions policies.

**Note**

This section discusses using IAM in the context of Amazon SQS. It doesn't provide detailed information about the IAM service. For complete IAM documentation, see What is IAM? in the IAM User Guide. For information about IAM policy syntax and descriptions, see AWS IAM Policy Reference in the IAM User Guide.

Policies attached to an IAM identity are referred to as identity-based policies (IAM polices) and policies attached to a resource are referred to as resource-based policies.

Identity-Based Policies (IAM Policies and Amazon SQS Policies)

There are two ways to give your users permissions to your Amazon SQS queues: using the Amazon SQS policy system and using the IAM policy system. You can use either system, or both, to attach policies to users or roles. In most cases, you can achieve the same result using either system. For example, you can do the following:

• **Attach a permission policy to a user or a group in your account** – To grant user permissions to create an Amazon SQS queue, attach a permissions policy to a user or group that the user belongs to.
• **Attach a permission policy to a user in another AWS account** – To grant user permissions to create an Amazon SQS queue, attach an Amazon SQS permissions policy to a user in another AWS account.
• **Attach a permission policy to a role (grant cross-account permissions)** – To grant cross-account permissions, attach an identity-based permissions policy to an IAM role. For example, the AWS account
A administrator can create a role to grant cross-account permissions to AWS account B (or an AWS service) as follows:

- The account A administrator creates an IAM role and attaches a permissions policy—that grants permissions on resources in account A—to the role.
- The account A administrator attaches a trust policy to the role that identifies account B as the principal who can assume the role.
- The account B administrator delegates the permission to assume the role to any users in account B. This allows users in account B to create or access queues in account A.

**Note**
If you want to grant the permission to assume the role to an AWS service, the principal in the trust policy can also be an AWS service principal.

For more information about using IAM to delegate permissions, see Access Management in the IAM User Guide.

While Amazon SQS works with IAM policies, it has its own policy infrastructure. You can use an Amazon SQS policy with a queue to specify which AWS Accounts have access to the queue. You can specify the type of access and conditions (for example, a condition that grants permissions to use SendMessage, ReceiveMessage if the request is made before December 31, 2010). The specific actions you can grant permissions for are a subset of the overall list of Amazon SQS actions. When you write an Amazon SQS policy and specify * to “allow all Amazon SQS actions,” it means that a user can perform all actions in this subset.

The following diagram illustrates the concept of one of these basic Amazon SQS policies that covers the subset of actions. The policy is for queue_xyz, and it gives AWS Account 1 and AWS Account 2 permissions to use any of the allowed actions with the specified queue.

**Note**
The resource in the policy is specified as 123456789012/queue_xyz, where 123456789012 is the AWS Account ID of the account that owns the queue.

With the introduction of IAM and the concepts of Users and Amazon Resource Names (ARNs), a few things have changed about SQS policies. The following diagram and table describe the changes.
In addition to specifying which AWS Accounts have access to a queue, you can specify which users in your own AWS Account have access to the queue. If the users are in different accounts, see Tutorial: Delegate Access Across AWS Accounts Using IAM Roles in the IAM User Guide.

The subset of actions included in * has expanded. For a list of allowed actions, see Amazon SQS API Permissions: Actions and Resource Reference (p. 154).

You can specify the resource using the Amazon Resource Name (ARN), the standard means of specifying resources in IAM policies. For information about the ARN format for Amazon SQS queues, see Amazon Simple Queue Service Resource and Operations (p. 133).

For example, according to the Amazon SQS policy in the preceding figure, anyone who possesses the security credentials for AWS Account 1 or AWS Account 2 can access queue_xyz. In addition, Users Bob and Susan in your own AWS Account (with ID 123456789012) can access the queue.

Before the introduction of IAM, Amazon SQS automatically gave the creator of a queue full control over the queue (that is, access to all of the possible Amazon SQS actions on that queue). This is no longer true, unless the creator uses AWS security credentials. Any user who has permissions to create a queue must also have permissions to use other Amazon SQS actions in order to do anything with the created queues.

The following is an example policy that allows a user to use all Amazon SQS actions, but only with queues whose names are prefixed with the literal string bob_queue_.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": "sqs:*",
        "Resource": "arn:aws:sqs:*:123456789012:bob_queue_*"
    }]
}
```

For more information, see Using Identity-Based Policies (IAM) Policies for Amazon SQS (p. 137), and Identities (Users, Groups, and Roles) in the IAM User Guide.

Resource-Based Policies

Other AWS services, such as Amazon S3, support resource-based permissions policies. For example, to manage access permissions for an S3 bucket, you can attach a policy the S3 bucket.

Amazon SQS doesn't support resource-level permissions in identity-based policies (attached to a user or role), in which you can specify resources on which users are allowed to perform specified actions. For more information, see Overview of AWS IAM Permissions in the IAM User Guide.

Specifying Policy Elements: Actions, Effects, Resources, and Principals

For each Amazon Simple Queue Service resource (p. 133), the service defines a set of API operations. To grant permissions for these API operations, Amazon SQS defines a set of actions that you can specify in a policy.

Note
Performing an API operation can require permissions for more than one action. When granting permissions for specific actions, you also identify the resource for which the actions are allowed or denied.
The following are the most basic policy elements:

- **Resource** – In a policy, you use an Amazon Resource Name (ARN) to identify the resource to which the policy applies.

- **Action** – You use action keywords to identify resource operations that you want to allow or deny. For example, the `sqs:CreateQueue` permission allows the user to perform the Amazon Simple Queue Service `CreateQueue` operation.

- **Effect** – You specify the effect when the user requests the specific action—this can be either allow or deny. If you don’t explicitly grant access to a resource, access is implicitly denied. You can also explicitly deny access to a resource, which you might do to make sure that a user can’t access it, even if a different policy grants access.

- **Principal** – In identity-based policies (IAM policies), the user that the policy is attached to is the implicit principal. For resource-based policies, you specify the user, account, service, or other entity that you want to receive permissions (applies to resource-based policies only).

To learn more about Amazon SQS policy syntax and descriptions, see [AWS IAM Policy Reference](https://docs.aws.amazon.com/IAM/latest/userguide/access-policies-writing-policies.html) in the [IAM User Guide](https://docs.aws.amazon.com/IAM/latest/userguide/intro.html).

For a table of all Amazon Simple Queue Service API actions and the resources that they apply to, see [Amazon SQS API Permissions: Actions and Resource Reference](https://docs.aws.amazon.com/AWSSimpleQueueService/latest/APIReference/ApiReference-2012-11-01.html) (p. 154).

### Specifying Conditions in a Policy

When you grant permissions, you can use the Amazon SQS access policy language to specify the conditions for when a policy should take effect. For example, you might want a policy to be applied only after a specific date. For more information about specifying conditions in a policy language, see [Condition](https://docs.aws.amazon.com/IAM/latest/userguide/access-policies-writing-policies.html#condition) in the [IAM User Guide](https://docs.aws.amazon.com/IAM/latest/userguide/intro.html).

To express conditions, you use predefined condition keys. There are no condition keys specific to Amazon SQS. However, there are AWS-wide condition keys that you can use with Amazon SQS. Currently, Amazon SQS supports only a limited subset of the [condition keys available in IAM](https://docs.aws.amazon.com/IAM/latest/userguide/access-policies-writing-policies.html#condition):

- `aws:CurrentTime`
- `aws:EpochTime`
- `aws:SecureTransport`
- `aws:SourceArn`
- `aws:SourceIP`
- `aws:UserAgent`
- `aws:MultiFactorAuthAge`
- `aws:MultiFactorAuthPresent`
- `aws:TokenAge`

### Using Identity-Based Policies (IAM) Policies for Amazon SQS

This topic provides examples of identity-based policies in which an account administrator can attach permissions policies to IAM identities (users, groups, and roles).

**Important**

We recommend that you first review the introductory topics that explain the basic concepts and options available for you to manage access to your Amazon Simple Queue Service resources. For
Using Amazon SQS and IAM Policies

There are two ways to give your users permissions to your Amazon SQS resources: using the Amazon SQS policy system and using the IAM policy system. You can use one or the other, or both. For the most part, you can achieve the same result with either one.

For example, the following diagram shows an IAM policy and an Amazon SQS policy equivalent to it. The IAM policy grants the rights to the Amazon SQS ReceiveMessage and SendMessage actions for the queue called queue_xyz in your AWS Account, and the policy is attached to users named Bob and Susan (Bob and Susan have the permissions stated in the policy). This Amazon SQS policy also gives Bob and Susan rights to the ReceiveMessage and SendMessage actions for the same queue.

**Note**
This example shows simple policies without conditions. You can specify a particular condition in either policy and get the same result.

There is one major difference between IAM and Amazon SQS policies: the Amazon SQS policy system lets you grant permission to other AWS Accounts, whereas IAM doesn’t.

It is up to you how you use both of the systems together to manage your permissions. The following examples show how the two policy systems work together.

- In the first example, Bob has both an IAM policy and an Amazon SQS policy that apply to his account. The IAM policy grants his account permission for the ReceiveMessage action on queue_xyz, whereas the Amazon SQS policy gives his account permission for the SendMessage action on the same queue. The following diagram illustrates the concept.

  If Bob sends a ReceiveMessage request to queue_xyz, the IAM policy allows the action. If Bob sends a SendMessage request to queue_xyz, the Amazon SQS policy allows the action.

- In the second example, Bob abuses his access to queue_xyz, so it becomes necessary to remove his entire access to the queue. The easiest thing to do is to add a policy that denies him access to all actions for the queue. This policy overrides the other two because an explicit deny always overrides an allow. For more information about policy evaluation logic, see Creating Custom Policies Using the Amazon SQS Access Policy Language (p. 144). The following diagram illustrates the concept.
You can also add an additional statement to the Amazon SQS policy that denies Bob any type of access to the queue. It has the same effect as adding an IAM policy that denies Bob access to the queue. For examples of policies that cover Amazon SQS actions and resources, see Customer-Managed Policy Examples (p. 141). For more information about writing Amazon SQS policies, see Creating Custom Policies Using the Amazon SQS Access Policy Language (p. 144).

Permissions Required to Use the Amazon SQS Console

A user who wants to work with the Amazon SQS console must have the minimum set of permissions to work with the Amazon SQS queues in the user's AWS account. For example, the user must have the permission to call the `ListQueues` action to be able to list queues, or the permission to call the `CreateQueue` action to be able to create queues. In addition to Amazon SQS permissions, to subscribe an Amazon SQS queue to an Amazon SNS topic, the console also requires permissions for Amazon SNS actions.

If you create an IAM policy that is more restrictive than the minimum required permissions, the console might not function as intended for users with that IAM policy.

You don't need to allow minimum console permissions for users that make calls only to the AWS CLI or Amazon SQS API actions.

AWS-Managed (Predefined) Policies for Amazon SQS

AWS addresses many common use cases by providing standalone AWS-managed IAM policies. These AWS-managed policies simplify working with permissions by granting the permissions necessary for common use cases. For more information, see AWS Managed Policies in the IAM User Guide.

The following AWS-managed policies (that you can attach to users in your account) are specific to Amazon SQS:

- **AmazonSQSReadOnlyAccess** – Grants read-only access to Amazon SQS queues using the AWS Management Console.
- **AmazonSQSFullAccess** – Grants full access to Amazon SQS queues using the AWS Management Console.

You can search and review available policies on the IAM console. You can also create your own custom IAM policies to allow permissions for Amazon SQS actions and queues. You can attach these custom policies to the IAM users or groups that require permissions.

Writing Amazon SQS Policies

The following examples provide an introductory breakdown of a permissions policy.
Example 1: Allow a User to Create Queues

In the following example, we create a policy for Bob that lets him access all Amazon SQS actions, but only with queues whose names are prefixed with the literal string `bob_queue_`. Amazon SQS doesn't automatically grant the creator of a queue permissions to use the queue. Therefore, we must explicitly grant Bob permissions to use all Amazon SQS actions in addition to `CreateQueue` action in the IAM policy.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": "sqs:*",
        "Resource": "arn:aws:sqs:*:123456789012:bob_queue_*"
    }]
}
```

Example 2: Allow Developers to Write Messages to a Shared Queue

In the following example, we create a group for developers and attach a policy that lets the group use the Amazon SQS `SendMessage` action, but only with the queue that belongs to the specified AWS account and is named `CompanyTestQueue`.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": "sqs:SendMessage",
        "Resource": "arn:aws:sqs:*:123456789012:CompanyTestQueue"
    }]
}
```

Example 3: Allow Managers to Get the General Size of Queues

In the following example, we create a group for managers and attach a policy that lets the group use the Amazon SQS `GetQueueAttributes` action with all of the queues that belong to the specified AWS account.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": "sqs:GetQueueAttributes",
        "Resource": "*"
    }]
}
```

Example 4: Allow a Partner to Send Messages to a Specific Queue

You can accomplish this task using an Amazon SQS policy or an IAM policy. If your partner has an AWS account, it might be easier to use an Amazon SQS policy. However, any user in the partner’s company who possesses the AWS security credentials can send messages to the queue. If you want to limit access to a particular user or application, you must treat the partner like a user in your own company and use an IAM policy instead of an Amazon SQS policy.

