
Amazon Simple Queue Service

Developer Guide



Amazon Simple Queue Service: Developer Guide

Copyright © 2017 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

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	9
Step 3: Receive and Delete a Message	11
Step 4: Delete a Queue	13
Next Steps	14
Tutorials	15
Creating a Queue	15
AWS Management Console	15
Java	17
Listing All Queues	18
AWS Management Console	18
Java	19
Adding Permissions to a Queue	19
AWS Management Console	19
Sending a Message	21
AWS Management Console	21
Java	23
Receiving and Deleting a Message	24
AWS Management Console	24
Java	27
Purging a Queue	29
AWS Management Console	29
Deleting a Queue	30
AWS Management Console	30
Java	31
Subscribing a Queue to a Topic	31
AWS Management Console	32
How Queues Work	34
Basic Prerequisites	34
Standard Queues	35
Message Ordering	35
At-Least-Once Delivery	35
Retrieving Messages Using Short Polling	36
Getting Started with Standard Queues	36
FIFO Queues	39
Message Ordering	40
FIFO Queue Logic	40
Exactly-Once Processing	41
Getting Started with FIFO Queues	41
Moving From a Standard Queue to a FIFO Queue	44
Compatibility	45
Queue and Message Identifiers	45

General Identifiers	45
Additional Identifiers for FIFO Queues	46
Resources Required to Process Messages	47
Visibility Timeout	47
Inflight Messages	48
Configuring the Visibility Timeout	48
Changing a Message's Visibility Timeout	49
Terminating a Message's Visibility Timeout	49
API Actions Related to Visibility Timeout	49
Message Lifecycle	50
Dead Letter Queues	51
Setting up a Dead Letter Queue with the AWS Management Console	51
Using a Dead Letter Queue with the Amazon SQS API	52
Issue: Viewing Messages with the Amazon SQS Console Can Cause the Messages to be Moved to a Dead Letter Queue	53
Issue: NumberOfMessagesSent and NumberOfMessagesReceived for the Dead Letter Queue Do Not Match	54
Message Attributes	54
Message Attribute Items and Validation	55
Data Types	55
Using Message Attributes with the AWS Management Console	56
Using Message Attributes with the AWS SDKs	57
Using Message Attributes with the Amazon SQS Query API	59
MD5 Message-Digest Calculation	61
Long Polling	62
Benefits of Long Polling	62
Enabling Long Polling with the AWS Management Console	63
Enabling Long Polling Using the API	65
Enabling Long Polling Using the Query API	65
Delay Queues	66
Creating Delay Queues with the AWS Management Console	67
Creating Delay Queues with the Query API	69
Message Timers	70
Creating Message Timers Using the Console	70
Creating Message Timers Using the Query API	72
Managing Large Messages Using Amazon S3	73
Prerequisites	73
Using the Amazon SQS Extended Client Library for Java	74
Amazon SQS and JMS	77
Prerequisites	77
Getting Started with the Amazon SQS Java Messaging Library	78
Receiving Messages Asynchronously	81
Using Client Acknowledge Mode	82
Using Unordered Acknowledge Mode	82
Code Examples	83
Supported JMS 1.1 Implementations	96
Best Practices	98
General Recommendations	98
Processing Messages	98
Reducing Costs	99
Moving from a Standard Queue to a FIFO Queue	99
Recommendations for FIFO Queues	99
Using the Message Deduplication ID	100
Using the Message Group ID	100
Using the Receive Request Attempt ID	101
Limits	102
Limits Related to Queues	102
Limits Related to Messages	103

Limits Related to Policies	103
Monitoring and Logging	104
Monitoring Amazon SQS using CloudWatch	104
Common Monitoring Tasks	105
Access CloudWatch Metrics for Amazon SQS	105
Set CloudWatch Alarms for Amazon SQS Metrics	108
Available CloudWatch Metrics for Amazon SQS	110
Logging Amazon SQS API Actions Using CloudTrail	113
Amazon SQS Information in CloudTrail	113
Understanding Amazon SQS Log File Entries	114
Security	118
Authentication and Access Control	118
Authentication	118
Access Control	119
Overview of Managing Access	120
Using Identity-Based Policies (IAM) Policies for Amazon SQS	125
Creating Custom Policies Using the Access Policy Language	130
Using Temporary Security Credentials	138
Amazon SQS API Permissions Reference	140
Working with APIs	142
Making API Requests	142
Endpoints	143
Making Query Requests	144
Request Authentication	147
Responses	153
Shared Queues	155
Programming Languages	158
Batch API Actions	158
Maximum Message Size for SendMessageBatch	159
Client-Side Buffering and Request Batching	160
Increasing Throughput with Horizontal Scaling and Batching	163
Related Resources	168
Document History	169

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. 35) and [FIFO queues](#) (p. 39).

Topics

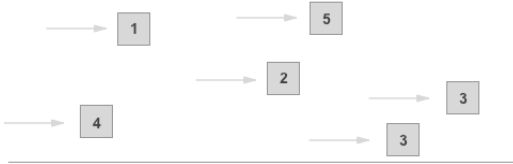

- [What Can I Use Amazon SQS For? \(p. 1\)](#)
- [What Type of Queue Do I Need? \(p. 2\)](#)
- [What Are the Main Features of Amazon SQS? \(p. 2\)](#)
- [What is the Basic Architecture of Amazon SQS? \(p. 3\)](#)

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 readers 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?

Standard Queue	FIFO Queue
<p>Available in all regions.</p> <p>High Throughput – Standard queues have nearly-unlimited transactions per second (TPS).</p> <p>At-Least-Once Delivery – A message is delivered at least once, but occasionally more than one copy of a message is delivered.</p> <p>Best-Effort Ordering – Occasionally, messages might be delivered in an order different from which they were sent.</p>	<p>Available in US West (Oregon) and US East (Ohio).</p> <p>First-In-First-Out Delivery – The order in which messages are sent and received is strictly preserved.</p> <p>Exactly-Once Processing – A message is delivered once and remains available until a consumer processes and deletes it. Duplicates are not introduced into the queue.</p> <p>Limited Throughput – 300 transactions per second (TPS).</p>
	
<p>Send data between applications when the throughput is important, for example:</p> <ul style="list-style-type: none">• 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.	<p>Send data between applications when the order of events is important, for example:</p> <ul style="list-style-type: none">• 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 sending and retrieving messages.
- **Multiple writers and readers** – 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 can 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.

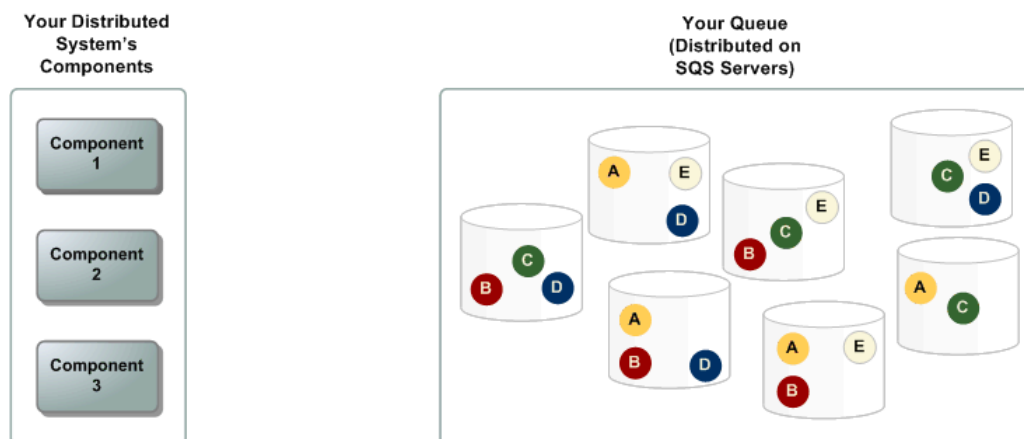
- **Access control** – You control who can send messages to a queue, and who can receive messages from a queue.
- **Delay queues** – You can set a default delay on a queue, so that delivery of all enqueued messages is postponed for the specified duration. You can set the delay value when you create a 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.
- **PCI compliance** – Amazon SQS supports the processing, storage, and transmission of credit card data by a merchant or service provider, and has been validated as compliant with Payment Card Industry (PCI) Data Security Standard (DSS). For more information about PCI DSS, including how to request a copy of the AWS PCI Compliance Package, see [PCI DSS Level 1](#).

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 (labeled A-E), is redundantly saved across multiple Amazon SQS servers.



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**.
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. Sign in to the Identity and Access Management (IAM) console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Users**, and then choose **Add user**.
3. For **User name**, type a user name, such as **Administrator**. The name can consist of letters, digits, and the following characters: plus (+), equal (=), comma (,), period (.), at (@), underscore (_), and hyphen (-). The name is not case sensitive and can be a maximum of 64 characters in length.
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 the name for the new group. The name can consist of letters, digits, and the following characters: plus (+), equal (=), comma (,), period (.), at (@), underscore (_), and hyphen (-). The name is not case sensitive and can be a maximum of 128 characters in length.
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 for Administering AWS Resources](#).

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 your access key ID and secret access key

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 by using the AWS Management Console. We recommend that you use IAM access keys instead of AWS root account access keys. IAM lets you securely control access to AWS services and resources in your AWS account.

Note

To create access keys, you must have permissions to perform the required IAM actions. For more information, see [Granting IAM User Permission to Manage Password Policy and Credentials](#) in the *IAM User Guide*.

1. Open the [IAM console](#).
2. In the navigation pane, 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 your access key, choose **Show User Security Credentials**. Your credentials will look something like this:
 - Access Key ID: AKIAIOSFODNN7EXAMPLE
 - Secret Access Key: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
6. Choose **Download Credentials**, and store the keys in a secure location.

Your secret key will no longer be available through the AWS Management Console; you will have the only copy. Keep it confidential in order to protect your account, and never email it. Do not

share it 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

- [What Is IAM?](#) in the *IAM User Guide*
- [AWS Security Credentials](#) in *AWS General Reference*

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

If you want to work with Amazon SQS programmatically, see the [Amazon SQS Tutorials \(p. 15\)](#) or skip to [Working with Amazon SQS APIs \(p. 142\)](#).

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 AWS Management Console and open the Amazon SQS console at <https://console.aws.amazon.com/sqs/>.
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 US West (Oregon) and US East (Ohio).

Create New Queue

Queue Name ⓘ

MyQueue.fifo

Region ⓘ US West (Oregon)

4. **Standard** is selected by default. Choose **FIFO**.

What type of queue do you need?

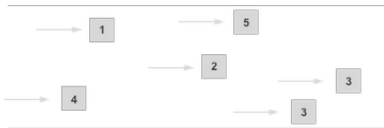
Standard

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 or a message is delivered.

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




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.

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 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. 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. 41).

<input type="checkbox"/>	Name	Queue Type	Content-Based Deduplication	Messages Available	Messages in Flight	Created
<input type="checkbox"/>	MyQueue	Standard Queue	N/A	0	0	2016-12-07 13:42:47 GMT-08:00
<input checked="" type="checkbox"/>	MyQueue.fifo	FIFO Queue	Disabled	0	0	2016-12-07 13:42:55 GMT-08:00

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

Details | Permissions | Redrive Policy | Monitoring

Name: MyQueue.fifo

URL: <https://sqs.us-west-2.amazonaws.com/.../MyQueue.fifo>

ARN: <arn:aws:sqs:us-west-2:...:MyQueue.fifo>

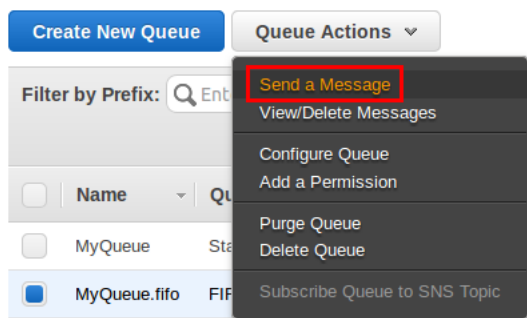
Step 2: Send a Message

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 the **Queue Actions** drop-down list, 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 is disabled).

Send a Message to MyQueue.fifo [X]

Message Body | **Message Attributes**

Enter the text of a message you want to send.

This is my message text.

Message Group ID ⓘ Type a FIFO message group (required).

Message Deduplication ID ⓘ Type a deduplication token (required).

Cancel **Send Message**

- 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. 40).

Note

The message group ID is always required. However, if content-based deduplication is enabled, the message deduplication ID is optional.

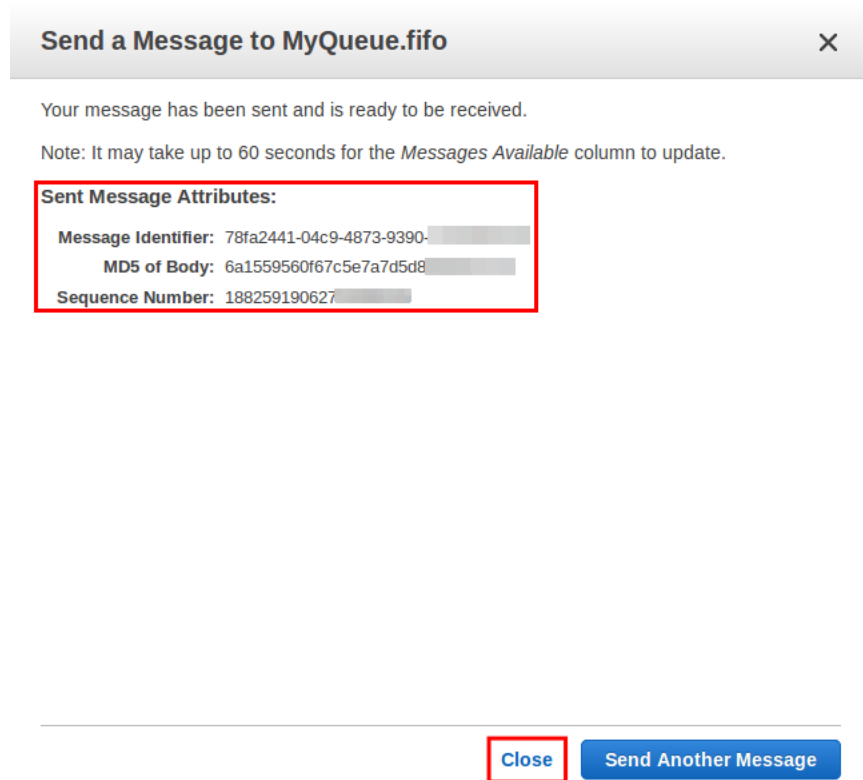
Message Group ID ⓘ MyMessageGroupId1234567890

Message Deduplication ID ⓘ MyMessageDeduplicationId1234567890

Cancel **Send Message**

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.



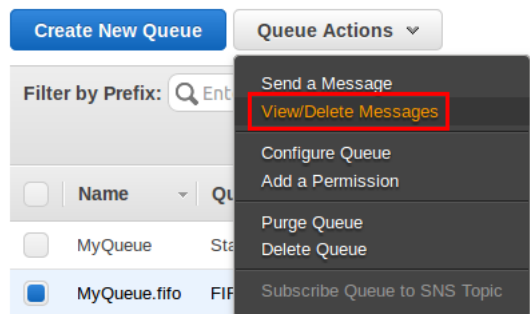
4. Choose **Close**.

Step 3: Receive and Delete a Message

After you send a message into a queue, you can receive 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 the **Queue Actions** drop-down list, select **View/Delete Messages**.

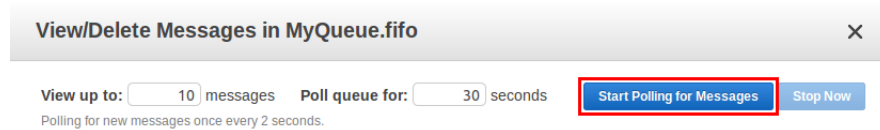


The **View/Delete Messages** in *QueueName* dialog box is displayed.

Note

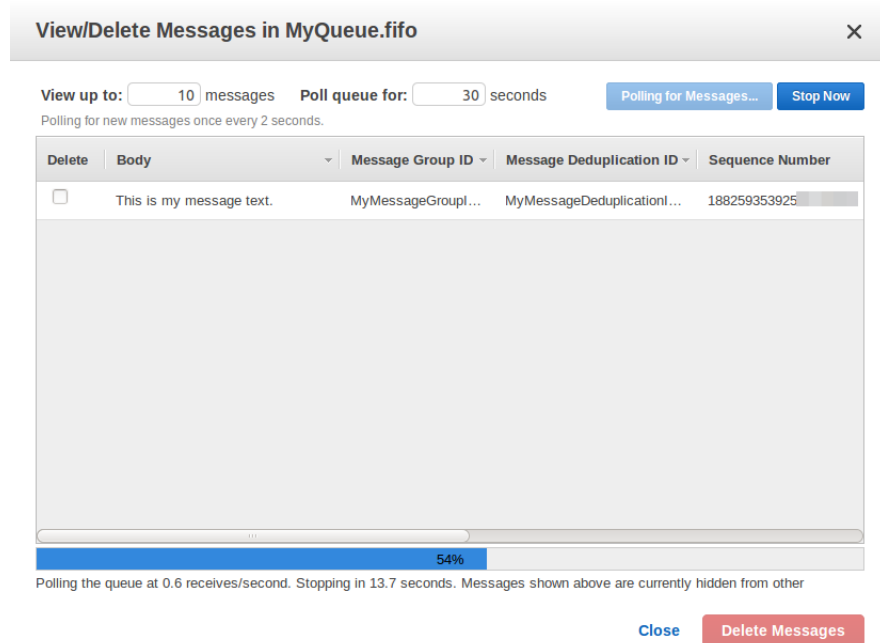
The first time you take this action, an information screen is displayed. To hide the screen, check the **Don't show this again** checkbox.