This example performs the following actions:
1. Create a group called WidgetCo to represent the partner company.
2. Create a user for the specific user or application at the partner's company who needs access.
3. Add the user to the group.
4. Attach a policy that gives the group access only to the SendMessage action for only the queue named WidgetPartnerQueue.

```json
{
   "Version": "2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": "sqs:SendMessage",
      "Resource": "arn:aws:sqs:*:123456789012:WidgetPartnerQueue"
   }]
}
```

### Customer-Managed Policy Examples

This section shows example policies for common Amazon SQS use cases.

You can use the console to verify the effects of each policy as you attach the policy to the user. Initially, the user doesn't have permissions and won't be able to do anything in the console. As you attach policies to the user, you can verify that the user can perform various actions in the console.

We recommend that you use two browser windows: one to grant permissions and the other to sign into the AWS Management Console using the user's credentials and verify permissions as you grant them to the user.

**Example 1: Grant One Permission to One AWS Account**

The following example policy grants AWS account number 111122223333 the SendMessage permission for the queue named 444455556666/queue1 in the US East (Ohio) region.

```json
{
   "Version": "2012-10-17",
   "Id": "Queue1_Policy_UUID",
   "Statement": [{
      "Sid": "Queue1_SendMessage",
      "Effect": "Allow",
      "Principal": {
         "AWS": ["111122223333"
      ],
      "Action": "sqs:SendMessage",
      "Resource": "arn:aws:sqs:us-east-2:444455556666:queue1"
   }]
}
```

**Example 2: Grant Two Permissions to One AWS Account**

The following example policy grants AWS account number 111122223333 both the SendMessage and ReceiveMessage permission for the queue named 444455556666/queue1.

```json
{
   "Version": "2012-10-17",
   "Id": "Queue1_Policy_UUID",
   "Statement": [{
      "Sid": "Queue1_SendMessage",
      "Effect": "Allow",
      "Principal": {
         "AWS": ["111122223333"
      ],
      "Action": "sqs:SendMessage",
      "Resource": "arn:aws:sqs:us-east-2:444455556666:queue1"
   },
   "Sid": "Queue1_ReceiveMessage",
   "Effect": "Allow",
   "Principal": {
      "AWS": ["111122223333"
   ],
   "Action": "sqs:ReceiveMessage",
   "Resource": "arn:aws:sqs:us-east-2:444455556666:queue1"
   }]
}
```
"Statement": [{
    "Sid":"Queue1_Send_Receive",
    "Effect": "Allow",
    "Principal": {
        "AWS": [
            "111122223333"
        ]
    },
    "Action": [
        "sqs:SendMessage",
        "sqs:ReceiveMessage"
    ],
    "Resource": "arn:aws:sqs:*:444455556666:queue1"
}]

Example 3: Grant All Permissions to Two AWS Accounts

The following example policy grants two different AWS accounts numbers (111122223333 and 444455556666) permission to use all actions to which Amazon SQS allows shared access for the queue named 123456789012/queue1 in the US East (Ohio) region.

{
    "Version": "2012-10-17",
    "Id": "Queue1_Policy_UUID",
    "Statement": [{
        "Sid":"Queue1_AllActions",
        "Effect": "Allow",
        "Principal": {
            "AWS": [
                "111122223333",
                "444455556666"
            ]
        },
        "Action": "sqs:*",
        "Resource": "arn:aws:sqs:us-east-2:123456789012:queue1"
    }]
}

Example 4: Grant Cross-Account Permissions to a Role and a User Name

The following example policy grants role1 and username1 under AWS account number 111122223333 cross-account permission to use all actions to which Amazon SQS allows shared access for the queue named 123456789012/queue1 in the US East (Ohio) region.

{
    "Version": "2012-10-17",
    "Id": "Queue1_Policy_UUID",
    "Statement": [{
        "Sid":"Queue1_AllActions",
        "Effect": "Allow",
        "Principal": {
            "AWS": [
                "arn:aws:iam::111122223333:role/role1",
                "arn:aws:iam::111122223333:user/username1"
            ]
        },
        "Action": "sqs:*",
        "Resource": "arn:aws:sqs:us-east-2:123456789012:queue1"
    }]
}
### Example 5: Grant a Permission to All Users

The following example policy grants all users ReceiveMessage permission for the queue named `111122223333/queue1`.

```json
{
    "Version": "2012-10-17",
    "Id": "Queue1_Policy_UUID",
    "Statement": [{
        "Sid": "Queue1_AnonymousAccess_ReceiveMessage",
        "Effect": "Allow",
        "Principal": "*",
        "Action": "sqs:ReceiveMessage",
        "Resource": "arn:aws:sqs:*:111122223333:queue1"
    }]
}
```

### Example 6: Grant a Time-Limited Permission to All Users

The following example policy grants all users ReceiveMessage permission for the queue named `111122223333/queue1`, but only between 12:00 p.m. (noon) and 3:00 p.m. on January 31, 2009.

```json
{
    "Version": "2012-10-17",
    "Id": "Queue1_Policy_UUID",
    "Statement": [{
        "Sid": "Queue1_AnonymousAccess_ReceiveMessage_TimeLimit",
        "Effect": "Allow",
        "Principal": "*",
        "Action": "sqs:ReceiveMessage",
        "Resource": "arn:aws:sqs:*:111122223333:queue1",
        "Condition": {
            "DateGreaterThan": {
                "aws:CurrentTime": "2009-01-31T12:00Z"
            },
            "DateLessThan": {
                "aws:CurrentTime": "2009-01-31T15:00Z"
            }
        }
    }]
}
```

### Example 7: Grant All Permissions to All Users in a CIDR Range

The following example policy grants all users permission to use all possible Amazon SQS actions that can be shared for the queue named `111122223333/queue1`, but only if the request comes from the `192.168.143.0/24` CIDR range.

```json
{
    "Version": "2012-10-17",
    "Id": "Queue1_Policy_UUID",
    "Statement": [{
        "Sid": "Queue1_AnonymousAccess_AllActions_WhitelistIP",
        "Effect": "Allow",
        "Principal": "*",
        "Action": "sqs:*",
        "Resource": "arn:aws:sqs:*:111122223333:queue1",
        "Condition": {
            "IpAddress": {
                "aws:SourceIp": "192.168.143.0/24"
            }
        }
    }]
}
```
Example 8: Whitelist and Blacklist Permissions for Users in Different CIDR Ranges

The following example policy has two statements:

- The first statement grants all users in the 192.168.143.0/24 CIDR range (except for 192.168.143.188) permission to use the SendMessage action for the queue named 111122223333/queue1.
- The second statement blacklists all users in the 10.1.2.0/24 CIDR range from using the queue.

```json
{
  "Version": "2012-10-17",
  "Id": "Queue1_Policy_UUID",
  "Statement": [{
    "Sid": "Queue1_AnonymousAccess_SendMessage_IPLimit",
    "Effect": "Allow",
    "Principal": "*",
    "Action": "sqs:SendMessage",
    "Resource": "arn:aws:sqs:*:111122223333:queue1",
    "Condition": {
      "IpAddress": {
        "aws:SourceIp": "192.168.143.0/24"
      },
      "NotIpAddress": {
        "aws:SourceIp": "192.168.143.188/32"
      }
    }
  }, {
    "Sid": "Queue1_AnonymousAccess_AllActions_IPLimit_Deny",
    "Effect": "Deny",
    "Principal": "*",
    "Action": "sqs:*",
    "Resource": "arn:aws:sqs:*:111122223333:queue1",
    "Condition": {
      "IpAddress": {
        "aws:SourceIp": "10.1.2.0/24"
      }
    }
  }]
}
```

Creating Custom Policies Using the Amazon SQS Access Policy Language

If you want to allow Amazon SQS access based only on an AWS account ID and basic permissions (such as for SendMessage or ReceiveMessage), you don't need to write your own policies. You can just use the Amazon SQS AddPermission action.

If you want to explicitly deny or allow access based on more specific conditions (such as the time the request comes in or the IP address of the requester), you need to write your own Amazon SQS policies and upload them to the AWS system using the Amazon SQS SetQueueAttributes action.

Key Concepts

To write your own policies, you must be familiar with JSON and a number of key concepts.
allow

The result of a statement (p. 146) that has effect (p. 145) set to allow.

action

The activity that the principal (p. 146) has permission to perform, typically a request to AWS.

default-deny

The result of a statement (p. 146) that that has no allow (p. 145) or explicit deny (p. 145) settings.

condition

Any restriction or detail about a permission (p. 145). Typical conditions are related to date and time and IP addresses.

effect

The result that you want the statement (p. 146) of a policy (p. 145) to return at evaluation time. You specify the deny or allow value when you write the policy statement. There can be three possible results at policy evaluation time: default-deny (p. 145), allow (p. 145), and explicit deny (p. 145).

explicit deny

The result of a statement (p. 146) that has effect (p. 145) set to deny.

evaluation

The process that Amazon SQS uses to determine whether an incoming request should be denied or allowed based on a policy (p. 145).

issuer

The user who writes a policy (p. 145) to grant permissions to a resource. The issuer, by definition is always the resource owner. AWS does not permit Amazon SQS users to create policies for resources they don't own.

key

The specific characteristic that is the basis for access restriction.

permission

The concept of allowing or disallowing access to a resource using a condition (p. 145) and a key (p. 145).

policy

The document that acts as a container for one or more statements (p. 146).

Amazon SQS uses the policy to determine whether to grant access to a user for a resource.
principal

The user who receives permission (p. 145) in the policy (p. 145).

resource

The object that the principal (p. 146) requests access to.

statement

The formal description of a single permission, written in the access policy language as part of a broader policy (p. 145) document.

requester

The user who sends a request for access to a resource (p. 146).

Architecture

The following figure and table describe the access control for your Amazon SQS resources.

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>You, the resource owner.</td>
</tr>
<tr>
<td>2</td>
<td>Your resources contained within the AWS service (for example, Amazon SQS queues).</td>
</tr>
<tr>
<td>3</td>
<td>Your policies. It is a good practice to have one policy per resource. The AWS service itself provides an API you use to upload and manage your policies.</td>
</tr>
<tr>
<td>4</td>
<td>Requesters and their incoming requests to the AWS service.</td>
</tr>
</tbody>
</table>
The access policy language evaluation code. This is the set of code within the AWS service that evaluates incoming requests against the applicable policies and determines whether the requester is allowed access to the resource.

Process Workflow

The following figure and table describe the general workflow of access control with the Amazon SQS access policy language.

1. You write an Amazon SQS policy for your queue.
2. You upload your policy to AWS. The AWS service provides an API that you use to upload your policies. For example, you use the Amazon SQS `SetQueueAttributes` action to upload a policy for a particular Amazon SQS queue.
3. Someone sends a request to use your Amazon SQS queue.
4. Amazon SQS examines all available Amazon SQS policies and determines which ones are applicable.
5. Amazon SQS evaluates the policies and determines whether the requester is allowed to use your queue.
6. Based on the policy evaluation result, Amazon SQS either returns an `Access denied` error to the requester or continues to process the request.

Evaluation Logic

At evaluation time, Amazon SQS determines whether a request from someone other than the resource owner should be allowed or denied. The evaluation logic follows several basic rules:

- By default, all requests to use your resource coming from anyone but you are denied.
- An `allow` (p. 145) overrides any `default-deny` (p. 145).
- An `explicit deny` (p. 145) overrides any `allow` (p. 145).
- The order in which the policies are evaluated is not important.

The following figure and table describe in detail how Amazon SQS evaluates decisions about access permissions.
1 The decision starts with a **default-deny** (p. 145).

2 The enforcement code evaluates all the policies that are applicable to the request (based on the resource, principal, action, and conditions). The order in which the enforcement code evaluates the policies is not important.

3 The enforcement code looks for an **explicit deny** (p. 145) instruction that can apply to the request. If it finds even one, the enforcement code returns a decision of **deny** and the process finishes.

4 If no **explicit deny** (p. 145) instruction is found, the enforcement code looks for any **allow** (p. 145) instructions that can apply to the request. If it finds even one, the enforcement code returns a decision of **allow** and the process finishes (the service continues to process the request).

5 If no **allow** (p. 145) instruction is found, then the final decision is **deny** (because there is no **explicit deny** (p. 145) or **allow** (p. 145), this is considered a **default-deny** (p. 145).
Relationships Between Explicit and Default Denials

If an Amazon SQS policy doesn't directly apply to a request, the request results in a default-deny (p. 145). For example, if a user requests permission to use Amazon SQS but the only policy that applies to the user can use DynamoDB, the requests results in a default-deny (p. 145).

If a condition in a statement isn't met, the request results in a default-deny (p. 145). If all conditions in a statement are met, the request results in either an allow (p. 145) or an explicit deny (p. 145) based on the value of the effect (p. 145) element of the policy. Policies don't specify what to do if a condition isn't met, so the default result in this case is a default-deny (p. 145). For example, you want to prevent requests that come from Antarctica. You write Policy A1 that allows a request only if it doesn't come from Antarctica. The following diagram illustrates the Amazon SQS policy.

If a user sends a request from the U.S., the condition is met (the request is not from Antarctica), and the request results in an allow (p. 145). However, if a user sends a request from Antarctica, the condition isn't met and the request defaults to a default-deny (p. 145). You can change the result to an explicit deny (p. 145) by writing Policy A2 that explicitly denies a request if it comes from Antarctica. The following diagram illustrates the policy.

If a user sends a request from Antarctica, the condition is met and the request results in an explicit deny (p. 145).

The distinction between a default-deny (p. 145) and an explicit deny (p. 145) is important because an allow (p. 145) can overwrite the former but not the latter. For example, Policy B allows requests if they arrive on June 1, 2010. The following diagram compares combining this policy with Policy A1 and Policy A2.
In Scenario 1, Policy A1 results in a default-deny (p. 145) and Policy B results in an allow (p. 145) because the policy allows requests that come in on June 1, 2010. The allow (p. 145) from Policy B overrides the default-deny (p. 145) from Policy A1, and the request is allowed.

In Scenario 2, Policy B2 results in an explicit deny (p. 145) and Policy B results in an allow (p. 145). The explicit deny (p. 145) from Policy A2 overrides the allow (p. 145) from Policy B, and the request is denied.

**Amazon SQS Access Policy Examples**

The following are examples of typical Amazon SQS access control policies.
Example 1: Give Permission to One Account

The following example Amazon SQS policy gives AWS account 111122223333 permission to send to and receive from queue2 owned by AWS account 444455556666.

```
{  
  "Version": "2012-10-17",
  "Id": "UseCase1",
  "Statement": [{
    "Sid": "1",
    "Effect": "Allow",
    "Principal": {
      "AWS": [
        "111122223333"
      ]
    },
    "Action": [
      "sqs:SendMessage",
      "sqs:ReceiveMessage"
    ],
  }]
}
```

Example 2: Give Permission to One or More Accounts

The following example Amazon SQS policy gives one or more AWS accounts access to queues owned by your account for a specific time period. It is necessary to write this policy and to upload it to Amazon SQS using the SetQueueAttributes action because the AddPermission action doesn't permit specifying a time restriction when granting access to a queue.

```
{  
  "Version": "2012-10-17",
  "Id": "UseCase2",
  "Statement": [{
    "Sid": "1",
    "Effect": "Allow",
    "Principal": {
      "AWS": [
        "111122223333",
        "444455556666"
      ]
    },
    "Action": [
      "sqs:SendMessage",
      "sqs:ReceiveMessage"
    ],
    "Condition": {
      "DateLessThan": {
        "AWS:CurrentTime": "2009-06-30T12:00Z"
      }
    }
  }]
}
```

Example 3: Give Permission to Requests from Amazon EC2 Instances

The following example Amazon SQS policy gives access to requests that come from Amazon EC2 instances. This example builds on the "Example 2: Give Permission to One or More Accounts (p. 151)" example: it restricts access to before June 30, 2009 at 12 noon (UTC), it restricts access to the IP range 10.52.176.0/24. It is necessary to write this policy and to upload it to Amazon SQS using the
**SetQueueAttributes** action because the **AddPermission** action doesn't permit specifying an IP address restriction when granting access to a queue.

```json
{"Version": "2012-10-17",
 "Id": "UseCase3",
 "Statement": [
  {
   "Sid": "1",
   "Effect": "Allow",
   "Principal": {
    "AWS": ["111122223333"
   ]
  },
  
  "Action": [
   "sqs:SendMessage",
   "sqs:ReceiveMessage"
  ],
  "Condition": {
   "DateLessThan": {
    "AWS:CurrentTime": "2009-06-30T12:00Z"
   },
   "IpAddress": {
    "AWS:SourceIp": "10.52.176.0/24"
   }
  }
 ]
}
```

**Example 4: Deny Access to a Specific Account**

The following example Amazon SQS policy denies a specific AWS account access to your queue. This example builds on the "**Example 1: Give Permission to One Account (p. 151)**" example: it denies access to the specified AWS account. It is necessary to write this policy and to upload it to Amazon SQS using the **SetQueueAttributes** action because the **AddPermission** action doesn't permit deny access to a queue (it allows only granting access to a queue).

```json
{"Version": "2012-10-17",
 "Id": "UseCase4",
 "Statement": [
  {
   "Sid": "1",
   "Effect": "Deny",
   "Principal": {
    "AWS": ["111122223333"
   ]
  },
  
  "Action": [
   "sqs:SendMessage",
   "sqs:ReceiveMessage"
  ],
 ]
}
```

**Using Temporary Security Credentials**

In addition to creating IAM users with their own security credentials, IAM also allows you to grant temporary security credentials to any user, allowing the user to access your AWS services and resources.
You can manage users who have AWS accounts (IAM users). You can also manage users for your system who do not have AWS accounts (federated users). In addition, applications that you create to access your AWS resources can also be considered to be "users."

You can use these temporary security credentials to make requests to Amazon SQS. The API libraries compute the necessary signature value using those credentials to authenticate your request. If you send requests using expired credentials, Amazon SQS denies the request.

**Note**
You can't set a policy based on temporary credentials.

**To get started with temporary security credentials**

1. Use IAM to create temporary security credentials:
   - Security token
   - Access Key ID
   - Secret Access Key
2. Prepare your string to sign with the temporary Access Key ID and the security token.
3. Use the temporary Secret Access Key instead of your own Secret Access Key to sign your Query API request.

**Note**
When you submit the signed Query API request, use the temporary Access Key ID instead of your own Access Key ID and to include the security token. For more information on IAM support for temporary security credentials, see [Granting Temporary Access to Your AWS Resources](https://docs.aws.amazon.com/IAM/latest/userguide/iam-temporary-access.html) in the IAM User Guide.

**To call an Amazon SQS Query API action using temporary security credentials**


IAM returns a security token, an Access Key ID, and a Secret Access Key.

2. Prepare your query using the temporary Access Key ID instead of your own Access Key ID and include the security token. Sign your request using the temporary Secret Access Key instead of your own.
3. Submit your signed query string with the temporary Access Key ID and the security token.

The following example demonstrates how to use temporary security credentials to authenticate an Amazon SQS request. How you structure `AUTHPARAMS` depends on how you sign your API request. For information about `AUTHPARAMS` in Signature Version 4, see [Examples of Signed Signature Version 4 Requests](https://docs.aws.amazon.com/general/latest/gr/sigv4-examples.html).

```
http://sqs.us-east-2.amazonaws.com/
?Action=CreateQueue
&DefaultVisibilityTimeout=40
&QueueName=testQueue
&Attribute.1.Name=VisibilityTimeout
&Attribute.1.Value=40
&Version=2012-11-05
&Expires=2015-12-18T22%3A52%3A43PST
&SecurityToken=EXAMPLE_SECURITY_TOKEN
&AWSAccessKeyId=EXAMPLE_ACCESS_KEY_ID_PROVIDED_BY_AWS_SECURITY_TOKEN_SERVICE
&AUTHPARAMS
```

The following example uses Temporary Security Credentials to send two messages with `SendMessageBatch`.
Amazon SQS API Permissions: Actions and Resource Reference

When you set up Access Control (p. 132) and write permissions policies that you can attach to an IAM identity, you can use the following table as a reference. The list includes each Amazon Simple Queue Service API operation, the corresponding actions for which you can grant permissions to perform the action, and the AWS resource for which you can grant the permissions.

Specify the actions in the policy’s `Action` field, and the resource value in the policy’s `Resource` field. To specify an action, use the `sqs:` prefix followed by the API operation name (for example, `sqs:CreateQueue`).

Currently, Amazon SQS supports only a limited subset of the condition keys available in IAM:

- `aws:CurrentTime`
- `aws:EpochTime`
- `aws:SecureTransport`
- `aws:SourceArn`
- `aws:SourceIP`
- `aws:UserAgent`
- `aws:MultiFactorAuthAge`
- `aws:MultiFactorAuthPresent`
- `aws:TokenAge`

**Amazon Simple Queue Service API and Required Permissions for Actions**

**AddPermission**

- **Action(s):** `sqs:AddPermission`
- **Resource:** `arn:aws:sqs:region:account_id:queue_name`

**ChangeMessageVisibility**

- **Action(s):** `sqs:ChangeMessageVisibility`
- **Resource:** `arn:aws:sqs:region:account_id:queue_name`

**ChangeMessageVisibilityBatch**

- **Action(s):** `sqs:ChangeMessageVisibilityBatch`
  - **Action(s)**: sqs:CreateQueue

  - **Action(s)**: sqs:DeleteMessage

  - **Action(s)**: sqs:DeleteMessageBatch

  - **Action(s)**: sqs:DeleteQueue

  - **Action(s)**: sqs:GetQueueAttributes

  - **Action(s)**: sqs:GetQueueUrl

  - **Action(s)**: sqs:ListDeadLetterSourceQueues

  - **Action(s)**: sqs:ListQueues

  - **Action(s)**: sqs:PurgeQueue

  - **Action(s)**: sqs:ReceiveMessage

  - **Action(s)**: sqs:RemovePermission
Protecting Data Using Server-Side Encryption (SSE) and AWS KMS

This section provides an overview of using server-side encryption with AWS KMS and information about configuring IAM and AWS KMS key policies for SSE.

Topics
- Benefits of Server-Side Encryption (p. 156)
- What Does SSE for Amazon SQS Encrypt? (p. 157)
- Key Terms (p. 157)
- How Does the Data Key Reuse Period Work? (p. 158)
- How Do I Estimate My AWS KMS Usage Costs? (p. 159)
- What Permissions Do I Need to Use SSE? (p. 160)
- Getting Started with SSE (p. 161)
- Errors (p. 161)

Benefits of Server-Side Encryption

Server-side encryption (SSE) for Amazon SQS is available in the US East (N. Virginia), US East (Ohio), and US West (Oregon) regions. SSE lets you transmit sensitive data in encrypted queues. SSE protects the contents of messages in Amazon SQS queues using keys managed in the AWS Key Management Service (AWS KMS).

SSE encrypts messages as soon as Amazon SQS receives them. The messages are stored in encrypted form and Amazon SQS decrypts messages only when they are sent to an authorized consumer.

Important
All requests to queues with SSE enabled must use HTTPS and Signature Version 4. Some features of AWS services that can send notifications to Amazon SQS using the AWS Security Token Service AssumeRole API action are compatible with SSE but work only with standard queues:

- Auto Scaling Lifecycle Hooks
- AWS Lambda Dead-Letter Queues

Other features of AWS services or third-party services that send notifications to Amazon SQS aren't compatible with SSE, despite allowing you to set an encrypted queue as a target:

- Amazon CloudWatch Events
- AWS IoT Rule Actions
What Does SSE for Amazon SQS Encrypt?

SSE encrypts the body of a message in an Amazon SQS queue.

SSE doesn't encrypt the following:

- Queue metadata (queue name and attributes)
- Message metadata (message ID, timestamp, and attributes)
- Per-queue metrics

Encrypting a message makes its contents unavailable to unauthorized or anonymous users. This doesn't affect the normal functioning of Amazon SQS:

- A message is encrypted only if it is sent after the encryption of a queue is enabled. Amazon SQS doesn't encrypt backlogged messages.
- Any encrypted message remains encrypted even if the encryption of its queue is disabled.

Moving a message to a dead-letter queue (p. 61) does not affect its encryption:

- When Amazon SQS moves a message from an encrypted source queue to a unencrypted dead-letter queue, the message remains encrypted.
- When Amazon SQS moves a message from a unencrypted source queue to an encrypted dead-letter queue, the message remains unencrypted.

Key Terms

The following key terms can help you better understand the functionality of SSE. For detailed descriptions, see the Amazon Simple Queue Service API Reference.

Data Key

The data encryption key (DEK) responsible for encrypting the contents of Amazon SQS messages.
Data Key Reuse Period

The length of time, in seconds, for which Amazon SQS can reuse a data key to encrypt or decrypt messages before calling AWS KMS again. An integer representing seconds, between 60 seconds (1 minute) and 86,400 seconds (24 hours). The default is 300 (5 minutes). For more information, see How Does the Data Key Reuse Period Work? (p. 158).

Note
In the unlikely event of being unable to reach AWS KMS, Amazon SQS continues to use the cached data key until a connection is reestablished.