3. Choose **Start Polling for messages**.

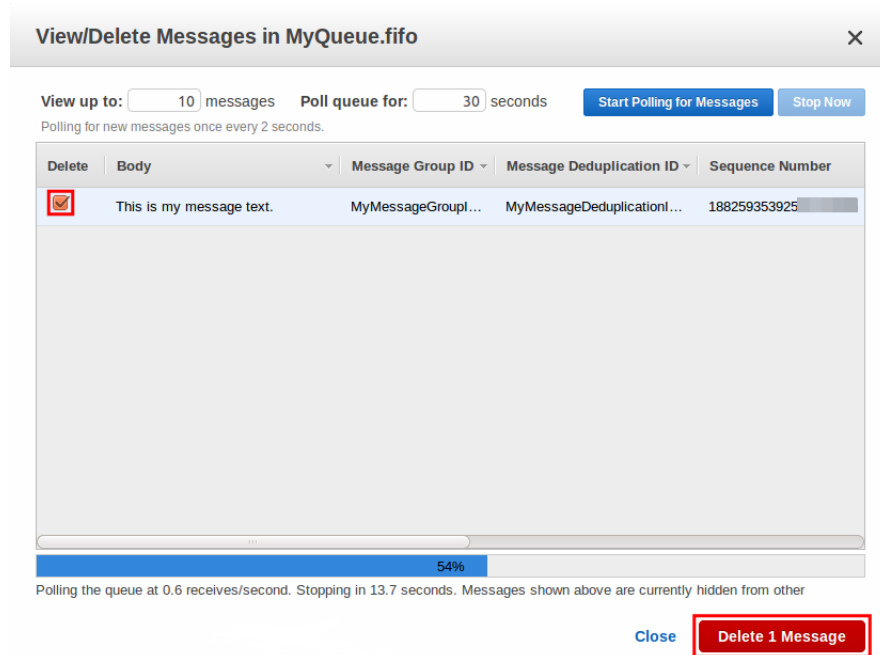


Amazon SQS begins to poll the messages in the queue. The dialog box displays a message from the queue. A progress bar at the bottom of the dialog box displays the status of the message's visibility timeout.

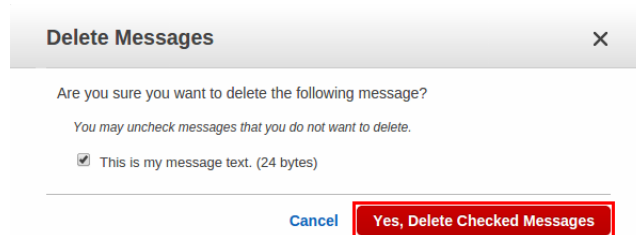
The following example shows the **Message Group ID**, **Message Deduplication ID**, and **Sequence Number** columns specific to FIFO queues.



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



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



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

The selected message is deleted.

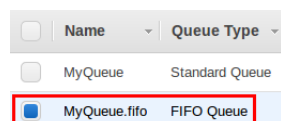
When the progress bar is filled in, the [visibility timeout \(p. 47\)](#) expires and the message becomes visible to consumers.

6. Select **Close**.

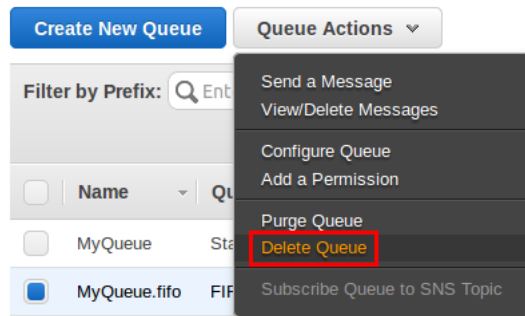
Step 4: Delete a Queue

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.

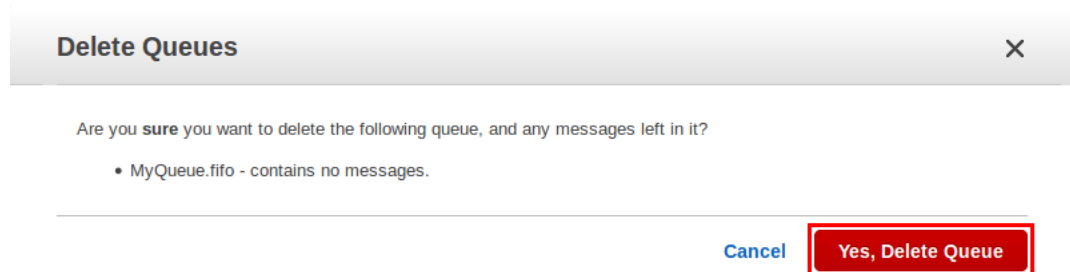
1. From the queue list, select the queue that you have created.



2. From the **Queue Actions** drop-down list, select **Delete Queue**.



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



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:

- [Add permissions to a queue.](#) (p. 19)
- [Purge a queue.](#) (p. 29)
- [Subscribe a queue to an Amazon SNS topic.](#) (p. 31)
- Learn more about Amazon SQS workflows and processes: Read [How Queues Work](#) (p. 34), [Best Practices](#) (p. 98), and [Limits](#) (p. 102). 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. 142) 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. 104) section.
- Learn about protecting your data and access to it: Start by reading the [Security](#) (p. 118) 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: Listing All Amazon SQS Queues in a Region \(p. 18\)](#)
- [Tutorial: Adding Permissions to an Amazon SQS Queue \(p. 19\)](#)
- [Tutorial: Sending a Message to an Amazon SQS Queue \(p. 21\)](#)
- [Tutorial: Receiving and Deleting a Message from an Amazon SQS Queue \(p. 24\)](#)
- [Tutorial: Purging Messages from an Amazon SQS Queue \(p. 29\)](#)
- [Tutorial: Deleting an Amazon SQS Queue \(p. 30\)](#)
- [Tutorial: Subscribing an Amazon SQS Queue to an Amazon SNS Topic \(p. 31\)](#)

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 AWS Management Console and open the Amazon SQS console at <https://console.aws.amazon.com/sqs/>.

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 US West (Oregon) and US East (Ohio).

Create New Queue

A screenshot of the 'Create New Queue' form. The 'Queue Name' field is highlighted with a red box and contains the text 'MyQueue.fifo'. Below it, the 'Region' dropdown menu is set to 'US West (Oregon)'. There are information icons (i) next to both labels.

4. **Standard** is selected by default. Choose **FIFO**.

What type of queue do you need?

Standard

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 or a message is delivered.

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

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.

FIFO

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 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. 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**.

The following example shows the **Content-Based Deduplication** parameter specific to FIFO queues.

Queue Attributes

Default Visibility Timeout 30 seconds Value must be between 0 seconds and 12 hours.

Message Retention Period 4 days Value must be between 1 minute and 14 days.

Maximum Message Size 256 KB Value must be between 1 and 256 KB.

Delivery Delay 0 seconds Value must be between 0 seconds and 15 minutes.

Receive Message Wait Time 0 seconds Value must be between 0 and 20 seconds.

Content-Based Deduplication ☐

Dead Letter Queue Settings

Use Redrive Policy ☐

Dead Letter Queue Value must be an existing queue name.

Maximum Receives Value must be between 1 and 1000.

Cancel **Create Queue**

Your new queue is created and selected in the queue list. 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 \(p. 41\)](#).

<input type="checkbox"/>	Name	Queue Type	Content-Based Deduplication	Messages Available	Messages in Flight	Created
<input type="checkbox"/>	MyQueue	Standard Queue	N/A	0	0	2016-12-07 13:42:47 GMT-08:00
<input checked="" type="checkbox"/>	MyQueue.fifo	FIFO Queue	Disabled	0	0	2016-12-07 13:42:55 GMT-08:00

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

Details Permissions Redrive Policy Monitoring

Name: MyQueue.fifo

URL: https://sqs.us-west-2.amazonaws.com/.../MyQueue.fifo

ARN: arn:aws:sqs:us-west-2:...:MyQueue.fifo

Java

Before you begin working with the example code, specify your AWS credentials. For more information, see [Set Up AWS Credentials for Development](#) in the *AWS SDK for Java Developer Guide*.

To create a standard queue

1. Copy the [example program \(p. 36\)](#).

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

```
// 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 \(p. 41\)](#).

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

```
// 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.

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.

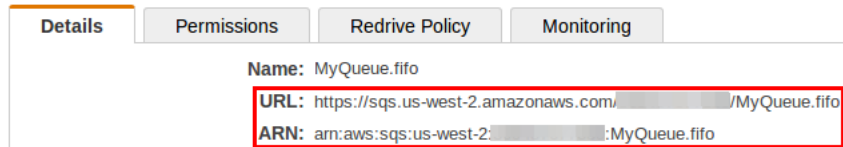
AWS Management Console

1. Sign in to the AWS Management Console and open the Amazon SQS console at <https://console.aws.amazon.com/sqs/>.
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 \(p. 41\)](#).

<input type="checkbox"/>	Name	Queue Type	Content-Based Deduplication	Messages Available	Messages in Flight	Created
<input type="checkbox"/>	MyQueue	Standard Queue	N/A	0	0	2016-12-07 13:42:47 GMT-08:00
<input checked="" type="checkbox"/>	MyQueue.fifo	FIFO Queue	Disabled	0	0	2016-12-07 13:42:55 GMT-08:00

Your queue's **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 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 \(p. 36\)](#) or the [FIFO queue example program \(p. 41\)](#).

The following section of the code list all queues in the current region:

```
// 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

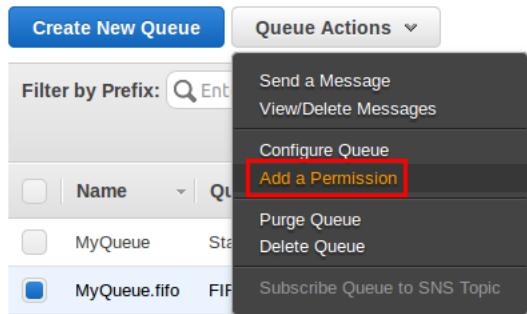
You can specify to whom you allow (or explicitly deny) the ability to interact with your queue in specific ways by adding permissions to a queue. The following example demonstrates adding the permission for anyone to get a queue's URL.

AWS Management Console

1. Sign in to the AWS Management Console and open the Amazon SQS console at <https://console.aws.amazon.com/sqs/>.
2. From the queue list, select a queue.

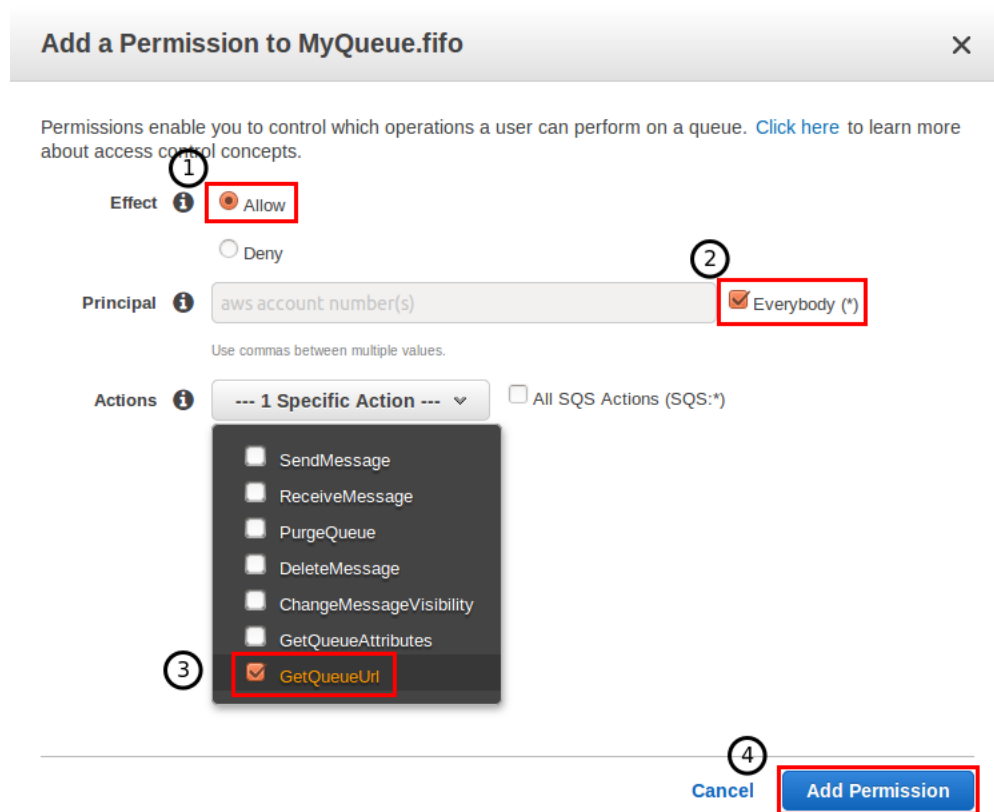


- From the **Queue Actions** drop-down list, select **Add a Permission**.



The **Add a Permission** dialog box is displayed.

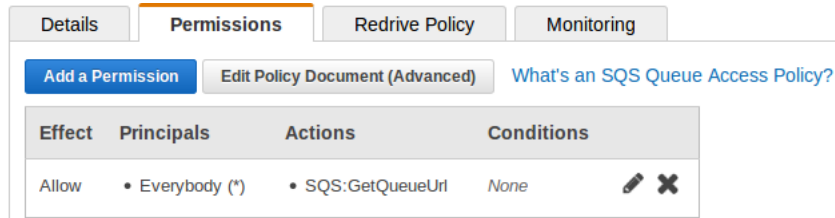
- In this example, you allow anyone to get the queue's URL:



- Ensure that next to **Effect**, **Allow** is selected.
- Next to **Principal**, check the **Everybody** box.
- From the **Actions** drop-down list, select **GetQueueUrl** box.
- 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

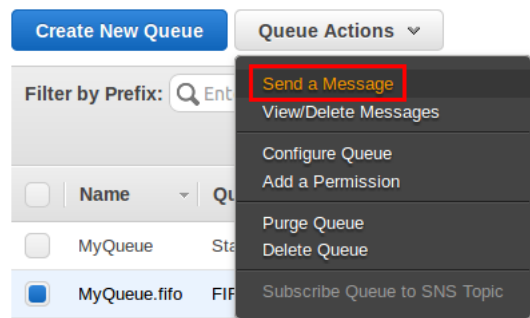
After you create your queue, you can send a message to it. The following example demonstrates sending a message to an existing queue.

AWS Management Console

1. Sign in to the AWS Management Console and open the Amazon SQS console at <https://console.aws.amazon.com/sqs/>.
2. From the queue list, select a queue.



3. From the **Queue Actions** drop-down list, 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 is disabled).

Send a Message to MyQueue.fifo X

Message Body Message Attributes

Enter the text of a message you want to send.

This is my message text.

Message Group ID ⓘ Type a FIFO message group (required).

Message Deduplication ID ⓘ Type a deduplication token (required).

Cancel Send Message

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. 40).

Note

The message group ID is always required. However, if content-based deduplication is enabled, the message deduplication ID is optional.

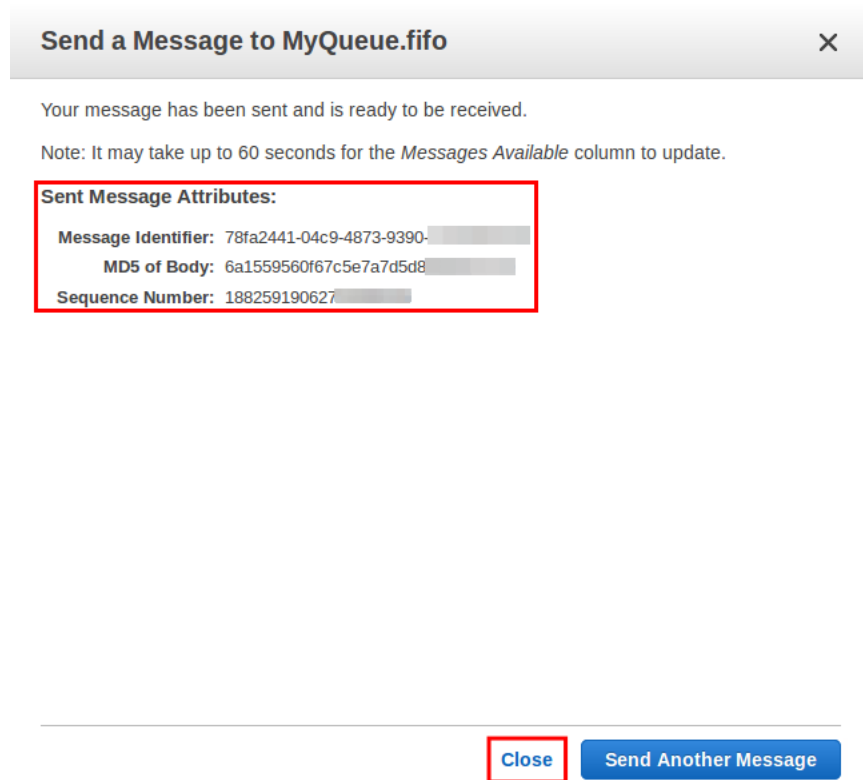
Message Group ID ⓘ MyMessageGroupId1234567890

Message Deduplication ID ⓘ MyMessageDeduplicationId1234567890

Cancel Send Message

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 for Development](#) in the *AWS SDK for Java Developer Guide*.

To send a message to a standard queue

1. Copy the [example program \(p. 36\)](#).

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

```
// Send a message
System.out.println("Sending a message to MyQueue.\n");
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](#)).
 - The *request ID* that Amazon SQS assigned to your request.

To send a message to a FIFO queue

1. Copy the [example program](#) (p. 41).

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

```
// 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 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.

Note

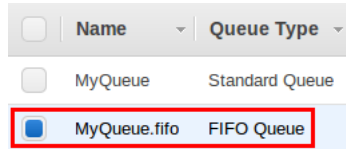
Because Amazon SQS is a distributed system, a queue with very few messages might display an empty response to a receive request. In this case, you can rerun the request to get your message. Depending on your application's needs, you might have to use short or [long polling](#) (p. 62) to receive messages.

Amazon SQS doesn't automatically delete a message after retrieving it for you, in case you don't successfully receive the message (for example, the receiver 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.

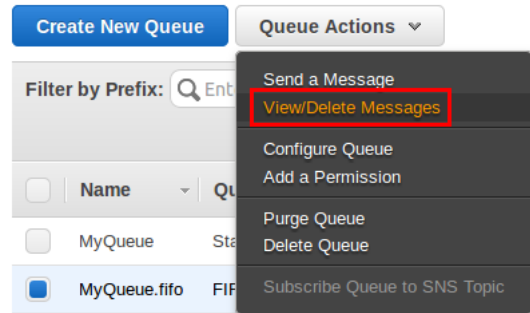
The following example demonstrates receiving and deleting a message.

AWS Management Console

1. Sign in to the AWS Management Console and open the Amazon SQS console at <https://console.aws.amazon.com/sqs/>.
2. From the queue list, select a queue.



3. From the **Queue Actions** drop-down list, select **View/Delete Messages**.

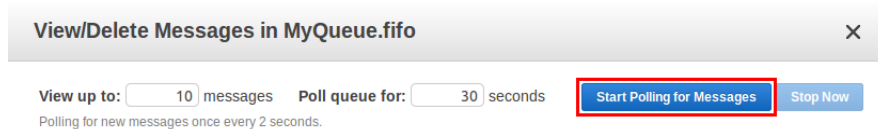


The **View/Delete Messages** in *QueueName* dialog box is displayed.

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.

4. 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.

View/Delete Messages in MyQueue.fifo

View up to: 10 messages

Poll queue for: 30 seconds

Polling for Messages...

Stop Now

View up to: 10 messages

Poll queue for: 30 seconds

Polling for new messages once every 2 seconds.

Delete	Body	Message Group ID	Message Deduplication ID	Sequence Number
<input type="checkbox"/>	This is my message text.	MyMessageGroupI...	MyMessageDeduplicati...	188259353925

54%

Close

Delete Messages

- Before the visibility timeout expires, select the message that you want to delete and then choose **Delete 1 Message**.

View/Delete Messages in MyQueue.fifo

View up to: 10 messages

Poll queue for: 30 seconds

Start Polling for Messages

Stop Now

View up to: 10 messages

Poll queue for: 30 seconds

Polling for new messages once every 2 seconds.

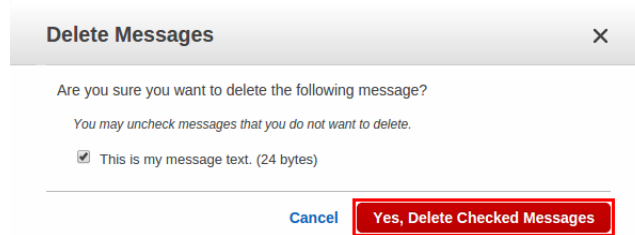
Delete	Body	Message Group ID	Message Deduplication ID	Sequence Number
<input checked="" type="checkbox"/>	This is my message text.	MyMessageGroupI...	MyMessageDeduplicati...	188259353925

54%

Close

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 \(p. 47\)](#) 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 for Development](#) in the *AWS SDK for Java Developer Guide*.

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

1. Copy the [example program \(p. 36\)](#).

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

```
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();
```

The following section of the code deletes the message:

```
// Delete a message
System.out.println("Deleting a message.\n");
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 a message from a FIFO queue

1. Copy the [example program \(p. 41\)](#).

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

```
// 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();
```

The following section of the code deletes the message:

```
// Delete the message
System.out.println("Deleting the message.\n");
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: Purging Messages from an Amazon SQS Queue

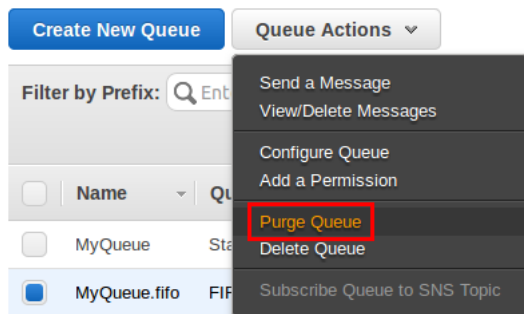
If you don't want to delete an Amazon SQS queue but need to delete all the messages from it, you can purge the queue. The following example demonstrates purging a queue.

AWS Management Console

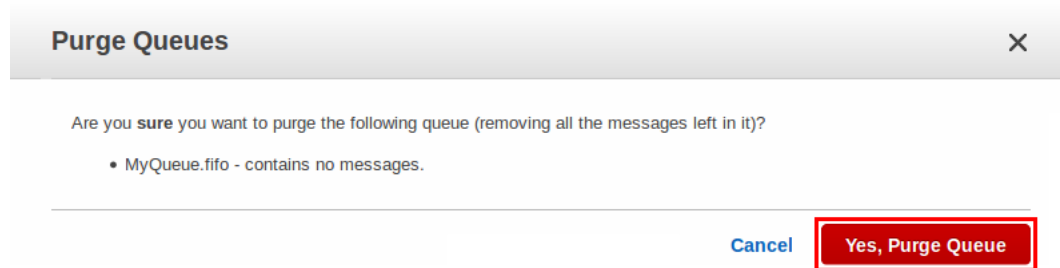
1. Sign in to the AWS Management Console and open the Amazon SQS console at <https://console.aws.amazon.com/sqs/>.
2. From the queue list, select a queue.



3. From the **Queue Actions** drop-down list, select **Purge Queue**.



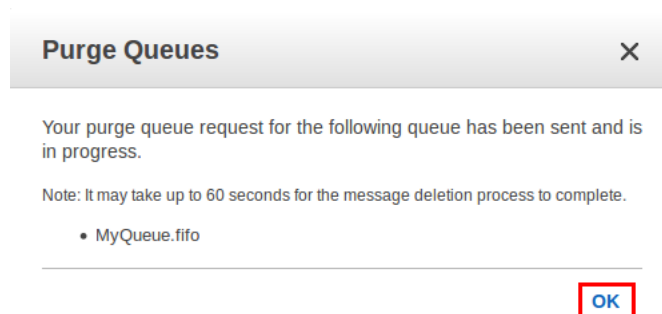
The **Purge Queues** dialog box is displayed.



4. Choose **Yes, Purge Queue**.

All messages are purged from the queue.

The **Purge Queues** confirmation dialog box is displayed.



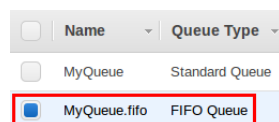
5. Choose **OK**.

Tutorial: Deleting an Amazon SQS Queue

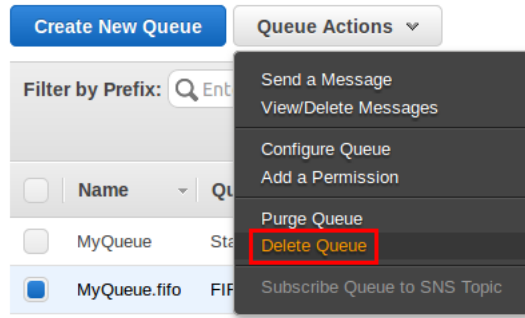
If you don't use an Amazon SQS queue (and don't foresee using it in the near future), it is a best practice to delete it from Amazon SQS. The following example demonstrates deleting a queue.

AWS Management Console

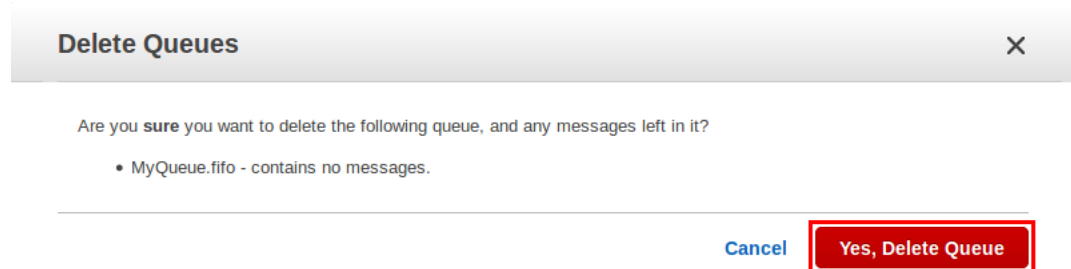
1. Sign in to the AWS Management Console and open the Amazon SQS console at <https://console.aws.amazon.com/sqs/>.
2. From the queue list, select a queue.



3. From the **Queue Actions** drop-down list, select **Delete Queue**.



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 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 \(p. 36\)](#) or the [FIFO queue example program \(p. 41\)](#).

The following section of the code deletes the queue:

```
// 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 an Amazon SQS message to a topic, Amazon SNS sends the message to any subscribed queues. 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.

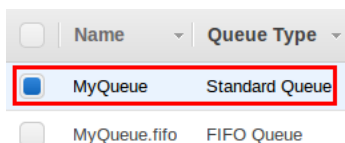
Note

Amazon SNS isn't currently compatible with FIFO queues.

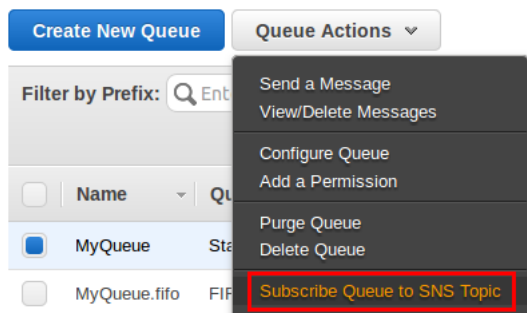
When you subscribe an Amazon SQS queue to an Amazon SNS topic, Amazon SNS uses HTTPS to forward messages to Amazon SQS.

AWS Management Console

1. Sign in to the AWS Management Console and open the Amazon SQS console at <https://console.aws.amazon.com/sqs/>.
2. From the list of queues, choose the queue (or queues) to which you want to subscribe an Amazon SNS topic.



3. From the **Queue Actions** drop-down list, 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**.

Subscribe to a Topic ✕

Select an SNS Topic from the *Choose a Topic* drop-down or enter a topic's ARN in the *Topic ARN* text box and then press *Subscribe* to allow your queue(s) to receive SNS notifications from the topic and to subscribe your queue(s) to the topic.

Topic Region ⓘ US West (Oregon) ▼

Choose a Topic ⓘ MyTopic ▼

Topic ARN ⓘ arn:aws:sns:us-west-2: :MyTopic

[Cancel](#) [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**.

Topic Subscription Result ✕

Successfully subscribed the following queue to the SNS topic MyTopic.
Permission to receive SNS notifications was added to the queue.

- MyQueue

[OK](#)

The queue is subscribed to the topic.

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*.

How Amazon SQS Queues Work

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

Topics

- [Basic Prerequisites](#) (p. 34)
- [Standard Queues](#) (p. 35)
- [FIFO \(First-In-First-Out\) Queues](#) (p. 39)
- [Queue and Message Identifiers](#) (p. 45)
- [Resources Required to Process Messages](#) (p. 47)
- [Visibility Timeout](#) (p. 47)
- [Message Lifecycle](#) (p. 50)
- [Using Amazon SQS Dead Letter Queues](#) (p. 51)
- [Using Amazon SQS Message Attributes](#) (p. 54)
- [Amazon SQS Long Polling](#) (p. 62)
- [Amazon SQS Delay Queues](#) (p. 66)
- [Amazon SQS Message Timers](#) (p. 70)
- [Managing Large Amazon SQS Messages Using Amazon S3](#) (p. 73)
- [Using JMS with Amazon SQS](#) (p. 77)

Basic Prerequisites

The following basic prerequisites will 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 could get a list of all your queues whose names start with `τ3`).
- 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.

To do this...	Use this action
Create a queue	CreateQueue
Get the URL of an existing queue	GetQueueUrl
List your queues	ListQueues
Delete a queue	DeleteQueue

Standard Queues

Amazon SQS offers *standard* as the default queue type. A standard queue allows you to have a nearly-unlimited number of transactions per second. 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 \(p. 98\)](#).

Topics

- [Message Ordering \(p. 35\)](#)
- [At-Least-Once Delivery \(p. 35\)](#)
- [Retrieving Messages Using Short Polling \(p. 36\)](#)
- [Getting Started with Standard Queues \(p. 36\)](#)

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 either using a *FIFO (First-In-First-Out) queue* ([p. 39](#)) or adding sequencing information in each message so you can reorder the messages when they're received.

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 will not be deleted on that unavailable server, and you might get that message copy again when you receive messages. You should design your applications to be

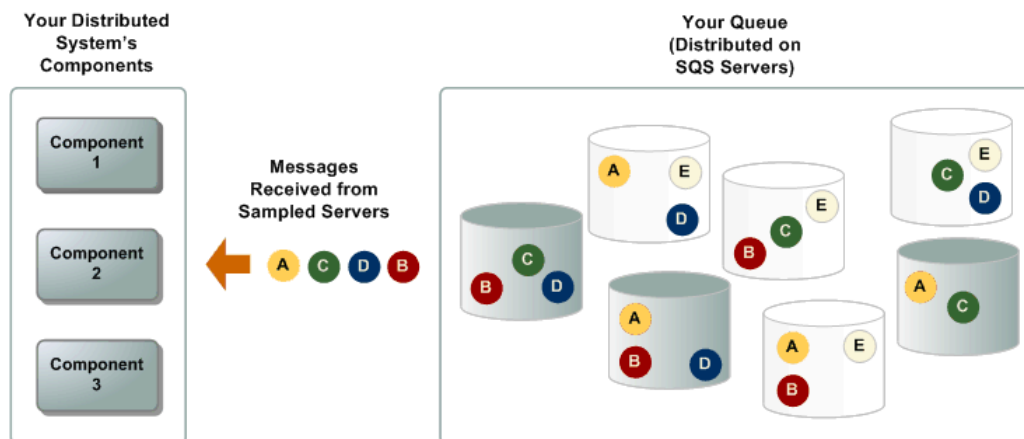
idempotent (they should not be affected adversely when processing the same message more than once).

Retrieving Messages Using Short Polling

The behavior of retrieving 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. 62).

When you retrieve 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 will. If you keep retrieving from your queues, Amazon SQS will sample all of the servers, and you'll 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 it will be 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.

```
/*
 * Copyright 2010-2016 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.
 * A copy of the License is located at
 *
 * http://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under 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.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;

/**
 * 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 on 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 will load the credentials from.
 *
 * WARNING: 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 will return 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 = new AmazonSQSClient(credentials);
        Region usWest2 = Region.getRegion(Regions.US_WEST_2);
        sqs.setRegion(usWest2);
    }
}
```

```
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.\\n"));

    // 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.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.");  
        System.out.println("Error Message: " + ace.getMessage());  
    }  
}
```

FIFO (First-In-First-Out) Queues

FIFO queues are available in US West (Oregon) and US East (Ohio). In addition to having all the capabilities of the [standard queue](#) (p. 35), *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 are limited to 300 transactions per second (TPS).

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. 39), 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. 99) and [General Recommendations](#) (p. 98).

For information on compatibility of clients and services with FIFO queues, see [Compatibility](#) (p. 45).

Topics

- [Message Ordering](#) (p. 40)
- [FIFO Queue Logic](#) (p. 40)
- [Exactly-Once Processing](#) (p. 41)
- [Getting Started with FIFO Queues](#) (p. 41)
- [Moving From a Standard Queue to a FIFO Queue](#) (p. 44)
- [Compatibility](#) (p. 45)

Message Ordering

The FIFO queue improves upon and complements the [standard queue \(p. 35\)](#). 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 will 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.

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 processed in a FIFO manner (however, messages in 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 sender 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 sender or receiver to attempt multiple retries:

- If the sender detects a failed `SendMessage` action, it can retry sending as many times as necessary, using the same message deduplication ID. Assuming that the sender receives at least one acknowledgement before the deduplication interval expires, multiple retries neither affect the ordering of messages nor introduce duplicates.
- If the receiver detects a failed `ReceiveMessage` action, it can retry as many times as necessary, using the same receive request attempt ID. Assuming that the receiver 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.

```
/*
 * Copyright 2010-2016 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.
 * A copy of the License is located at
 *
 * http://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 */

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("=====
\n");

        try {
            // 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();

        // 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 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");

        // 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.");
        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.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.");
        System.out.println("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 for your receiver. 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 will allow you to retry receive attempts in case of networking failures and will prevent queues from pausing due to failed receive attempts.

For more information, see the [Amazon Simple Queue Service API Reference](#).

Compatibility

Clients

The following clients don't currently support FIFO queues:

- Amazon SQS Buffered Asynchronous Client
- Amazon SQS Extended Client Library for Java
- Amazon SQS Java Message Service (JMS) Client

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:

- [Amazon CloudWatch Events](#)
- [Amazon S3 Event Notifications](#)
- [Amazon SNS Topic Subscriptions](#)

Note

You can use Amazon SNS to forward messages to Amazon SQS standard queues.

- [Auto Scaling Lifecycle Hooks](#)
- [AWS IoT Rule Actions](#)
- [AWS Lambda Dead Letter Queues](#)

For information about compatibility of other services with FIFO queues, see your service documentation.

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 \(p. 39\)](#), 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+JvwWJaBV+3dcjk2YW2vA3+STFFljTM8tJJg6HRG6PYSasuWXPJB+Cw  
Ljl1FjgXUv1uSj1gUPAWV66FU/WeR4mq2OKpEGYWbnLmpRCJVAyeMjeU5ZBdtcQ+QE  
auMZc8ZRv37sIW2iJKq3M9MFx1YvV11A2x/KSbkJ0=
```

Additional Identifiers for FIFO Queues

For more information about the following identifiers, see [Exactly-Once Processing \(p. 41\)](#) 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 processed in a FIFO manner (however, messages in 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. 47).

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.