Customer Master Key ID

The alias, alias ARN, key ID, or key ARN of an AWS-managed customer master key (CMK) or a custom CMK—in your account or in another account. While the alias of the AWS-managed CMK for Amazon SQS is always alias/aws/sqs, the alias of a custom CMK can, for example, be alias/MyAlias. You can use these CMKs to protect the messages in Amazon SQS queues.

Note
Keep the following in mind:
- If you don't specify a custom CMK, Amazon SQS uses the AWS-managed CMK for Amazon SQS. For instructions on creating custom CMKs, see Creating Keys in the AWS Key Management Service Developer Guide.
- The first time you use the AWS Management Console to specify the AWS-managed CMK for Amazon SQS for a queue, AWS KMS creates the AWS-managed CMK for Amazon SQS.
- Alternatively, the first time you use the SendMessage or SendMessageBatch API action on a queue with SSE enabled, AWS KMS creates the AWS-managed CMK for Amazon SQS.

You can create CMKs, define the policies that control how CMKs can be used, and audit CMK usage using the Encryption Keys section of the AWS KMS console or using AWS KMS API actions. For more information about CMKs, see Customer Master Keys in the AWS Key Management Service Developer Guide. For more examples of CMK identifiers, see KeyId in the AWS Key Management Service API Reference.

Important
There are additional charges for using AWS KMS. For more information, see How Do I Estimate My AWS KMS Usage Costs? (p. 159) and AWS Key Management Service Pricing.

How Does the Data Key Reuse Period Work?

Amazon SQS uses a single customer master key (either the AWS-managed CMK for Amazon SQS or a custom CMK) to provide envelope encryption and decryption of multiple Amazon SQS messages during the data key reuse period. To make the most of the data key reuse period (p. 157), keep the following in mind:

- A shorter reuse period provides better security but results in more calls to AWS KMS, which might incur charges beyond the Free Tier.
- Although the data key is cached separately for encryption and for decryption, the reuse period applies to both copies of the data key.
- Principals (AWS accounts or IAM users) don't share data keys (messages sent by unique principals always get unique data keys). Thus, the volume of calls to AWS KMS is a multiple of the number of unique principals in use during the data key reuse period:
- When you send messages using the SendMessage or SendMessageBatch action, Amazon SQS typically calls the AWS KMS GenerateDataKey and Decrypt actions once per every data key reuse period.
Note
For each data key that AWS KMS generates, SSE calls the Decrypt action to verify the integrity of the data key before using it.

- When you receive messages using the ReceiveMessage action, Amazon SQS typically calls the AWS KMS Decrypt action once per every data key reuse period.

How Do I Estimate My AWS KMS Usage Costs?

To predict costs and better understand your AWS bill, you might want to know how often Amazon SQS uses your customer master key (CMK).

Note
Although the following formula can give you a very good idea of expected costs, actual costs might be higher because of the distributed nature of Amazon SQS.

To calculate the number of API requests \( R \) per queue, use the following formula:

\[
R = \frac{B}{D} \times (2 \times P + C)
\]

- \( B \) is the billing period (in seconds).
- \( D \) is the data key reuse period (p. 157) (in seconds).
- \( P \) is the number of producing principals that send to the Amazon SQS queue.
- \( C \) is the number of consuming principals that receive from the Amazon SQS queue.

Important
In general, producing principals incur double the cost of consuming principals. For more information, see How Does the Data Key Reuse Period Work? (p. 158). If the producer and consumer have different IAM users, the cost increases.

The following are example calculations. For exact pricing information, see AWS Key Management Service Pricing.

Example 1: Calculating the Number of AWS KMS API Calls for 2 Principals and 1 Queue

This example assumes the following:

- The billing period is January 1-31 (2,678,400 seconds).
- The data key reuse period is set to 5 minutes (300 seconds).
- There is 1 queue.
- There is 1 producing principal and 1 consuming principal.

\[
\frac{2,678,400}{300} \times (2 \times 1 + 1) = 26,784
\]

Example 2: Calculating the Number of AWS KMS API Calls for Multiple Producers and Consumers and 2 Queues

This example assumes the following:
What Permissions Do I Need to Use SSE?

Before you can use SSE, you must configure AWS KMS key policies to allow encryption of queues and encryption and decryption of messages. For examples and more information about AWS KMS permissions, see AWS KMS API Permissions: Actions and Resources Reference in the AWS Key Management Service Developer Guide.

Note
You can also manage permissions for KMS keys using IAM policies. For more information, see Using IAM Policies with AWS KMS.

While you can configure global permissions to send to and receive from Amazon SQS, AWS KMS requires explicitly naming the full ARN of CMKs in specific regions in the Resource section of an IAM policy.

You must also ensure that the key policies of the customer master key (CMK) allow the necessary permissions. To do this, name the principals which produce and consume encrypted messages in Amazon SQS as users in the CMK key policy.

Alternatively, you can specify the required AWS KMS actions and CMK ARN in an IAM policy assigned to the principals which produce and consume encrypted messages in Amazon SQS. For more information, see Managing Access to AWS KMS CMKs in the AWS Key Management Service Developer Guide.

Example 1: Allow a User to Send Single or Batched Messages to an Encrypted Queue

The producer must have the kms:GenerateDataKey and kms:Decrypt permissions for the customer master key (CMK).

```json
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "kms:GenerateDataKey",
      "kms:Decrypt"
    ],
    "Resource": "arn:aws:kms:us-east-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
  }, {
    "Effect": "Allow",
    "Action": [
      "sqs:SendMessage",
      "sqs:SendMessageBatch"
    ],
    "Resource": "arn:aws:sqs:*:123456789012:MyQueue"
  }]
}
```
Example 2: Allow a User to Receive Messages from an Encrypted Queue

The consumer must have the `kms:Decrypt` permission for any customer master key (CMK) that is used to encrypt the messages in the specified queue. If the queue acts as a dead-letter queue (p. 61), the consumer must also have the `kms:Decrypt` permission for any CMK that is used to encrypt the messages in the source queue.

```json
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": ["kms:Decrypt"]
    }, {
        "Effect": "Allow",
        "Action": ["sqs:ReceiveMessage"]
    }, {
        "Resource": "arn:aws:kms:us-east-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    }, {
        "Resource": "arn:aws:sqs:*:123456789012:MyQueue"
    }
}
```

Getting Started with SSE

For information about how to manage SSE using the AWS Management Console or using API actions, see the following tutorials:

- Creating an Amazon SQS queue with SSE (p. 20)
- Configuring SSE for an existing Amazon SQS queue (p. 24)

You can enable and disable SSE for an Amazon SQS queue using the following API actions.

<table>
<thead>
<tr>
<th>Task</th>
<th>API Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a new queue with SSE enabled.</td>
<td>CreateQueue</td>
</tr>
<tr>
<td>Enable SSE for an existing queue.</td>
<td>SetQueueAttributes</td>
</tr>
<tr>
<td>Determine whether SSE is enabled for an existing queue.</td>
<td>GetQueueAttributes</td>
</tr>
</tbody>
</table>

Errors

When you work with Amazon SQS and AWS KMS, you might encounter errors. The following list describes the errors and possible troubleshooting solutions.

**KMSAccessDeniedException**

The ciphertext references a key that doesn't exist or that you don't have access to.

HTTP Status Code: 400
KMSDisabledException
The request was rejected because the specified CMK isn't enabled.
HTTP Status Code: 400

KMSInvalidStateException
The request was rejected because the state of the specified resource isn't valid for this request.
For more information, see How Key State Affects Use of a Customer Master Key in the AWS Key Management Service Developer Guide.
HTTP Status Code: 400

KMSNotFoundException
The request was rejected because the specified entity or resource can't be found.
HTTP Status Code: 400

KMSOptInRequired
The AWS access key ID needs a subscription for the service.
HTTP Status Code: 403

KMSThrottlingException
The request was denied due to request throttling. For more information about throttling, see Limits in the AWS Key Management Service Developer Guide.
HTTP Status Code: 400
Working with Amazon SQS APIs

This section provides information about working with Amazon SQS APIs.

For detailed information about API actions (including parameters, errors, and examples) and data types, see the Amazon Simple Queue Service API Reference.

Topics
• Making API Requests (p. 163)
• Amazon SQS Batch API Actions (p. 176)

Making API Requests

Topics
• Endpoints (p. 164)
• Making Query Requests (p. 164)
• Request Authentication (p. 166)
• Responses (p. 172)
• Shared Queues (p. 173)
• Programming Languages (p. 175)

This section describes how to make requests to Amazon SQS. The topics acquaint you with the basic differences between the interfaces, the components of a request, how to authenticate a request, and the content of responses.

We also provide SDKs that enable you to access Amazon SQS from your preferred programming language. The SDKs contain functionality that automatically takes care of tasks such as:

• Cryptographically signing your service requests
• Retrying requests
• Handling error responses

For a list of available SDKs, see Tools for Amazon Web Services

Important
As of August 8, 2011, Amazon SQS no longer supports SOAP requests.
Endpoints

For information about Amazon SQS regions and endpoints, see Regions and Endpoints in the Amazon Web Services General Reference.

Every Amazon SQS endpoint is entirely independent. For example, two queues named MyQueue, one in sqs.us-east-1.amazonaws.com and one in sqs.eu-west-1.amazonaws.com, would be completely independent and would not share any data. Queue names and queue URLs are case-sensitive.

The following are general examples of query requests that create queues in different regions. The structure of AUTHPARAMS depends on how you sign your API request.

Example of Creating a Queue in EU (Ireland)

```
http://sqs.eu-west-1.amazonaws.com/
?Action=CreateQueue
&DefaultVisibilityTimeout=40
&QueueName=MyQueue
&Version=2012-11-05
&AUTHPARAMS
```

Making Query Requests

Topics

- Structure of a GET Request (p. 164)
- Structure of a POST Request (p. 165)
- Related Topics (p. 166)

Amazon SQS supports Query requests for calling service actions. Query requests are simple HTTP or HTTPS requests, using the GET or POST method. Query requests must contain an Action parameter to indicate the action to be performed. The response is an XML document that conforms to a schema.

Structure of a GET Request

This guide presents the Amazon SQS GET requests as URLs, which can be used directly in a browser. The URL consists of:

- **Endpoint**—The resource the request is acting on (in the case of Amazon SQS, the endpoint is a queue)
- **Action**—The action you want to perform on the endpoint; for example: sending a message
- **Parameters**—Any request parameters

The following is an example GET request to send a message to an Amazon SQS queue.

How you structure the AUTHPARAMS depends on how you're signing your API request. For information about AUTHPARAMS in Signature Version 4, see Examples of Signed Signature Version 4 Requests.

```
%20Message%20Text&Version=2012-11-05&AUTHPARAMS
```

**Important**

Because the GET requests are URLs, you must URL encode the parameter values. For example, in the preceding example request, the value for the MessageBody parameter is actually Your
Message Text. However, spaces are not allowed in URLs, so each space is URL encoded as "%20". The rest of the example has not been URL encoded to make it easier for you to read.

**Note**
Queue names and queue URLs are case-sensitive.

To make the GET examples even easier to read, this guide presents them in the following parsed format.

```
http://sqs.us-east-2.amazonaws.com/123456789012/queue1
?Action=SendMessage
&MessageBody=Your%20Message%20Text
&Version=2012-11-05
&Expires=2011-10-15T12:00:00Z
&AUTHPARAMS
```

**Note**
In the example Query requests we present in this guide, we use a false AWS Access Key ID and false signature, each with `EXAMPLE` appended. We do this to indicate that you shouldn’t expect the signature in the example to be accurate based on the request parameters presented in the example. The one exception to this is in the instructions for creating Query request signatures. The example there shows a real signature based on a particular AWS Access Key ID we specify and the request parameters in the example (for more information, see Query Request Authentication (p. 171)).

In Amazon SQS, all parameters except `MessageBody` always have values that have no spaces. The value you provide for `MessageBody` in `SendMessage` requests can have spaces. In this guide, any example `SendMessage` Query requests with a `MessageBody` that includes spaces is displayed with the spaces URL encoded (as `%20`). For clarity, the rest of the URL isn't displayed in a URL encoded format.

The first line represents the endpoint of the request. This is the resource the request acts on. The preceding example acts on a queue, so the request's endpoint is the queue's identifier, known as the queue URL. For more details about the queue URL, see Queue Name and URL (p. 57).

After the endpoint is a question mark (?), which separates the endpoint from the parameters. Each parameter is separated by an ampersand (&).

The `Action` parameter indicates the action to perform (for a list of the actions, see API Actions in the Amazon SQS API Reference). For a list of the other parameters that are common to all Query requests, see Common Parameters in the Amazon SQS API Reference.

### Structure of a POST Request

Amazon SQS also accepts POST requests. With a POST request, you send the query parameters as a form in the HTTP request body as described in the following procedure.