To do this...	Use this action	Use this AttributeName
Get the approximate number of messages in the queue.	GetQueueAttributes	<code>ApproximateNumberOfMessages</code>
Get the approximate number of messages that are pending to be added to the queue.	GetQueueAttributes	<code>ApproximateNumberOfMessagesDelayed</code>
Get the approximate number of messages in the queue that are not visible (messages in flight).	GetQueueAttributes	<code>ApproximateNumberOfMessagesNotVisible</code>

Visibility Timeout

Topics

- [Inflight Messages](#) (p. 48)
- [Configuring the Visibility Timeout](#) (p. 48)
- [Changing a Message's Visibility Timeout](#) (p. 49)
- [Terminating a Message's Visibility Timeout](#) (p. 49)
- [API Actions Related to Visibility Timeout](#) (p. 49)

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 it's a distributed system, there is no guarantee that the component will actually receive the message (the connection can break or a component can fail to receive the message). 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 receivers from processing the message again, Amazon SQS sets a *visibility timeout*, a period of time during which Amazon SQS prevents other consuming components 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 \(p. 35\)](#).

FIFO queues allow the sender or receiver to attempt multiple retries:

- If the sender detects a failed `SendMessage` action, it can retry sending as many times as necessary, using the same message deduplication ID. Assuming that the sender receives at least one acknowledgement before the deduplication interval expires, multiple retries neither affect the ordering of messages nor introduce duplicates.
- If the receiver detects a failed `ReceiveMessage` action, it can retry as many times as necessary, using the same receive request attempt ID. Assuming that the receiver 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.

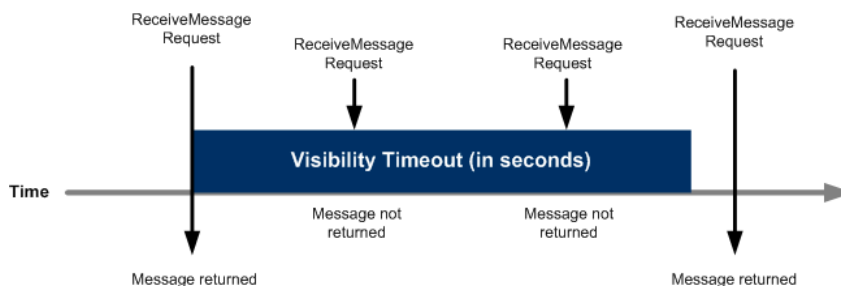
Inflight Messages

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.

The following figure illustrates the visibility timeout.



Configuring the Visibility Timeout

The visibility timeout clock starts ticking once Amazon SQS returns the message. During that time, the component processes and deletes the message. But what happens if the component fails before deleting the message? If your system doesn't call `DeleteMessage` for that message before the visibility

timeout expires, the message again becomes visible to the [ReceiveMessage](#) calls placed by the components in your system and it will be received again. If a message should only be received once, your system should delete it within the duration of the visibility timeout.

Each queue starts with a default setting of 30 seconds for the visibility timeout. You can change that setting for the entire queue. Typically, you'll 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 a Message's Visibility Timeout

When you receive a message for 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](#) action.

For example, if the timeout for a queue is 60 seconds, 15 seconds have elapsed, and you send a [ChangeMessageVisibility](#) call with `VisibilityTimeout` set to 10 seconds, the total timeout value will be the elapsed time (15 seconds) plus the new timeout value (10 seconds), a total of 25 seconds. Sending a call after 25 seconds will result in an error.

Note

The new timeout period will take effect from the time you call the [ChangeMessageVisibility](#) action. In addition, the new timeout period will apply only to the particular receipt of the message. The [ChangeMessageVisibility](#) action does not affect the timeout of later receipts of the message or later queues.

Terminating a Message's Visibility Timeout

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.

API Actions Related to Visibility Timeout

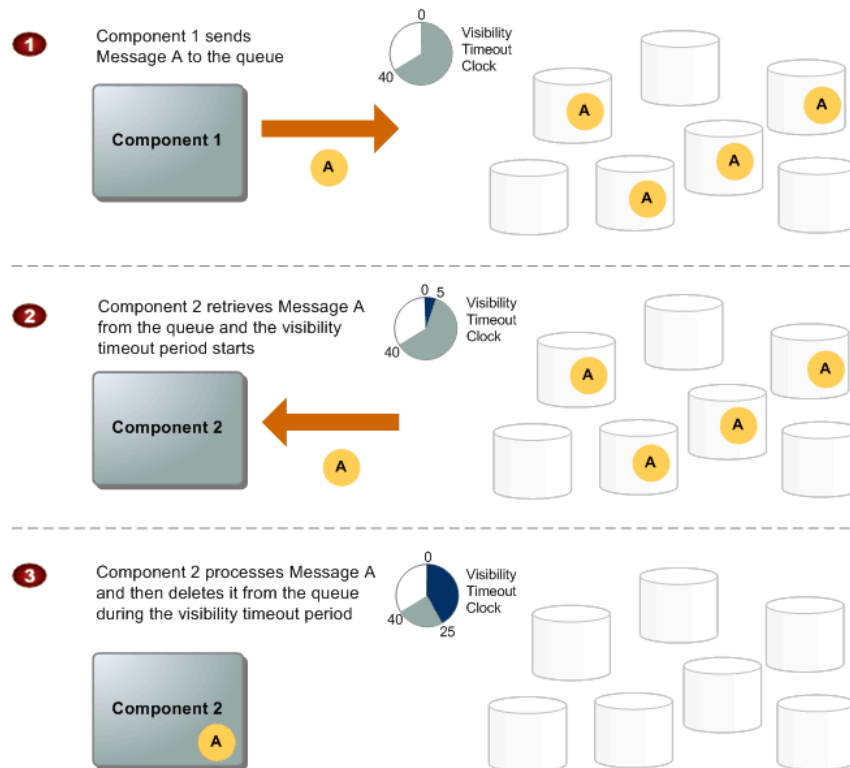
The following table lists the API actions to use to manipulate the visibility timeout. Use each action's `VisibilityTimeout` parameter to set or get the value.

To do this...	Use this action
Set the visibility timeout for a queue	SetQueueAttributes
Get the visibility timeout for a queue	GetQueueAttributes
Set the visibility timeout for the received messages without affecting the queue's visibility timeout	ReceiveMessage
Extending or terminating a message's visibility timeout	ChangeMessageVisibility

To do this...	Use this action
Extending or terminating the visibility timeout for up to ten messages.	ChangeMessageVisibilityBatch

Message Lifecycle

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



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 retrieves 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.
3	Component 2 deletes Message A from the queue to prevent the message from being received and processed again once the visibility timeout expires.

Note

Amazon SQS automatically deletes messages that have been in a queue for more than maximum message retention period. The default message retention period is 4 days. However, you can set the message retention period to a value from 60 seconds to 1,209,600 seconds (14 days) using the [SetQueueAttributes](#) action.

Using Amazon SQS Dead Letter Queues

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

Note

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.

To specify a dead letter queue, you can use the AWS Management Console or the query API. You must do this for each queue that will send messages to a dead letter queue. Multiple queues can target a single dead letter queue.

Important

You must use the same AWS account to create the dead letter queue and the other queues that will 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, then both queues must be in the US East (Ohio) region.

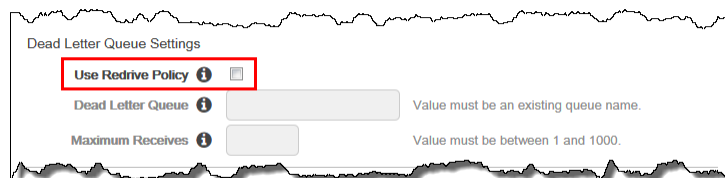
Topics

- [Setting up a Dead Letter Queue with the AWS Management Console \(p. 51\)](#)
- [Using a Dead Letter Queue with the Amazon SQS API \(p. 52\)](#)
- [Issue: Viewing Messages with the Amazon SQS Console Can Cause the Messages to be Moved to a Dead Letter Queue \(p. 53\)](#)
- [Issue: NumberOfMessagesSent and NumberOfMessagesReceived for the Dead Letter Queue Do Not Match \(p. 54\)](#)

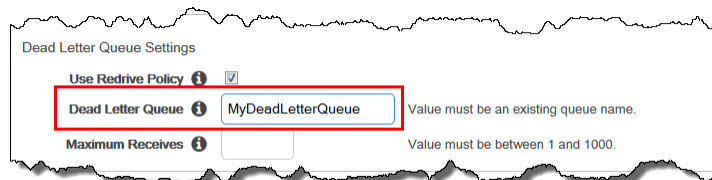
Setting up a Dead Letter Queue with the AWS Management Console

You can use the AWS Management Console to send messages that could not be processed successfully to a specified dead letter queue.

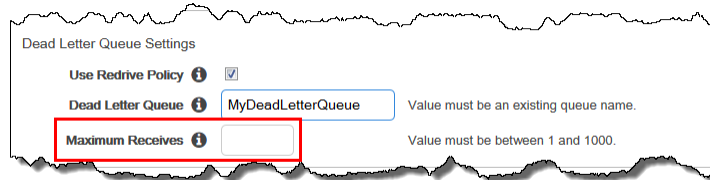
1. For existing and newly-created source queues, select **Use Redrive Policy**.



2. Type the name of the queue to which source queues will send messages.



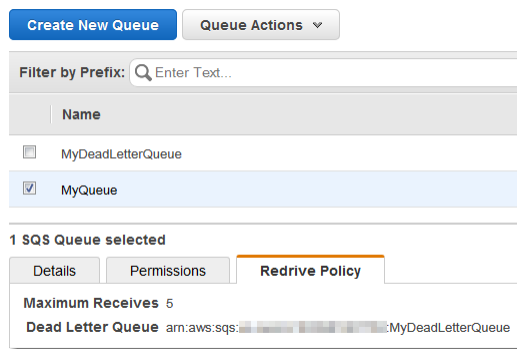
3. Set **Maximum Receives** to a value between 1 and 1,000.



Note

The **Maximum Receives** setting (the number of times that a message can be received before being sent to a dead letter queue) applies only to individual messages.

The following figure shows *MyQueue* with a **Redrive Policy** configured to send messages to *MyDeadLetterQueue*.



Using a Dead Letter Queue with the Amazon SQS API

To specify a dead letter queue using the query API, call either the `CreateQueue` or `SetQueueAttributes` actions and set the `maxReceiveCount` and `deadLetterTargetArn` parameters for the `RedrivePolicy` queue attribute.

You can set `maxReceiveCount` to a value between 1 and 1,000. The `deadLetterTargetArn` value is the Amazon Resource Name (ARN) of the queue that will receive the dead letter messages.

The following Java example shows how to use `SetQueueAttributes` to set the `maxReceiveCount` and `deadLetterTargetArn` parameters for the `RedrivePolicy` queue attribute. This example is based on the `SimpleQueueServiceSample.java` sample from the AWS SDK for Java.

First, set a string that contains JSON-formatted parameters and values for the `RedrivePolicy` queue attribute:

```
String redrivePolicy = "{\"maxReceiveCount\": \"5\", \"deadLetterTargetArn\": \"arn:aws:sqs:us-east-2:123456789012:MyDeadLetterQueue\"}";
```

Next, use `SetQueueAttributesRequest` to set the `RedrivePolicy` queue attribute:

```
SetQueueAttributesRequest queueAttributes = new SetQueueAttributesRequest();  
Map<String,String> attributes = new HashMap<String,String>();
```

Amazon Simple Queue Service Developer Guide
Issue: Viewing Messages with the Amazon
SQS Console Can Cause the Messages
to be Moved to a Dead Letter Queue

```
attributes.put("RedrivePolicy", redrivePolicy);  
queueAttributes.setAttributes(attributes);  
queueAttributes.setQueueUrl(myQueueUrl);  
sqs.setQueueAttributes(queueAttributes);
```

An API query request for this example should look similar to the following:

```
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%3A123456789012%3AMyDeadLetterQueue%22%7D  
&Version=2012-11-05  
&Attribute.1.Name=RedrivePolicy
```

Note

Queue names and queue URLs are case-sensitive.

The API query response should look similar to the following:

```
<SetQueueAttributesResponse xmlns="http://queue.amazonaws.com/  
doc/2012-11-05/">  
  <ResponseMetadata>  
    <RequestId>40945605-b328-53b5-aed4-1cc24a7240e8</RequestId>  
  </ResponseMetadata>  
</SetQueueAttributesResponse>
```

Issue: Viewing Messages with the Amazon SQS Console Can Cause the Messages to be Moved to a Dead Letter Queue

Amazon SQS counts viewing a message in the Amazon SQS console against the corresponding queue's redrive policy. Thus, if you view a message in the Amazon SQS 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 either increase the **Maximum Receives** setting for the corresponding queue's redrive policy, or avoid viewing the corresponding queue's messages in the Amazon SQS console.

To reproduce this behavior:

1. In the Amazon SQS console, [create queues \(p. 15\)](#) named *MyQueueA* and *MyQueueB*.
2. From the list of queues, select **MyQueueA**. Then choose **Queue Actions**, **Configure Queue**.

The **Configure MyQueueA** dialog box is displayed.

3. For **Dead Letter Queue Settings**, select **Use Redrive Policy**.

For **Dead Letter Queue**, type *MyQueueB*. (*MyQueueB* will become the dead letter queue for *MyQueueA*.)

For **Maximum Receives**, type 2 and then choose **Save Changes**. (After a message is viewed twice, it will be sent to *MyQueueB*, the dead letter queue.)

4. With **MyQueueA** still selected in the list of queues, choose **Queue Actions, Send a Message**.

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

5. On the **Message Body** tab, type any message text, and then choose **Send Message**.

Choose **Send Another Message**, and then send another message to *MyQueueA*. Then choose **Close**.

6. With **MyQueueA** still selected in the list of queues, choose **Queue Actions, View/Delete Messages**.

The **Start Polling for Messages** dialog box is displayed.

7. Choose **Start Polling for Messages**. The messages you sent are displayed.
8. Wait for 1 minute, and then choose **Start Polling for Messages** again. The messages you sent are displayed again. (The **Receive Count** for both messages becomes 2. This is equal to the **Maximum Receives** value that you specified.)
9. Wait for 1 minute, and then choose **Start Polling for Messages** again. This time, the messages you sent are no longer displayed. (You have already viewed the messages twice. This is equal to the **Maximum Receives** value that you specified.) Choose **Close**.
10. In the list of queues, clear **MyQueueA**. Then, select *MyQueueB*, and choose **Queue Actions, View/Delete Messages**.

Choose **Start Polling for Messages**. The messages you sent are displayed. (*MyQueueB* becomes the dead letter queue for *MyQueueA*.)

Issue: NumberOfMessagesSent and NumberOfMessagesReceived for the Dead Letter Queue Do Not Match

If you send a message to a dead letter queue manually, it will be captured by the `NumberOfMessagesSent` metric.

However, if a message is sent to a dead letter queue as a result of a failed processing attempt, it won't be captured by the `NumberOfMessagesSent` metric. In this case, it's possible for the values of the `NumberOfMessagesSent` and `NumberOfMessagesReceived` metrics to be different.

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 receiver 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 \(p. 55\)](#)
- [Message Attribute Data Types and Validation \(p. 55\)](#)

- [Using Message Attributes with the AWS Management Console \(p. 56\)](#)
- [Using Message Attributes with the AWS SDKs \(p. 57\)](#)
- [Using Message Attributes with the Amazon SQS Query API \(p. 59\)](#)
- [MD5 Message-Digest Calculation \(p. 61\)](#)

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 on 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 \(p. 55\)](#) 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 will validate that it's a number.

Amazon SQS supports the following logical data types (with optional custom type labels):

String	.<Custom Type> (Optional)
Number	.<Custom Type> (Optional)
Binary	.<Custom Type> (Optional)

- **String** – Strings are Unicode with UTF-8 binary encoding. For a list of code values, see http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters.
- **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} to 10^{+126} . Leading and trailing zeroes are trimmed.
- **Binary** – Binary type attributes can store any binary data, for example, compressed data, encrypted data, or images.
- **Custom Type** – You can append a custom type label to the supported data types (String, Number, and Binary) 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, then you could 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.

The screenshot shows the 'Send a Message to MyQueue' dialog box with the 'Message Attributes' tab selected. The 'Name' field is empty. The 'Type' dropdown is set to 'String', and the '(Custom Type: Optional)' field is empty. The 'Value' field is empty. Below the 'Value' field is a small text prompt 'Enter a string value.' and a blue 'Add Attribute' button. To the right of the button is a link 'What is Message Attribute?'. Below these fields is a table with three columns: 'Name', 'Type', and 'Values'. The table is currently empty. At the bottom of the dialog are 'Cancel' and 'Send Message' buttons.

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. 55) section.

This screenshot shows the same 'Send a Message to MyQueue' dialog box, but with example values entered. The 'Name' field contains 'MyMessageAttributeName'. The 'Type' dropdown is set to 'Number', and the '(Custom Type: Optional)' field contains 'byte'. The 'Value' field contains '24'. Below the 'Value' field is a small text prompt 'Enter a numerical value.' and a blue 'Add Attribute' button. To the right of the button is a link 'What is Message Attribute?'. Below these fields is a table with three columns: 'Name', 'Type', and 'Values'. The table is currently empty. At the bottom of the dialog are 'Cancel' and 'Send Message' buttons.

To add an attribute, choose **Add Attribute**. The attribute information will then appear in the **Name**, **Type**, and **Values** list.

Send a Message to MyQueue

Message Body | Message Attributes

Name:

Type: String (Custom Type: Optional)

Value:

Enter a string value.

Add Attribute [What is Message Attribute?](#)

Name	Type	Values
MyMessage...	Number.byte	24

Cancel **Send Message**

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.

Message Details

Message Body | Message Attributes

Name	Type	Value
MyMessageAtt...	Number.byte	24

Message Attributes Size: 35 bytes

MD5 of Message Attributes: 65fbeb22c3b0bf5014c83ab49...

Close

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

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

Number

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

Binary

```
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

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

Number—Custom

```
// NOTE Because the type is a number, the result in the receive message call
// will be 123456.
Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
messageAttributes.put("AccountId", new
    MessageAttributeValue().withDataType("Number.AccountId").withStringValue("000123456"));
```

Binary—Custom

```
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:

```
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.

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

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

```
POST 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:

```
HTTP/1.1 200 OK
...
<SendMessageResponse>
  <SendMessageResult>
    <MD5OfMessageBody>
      fafb00f5732ab283681e124bf8747ed1
    </MD5OfMessageBody>
    <MD5OfMessageAttributes>
      3ae8f24a165a8cedc005670c81a27295
    </MD5OfMessageAttributes>
    <MessageId>
```

```
5fea7756-0ea4-451a-a703-a558b933e274
</MessageId>
</SendMessageResult>
<ResponseMetadata>
  <RequestId>
    27daac76-34dd-47df-bd01-1f6e873584a0
  </RequestId>
</ResponseMetadata>
</SendMessageResponse>
```

When using [SendMessageBatch](#), the message attributes need to be specified on each individual message in the batch.

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

```
POST 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
...
<SendMessageBatchResponse>
<SendMessageBatchResult>
  <SendMessageBatchResultEntry>
    <Id>test_msg_001</Id>
    <MessageId>0a5231c7-8bff-4955-be2e-8dc7c50a25fa</MessageId>
    <MD5OfMessageBody>0e024d309850c78cba5eabbef77cae71</MD5OfMessageBody>
  </SendMessageBatchResultEntry>
  <SendMessageBatchResultEntry>
    <Id>test_msg_002</Id>
    <MessageId>15eeled3-87e7-40c1-bdaa-2e49968ea7e9</MessageId>
    <MD5OfMessageBody>7fb8146a82f95e0af155278f406862c2</MD5OfMessageBody>
    <MD5OfMessageAttributes>295c5fa15a51aae6884d1d7c1d99ca50</MD5OfMessageAttributes>
  </SendMessageBatchResultEntry>
</SendMessageBatchResult>
<ResponseMetadata>
  <RequestId>calad5d0-8271-408b-8d0f-1351bf547e74</RequestId>
</ResponseMetadata>
</SendMessageBatchResponse>
```

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:

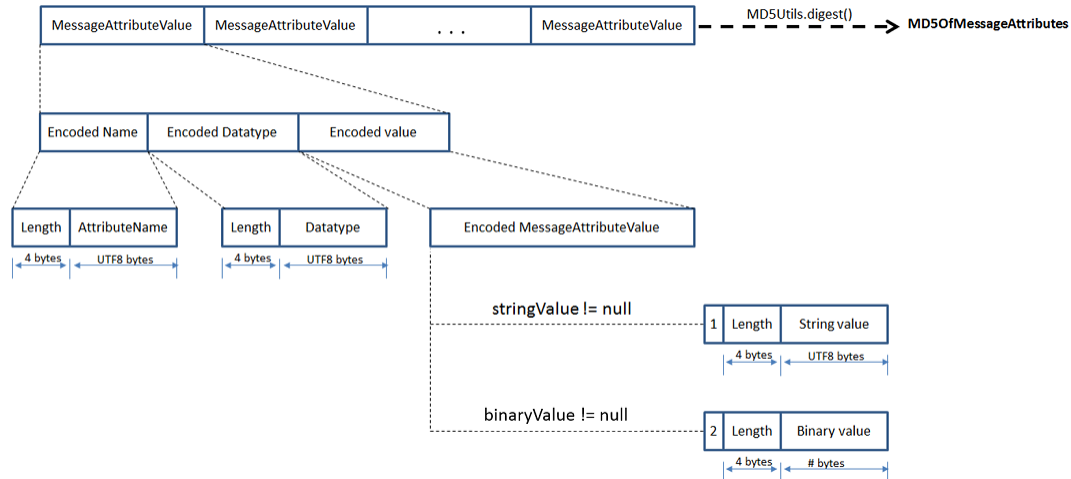
1. Encode the name (length of name [4 bytes] + UTF-8 bytes of the name).
2. Encode the type (length of type [4 bytes] + UTF-8 bytes of the type).
3. Encode the transport type (string or binary) of the value [1 byte].
 - a. For the *string* transport type, encode 1.
 - b. For the *binary* transport type, 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

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`.

Topics

- [Benefits of Long Polling \(p. 62\)](#)
- [Enabling Long Polling with the AWS Management Console \(p. 63\)](#)
- [Enabling Long Polling Using the API \(p. 65\)](#)
- [Enabling Long Polling Using the Query API \(p. 65\)](#)

Benefits of 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.

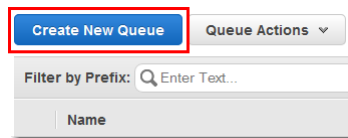
- Long polling returns messages as soon any message becomes available.

Enabling Long Polling with 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 AWS Management Console and open the Amazon SQS console at <https://console.aws.amazon.com/sqs/>.
2. Select **Create New Queue**.



3. In the **Create New Queue** dialog box, type the **Queue Name**.

A screenshot of the 'Create New Queue' dialog box. The 'Region' is set to 'US West (Oregon)'. The 'Queue Name' field, containing 'MyQueue', is highlighted with a red rectangular box. Below this, there are several configuration options: 'Default Visibility Timeout' (30 seconds), 'Message Retention Period' (4 days), 'Maximum Message Size' (256 KB), 'Delivery Delay' (0 seconds), and 'Receive Message Wait Time' (0 seconds). Each option has a help icon and a validation message. At the bottom, there are 'Dead Letter Queue Settings' including 'Use Redrive Policy' (unchecked), 'Dead Letter Queue' (empty field), and 'Maximum Receives' (empty field). The dialog box has 'Cancel' and 'Create Queue' buttons at the bottom right.

4. For **Receive Message Wait Time**, type a positive integer value, from 1 to 20 seconds, .

Create New Queue [X]

Region ⓘ US West (Oregon)

Queue Name ⓘ MyQueue

Configure your new queue by setting queue attributes (optional).

Default Visibility Timeout ⓘ 30 seconds Value must be between 0 seconds and 12 hours.

Message Retention Period ⓘ 4 days Value must be between 1 minute and 14 days.

Maximum Message Size ⓘ 256 KB Value must be between 1 and 256 KB.

Delivery Delay ⓘ 0 seconds Value must be between 0 seconds and 15 minutes.

Receive Message Wait Time ⓘ 10 seconds Value must be between 0 and 20 seconds.

Dead Letter Queue Settings

Use Redrive Policy ⓘ ☐

Dead Letter Queue ⓘ Value must be an existing queue name.

Maximum Receives ⓘ Value must be between 1 and 1000.

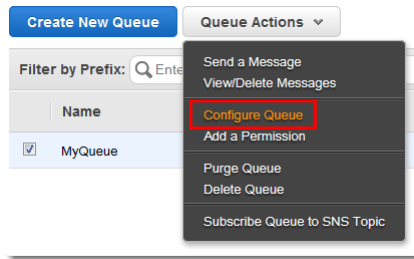
Cancel Create Queue

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.

Configure MyQueue

Queue Settings

Default Visibility Timeout 30 seconds Value must be between 0 seconds and 12 hours.

Message Retention Period 4 days Value must be between 1 minute and 14 days.

Maximum Message Size 256 KB Value must be between 1 and 256 KB.

Delivery Delay 0 seconds Value must be between 0 seconds and 15 minutes.

Receive Message Wait Time 5 seconds Value must be between 0 and 20 seconds.

Dead Letter Queue Settings

Use Redrive Policy ☒

Dead Letter Queue Value must be an existing queue name.

Maximum Receives Value must be between 1 and 1000.

Cancel Save Changes

4. Choose **Save Changes**.

Enabling Long Polling Using the API

The following table lists the API actions to use.

Use this action	Use...
ReceiveMessage	WaitTimeSeconds parameter
CreateQueue	ReceiveMessageWaitTimeSeconds attribute
SetQueueAttributes	ReceiveMessageWaitTimeSeconds attribute

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.

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

```
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

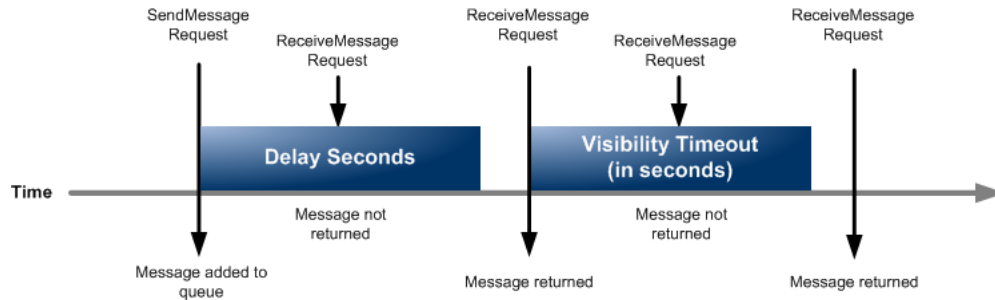
- [Creating Delay Queues with the AWS Management Console \(p. 67\)](#)
- [Creating Delay Queues with the Query API \(p. 69\)](#)

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 retrieved from the queue. The following figure illustrates the relationship between delay queues and visibility timeouts.



Note

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.

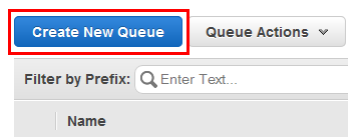
To set delay seconds on individual messages, rather than for an entire queue, use message timers. If you send a message with a message timer, Amazon SQS uses the message timer's delay seconds value instead of the delay queue's delay seconds value. For more information, see [Amazon SQS Message Timers \(p. 70\)](#).

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 AWS Management Console and open the Amazon SQS console at <https://console.aws.amazon.com/sqs/>.
2. Choose **Create New Queue**.



3. In the **Create New Queue** dialog box, type your **Queue Name**.

Amazon Simple Queue Service Developer Guide

Creating Delay Queues with the AWS Management Console

Create New Queue [X]

Region ⓘ US West (Oregon)

Queue Name ⓘ MyQueue

Configure your new queue by setting queue attributes (optional).

Default Visibility Timeout ⓘ 30 seconds Value must be between 0 seconds and 12 hours.

Message Retention Period ⓘ 4 days Value must be between 1 minute and 14 days.

Maximum Message Size ⓘ 256 KB Value must be between 1 and 256 KB.

Delivery Delay ⓘ 0 seconds Value must be between 0 seconds and 15 minutes.

Receive Message Wait Time ⓘ 0 seconds Value must be between 0 and 20 seconds.

Dead Letter Queue Settings

Use Redrive Policy ⓘ ☐

Dead Letter Queue ⓘ Value must be an existing queue name.

Maximum Receives ⓘ Value must be between 1 and 1000.

Cancel Create Queue

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

Create New Queue [X]

Region ⓘ US West (Oregon)

Queue Name ⓘ MyQueue

Configure your new queue by setting queue attributes (optional).

Default Visibility Timeout ⓘ 30 seconds Value must be between 0 seconds and 12 hours.

Message Retention Period ⓘ 4 days Value must be between 1 minute and 14 days.

Maximum Message Size ⓘ 256 KB Value must be between 1 and 256 KB.

Delivery Delay ⓘ 30 seconds Value must be between 0 seconds and 15 minutes.

Receive Message Wait Time ⓘ 0 seconds Value must be between 0 and 20 seconds.

Dead Letter Queue Settings

Use Redrive Policy ⓘ ☐

Dead Letter Queue ⓘ Value must be an existing queue name.

Maximum Receives ⓘ Value must be between 1 and 1000.

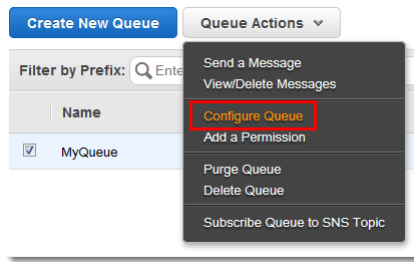
Cancel Create Queue

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.

A screenshot of the 'Configure MyQueue' dialog box. It has a title bar with a close button. Under 'Queue Settings', there are several fields: 'Default Visibility Timeout' (30 seconds), 'Message Retention Period' (4 days), 'Maximum Message Size' (256 KB), 'Delivery Delay' (0 seconds, highlighted with a red box), and 'Receive Message Wait Time' (0 seconds). Each field has a help icon and a validation message. Below this is the 'Dead Letter Queue Settings' section with 'Use Redrive Policy' (checkbox), 'Dead Letter Queue' (text field), and 'Maximum Receives' (text field). At the bottom right are 'Cancel' and 'Save Changes' buttons.

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 on `AUTHPARAMS` in Signature Version 4, see [Examples of Signed Signature Version 4 Requests](#).

```
http://sqs.us-east-2.amazonaws.com/  
?Action=CreateQueue  
&QueueName=testQueue  
&Attribute.1.Name=DelaySeconds  
&Attribute.1.Value=45  
&Version=2012-11-05  
&Expires=2015-12-20T22%3A52%3A43PST  
&AUTHPARAMS
```

Note

Queue names and queue URLs are case-sensitive.

You can also change an existing queue into a delay queue by changing the `DelaySeconds` attribute from its default value of 0 to a positive integer value that is less than or equal to 900. The following

example calls `SetQueueAttributes` to set the `DelaySeconds` attribute of a queue named `testQueue` to 45 seconds.

```
http://sqs.us-east-2.amazonaws.com/123456789012/testQueue/  
?Action=SetQueueAttributes  
&DelaySeconds=45  
&Version=2012-11-05  
&Expires=2015-12-20T22%3A52%3A43PST  
&AUTHPARAMS
```

Amazon SQS Message Timers

Amazon SQS message timers allow you to specify an initial invisibility period for a message that you're add to a queue. For example, if you send a message with the `DelaySeconds` parameter set to 45, the message won't be 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. 66). A message timer setting for an individual message overrides any `DelaySeconds` value that applies to the entire delay queue.

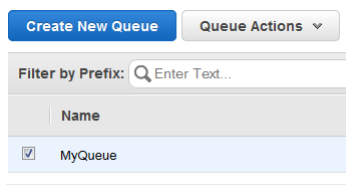
Topics

- [Creating Message Timers Using the Console](#) (p. 70)
- [Creating Message Timers Using the Query API](#) (p. 72)

Creating Message Timers Using the Console

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

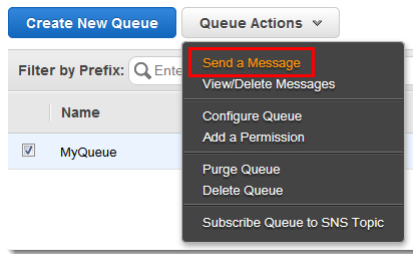
1. Sign in to the AWS Management Console and open the Amazon SQS console at <https://console.aws.amazon.com/sqs/>.
2. Select a queue.



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.



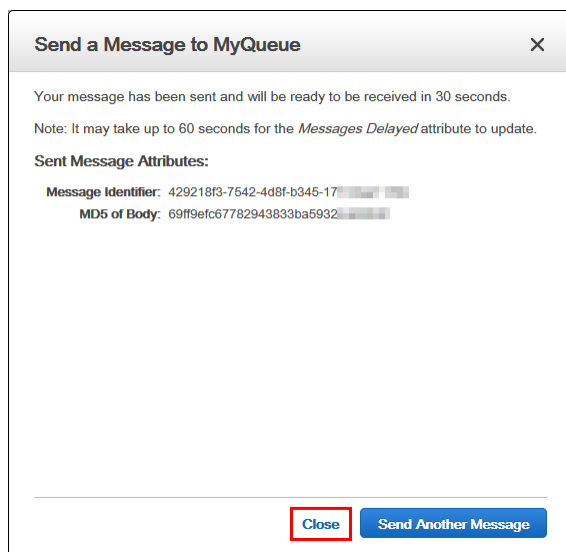
4. In the **Send a Message to MyQueue** dialog box, type a message.

A screenshot of the 'Send a Message to MyQueue' dialog box. It has a title bar with a close button (X). There are two tabs: 'Message Body' (active) and 'Message Attributes'. The 'Message Body' tab contains a text area with the text 'This is a test message with a message timer.' Below the text area is a checkbox labeled 'Delay delivery of this message by' followed by a text input field containing '0', a dropdown menu set to 'seconds', and the text '(up to 15 minutes)'. At the bottom are 'Cancel' and 'Send Message' buttons.

5. In the **Delay delivery of this message by** text box enter a delay value (for example, 30) .

A screenshot of the 'Send a Message to MyQueue' dialog box, similar to the previous one. In this state, the 'Delay delivery of this message by' checkbox is checked. The text input field now contains the value '30'. The 'Send Message' button at the bottom right is highlighted with a blue glow.

6. Choose **Send Message**.
7. In the **Send a Message to MyQueue** confirmation box choose **Close**.



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 on `AUTHPARAMS` in Signature Version 4, see [Examples of Signed Signature Version 4 Requests](#).

```
http://sqs.us-east-2.amazonaws.com/123456789012/testQueue/  
?Action=SendMessage  
&MessageBody=This+is+a+test+message  
&Attribute.Name=DelaySeconds  
&Attribute.Value=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 ten 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 \(p. 66\)](#). The following example uses `SendMessageBatch` to send three messages: one message without a message timer and two messages with different values for `DelaySeconds`.

```
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 retrieving 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.

The Amazon SQS Extended Client Library for Java doesn't currently support FIFO queues.

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, 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 \(p. 74\)](#) following this section.

Note

The Amazon SQS Extended Client Library for Java includes support for [Maven](#) as follows:

```
<dependency>
```

```
<groupId>com.amazonaws</groupId>
<artifactId>amazon-sqs-java-extended-client-lib</artifactId>
<version>1.0.0</version>
</dependency>
```

Using the Amazon SQS Extended Client Library for Java

After you have met the [prerequisites \(p. 73\)](#), 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 retrieves the message and prints out information about the retrieved message. The example then deletes the message, queue, and bucket.

```
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.auth.profile.ProfileCredentialsProvider;
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 DateTime());

    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("\nMessage 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 \(p. 96\)](#) and the [Amazon SQS FAQs](#).

Note

The Amazon SQS Java Message Service (JMS) Client doesn't currently support FIFO queues.

Topics

- [Prerequisites \(p. 77\)](#)
- [Getting Started with the Amazon SQS Java Messaging Library \(p. 78\)](#)
- [Receiving Messages Asynchronously \(p. 81\)](#)
- [Using Client Acknowledge Mode \(p. 82\)](#)
- [Using Unordered Acknowledge Mode \(p. 82\)](#)
- [Code Examples \(p. 83\)](#)
- [Supported JMS 1.1 Implementations \(p. 96\)](#)

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 6.0 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 \(p. 83\)](#).

```
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>amazon-sqs-java-messaging-lib</artifactId>
  <version>1.0.0</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 \(p. 78\)](#).

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 on providing the required credentials to the factory, see [Interface AWSCredentialsProvider](#).