How you structure the AUTHPARAMS depends on how you're signing your API request. For information about AUTHPARAMS in Signature Version 4, see Examples of Signed Signature Version 4 Requests.

**To create a POST request**

1. Assemble the query parameter names and values into a form.

   This means you put the parameters and values together like you would for a GET request (with an ampersand separating each name-value pair). The following example shows a `SendMessage` request with the line breaks we use in this guide to make the information easier to read.

```
Action=SendMessage
&MessageBody=Your Message Text
```
2. Form-URL-encode the form according to the *Form Submission* section of the HTML specification (for more information, see [http://www.w3.org/MarkUp/html-spec/html-spec_toc.html#SEC8.2.1](http://www.w3.org/MarkUp/html-spec/html-spec_toc.html#SEC8.2.1)).

```
Action=SendMessage
&MessageBody=Your+Message+Text
&Version=2012-11-05
&Expires=2011-10-15T12%3A00%3A00Z
&AUTHPARAMS
```

3. Add the request signature to the form (for more information, see [Query Request Authentication (p. 171)]).

```
Action=SendMessage
&MessageBody=Your+Message+Text
&Version=2012-11-05
&Expires=2011-10-15T12%3A00%3A00Z
&AUTHPARAMS
```

4. Provide the resulting form as the body of the POST request.

5. Include the Content-Type HTTP header with the value set to application/x-www-form-urlencoded.

The following example shows the final POST request.

```
POST /queue1 HTTP/1.1
Host: sqs.us-east-2.amazonaws.com
Content-Type: application/x-www-form-urlencoded

Action=SendMessage
&MessageBody=Your+Message+Text
&Version=2012-11-05
&Expires=2011-10-15T12%3A00%3A00Z
&AUTHPARAMS
```

Amazon SQS requires no other HTTP headers in the request besides Content-Type. The authentication signature you provide is the same signature you'd provide if you sent a GET request (for information about the signature, see [Query Request Authentication (p. 171)]).

**Note**

Your HTTP client typically adds other items to the HTTP request as required by the version of HTTP the client uses. We don't include those additional items in the examples in this guide.

**Related Topics**

- [Query Request Authentication (p. 171)]
- [Responses (p. 172)]

**Request Authentication**

The topics in this section describe how Amazon SQS authenticates your requests. In this section you can learn about the basics of authentication, how your AWS account and access keys support authentication, and how to create a signature. This section also covers the request authentication requirements for Query requests.
What Is Authentication?

Authentication is a process for identifying and verifying who is sending a request. The following diagram shows a simplified version of an authentication process.

General Process of Authentication

1. The producer (sender) obtains the necessary credential.
2. The producer sends a request with the credential to the consumer (receiver).
The consumer uses the credential to verify whether the producer sent the request.

If authentication succeeds, the consumer processes the request. If no, the consumer rejects the request and responds accordingly.

During authentication, AWS verifies both the identity of the producer and whether the producer is registered to use services offered by AWS. If either test fails, the request isn't processed further.

The subsequent sections describe how Amazon SQS implements authentication to protect your data.

Your AWS Account

To access any web services offered by AWS, you must first create an AWS account at http://aws.amazon.com. An AWS account is simply an Amazon.com account that is enabled to use AWS products; you can use an existing Amazon.com account login and password when creating the AWS account.

Alternatively, you can create a new AWS-enabled Amazon.com account using a new login and password. The email address you provide as the account login must be valid. You must provide a credit card or other payment method to cover the charges for any AWS products you use.

From your AWS account you can view your AWS account activity and view usage reports.

For more information, see Step 1: Create an AWS Account (p. 4) and Step 2: Create an IAM User (p. 4).

Related Topics
• Your Access Keys (p. 168)

Your Access Keys

For API access, you need an access key ID and secret access key. Use IAM user access keys instead of AWS root account access keys. IAM lets you securely control access to AWS services and resources in your AWS account. For more information about creating access keys, see How Do I Get Security Credentials? in the AWS General Reference.

Related Topics
• HMAC-SHA Signatures (p. 168)
• Query Request Authentication (p. 171)

HMAC-SHA Signatures

The topics in this section describe how Amazon SQS uses HMAC-SHA signatures to authenticate Query requests.

Topics
• Required Authentication Information (p. 169)
• Basic Authentication Process (p. 169)
• About the String to Sign (p. 170)
• About the Time Stamp (p. 171)
• Java Sample Code for Base64 Encoding (p. 171)
Required Authentication Information

When accessing Amazon SQS using the Query API, you must provide the following items so the request can be authenticated:

- **AWS Access Key ID**—Your AWS account is identified by your Access Key ID, which AWS uses to look up your Secret Access Key.
- **Signature**—Each request must contain a valid HMAC-SHA request signature, or the request is rejected.
  
  You calculate the request signature using your Secret Access Key, which is a shared secret known only to you and AWS.
- **Date**—Each request must contain the time stamp of the request. You can provide an expiration date and time for the request instead of or in addition to the time stamp.

Related Topics

- Your Access Keys (p. 168)

Basic Authentication Process

Following is the series of tasks required to authenticate requests to AWS using an HMAC-SHA request signature. It’s assumed you have already created an AWS account and created an Access Key ID and Secret Access Key. For more information about those, see Your AWS Account (p. 168) and Your Access Keys (p. 168).

You perform the first three tasks.
Process for Authentication: Tasks You Perform

1. You construct a request to AWS.

2. You calculate a keyed-hash message authentication code (HMAC-SHA) signature using your Secret Access Key (for information about HMAC, see http://www.faqs.org/rfcs/rfc2104.html)

3. You include the signature and your Access Key ID in the request, and then send the request to AWS.

AWS performs the next three tasks.

4. AWS uses the Access Key ID to look up your Secret Access Key.

5. AWS generates a signature from the request data and the Secret Access Key using the same algorithm you used to calculate the signature you sent in the request.

6. If the signature generated by AWS matches the one you sent in the request, the request is considered authentic. If the comparison fails, the request is discarded, and AWS returns an error response.

About the String to Sign

Each AWS request you send must include an HMAC-SHA request signature calculated with your Secret Access Key. The details are covered in Query Request Authentication (p. 171).
About the Time Stamp

The time stamp (or expiration time) you use in the request must be a `dateTime` object, with the complete date plus hours, minutes, and seconds (for more information, see `http://www.w3.org/TR/xmlschema-2/#dateTime`). For example: 2007-01-31T23:59:59Z. Although it's not required, we recommend you provide the time stamp in the Coordinated Universal Time (Greenwich Mean Time) time zone.

If you specify a time stamp (instead of an expiration time), the request automatically expires 15 minutes after the time stamp (in other words, AWS does not process a request if the request time stamp is more than 15 minutes earlier than the current time on AWS servers). Make sure your server's time is set correctly.

**Important**

If you’re using .NET you must not send overly specific time stamps, due to different interpretations of how extra time precision should be dropped. To avoid overly specific time stamps, manually construct `dateTime` objects with no more than millisecond precision.

Java Sample Code for Base64 Encoding

Request signatures must be base64 encoded. The following Java sample code shows how to perform base64 encoding.

```java
package amazon.webservices.common;
/**
 * This class defines common routines for encoding data in AWS requests.
 */
public class Encoding {
    /**
     * Performs base64-encoding of input bytes.
     *
     * @param rawData * Array of bytes to be encoded.
     * @return * The base64 encoded string representation of rawData.
     */
    public static String EncodeBase64(byte[] rawData) {
        return Base64.encodeBytes(rawData);
    }
}
```

Java Sample Code for Calculating SHA256 Signatures

For an example of how to derive a Signature Version 4 Signing Key, see Deriving the Signing Key with Java.

**Note**

While both Signature Version 2 and Signature Version 4 support SHA256, only Signature Version 2 supports SHA1.

Query Request Authentication

When you programmatically call the functionality exposed by the Amazon SQS API, all calls sent to Amazon SQS must be signed. If you use an AWS SDK, the SDK handles the signing process for you so that you do not have to manually complete the tasks. On the other hand, if you submit a Query request over HTTP/HTTPS, then you must include a signature in every Query request.

Amazon SQS supports Signature Version 4. Signature Version 4 provides improved security and performance over previous versions. If you’re creating new applications that use Amazon SQS, then you should use Signature Version 4.
Responses

Topics
- Structure of a Successful Response (p. 172)
- Structure of an Error Response (p. 172)
- Related Topics (p. 173)

In response to an action request, Amazon SQS returns an XML data structure that contains the results of the request. This data conforms to the Amazon SQS schema. For more information, see the API version in the Amazon SQS API Reference.

Structure of a Successful Response

If the request succeeded, the main response element is named after the action, but with "Response" appended. For example, CreateQueueResponse is the response element returned for a successful CreateQueue request. This element contains the following child elements:

- ResponseMetadata, which contains the RequestId child element
- An optional element containing action-specific results; for example, the CreateQueueResponse element includes an element called CreateQueueResult

The XML schema describes the XML response message for each Amazon SQS action.

The following is an example of a successful response.

```
<CreateQueueResponse
 xmlns=http://sqs.us-east-2.amazonaws.com/doc/2012-11-05/
 xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
 xsi:type=CreateQueueResponse>
 <CreateQueueResult>
  <QueueUrl>
   http://sqs.us-east-2.amazonaws.com/770098461991/queue2
  </QueueUrl>
 </CreateQueueResult>
 <ResponseMetadata>
  <RequestId>cb919c0a-9bce-4afe-9b48-9bdf2412bb67</RequestId>
 </ResponseMetadata>
</CreateQueueResponse>
```

Structure of an Error Response

If a request is unsuccessful, the main response element is called ErrorResponse regardless of the action that was called. This element contains an Error element and a RequestId element. Each Error includes:

- A Type element that identifies whether the error was a producer or consumer error
- A Code element that identifies the type of error that occurred
- A Message element that describes the error condition in a human-readable form
- A Detail element that might give additional details about the error or might be empty

The following is an example of an error response.

```
<ErrorResponse>
</ErrorResponse>
```
Amazon SQS includes methods to share your queues so others can use them, using permissions set in an access control policy. A permission gives access to another user to use your queue in some particular way. A policy is the actual document that contains the permissions you've granted.

Amazon SQS offers two methods for setting a policy: a simple API and an advanced API. In the simple API, Amazon SQS generates an access control policy for you. In the advanced API, you create the access control policy.

**Simple API for Shared Queues**

The simple API for sharing a queue has two operations:

- AddPermission
- RemovePermission

With the simple API, Amazon SQS writes the policy in the required language for you based on the information you include in the AddPermission operation. However, the policy that Amazon SQS generates is limited in scope. You can grant permissions to principals, but you can't specify restrictions.

**Advanced API for Shared Queues**

With the advanced API, you write the policy yourself directly in the IAM policy language and upload the policy with the SetQueueAttributes operation. The advanced API allows you to deny access or to apply finer access restrictions (for example, based on time or based on IP address).
If you choose to write your own policies, you need to understand how policies are structured. For complete reference information about policies, see Creating Custom Policies Using the Amazon SQS Access Policy Language (p. 144). For examples of policies, see Amazon SQS Access Policy Examples (p. 150).

### Understanding Resource-Level Permissions

A permission is the type of access you give to a principal (the user receiving the permission). You give each permission a label that identifies that permission. If you want to delete that permission in the future, you use that label to identify the permission. If you want to see what permissions are on a queue, use the `GetQueueAttributes` operation. Amazon SQS returns the entire policy (containing all the permissions). Amazon SQS supports the permission types shown in the following table.

#### Note

To allow anonymous access, you must write your own policy.

<table>
<thead>
<tr>
<th>Permission</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>This permission type grants the following actions to a principal on a shared queue: change a message's visibility, delete messages, get a queue's attributes, get a queue's URL, receive messages, and send messages.</td>
</tr>
<tr>
<td>ChangeMessageVisibility</td>
<td><strong>This</strong> grants permission to extend or terminate the read lock timeout of a specified message. ChangeMessageVisibilityBatch inherits permissions associated with ChangeMessageVisibility. For more information about visibility timeout, see Visibility Timeout (p. 59). For more information, see the <code>ChangeMessageVisibility</code> operation.</td>
</tr>
<tr>
<td>DeleteMessage</td>
<td>This grants permission to delete messages from the queue. DeleteMessageBatch inherits permissions associated with DeleteMessage. For more information, see the <code>DeleteMessage</code> operation.</td>
</tr>
<tr>
<td>GetQueueAttributes</td>
<td>This grants permission to get all of the queue attributes except the policy, which can only be accessed by the queue's owner. For more information, see the <code>GetQueueAttributes</code> operation.</td>
</tr>
<tr>
<td>GetQueueUrl</td>
<td>This grants permission to get a queue's URL. For more information, see the <code>GetQueueUrl</code> operation.</td>
</tr>
<tr>
<td>ReceiveMessage</td>
<td>This grants permission to receive messages in the queue. For more information, see the <code>ReceiveMessage</code> operation.</td>
</tr>
<tr>
<td>SendMessage</td>
<td>This grants permission to send messages to the queue. SendMessageBatch inherits permissions associated with SendMessage. For more information, see the <code>SendMessage</code> operation.</td>
</tr>
</tbody>
</table>

#### Note

Setting permissions for `SendMessage`, `DeleteMessage`, or `ChangeMessageVisibility` also sets permissions for the corresponding batch versions of those actions: `SendMessageBatch`, `DeleteMessageBatch`, and `ChangeMessageVisibilityBatch`. Setting permissions explicitly on `SendMessageBatch`, `DeleteMessageBatch`, and `ChangeMessageVisibilityBatch` isn't allowed.