```
// 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 =
    SQSConnectionFactory.builder()
        .withRegion(Region.getRegion(Regions.US_EAST_2))
```

```
.withAWSCredentialsProvider(new
EnvironmentVariableCredentialsProvider())
.build();

// 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 is expecting 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 will be 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, then the function does not return anything. For more information, see the "Create the queue if needed" section in the [TextMessageSender.java \(p. 85\)](#) example.

```
// 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");
}
```

Sending Messages Synchronously

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

```
// Create the non-transacted 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 with name 'TestQueue' in 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.

```
// Create the text message.
TextMessage message = session.createTextMessage("Hello World!");

// Send the message.
producer.send(message);
System.out.println("JMS Message " + message.getJMSMessageID());
```

Receiving Messages Synchronously

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

You can call the `start` method on the connection at any time. However, the consumer will not begin to receiving 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 print the content of the received message.

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

// Cast the received message as TextMessage and print the text to screen.
if (receivedMessage != null) {
    System.out.println("Received: " + ((TextMessage)
        receivedMessage).getText());
}
```

3. Close the connection and the session.

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

The output will look similar to the following:

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

Note

You can use the Spring Framework to initialize these objects.

For additional information, see `SpringExampleConfig.xml`, `SpringExample.java`, and the other helper classes in `ExampleConfiguration.java` and `ExampleCommon.java` in [Code Examples \(p. 83\)](#).

For complete examples of sending and receiving objects, see [TextMessageSender.java \(p. 85\)](#) and [SyncMessageReceiver.java \(p. 86\)](#).

Receiving Messages Asynchronously

In the example in [Getting Started with the Amazon SQS Java Messaging Library \(p. 78\)](#), 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.

```
class MyListener implements MessageListener {  
  
    @Override  
    public void onMessage(Message message) {  
        try {  
            // Cast the received message as TextMessage and print the text  
            // to screen.  
            if (message != null) {  
                System.out.println("Received: " + ((TextMessage)  
message).getText());  
            }  
        } catch (JMSEException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

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 sleeps for one second.

```
// 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 will be 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 \(p. 78\)](#) example. For a complete example of an asynchronous receiver, see `AsyncMessageReceiver.java` in [Code Examples \(p. 83\)](#).

The output for this example will look 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 \(p. 78\)](#) 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.

```
// 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.

```
// 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, then all messages received prior to 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 9 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 \(p. 78\)](#) example. For a complete example of a synchronous receiver with client acknowledge mode, see `SyncMessageReceiverClientAcknowledge.java` in [Code Examples \(p. 83\)](#).

The output for this example will look 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 \(p. 82\)](#).

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.

Create a session with `UNORDERED_ACKNOWLEDGE` mode.


```
// 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 \(p. 82\)](#) example. For a complete example of a synchronous receiver with UNORDERED_ACKNOWLEDGE mode, see `SyncMessageReceiverUnorderedAcknowledge.java`.

In this example, the output will look similar to the following:

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

Important

When you use the JMS library with UNORDERED_ACKNOWLEDGE mode, [visibility timeout \(p. 47\)](#) always takes effect. However, in the case of [asynchronous message processing \(p. 81\)](#), visibility timeout takes effect only when the message listener `onMessage()` returns successfully.

Code Examples

The following code examples show how to use JMS with Amazon SQS.

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.

```
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
     */
    public static ExampleConfiguration parseConfig(String app, String args[])
    {
        try {
            return new ExampleConfiguration(args);
        }
    }
}
```

```

        } catch (IllegalArgumentException e) {
            System.err.println( "ERROR: " + e.getMessage() );
            System.err.println();
            System.err.println( "Usage: " + app + " [--queue <queue>] [--
region <region>] [--credentials <credentials>] " );
            System.err.println( "   or " );
            System.err.println( "       " + app + " <spring.xml>" );
            System.exit(-1);
            return null;
        }
    }

    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.credentialsProvider = credentialsProvider;
    }
}
```

TextMessageSender.java

The following Java code example creates a text message sender.

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

        ExampleCommon.setupLogging();

        // Create the connection factory based on the config
        SQSConnectionFactory connectionFactory =
            SQSConnectionFactory.builder()
                .withRegion(config.getRegion())

.withAWSCredentialsProvider(config.getCredentialsProvider())
                .build();

        // 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 will close 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 receiver.

```
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 =
        SQSConnectionFactory.builder()
            .withRegion(config.getRegion())
            .withAWSCredentialsProvider(config.getCredentialsProvider())
            .build();

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

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));
            if( message == null ) {
                System.out.println( "Shutting down after 1 minute of
silence" );
                break;
            }
            ExampleCommon.handleMessage(message);
            message.acknowledge();
            System.out.println( "Acknowledged message " +
message.getMessageID() );
        }
    } catch (JMSEException e) {
        System.err.println( "Error receiving from SQS: " + e.getMessage() );
        e.printStackTrace();
    }
}
}
```

AsyncMessageReceiver.java

The following Java code example creates an asynchronous message receiver.

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

        ExampleCommon.setupLogging();

        // 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 will be processed until this is called
```

```
        connection.start();

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

        // Close the connection. This will close 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.getMessageID() );
                timeOfLastMessage = System.nanoTime();
            } catch (JMSEException e) {
                System.err.println( "Error processing message: " +
e.getMessage() );
                e.printStackTrace();
            }
        }
    }
}
```

SyncMessageReceiverClientAcknowledge.java

The following Java code example creates a synchronous receiver 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 {@link
 * 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 {@link
* 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 JMSEException,
    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 =
            SQSConnectionFactory.builder()
                .withRegion(config.getRegion())

        .withAWSCredentialsProvider(config.getCredentialsProvider())
            .build();

        // 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 will
        result 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 will close 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 JMSEException
     */
    private static void sendMessage(MessageProducer producer, Session
    session, String messageText) throws JMSEException {
        // 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 JMSEException
     */
    private static void receiveMessage(MessageConsumer consumer, boolean
    acknowledge) throws JMSEException {
        // 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 receiver 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 JMSEException,
    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 =
        SQSConnectionFactory.builder()
```

```
        .withRegion(config.getRegion())

.withAWSCredentialsProvider(config.getCredentialsProvider())
    .build();

    // 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 will
    result 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 will close 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 JMSEException
 */
```

```
private static void sendMessage(MessageProducer producer, Session
session, String messageText) throws JMSEException {
    // 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 JMSEException
 */
private static void receiveMessage(MessageConsumer consumer, boolean
acknowledge) throws JMSEException {
    // 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();
    }
}
}
```

SpringExampleConfig.xml

The following XML code example is a bean configuration file for [SpringExample.java](#) (p. 94).

```
<?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"
    xsi:schemaLocation="
        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
">

    <bean id="CredentialsProviderBean"
        class="com.amazonaws.auth.DefaultAWSCredentialsProviderChain"/>
```

```
<bean id="ConnectionFactoryBuilder"
class="com.amazon.sqs.javamessaging.SQSConnectionFactory$Builder">
  <property name="regionName" value="us-east-2"/>
  <property name="numberOfMessagesToPrefetch" value="5"/>
  <property name="awsCredentialsProvider" ref="CredentialsProviderBean"/>
</bean>

<bean id="ConnectionFactory"
class="com.amazon.sqs.javamessaging.SQSConnectionFactory"
  factory-bean="ConnectionFactoryBuilder"
  factory-method="build" />

<bean id="Connection" class="javax.jms.Connection"
  factory-bean="ConnectionFactory"
  factory-method="createConnection"
  init-method="start"
  destroy-method="close" />

  <bean id="QueueName" class="java.lang.String">
    <constructor-arg value="SQSJMSClientExampleQueue"/>
  </bean>

</beans>
```

SpringExample.java

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

```
public class SpringExample {
    public static void main(String args[]) throws JMSEException {
        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( "Could not find the JMS connection to use: "
+ e.getMessage() );
            System.exit(3);
            return;
        }
    }
}
```

```
String queueName;
try {
    queueName = context.getBean("QueueName", String.class);
} catch( NoSuchBeanDefinitionException e ) {
    System.err.println( "Could not find the name of the queue to use:"
+ e.getMessage() );
    System.exit(3);
    return;
}

if( connection instanceof SQSConnection ) {
    ExampleCommon.ensureQueueExists( (SQSConnection) connection,
queueName );
}

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

receiveMessages(session, consumer);

// The context can be setup to close the connection for us
context.close();
System.out.println( "Context closed" );
}

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));
            if( message == null ) {
                System.out.println( "Shutting down after 1 minute of
silence" );
                break;
            }
            ExampleCommon.handleMessage(message);
            message.acknowledge();
            System.out.println( "Acknowledged message" );
        }
    } catch (JMSEException e) {
        System.err.println( "Error receiving from SQS: " +
e.getMessage() );
        e.printStackTrace();
    }
}
}
```

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.

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

        /**
         * For most cases this could be done 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 );
        }
    }

    public static void setupLogging() {
        // Setup logging
        BasicConfigurator.configure();
        Logger.getRootLogger().setLevel(Level.WARN);
    }

    public static void handleMessage(Message message) throws JMSEException {
        System.out.println( "Got message " + message.getJMSMessageID() );
        System.out.println( "Content: " );
        if( message instanceof TextMessage ) {
            TextMessage txtMessage = ( TextMessage ) message;
            System.out.println( "\t" + txtMessage.getText() );
        } else if( message instanceof BytesMessage ) {
            BytesMessage byteMessage = ( BytesMessage ) message;
            // Assume the length fits in an int - SQS only supports sizes up
            // to 256k so that
            // should be true
            byte[] bytes = new byte[(int)byteMessage.getBodyLength()];
            byteMessage.readBytes(bytes);
            System.out.println( "\t" + Base64.encodeAsString( bytes ) );
        } else if( message instanceof ObjectMessage ) {
            ObjectMessage objMessage = (ObjectMessage) message;
            System.out.println( "\t" + objMessage.getObject() );
        }
    }
}
```

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 FAQs](#).

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

The Amazon SQS JMS client sets the following JMS-defined headers:

- `JMSDestination`
- `JMSMessageID`
- `JMSRedelivered`

The Amazon SQS JMS client sets the following JMS reserved property:

- `JMSXDeliveryCount`

Best Practices for Amazon SQS

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

Topics

- [General Recommendations](#) (p. 98)
- [Recommendations for FIFO \(First-In-First-Out\) Queues](#) (p. 99)

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. 48) and [Changing a Message's Visibility Timeout](#) (p. 49).
 - 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 \(p. 51\)](#).
- 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 \(p. 110\)](#) 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.

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 \(p. 158\)](#).
 - To combine client-side buffering with request batching, use long polling together with the [buffered asynchronous client \(p. 160\)](#) 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 retrieve 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 \(p. 62\)](#).
 - 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 \(p. 44\)](#).

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.

- If you have a single sender and a single receiver 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 sender can omit the message deduplication ID.
 - Although the receiver 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 sender 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 sender and the receiver can recover in case of a client or network outage.
 - The sender 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 receiver 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 receiver 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. 51).

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 processed in a FIFO manner (however, messages in 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 readers 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 receiver can process messages with the same message group ID.

- To avoid processing duplicate messages in a system with multiple readers and writers where throughput and latency are more important than ordering, the sender 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 readers and writers 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.

Limits in Amazon SQS

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

Topics

- [Limits Related to Queues \(p. 102\)](#)
- [Limits Related to Messages \(p. 103\)](#)
- [Limits Related to Policies \(p. 103\)](#)

Limits Related to Queues

The following table lists limits related to queues.

Limit	Description
Queue name	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, <code>Test-queue</code> and <code>test-queue</code> are different queues).
	The name of a FIFO queue must end with the <code>.fifo</code> suffix. The suffix counts towards the 80-character queue name limit. To determine whether a queue is FIFO (p. 39) , you can check whether the queue name ends with the suffix.
Inflight messages per 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 <code>OverLimit</code> 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.

Limits Related to Messages

The following table lists limits related to messages.

Limit	Description
Message attributes	A message can contain up to 10 metadata attributes.
Message content	<p>A message can include only XML, JSON, and unformatted text. The following Unicode characters are allowed:</p> <p>#x9 #xA #xD #x20 to #xD7FF #xE000 to #xFFFD #x10000 to #x10FFFF</p> <p>Any characters not included in this list will be rejected. For more information, see the W3C specification for characters.</p>
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 size	<p>The minimum message size is 1,024 bytes (1 KB). The maximum is 262,144 bytes (256 KB).</p> <p>To send messages larger than 256 KB, you can use the Amazon SQS Extended Client Library for Java. This library allows you to send an Amazon SQS message that contains a reference to a message payload in Amazon S3. The maximum payload size is 2 GB.</p> <p>Note The Amazon SQS Extended Client Library for Java doesn't currently support FIFO queues.</p>
Message visibility timeout	The maximum visibility timeout for a message is 12 hours.
Policy information	The maximum limit is 8,192 bytes, 20 statements, 50 principals, or 10 conditions. For more information, see Limits Related to Policies (p. 103) .

Limits Related to Policies

The following table lists limits related to policies.

Name	Maximum Size
Bytes	8192
Statements	20
Principals	50
Conditions	10

Monitoring and Logging Amazon SQS Queues

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

Topics

- [Monitoring Amazon SQS using CloudWatch \(p. 104\)](#)
- [Logging Amazon SQS API Actions Using AWS CloudTrail \(p. 113\)](#)

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. (Detailed monitoring, or one-minute metrics, is currently unavailable for Amazon SQS.) These metrics are gathered on all queues that meet the CloudWatch guidelines for being active. A queue is considered active by CloudWatch for up to six hours from the last activity (for example, any API call) on the queue.

Note

There is no charge for the Amazon SQS metrics reported in CloudWatch. They're provided as part of the Amazon SQS service.

CloudWatch metrics are supported for both standard and FIFO queues.

Topics

- [Common Monitoring Tasks \(p. 105\)](#)
- [Access CloudWatch Metrics for Amazon SQS \(p. 105\)](#)
- [Set CloudWatch Alarms for Amazon SQS Metrics \(p. 108\)](#)

- [Available CloudWatch Metrics for Amazon SQS \(p. 110\)](#)

Common Monitoring Tasks

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

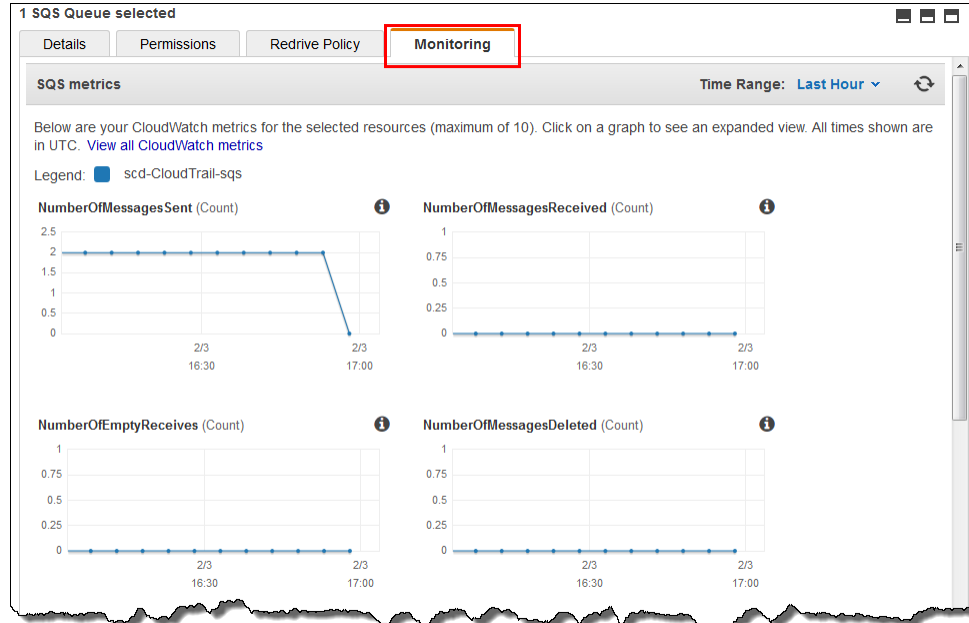
Task	Instructions
Quickly display a default view of CloudWatch metrics over time for up to 10 queues at once.	Access Metrics Using the Amazon SQS Console (p. 105)
Further customize the default views of CloudWatch metrics.	Access Metrics Using the CloudWatch Console (p. 107)
Set alarms when metrics meet or exceed specified conditions.	Set CloudWatch Alarms for Amazon SQS Metrics (p. 108)
Create complex dashboards that display metrics for multiple Amazon SQS queues together.	
Access CloudWatch metrics from the command line or programmatically.	Access Metrics Using the AWS CLI (p. 108) Access Metrics Using the CloudWatch CLI (p. 108) Access Metrics Using the CloudWatch API (p. 108)

Access CloudWatch Metrics for Amazon SQS

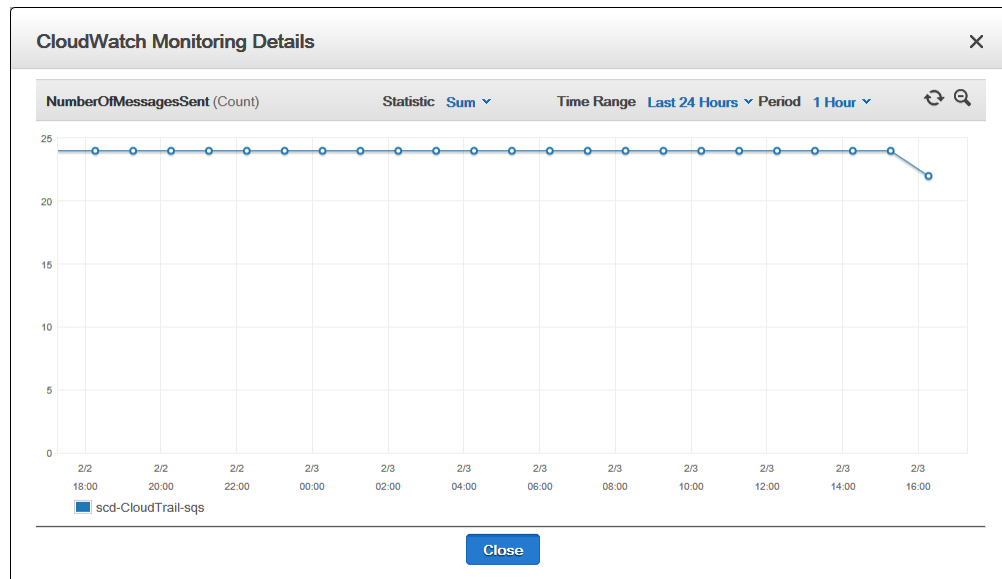
You can access metrics for Amazon SQS using the Amazon SQS console, the CloudWatch console, the AWS CLI, CloudWatch's own CLI, or programmatically using the CloudWatch API. The following procedures show you how to access the metrics using these different options.