Permissions for each of the different permission types are considered separate permissions by Amazon SQS, even though * includes the access provided by the other permission types. For example, it's possible to grant both * and `SendMessage` permissions to a user, even though a * includes the access provided by `SendMessage`. 
This concept applies when you remove a permission. If a principal has only a * permission, requesting to remove a SendMessage permission does not leave the principal with an "everything but" permission. Instead, the request does nothing, because the principal did not previously possess an explicit SendMessage permission.

If you want to remove * and leave the principal with just the ReceiveMessage permission, first add the ReceiveMessage permission, then remove the * permission.

**Note**  
You give each permission a label that identifies that permission. If you want to delete that permission in the future, you use that label to identify the permission.

**Note**  
If you want to see what permissions are on a queue, use the GetQueueAttributes operation. The entire policy (containing all the permissions) is returned.

**Granting Anonymous Access to a Queue**

You can allow shared queue access to anonymous users. Such access requires no signature or Access Key ID.

To allow anonymous access you must write your own policy, setting the Principal to *. For information about writing your own policies, see Creating Custom Policies Using the Amazon SQS Access Policy Language (p. 144).

**Important**  
The queue owner is responsible for all costs associated with the queue. Thus, it's a good idea to limit anonymous access in some other way (for example, by time or by IP address).

**Programming Languages**

AWS provides libraries, sample code, tutorials, and other resources for software developers who prefer to build applications using language-specific APIs instead of Amazon SQS's Query API. These libraries provide basic functions (not included in Amazon SQS's Query API), such as request authentication, request retries, and error handling so you can get started more easily. Libraries and resources are available for the following languages:

- Go
- Java
- JavaScript
- PHP
- Python
- Ruby
- Windows and .NET
- C++

For mobile application development, see:

- AWS Mobile SDK for Android
- AWS Mobile SDK for iOS
- AWS SDK for Unity

**Note**  
There are also command-line tools available for interacting with Amazon SQS:
Amazon Simple Queue Service Developer Guide

Batch API Actions

Topics

- Maximum Message Size for SendMessageBatch (p. 176)
- Client-Side Buffering and Request Batching (p. 176)
- Increasing Throughput with Horizontal Scaling and Batching (p. 179)

From the 2011-10-01 API version of Amazon SQS, you can use batch functionality to send and delete messages and to change the message visibility timeout:

- To send up to ten messages at once, use the **SendMessageBatch** action.
- To delete up to ten messages with one API call, use the **DeleteMessageBatch** action.
- To change the visibility timeout value for up to ten messages, use the **ChangeMessageVisibilityBatch** action.

To reduce costs, take advantage of batch functionality using the Query API or a Software Development Kit (SDK) that supports the new Amazon SQS batch actions.

**Note**

The Amazon SQS console does not support batch API actions.

For details and examples of the following three batch API actions, see the *Amazon Simple Queue Service API Reference*:

- **ChangeMessageVisibilityBatch**
- **DeleteMessageBatch**
- **SendMessageBatch**

**Maximum Message Size for SendMessageBatch**

You can send a message as large as 262,144 bytes (256 KB) with **SendMessageBatch**. However, the total size of all the messages that you send in a single call to **SendMessageBatch** can't exceed 262,144 bytes (256 KB).

**Client-Side Buffering and Request Batching**

The AWS SDK for Java ([http://aws.amazon.com/sdkforjava/](http://aws.amazon.com/sdkforjava/)) includes a buffered asynchronous client, **AmazonSQSBufferedAsyncClient**, for accessing Amazon SQS. This new client allows for easier request batching by enabling client-side buffering, where calls made from the client are first buffered and then sent as a batch request to Amazon SQS.

Client-side buffering allows up to 10 requests to be buffered and sent as a batch request instead of sending each request separately. As a result, your cost of using Amazon SQS decreases as you reduce the number of requests sent to the service. **AmazonSQSBufferedAsyncClient** buffers both synchronous
and asynchronous calls. Batched requests and support for long polling can also help increase throughput (the number of messages transmitted per second). For more information, see Amazon SQS Long Polling (p. 73) and Increasing Throughput with Horizontal Scaling and Batching (p. 179).

Migrating from the asynchronous client, AmazonSQSAsyncClient, to the buffered asynchronous client, AmazonSQSBufferedAsyncClient, should require only minimal changes to your existing code. This is because AmazonSQSBufferedAsyncClient implements the same interface as AmazonSQSAsyncClient.

Note
The Amazon SQS Buffered Asynchronous Client doesn't currently support FIFO queues.

Getting Started with AmazonSQSBufferedAsyncClient

Before you begin using the example code in this section, you must first install the AWS SDK for Java and set up your AWS credentials. For instructions, see Getting Started in the AWS SDK for Java Developer Guide.

The following code example shows how to create a new AmazonSQSBufferedAsyncClient based on the AmazonSQSAsyncClient.

```java
// Create the basic Amazon SQS async client
AmazonSQSAsync sqsAsync = new AmazonSQSAsyncClient();

// Create the buffered client
AmazonSQSAsync bufferedSqs = new AmazonSQSBufferedAsyncClient(sqsAsync);
```

After you have created the new AmazonSQSBufferedAsyncClient, you can make calls to it as you do with the AmazonSQSAsyncClient, as the following code example demonstrates.

```java
CreateQueueRequest createRequest = new CreateQueueRequest().withQueueName("MyTestQueue");
CreateQueueResult res = bufferedSqs.createQueue(createRequest);

SendMessageRequest request = new SendMessageRequest();
String body = "test message_" + System.currentTimeMillis();
request.setMessageBody(body);
request.setQueueUrl(res.getQueueUrl());
SendMessageResult sendResult = bufferedSqs.sendMessage(request);

ReceiveMessageRequest receiveRq = new ReceiveMessageRequest()
    .withMaxNumberOfMessages(1)
    .withQueueUrl(queueUrl);
ReceiveMessageResult rx = bufferedSqs.receiveMessage(receiveRq);
```

Advanced Configuration

AmazonSQSBufferedAsyncClient is pre-configured with settings that work for most use cases. If you'd like to configure it yourself, you can use the QueueBufferConfig class to do so. Just create an instance of QueueBufferConfig with the settings you want and supply it to the AmazonSQSBufferedAsyncClient constructor, as the following sample code shows.

```java
// Create the basic Amazon SQS async client
AmazonSQSAsync sqsAsync = new AmazonSQSAsyncClient();

QueueBufferConfig config = new QueueBufferConfig()
    .withMaxInflightReceiveBatches(5)
    .withMaxDoneReceiveBatches(15);
```
// Create the buffered client
AmazonSQSAsync bufferedSqs = new AmazonSQSBufferedAsyncClient(sqsAsync, config);

The parameters you can use to configure QueueBufferConfig are as follows:

- **longPoll**—if this parameter is set to `true`, `AmazonSQSBufferedAsyncClient` attempts to use long-polling when consuming messages. The default value is `true`.

- **longPollWaitTimeoutSeconds**—the maximum amount of time, in seconds, that a receive message call blocks on the server waiting for messages to appear in the queue before returning with an empty receive result. This setting has no impact if long polling is disabled. The default value of this setting is 20 seconds.

- **maxBatchOpenMs**—the maximum amount of time, in milliseconds, that an outgoing call waits for other calls of the same type to batch with. The higher the setting, the fewer batches are required to perform the same amount of work. Of course, the higher the setting, the more the first call in a batch has to spend waiting. If this parameter is set to zero, submitted requests do not wait for other requests, effectively disabling batching. The default value of this setting is 200 milliseconds.

- **maxBatchSize**—the maximum number of messages that are batched together in a single request. The higher the setting, the fewer batches are required to carry out the same number of requests. The default value of this setting is 10 requests per batch, which is also the maximum batch size currently allowed by Amazon SQS.

- **maxBatchSizeBytes**—the maximum size of a message batch, in bytes, that the client attempts to send to Amazon SQS. The default value is 256 KB, which is also the maximum message and batch size currently allowed by Amazon SQS.

- **maxDoneReceiveBatches**—the maximum number of receive batches `AmazonSQSBufferedAsyncClient` prefetches and stores on the client side. The higher the setting, the more receive requests can be satisfied without having to make a call to Amazon SQS server. However, the more messages are pre-fetched, the longer they sit in the buffer, which means that their visibility timeout expires. If this parameter is set to zero, all pre-fetching of messages is disabled and messages are consumed only on demand. The default value is 10 batches.

- **maxInflightOutboundBatches**—the maximum number of active outbound batches that can be processed at the same time. The higher the setting, the faster outbound batches can be sent (subject to other limits, such as CPU or bandwidth). The higher the setting, the more threads are consumed by the `AmazonSQSBufferedAsyncClient`. The default value is 5 batches.

- **maxInflightReceiveBatches**—the maximum number of active receive batches that can be processed at the same time. The higher the setting, the more messages can be received (subject to other limits, such as CPU or bandwidth, being reached). Although, the higher the setting, the more threads are consumed by the `AmazonSQSBufferedAsyncClient`. If this parameter is set to 0, all pre-fetching of messages is disabled and messages are only consumed on demand. The default value is 10 batches.

- **visibilityTimeoutSeconds**—if this parameter is set to a positive nonzero value, this visibility timeout overrides the visibility timeout set on the queue from which messages are consumed. A visibility timeout of zero seconds isn't supported. The default value is -1, which means the default queue setting is used.
Increasing Throughput with Horizontal Scaling and Batching

Amazon SQS queues can deliver very high throughput (many thousands of messages per second). To achieve this throughput, you must scale message producers and consumers horizontally (add more producers and consumers).

In addition to horizontal scaling, batching provides a throughput with fewer threads, connections, and requests than would be required by individual message requests. You can use batched Amazon SQS API actions to send, receive, or delete up to 10 messages at a time. Because Amazon SQS charges by the request instead of by the message, batching can also substantially reduce costs.

Horizontal Scaling

Because you access Amazon SQS through an HTTP request-response protocol, the request latency (the time interval between initiating a request and receiving a response) limits the throughput that you can achieve from a single thread over a single connection. For example, if the latency from an Amazon Elastic Compute Cloud (Amazon EC2) based client to Amazon SQS in the same region averages around 20 ms, the maximum throughput from a single thread over a single connection averages 50 operations per second.

Horizontal scaling means increasing the number of your message producers (making SendMessage requests) and consumers (making ReceiveMessage and DeleteMessage requests) in order to increase your overall queue throughput. You can scale horizontally by increasing the number of threads on a client, adding clients, or both. You should achieve essentially linear gains in queue throughput as you add more clients. For example, if you double the number of clients, you can get twice the throughput.

Important

As you scale horizontally, you need to ensure that the Amazon SQS queue that you use has enough connections or threads to support the number of concurrent message producers and consumers that send requests and receive responses. For example, by default, instances of the AWS SDK for Java AmazonSQSClient class maintain at most 50 connections to Amazon SQS.

To create additional concurrent producers and consumers, you'll need to adjust that limit. For example, in the AWS SDK for Java, you can adjust the maximum number of allowable producer and consumer threads on an AmazonSQSClient object with this line of code:

```java
AmazonSQS sqsClient = new AmazonSQSClient(credentials,
new
ClientConfiguration().withMaxConnections(producerCount + consumerCount));
```

For the SDK for Java asynchronous client AmazonSQSAyncClient, you'll also need to make sure there are enough threads available. For more information, consult the documentation for the SDK library that you're using.

Batching

The batching actions in the Amazon SQS API (SendMessageBatch and DeleteMessageBatch) can further optimize throughput by processing up to ten messages at a time. ReceiveMessage can process ten messages at a time, so there is no ReceiveMessageBatch action.

The basic idea of batching is to perform more work in each round trip to the service (e.g., sending multiple messages with a single SendMessageBatch request), and to distribute the latency of the batch operation over the multiple messages in the batch request, as opposed to accepting the entire latency for a single message (for example, a SendMessage request). Because each round-trip carries more work, batch requests make more efficient use of threads and connections and so improve throughput.
SQS charges by the request, so the cost can be greatly reduced when fewer requests are processing the same number of messages. Moreover, fewer threads and connections reduce client-side resource utilization and can reduce client-side cost by doing the same work with smaller or fewer hosts.

Batching does introduce a bit of complication for the application. For example, the application has to accumulate the messages before sending them and it sometimes has to wait longer for a response, but batching can be effective in the following circumstances:

- Your application is generating a lot of messages in a short time, so the delay is never very long.
- A message consumer fetches messages from a queue at its discretion, as opposed to typical message producers that need to send messages in response to events they do not control.

**Important**

A batch request (SendMessageBatch or DeleteMessageBatch) may succeed even though individual messages in the batch have failed. After a batch request, you should always check for individual message failures and retry them if necessary.

You can take advantage of batching without changing your producers and consumers using the Amazon SQS Buffered Asynchronous Client (p. 176).

**Example**

The example presented in this section implements a simple producer-consumer pattern. The complete example is available as a free download at https://s3.amazonaws.com/cloudformation-examples/sqs-producer-consumer-sample.tar. The resources that are deployed by each template are described later in this section.

The code for the samples is available on the provisioned instances in /tmp/sqs-producer-consumer-sample/src. The command line for the configured run is in /tmp/sqs-producer-consumer-sample/command.log.

The main thread spawns a number of producer and consumer threads that process 1 KB messages for a specified time. The example includes producers and consumers that make single-operation requests and others that make batch requests.

In the program, each producer thread sends messages until the main thread stops the producer thread. The producedCount object tracks the number of messages produced by all producer threads. Error handling is simple: if there is an error, the program exits the run() method. Requests that fail on transient errors are, by default, retried three times by the AmazonSQSClient, so very few such errors are surfaced. The retry count can be configured as necessary to reduce the number of exceptions that are thrown. The run() method on the message producer is implemented as follows:

```java
try {
    while (!stop.get()) {
        sqsClient.sendMessage(new SendMessageRequest(queueUrl, theMessage));
        producedCount.incrementAndGet();
    }
} catch (AmazonClientException e) {
    // By default AmazonSQSClient retries calls 3 times before failing,
    // so when this rare condition occurs, simply stop.
    log.error("Producer: " + e.getMessage());
    System.exit(1);
}
```

The batch producer is much the same. One noteworthy difference is the need to retry failed individual batch entries:

```java
SendMessageBatchResult batchResult = sqsClient.sendMessageBatch(batchRequest);
```
if (!batchResult.getFailed().isEmpty()) {
    log.warn("Producer: retrying sending ", batchResult.getFailed().size() + " messages");
    for (int i = 0, n = batchResult.getFailed().size(); i < n; i++)
        sqsClient.sendMessage(new SendMessageRequest(queueUrl, theMessage));
}

The consumer run() method is as follows:

while (!stop.get()) {
    result = sqsClient.receiveMessage(new ReceiveMessageRequest(queueUrl));
    if (!result.getMessages().isEmpty()) {
        m = result.getMessages().get(0);
        sqsClient.deleteMessage(new DeleteMessageRequest(queueUrl,
                                                   m.getReceiptHandle()));
        consumedCount.incrementAndGet();
    }
}

Each consumer thread receives and deletes messages until it's stopped by the main thread. The consumedCount object tracks the number of messages that are consumed by all consumer threads, and the count is periodically logged. The batch consumer is similar, except that up to ten messages are received at a time, and it uses DeleteMessageBatch instead of DeleteMessage.

Running the Example

You can use the AWS CloudFormation templates provided to run the example code in three different configurations: single host with the single operation requests, two hosts with the single operation requests, one host with the batch requests.

Important
The complete sample is available in a single .tar file. The resources that are deployed by each template are described later in this section.
The code for the samples is available on the provisioned instances in /tmp/sqs-producer-consumer-sample/src. The command line for the configured run is in /tmp/sqs-producer-consumer-sample/command.log.
The default duration (20 minutes) is set to provide three or four 5-minute CloudWatch data points of volume metrics. The Amazon EC2 cost for each run is the m1.large instance cost. The Amazon SQS cost varies based on the API call rate for each sample, and this ranges between approximately 38,000 API calls per minute for the batching sample and 380,000 API calls per minute for the double-host, single-API sample.

If you want to deploy the AWS CloudFormation stack in a region other than the US East (N. Virginia) region, in the Region box of the AWS CloudFormation console, choose a region.

To run the example

1. Choose the link below that corresponds to the stack that you want to launch:
   - **Single Operation API, One Host**: This example template uses the single operation form of Amazon SQS API requests: SendMessage, ReceiveMessage, and DeleteMessage. A single m1.large Amazon EC2 instance animates 16 producer threads and 32 consumer threads. View template
   - **Single Operation API, Two Hosts**: This example template uses the single operation form of Amazon SQS API requests, but instead of a single m1.large Amazon EC2 instance, it uses two, each with 16 producer threads and 32 consumer threads for a total of 32 producers and 64 consumers. It illustrates the elasticity of Amazon SQS with throughput increasing proportionally to the greater number of producers and consumers. View Template
Increasing Throughput with Horizontal Scaling and Batching

- **Batch API, One Host**: This example template uses the batch form of Amazon SQS API requests on a single m1.large Amazon EC2 instance with 12 producer threads and 20 consumer threads. View Template

1. If you're prompted, sign in to the AWS Management Console.
2. In the **Create Stack** wizard, on the **Select Template** page, choose **Continue**.
3. On the **Specify Parameters** page, specify how long the program should run, whether or not you want to automatically terminate the Amazon EC2 instances when the run is complete, and provide an Amazon EC2 key pair so that you can access the instances that are running the sample.
4. Select the **I acknowledge that this template may create IAM resources** check box. All templates create an AWS Identity and Access Management (IAM) user so that the producer-consumer program can access the queue.
5. When all the settings are as you want them, choose **Continue**.
6. On the **Review** page, review the settings. If they're as you want them, choose **Continue**. If not, choose **Back** and make the necessary changes.
7. On the final page of the wizard, choose **Close**. Stack deployment may take several minutes.

To follow the progress of stack deployment, in the AWS CloudFormation console, choose the sample stack. In the lower pane, choose the **Events** tab. After the stack is created, it should take less than 5 minutes for the sample to start running. When it does, you can see the queue in the Amazon SQS console.

To monitor queue activity, you can do the following:

- Access the client instance, and open its output log file /tmp/sqs-producer-consumer-sample/output.log for a tally of messages produced and consumed so far. This tally is updated once per second.
- In the Amazon SQS console, observe changes in the **Message Available** and **Messages in Flight** numbers.

In addition, after a delay of up to 15 minutes after the queue is started, you can monitor the queue in CloudWatch as described later in this topic.

Although the templates and samples have safeguards to prevent excessive use of resources, it’s best to delete your AWS CloudFormation stacks when you’re done running the samples. To do so, in the Amazon SQS console, choose the stack that you want to delete, and then choose **Delete Stack**. When the resources are all deleted, all CloudWatch metrics drop to zero.

**Monitoring Volume Metrics from Example Run**

Amazon SQS automatically generates volume metrics for messages sent, received, and deleted. You can access those metrics and others through the CloudWatch console. The metrics can take up to 15 minutes after the queue starts to become available. To manage the search result set, choose **Search**, and then select the check boxes that correspond to the queues and metrics that you want to monitor.

Here is the NumberOfMessageSent metric for consecutive runs of the three samples. Your results may vary somewhat, but the results should be qualitatively similar:
- The `NumberOfMessagesReceived` and `NumberOfMessagesDeleted` metrics show the same pattern, but we have omitted them from this graph to reduce clutter.
- The first sample (single operation API on a single `m1.large` instance) delivers approximately 210,000 messages over 5 minutes, or about 700 messages per second, with the same throughput for receive and delete operations.
- The second sample (single operation API on two `m1.large` instances) delivers roughly double that throughput: approximately 440,000 messages in 5 minutes, or about 1,450 messages per second, with the same throughput for receive and delete operations.
- The last sample (batch API on a single `m1.large` instance) delivers over 800,000 messages in 5 minutes, or about 2,500 messages per second, with the same throughput for received and deleted messages. With a batch size of 10, these messages are processed with far fewer requests and therefore at lower cost.
Related Amazon SQS Resources

The following table lists related resources that you might find useful as you work with this service.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon Simple Queue Service API Reference</td>
<td>The API reference gives complete descriptions of API actions, parameters, and data types and a list of errors that the service returns.</td>
</tr>
<tr>
<td>Amazon SQS Release Notes</td>
<td>The release notes give a high-level overview of the current release. They specifically note any new features, corrections, and known issues.</td>
</tr>
<tr>
<td>Product information for Amazon SQS</td>
<td>The primary web page for information about Amazon SQS.</td>
</tr>
<tr>
<td>Discussion Forums</td>
<td>A community-based forum for developers to discuss technical questions related to Amazon SQS.</td>
</tr>
<tr>
<td>AWS Premium Support Information</td>
<td>The primary web page for information about AWS Premium Support, a one-on-one, fast-response support channel to help you build and run applications on AWS infrastructure services.</td>
</tr>
</tbody>
</table>
### Document History

The following table describes the important changes to the documentation since the last release of the *Amazon Simple Queue Service Developer Guide*.

- **API version:** 2012-11-05
- **Latest documentation update:** October 30, 2017