Access Metrics Using the Amazon SQS Console

1. Sign in to the AWS Management Console and open the Amazon SQS console at <https://console.aws.amazon.com/sqs/>.
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.



- 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 \(p. 110\)](#).
- 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**).
- 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 \(p. 110\)](#).



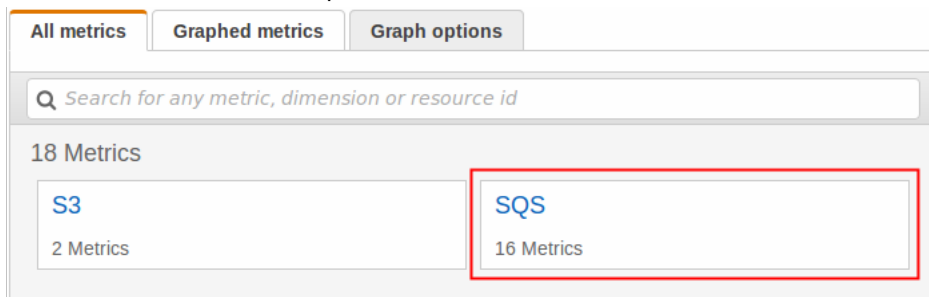
- To change the time range and time interval that an individual graph displays (for example, to show a time range of the last 24 hours instead of the last 5 minutes, or to show a time period of every hour instead of every 5 minutes), with the graph's dialog box still displayed, for **Time Range**, choose 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. 107) procedure.

Access Metrics Using the CloudWatch Console

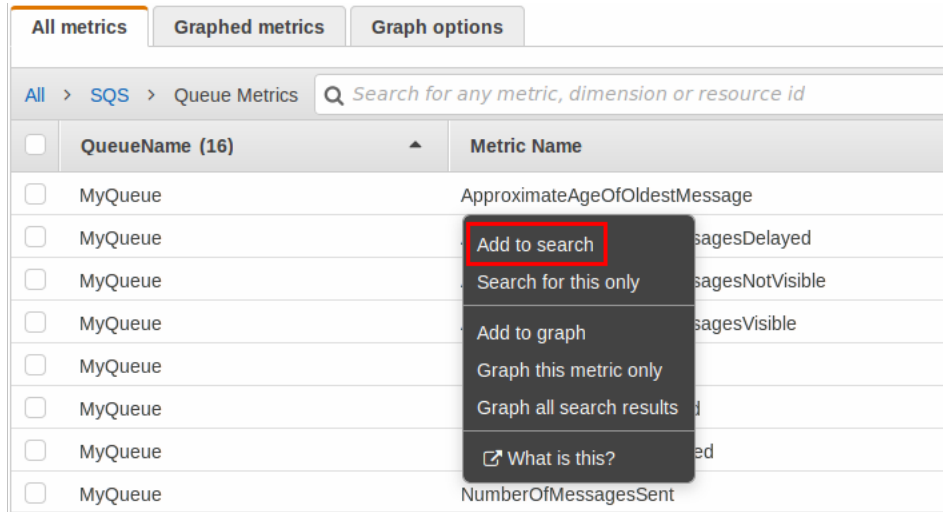
1. Sign in to the AWS Management Console and open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. Select the **SQS** metric namespace.



4. Select the **Queue Metrics** metric dimension.



5. 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

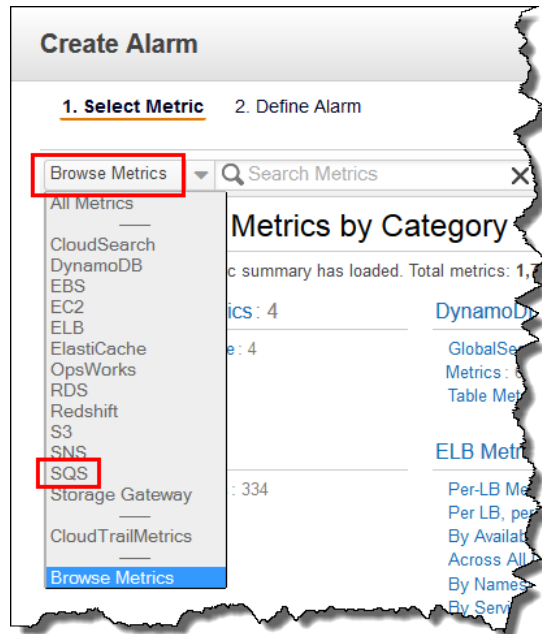
Call the `GetMetricStatistics` operation. For more information, see [Get Statistics for a Metric](#).

Set CloudWatch Alarms for Amazon SQS Metrics

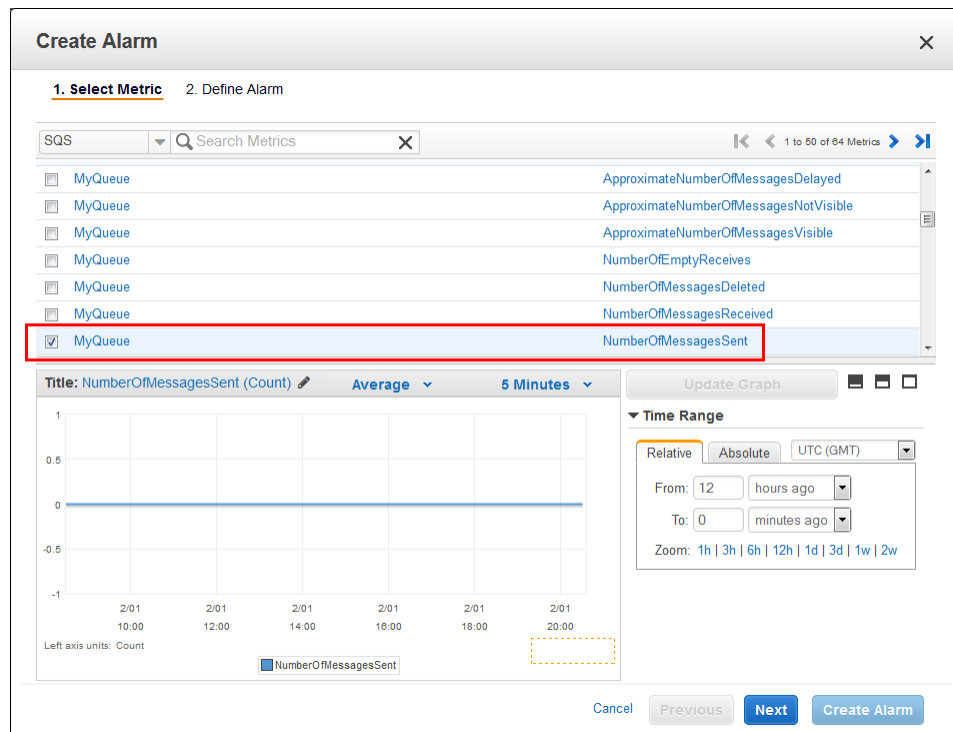
CloudWatch allows you to trigger alarms when a threshold is met for a metric. For example, you could set an alarm for the `NumberOfMessagesSent` metric so that when the number of messages exceeds a specified limit over a specified time period, then an email notification could be sent to inform you of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



- 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. 110)). For example, choosing (checking) the box for **MyQueue, NumberOfMessagesSent** sets an alarm based on the number of messages sent to the MyQueue queue.



- 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**.

The screenshot shows the 'Create Alarm' console with the 'Define Alarm' step selected. The 'Alarm Threshold' section contains the following fields:

- Name:** `NumberOfMessagesSent>100`
- Description:** `100+ messages sent in any hour`
- Whenever:** `NumberOfMessagesSent`
- is:** `>`
- for:** `1` consecutive period(s)

The 'Actions' section contains the following fields:

- Whenever this alarm:** `State is ALARM`
- Send notification to:** `Select a notification list`

The 'Alarm Preview' section shows a graph of `NumberOfMessagesSent > 100` and the following fields:

- Namespace:** `AWS/SQS`
- QueueName:** `MyQueue`
- Metric Name:** `NumberOfMessagesSent`
- Period:** `1 Hour`
- Statistic:** `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 will be saved and appear for future alarms.

Note

If you choose **New list** to create a new Amazon SNS topic, the email addresses must be verified before they'll receive 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.

Metric	Description
<code>ApproximateAgeOfOldestMessage</code>	<p>The approximate age of the oldest non-deleted message in the queue.</p> <p>Units: <i>Seconds</i></p> <p>Valid Statistics: Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console)</p>
<code>ApproximateNumberOfMessagesDelayed</code>	<p>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.</p> <p>Units: <i>Count</i></p> <p>Valid Statistics: Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console)</p>
<code>ApproximateNumberOfMessagesNotVisible</code>	<p>The number of messages that are "in flight." 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.</p> <p>Units: <i>Count</i></p> <p>Valid Statistics: Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console)</p>
<code>ApproximateNumberOfMessagesVisible</code>	<p>The number of messages available for retrieval from the queue.</p> <p>Units: <i>Count</i></p> <p>Valid Statistics: Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console)</p>
<code>NumberOfEmptyReceives</code>	<p>The number of <code>ReceiveMessage</code> API calls that did not return a message.</p> <p>Units: <i>Count</i></p>

Metric	Description
	Valid Statistics: Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console)
NumberOfMessagesDeleted	<p>The number of messages deleted from the queue.</p> <p>Units: <i>Count</i></p> <p>Valid Statistics: Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console)</p> <p>Amazon SQS emits the <code>NumberOfMessagesDeleted</code> metric for every successful deletion operation that uses a valid receipt handle, including duplicate deletions. The following scenarios might cause the value of the <code>NumberOfMessagesDeleted</code> metric to be higher than expected:</p> <ul style="list-style-type: none"> Calling the <code>DeleteMessage</code> 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 <code>NumberOfMessagesDeleted</code> metric. Calling the <code>DeleteMessage</code> action on the same receipt handle: If the message is processed and deleted but you call the <code>DeleteMessage</code> action again using the same receipt handle, a success status is returned, increasing the value of the <code>NumberOfMessagesDeleted</code> metric.
NumberOfMessagesReceived	<p>The number of messages returned by calls to the <code>ReceiveMessage</code> API action.</p> <p>Units: <i>Count</i></p> <p>Valid Statistics: Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console)</p>
NumberOfMessagesSent	<p>The number of messages added to a queue.</p> <p>Units: <i>Count</i></p> <p>Valid Statistics: Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console)</p>

Metric	Description
SentMessageSize	<p>The size of messages added to a queue.</p> <p>Units: <i>Bytes</i></p> <p>Valid Statistics: Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console)</p> <p>Note that <code>SentMessageSize</code> does not display as an available metric in the CloudWatch console until at least one message is sent to the corresponding queue.</p>

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 by 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:

```
{
  "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": "334cccd-b9bb-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": "49ebdbb7-5cd3-5323-8a00-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": "af1bb158-6443-4b4d-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"  
      },  
      "queueUrl": "http://test-sqs.amazon.com/123456789012/hello1"  
    },  
    "responseElements": null,  
    "requestID": "7f15d706-f3d7-5221-b9ca-9b393f349b79",  
    "eventID": "8b6fb2dc-2661-49b1-b328-94317815088b"  
  }  
]
```

Amazon SQS Security

This section provides information on Amazon SQS security, authentication and access control, and the access policy language.

Topics

- [Authentication and Access Control for Amazon SQS \(p. 118\)](#)

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 \(p. 118\)](#)
- [Access Control \(p. 119\)](#)

Authentication

You can access AWS as any of the following types of identities:

- **AWS account root user** – When you sign up for AWS, you provide an email address and password that is associated with your AWS account. These are your *root credentials* and they provide complete access to all of your AWS resources.

Important

For security reasons, we recommend that you use the root credentials only to create an *administrator user*, which is an *IAM user* with full permissions to your AWS account. Then, you can use this administrator user to create other IAM users and roles with limited permissions. For more information, see [IAM Best Practices](#) and [Creating an Admin User and Group](#) in the *IAM User Guide*.

- **IAM user** – An [IAM user](#) is simply 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 the 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 another IAM identity 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 preexisting 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*.
 - **Cross-account access** – You can use an IAM role in your account to grant another AWS account permissions to access your account's resources. For an example, see [Tutorial: Delegate Access Across AWS Accounts Using IAM 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 stored in the 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** – Instead of storing access keys within the EC2 instance for use by applications running on the instance and making AWS API requests, you can use an IAM role to manage temporary credentials for these applications. To assign an AWS role to an EC2 instance and make it available to all of its applications, you can create an instance profile that is attached to the instance. An instance profile contains the role and enables programs 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. 140).

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](#) (p. 120)
- [Using Identity-Based Policies \(IAM\) Policies for Amazon SQS](#) (p. 125)

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. 120)
- [Understanding Resource Ownership](#) (p. 121)
- [Managing Access to Resources](#) (p. 121)
- [Specifying Policy Elements: Actions, Effects, Resources, and Principals](#) (p. 124)

Amazon Simple Queue Service Resource and Operations

For Amazon SQS, the *queue* is the only resource type you can specify in a policy. The following are examples of the Amazon Resource Name (ARN) format for queues:

- An ARN for a normal queue:

```
arn:aws:sqs:region:account_id:queue_name
```

For more information about ARNs, see [IAM ARNs](#) in the *IAM User Guide*.

- 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 retrieve the ARN value for an existing queue by calling the [GetQueueAttributes](#) action. The value of the `Arn` attribute is the ARN of the queue.

Understanding Resource Ownership

A *resource owner* is the AWS account that creates the resource. That is, the resource owner is the AWS account of the *principal entity* (the root account, an IAM user, or an IAM role) that authenticates the request that creates the resource. 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 Amazon SQS user in your AWS account and grant permissions to create an Amazon SQS queue to the user, the user can create an Amazon SQS queue. However, your AWS account (to which the user belongs) owns the Amazon SQS 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 Amazon SQS queue. Your AWS account (to which the role belongs) owns the Amazon SQS 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 Simple Queue Service. 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 policies) and policies attached to a resource are referred to as *resource-based* policies.

Topics

- [Identity-Based \(IAM\) Features of Resource-Based \(Amazon SQS\) Policies](#) (p. 121)
- [Resource-Based \(Amazon SQS\) and Identity-Based \(IAM\) Policies](#) (p. 123)

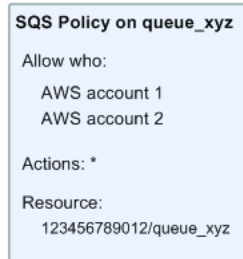
Identity-Based (IAM) Features of Resource-Based (Amazon SQS) Policies

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 permission to use `SendMessage`, `ReceiveMessage` if the request is made before December 31, 2010). The specific actions you can grant permission for are a subset of the overall list of Amazon SQS actions. When you write an Amazon SQS policy and specify `*` to mean "all the Amazon SQS actions", that means all actions in that subset.

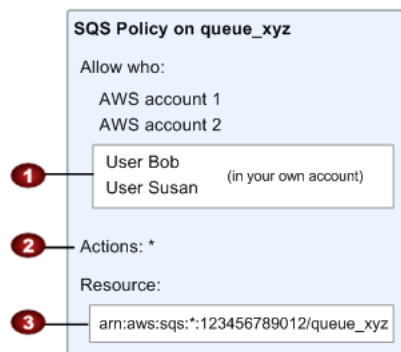
The following diagram illustrates the concept of one of these basic Amazon SQS policies that covers the subset of actions. The policy is for `queue_xyz`, and it gives AWS Account 1 and AWS Account 2 permission to use any of the allowed actions with the queue.

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.



1	In addition to specifying which AWS Accounts have access to a queue, you can specify which users <i>in your own AWS Account</i> have access to the queue. If the users are in different accounts, see Tutorial: Delegate Access Across AWS Accounts Using IAM Roles in the <i>IAM User Guide</i> .
2	The subset of actions included in <code>*</code> has expanded. For a list of allowed actions, see Amazon SQS API Permissions: Actions and Resource Reference (p. 140).
3	You can specify the resource using the <i>Amazon Resource Name (ARN)</i> , 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. 120).

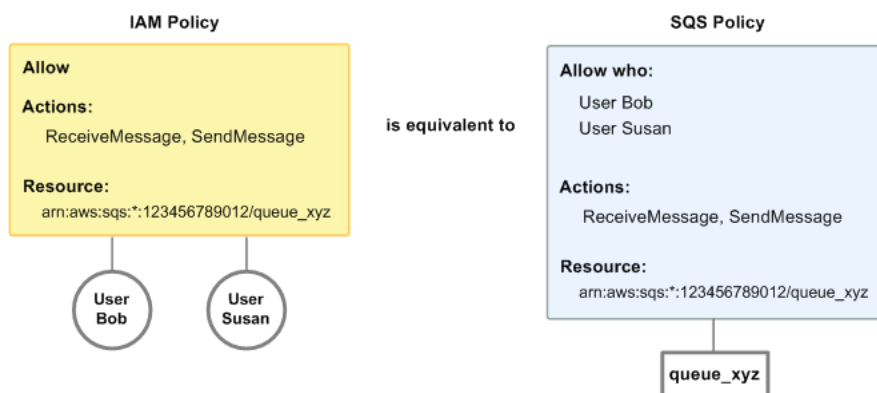
For example, according to the Amazon SQS policy in the preceding figure, anyone who possesses the security credentials for AWS Account 1 or AWS Account 2 can access `queue_xyz`. In addition, Users Bob and Susan in your own AWS Account (with ID `123456789012`) can access the queue.