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date Changed</th>
</tr>
</thead>
</table>
| Update       | • Corrected and reorganized the table of contents.  
• Rewrote the *Visibility Timeout* (p. 59) section.                                                                                        | October 30, 2017  |
<p>| Update       | Clarified the explanation of throughput for FIFO queues in the <em>FIFO (First-in-First-Out) Queues</em> (p. 51) section.                                                                                       | October 27, 2017  |
| New feature  | You can track cost allocation by adding, updating, removing, and listing metadata tags for Amazon SQS queues using the <em>TagQueue</em>, <em>UntagQueue</em>, and <em>ListQueueTags</em> actions and the AWS Management Console. For more information, see <em>Tagging Your Amazon SQS Queues</em> (p. 65) and the <em>Adding, Updating, and Removing Tags from an Amazon SQS Queue</em> (p. 44) tutorial. | October 19, 2017  |
| Update       | Added a note about the Amazon SQS Buffered Asynchronous Client to the <em>Increasing Throughput with Horizontal Scaling and Batching</em> (p. 179) section.                                                   | September 29, 2017|
| Update       | Corrected the diagrams in the <em>Using Amazon SQS and IAM Policies</em> (p. 138) section.                                                                                                                        | September 19, 2017|
| New feature  | The complete set of Amazon SQS actions is displayed in the <em>Actions</em> list on the <em>Add a Permission to MyQueue</em> dialog box. For more information, see the <em>Tutorial: Adding Permissions to an Amazon SQS Queue</em> (p. 28) tutorial. | September 1, 2017 |
| Update       | Clarified the information in the <em>Changing the Visibility Timeout for a Message</em> (p. 60) section.                                                                                                          | August 29, 2017   |
| Update       | Clarified the permissions for the <em>SendMessage</em> and <em>SendMessageBatch</em> API actions in <em>Amazon SQS API Permissions: Actions and Resource Reference</em> (p. 154).                                      | August 17, 2017   |</p>
<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date Changed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update</td>
<td>Updated information about dead-letter queues in the General Recommendations (p. 109) section.</td>
<td>August 15, 2017</td>
</tr>
</tbody>
</table>
| Update | • The Amazon SQS Java Messaging Library has been updated to 1.0.4. For more information, see Using JMS with Amazon SQS (p. 88).  
• Updated the Using JMS with Amazon SQS (p. 88) section. | August 9, 2017 |
| Update | Changed the deprecated AmazonSQSClient method to AmazonSQSClientBuilder and revised the corresponding region specification in the Getting Started with Standard Queues (p. 48) section. | July 27, 2017 |
| Update | Clarified the throughput for standard and FIFO queues throughout this guide:  
• Standard queues can support a nearly unlimited number of transactions per second (TPS) per API action.  
• Without batching, FIFO queues can support up to 300 messages per second (300 send, receive, or delete operations per second).  
• If you take advantage of the maximum batching (p. 176) of 10 messages per operation, FIFO queues can support up to 3,000 messages per second. | July 25, 2017 |
| Update | Clarified the compatibility between Amazon SQS SSE queues and AWS and third-party service features throughout this guide:  
Some features of AWS services that can send notifications to Amazon SQS using the AWS Security Token Service AssumeRole API action are compatible with SSE but work only with standard queues:  
• Auto Scaling Lifecycle Hooks  
• AWS Lambda Dead-Letter Queues  
Other features of AWS services or third-party services that send notifications to Amazon SQS aren’t compatible with SSE, despite allowing you to set an encrypted queue as a target:  
• Amazon CloudWatch Events  
• AWS IoT Rule Actions  
• Amazon Simple Storage Service Event Notifications  
• Amazon Simple Notification Service Topic Subscriptions  
For information about compatibility of other services with encrypted queues, see your service documentation. | July 20, 2017 |
<p>| Update | Corrected the information in the Limits Related to Messages (p. 114) section. | June 23, 2017 |
| Update | Clarified the information in the Using Amazon SQS Dead-Letter Queues (p. 61) section. | June 20, 2017 |</p>
<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date Changed</th>
</tr>
</thead>
<tbody>
<tr>
<td>New feature</td>
<td>FIFO (First-In-First-Out) queues are available in the US East (N. Virginia) region, in addition to the EU (Ireland), US East (Ohio), and US West (Oregon) regions. For more information about how FIFO queues work and how to get started using them, see FIFO (First-In-First-Out) Queues (p. 51).</td>
<td>June 14, 2017</td>
</tr>
<tr>
<td>New feature</td>
<td>FIFO (First-In-First-Out) queues are available in the EU (Ireland) region, in addition to the US East (Ohio) and US West (Oregon) regions. For more information about how FIFO queues work and how to get started using them, see FIFO (First-In-First-Out) Queues (p. 51).</td>
<td>June 8, 2017</td>
</tr>
</tbody>
</table>
| Update | • Restructured and updated the Using Amazon SQS Dead-Letter Queues (p. 61) section.  
• Created the Tutorial: Configuring an Amazon SQS Dead-Letter Queue (p. 37) section. | June 2, 2017 |
| Update | Updated the What is Amazon Simple Queue Service? (p. 1) section. | June 1, 2017 |
| Update | • Restructured the Using JMS with Amazon SQS (p. 88) section.  
• Created the Using the Amazon SQS Java Message Service (JMS) Client with Other Amazon SQS Clients (p. 94) section. | May 24, 2017 |
| New feature | Server-side encryption (SSE) for Amazon SQS is available in the US East (N. Virginia) region, in addition to the US East (Ohio) and US West (Oregon) regions. For more information on server-side encryption and how to get started using it, see Protecting Data Using Server-Side Encryption (SSE) and AWS KMS (p. 156). | May 23, 2017 |
| New feature | • You can use the Amazon SQS Extended Client Library for Java together with the Amazon SQS Java Message Service (JMS) Client.  
• The Amazon SQS Java Messaging Library has been updated to 1.0.3. For more information, see Using JMS with Amazon SQS (p. 88).  
• Updated the Using JMS with Amazon SQS (p. 88) section. | May 19, 2017 |
<p>| New feature | AWS has expanded its HIPAA compliance program to include Amazon SQS as a HIPAA Eligible Service. | May 1, 2017 |</p>
<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date Changed</th>
</tr>
</thead>
</table>
| New feature  | Server-side encryption (SSE) for Amazon SQS is available in the US East (Ohio) and US West (Oregon) regions. SSE lets you protect the contents of messages in Amazon SQS queues using keys managed in the AWS Key Management Service (AWS KMS). For more information on server-side encryption and how to get started using it, see Protecting Data Using Server-Side Encryption (SSE) and AWS KMS (p. 156). For tutorials, see the following:  
  - Creating an Amazon SQS queue with SSE (p. 20)  
  - Configuring SSE for an existing Amazon SQS queue (p. 24)  
  SSE adds the `KmsKeyId` and `KmsDataKeyReusePeriodSeconds` attributes to the `CreateQueue`, `GetQueueAttributes`, and `SetQueueAttributes` actions. Important
  Some features of AWS services that can send notifications to Amazon SQS using the AWS Security Token Service `AssumeRole` API action are compatible with SSE but work only with standard queues:  
  - Auto Scaling Lifecycle Hooks  
  - AWS Lambda Dead-Letter Queues  
  Other features of AWS services or third-party services that send notifications to Amazon SQS aren't compatible with SSE, despite allowing you to set an encrypted queue as a target:  
  - Amazon CloudWatch Events  
  - AWS IoT Rule Actions  
  - Amazon Simple Storage Service Event Notifications  
  - Amazon Simple Notification Service Topic Subscriptions  
  For information about compatibility of other services with encrypted queues, see your service documentation.                                                                 | April 28, 2017      |
<p>| Update       | Restructured and updated the Amazon SQS Long Polling (p. 73) section.                                                                                                                                     | April 25, 2017      |
| New feature  | The Amazon SQS Extended Client Library for Java and Amazon SQS Java Message Service (JMS) Client support FIFO queues.                                                                                       | April 24, 2017      |
| New feature  | The Amazon SQS Java Messaging Library has been updated to 1.0.2.                                                                                                                                            |                     |
|              | Updated the Using JMS with Amazon SQS (p. 88) section.                                                                                                                                                     |                     |
| New feature  | AWS CloudFormation lets you create FIFO queues. Added the Create a Queue Using AWS CloudFormation (p. 18) tutorial.                                                                                         | March 28, 2017      |
| Update       | Updated the Authentication and Access Control (p. 131) section with new content.                                                                                                                             | February 6, 2017    |</p>
<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date Changed</th>
</tr>
</thead>
</table>
| Update | Retired the *Amazon Simple Queue Service Getting Started Guide* and incorporated some of its content into the following sections of this guide:  
- Setting Up Amazon SQS (p. 4)  
- Getting Started with Amazon SQS (p. 7)  
- Amazon SQS Tutorials (p. 15) | December 16, 2016 |
<p>| Update | Restructured and updated the <em>Authentication and Access Control (p. 131)</em> section. | December 2, 2016 |</p>
<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date Changed</th>
</tr>
</thead>
<tbody>
<tr>
<td>New feature</td>
<td><strong>FIFO (First-In-First-Out) queues or standard queues</strong> (the new name for existing queues) are available in the US West (Oregon) and US East (Ohio) regions. For more information about how FIFO queues work and how to get started using them, see the following:</td>
<td>November 17, 2016</td>
</tr>
<tr>
<td></td>
<td>• FIFO (First-In-First-Out) Queues (p. 51)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Moving from a Standard Queue to a FIFO Queue (p. 56)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Recommendations for FIFO (First-In-First-Out) Queues (p. 111)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For revised Amazon SQS tutorials, see the following:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Creating an Amazon SQS Queue (p. 15)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Sending a Message to an Amazon SQS Queue (p. 29)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Receiving and Deleting a Message from an Amazon SQS Queue (p. 32)</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>FIFO queues add the following API functionality:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The <strong>FifoQueue</strong> and <strong>ContentBasedDeduplication</strong> attributes for the <strong>CreateQueue</strong>, <strong>GetQueueAttributes</strong>, and <strong>SetQueueAttributes</strong> actions.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The <strong>MessageDeduplicationId</strong> and <strong>MessageGroupId</strong> request parameters for the <strong>SendMessage</strong> and <strong>SendMessageBatch</strong> actions and attributes for the <strong>ReceiveMessage</strong> action.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The <strong>ReceiveRequestAttemptId</strong> request parameter for the <strong>ReceiveMessage</strong> action.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The <strong>SequenceNumber</strong> response parameter for the <strong>SendMessage</strong> and <strong>SendMessageBatch</strong> actions and the <strong>SequenceNumber</strong> attribute for the <strong>ReceiveMessage</strong> action.</td>
<td></td>
</tr>
<tr>
<td><strong>Important</strong></td>
<td>As of November 17, 2016, Amazon SQS no longer publishes a WSDL.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The Amazon SQS Buffered Asynchronous Client doesn't currently support FIFO queues.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Some AWS or external services that send notifications to Amazon SQS might not be compatible with FIFO queues, despite allowing you to set a FIFO queue as a target.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The following features of AWS services aren't currently compatible with FIFO queues:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Auto Scaling Lifecycle Hooks</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Amazon CloudWatch Events</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• AWS IoT Rule Actions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• AWS Lambda Dead-Letter Queues</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Amazon S3 Event Notifications</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Amazon SNS Topic Subscriptions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For information about compatibility of other services with FIFO queues, see your service documentation.</td>
<td></td>
</tr>
<tr>
<td>Change</td>
<td>Description</td>
<td>Date Changed</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>Update</td>
<td>FIFO queues don't support timers on individual messages.</td>
<td></td>
</tr>
<tr>
<td>New feature</td>
<td>The \texttt{ApproximateAgeOfOldestMessage} CloudWatch metric lets you find the approximate age of the oldest non-deleted message in the queue. For more information, see Available CloudWatch Metrics for Amazon SQS (p. 123).</td>
<td>August 31, 2016</td>
</tr>
<tr>
<td>Update</td>
<td>Renamed the Walkthroughs section to Amazon SQS Tutorials (p. 15).</td>
<td>November 2, 2016</td>
</tr>
<tr>
<td>Update</td>
<td>Added the Best Practices for Amazon SQS (p. 109) section.</td>
<td>May 27, 2016</td>
</tr>
<tr>
<td>Update</td>
<td>Added the Amazon SQS Limits (p. 113) section.</td>
<td>May 12, 2016</td>
</tr>
<tr>
<td>New feature</td>
<td>You can view CloudWatch metrics from within the Amazon SQS console for up to 10 of your queues at a time. For more information, see Monitoring Amazon SQS using CloudWatch (p. 116).</td>
<td>February 12, 2016</td>
</tr>
<tr>
<td>Update</td>
<td>Updated Amazon SQS console screenshots.</td>
<td>December 7, 2015</td>
</tr>
<tr>
<td>New feature</td>
<td>The Amazon SQS Extended Client Library for Java lets you manage Amazon SQS messages with Amazon S3. For more information, see Managing Large Amazon SQS Messages Using Amazon S3 (p. 84) in the Amazon Simple Queue Service Developer Guide.</td>
<td>October 27, 2015</td>
</tr>
<tr>
<td>New feature</td>
<td>Amazon SQS lets you use JMS (Java Message Service) with Amazon SQS queues. For more information, see Using JMS with Amazon SQS (p. 88) in the Amazon Simple Queue Service Developer Guide.</td>
<td>December 29, 2014</td>
</tr>
<tr>
<td>New feature</td>
<td>Amazon SQS lets you delete the messages in a queue using the \texttt{PurgeQueue} API action. For more information, see \texttt{PurgeQueue} in the Amazon SQS API Reference.</td>
<td>December 8, 2014</td>
</tr>
<tr>
<td>Update</td>
<td>Updated information about access keys. For more information, see Your Access Keys (p. 168).</td>
<td>August 4, 2014</td>
</tr>
<tr>
<td>New feature</td>
<td>Amazon SQS lets you log API actions using AWS CloudTrail. For more information, see Logging Amazon SQS API Actions Using AWS CloudTrail (p. 126).</td>
<td>July 16, 2014</td>
</tr>
<tr>
<td>New feature</td>
<td>Amazon SQS provides support for message attributes. For more information, see Using Amazon SQS Message Attributes (p. 66).</td>
<td>May 6, 2014</td>
</tr>
<tr>
<td>New feature</td>
<td>Amazon SQS provides support for dead-letter queues. For more information, see Using Amazon SQS Dead-Letter Queues (p. 61).</td>
<td>January 29, 2014</td>
</tr>
<tr>
<td>New feature</td>
<td>You can subscribe an Amazon SQS queue to an Amazon SNS topic using the AWS Management Console for Amazon SQS, which simplifies the process. For more information, see Tutorial: Subscribing an Amazon SQS Queue to an Amazon SNS Topic (p. 42).</td>
<td>November 21, 2012</td>
</tr>
<tr>
<td>New feature</td>
<td>The 2012-11-05 API version of Amazon SQS adds support for Signature Version 4, which provides improved security and performance. For more information about Signature Version 4, see Query Request Authentication (p. 171).</td>
<td>November 5, 2012</td>
</tr>
<tr>
<td>Change</td>
<td>Description</td>
<td>Date Changed</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>New feature</td>
<td>The AWS SDK for Java includes a buffered asynchronous client, <strong>AmazonSQSBufferedAsyncClient</strong>, for accessing Amazon SQS. This client allows for easier request batching by enabling client-side buffering, where calls made from the client are first buffered and then sent as a batch request to Amazon SQS. For more information about client-side buffering and request batching, see <a href="https://console.aws.amazon.com/sqs">Client-Side Buffering and Request Batching (p. 176)</a>.</td>
<td>November 5, 2012</td>
</tr>
<tr>
<td>New feature</td>
<td>The 2012-11-05 API version of Amazon SQS adds long polling support. Long polling allows Amazon SQS to wait for a specified amount time for a message to be available instead of returning an empty response if one isn't available. For more information about long polling, see <a href="https://console.aws.amazon.com/sqs">Amazon SQS Long Polling (p. 73)</a>.</td>
<td>November 5, 2012</td>
</tr>
</tbody>
</table>
AWS Glossary

For the latest AWS terminology, see the AWS Glossary in the AWS General Reference.