Before the introduction of IAM, Amazon SQS automatically gave the creator of a queue full control over the queue (that is, access to all possible Amazon SQS actions on that queue). This is no longer true, unless the creator uses AWS security credentials. Any user who has permission to create a queue must also have permission to use other Amazon SQS actions in order to do anything with the created queues.

Resource-Based (Amazon SQS) and Identity-Based (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.

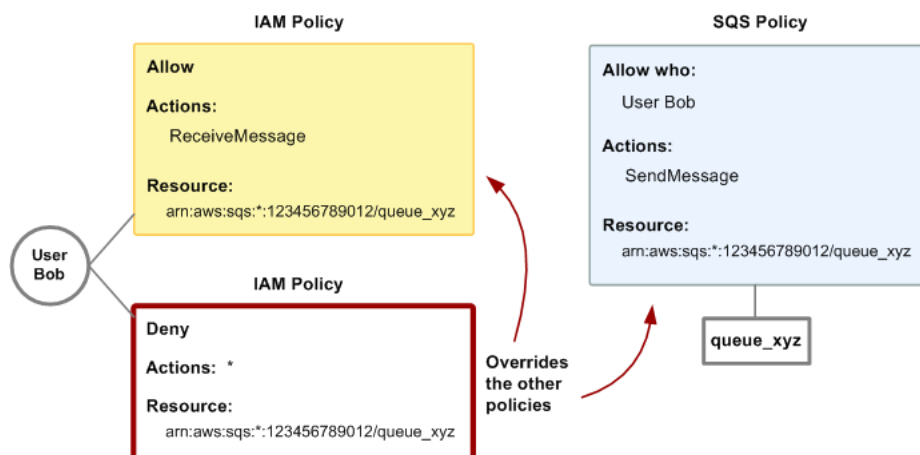
It is up to you how you use both of the systems together to manage your permissions. The following examples show how the two policy systems work together.

- 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 Access Policy Language \(p. 130\)](#). 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 \(p. 126\)](#). For more information about writing Amazon SQS policies, see [Creating Custom Policies Using the Access Policy Language \(p. 130\)](#).

Specifying Policy Elements: Actions, Effects, Resources, and Principals

For each [Amazon Simple Queue Service resource \(p. 120\)](#), 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](#) in the *IAM User Guide*.

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](#) (p. 140).

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 more information, see [Overview of Managing Access Permissions to Your Amazon Simple Queue Service Resource](#) (p. 120).

Writing an Amazon SQS Policy

The following examples provide an introductory breakdown of a permission 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 permission to use the queue. Therefore, we must explicitly grant Bob permission 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. Using an Amazon SQS policy might be easier if your partner has an AWS account. However, anyone 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.

In the following example, we perform 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`.

```
{  
  "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.

```
{
  "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.

```
{
  "Version": "2012-10-17",
  "Id": "Queue1_Policy_UUID",
  "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 a Role and a User Name Cross-Account Permission

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`.

```
{
  "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.

```
{
  "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.

```
{
  "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.

```
{
  "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 Access Policy Language

If you want to allow Amazon SQS access based only on 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 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. 131) that has [effect](#) (p. 131) set to `allow`.

action

The activity that the [principal](#) (p. 131) has permission to perform, typically a request to AWS.

default-deny

The result of a [statement](#) (p. 131) that has no [allow](#) (p. 130) or [explicit deny](#) (p. 131) settings.

condition

Any restriction or detail about a [permission](#) (p. 131). Typical conditions are related to date and time and IP addresses.

effect

The result that you want the [statement](#) (p. 131) of a [policy](#) (p. 131) 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. 130), [allow](#) (p. 130), and [explicit deny](#) (p. 131).

explicit deny

The result of a [statement](#) (p. 131) that has [effect](#) (p. 131) 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. 131).

issuer

The user who writes a [policy](#) (p. 131) 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

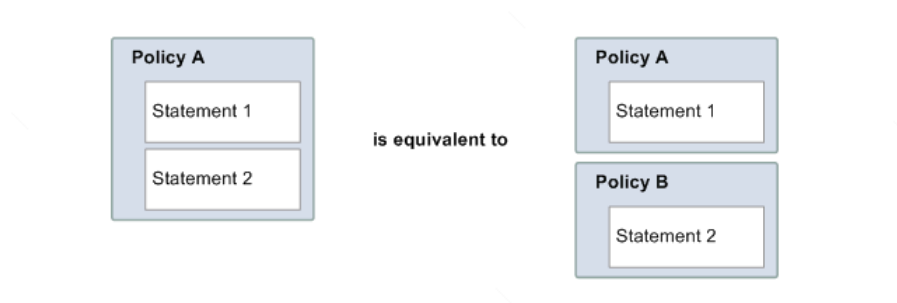
The specific characteristic that is the basis for access restriction.

permission

The concept of allowing or disallowing access to a resource using a [condition](#) (p. 131) and a [key](#) (p. 131).

policy

The document that acts as a container for one or more [statements](#) (p. 131).



Amazon SQS uses the policy to determine whether to grant access to a user for a resource.

principal

The user who receives [permission](#) (p. 131) in the [policy](#) (p. 131).

resource

The object that the [principal](#) (p. 131) requests access to.

statement

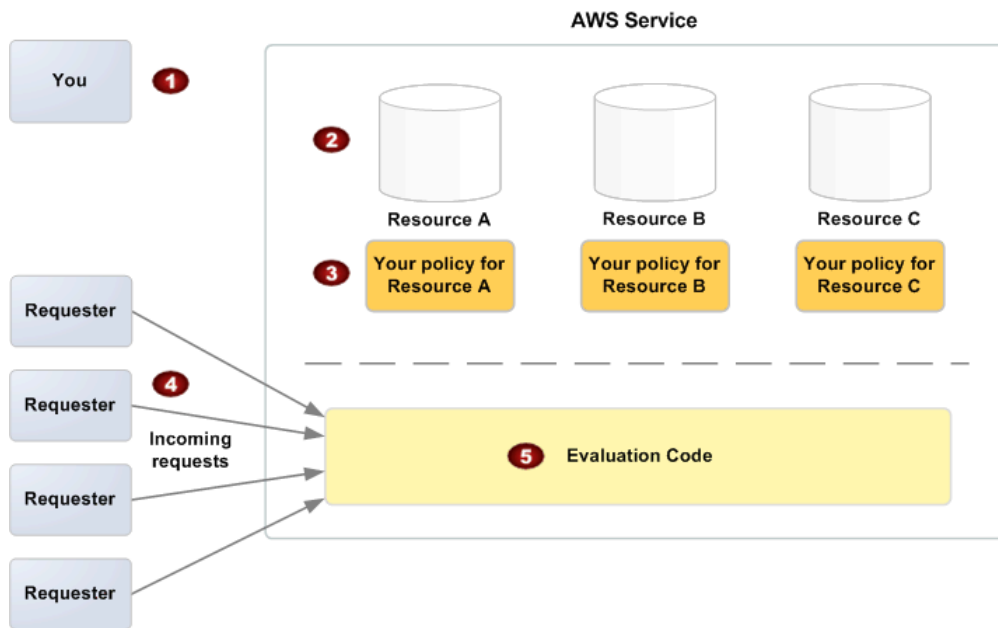
The formal description of a single permission, written in the access policy language as part of a broader [policy](#) (p. 131) document.

requester

The user who sends a request for access to a [resource](#) (p. 131).

Architecture

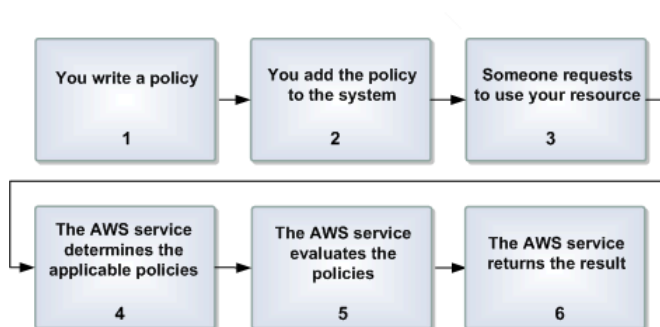
The following figure and table describe the main components that together provide access control for your resources.



1	You, the resource owner.
2	Your resources contained within the AWS service (for example, Amazon SQS queues).
3	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.
4	Requesters and their incoming requests to the AWS service.
5	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 the general workflow of access control with the access policy language.



1	You write a policy for your Amazon SQS queue.
---	---

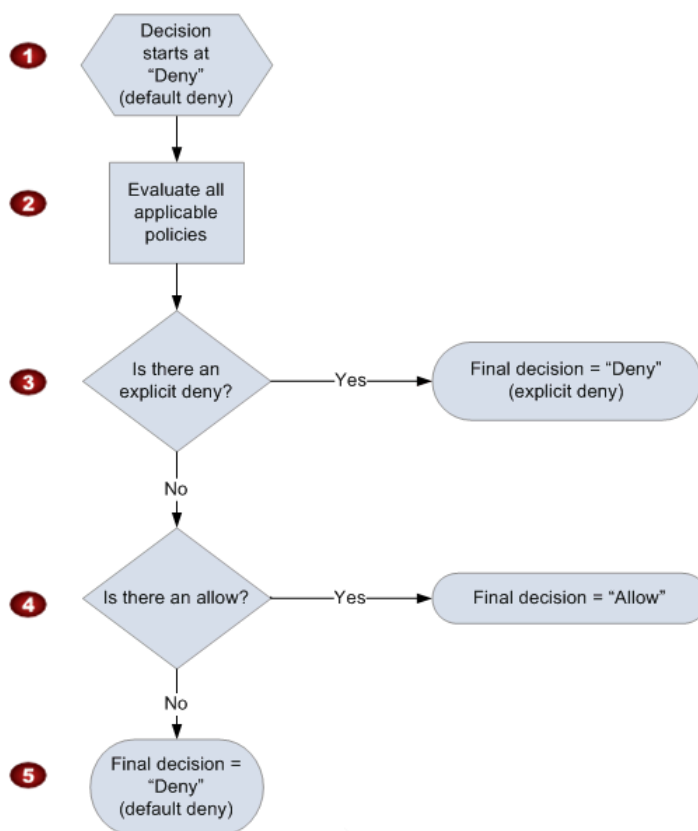
2	You upload your policy to AWS. The AWS service itself provides an API you use to upload your policies. For example, you use the Amazon SQS <code>SetQueueAttributes</code> action to upload a policy for a particular Amazon SQS queue.
3	Someone sends a request to use your Amazon SQS queue.
4	Amazon SQS 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 <code>Access denied</code> 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. 130) overrides any [default-deny](#) (p. 130).
- An [explicit deny](#) (p. 131) overrides any [allow](#) (p. 130).
- 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 permission.



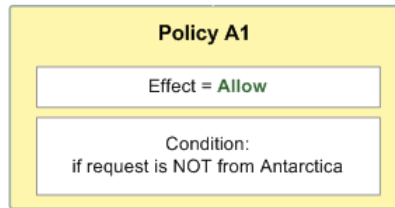
1	The decision starts with a default-deny (p. 130).
2	The enforcement code evaluates all the policies that are applicable to the request (based on the resource, principal, action, and conditions). The order in which the enforcement code evaluates the policies is not important
3	The enforcement code looks for an explicit deny (p. 131) instruction that could apply to the request. If it finds even one, the enforcement code returns a decision of <i>deny</i> and the process finishes.
4	If no explicit deny (p. 131) instruction is found, the enforcement code looks for any allow (p. 130) instructions that could apply to the request. If it finds even one, the enforcement code returns a decision of <i>allow</i> and the process finishes (the service continues to process the request).
5	If no allow (p. 130) instruction is found, then the final decision is <i>deny</i> (because there is no explicit deny (p. 131) or allow (p. 130), this is considered a default-deny (p. 130).

Relationships Between Explicit and Default Denials

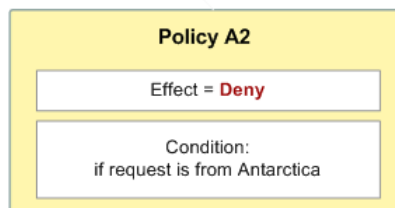
If a policy doesn't directly apply to a requests, the request results in an [default-deny](#) (p. 130). For example, if a user requests permission to use Amazon SQS but the only policy that applies to the user can use DynamoDB, the requests results in a [default-deny](#) (p. 130).

If a condition in a statement isn't met, the request results in an [default-deny](#) (p. 130). If all conditions in a statement are met, then the request results in either an [allow](#) (p. 130) or an [explicit deny](#) (p. 131) based on the value of the [effect](#) (p. 131) 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. 130). 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 policy.

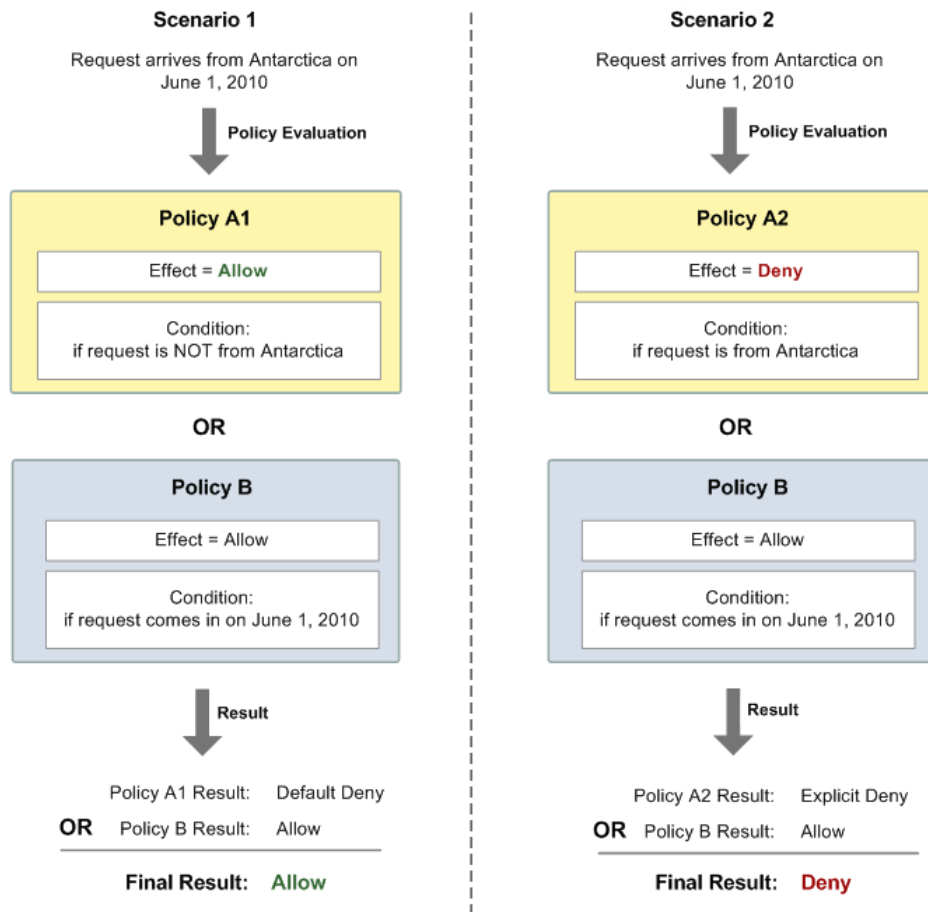


If a user sends a request from the U.S., the condition is met (the request is not from Antarctica), and the request results in an **allow** (p. 130). However, if a user sends a request from Antarctica, the condition isn't met and the request defaults to a **default-deny** (p. 130). You can change the result to an **explicit deny** (p. 131) 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. 131).

The distinction between a **default-deny** (p. 130) and an **explicit deny** (p. 131) is important because an **allow** (p. 130) 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 \(p. 130\)](#) and Policy B results in an [allow \(p. 130\)](#) because the policy allows requests that come in on June 1, 2010. The [allow \(p. 130\)](#) from Policy B overrides the [default-deny \(p. 130\)](#) from Policy A1, and the request is allowed.

In Scenario 2, Policy B2 results in an [explicit deny \(p. 131\)](#) and Policy B results in an [allow \(p. 130\)](#). The [explicit deny \(p. 131\)](#) from Policy A2 overrides the [allow \(p. 130\)](#) from Policy B, and the request is denied.

Access Policy Examples

The following are examples of typical access control policies.

Example 1: Give Permission to One Account

The following example policy gives AWS account 1111-2222-3333 permission to send to and receive from queue2 owned by AWS account 4444-5555-6666.

```
{
  "Version": "2012-10-17",
  "Id": "UseCase1",
  "Statement": [
    {
      "Sid": "1",
      "Effect": "Allow",
```

```
"Principal" : {
  "AWS": "111122223333"
},
"Action":["sqs:SendMessage", "sqs:ReceiveMessage"],
"Resource": "arn:aws:sqs:us-east-2:444455556666:queue2",
}
]
```

Example 2: Give Permission to One or More Accounts

The following example 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"],
      "Resource": "arn:aws:sqs:us-east-2:444455556666:queue2",
      "Condition" : {
        "DateLessThan" : {
          "AWS:CurrentTime": "2009-06-30T12:00Z"
        }
      }
    }
  ]
}
```

Example 3: Give Permission to Requests from Amazon EC2 Instances

The following example 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 \(p. 137\)](#)" 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.

```
{
  "Version": "2012-10-17",
  "Id": "UseCase3",
  "Statement" : [
    {
      "Sid": "1",
      "Effect": "Allow",
      "Principal" : {
        "AWS": "111122223333"
      },

```

```
    "Action":["sqs:SendMessage","sqs:ReceiveMessage"],
    "Resource": "arn:aws:sqs:us-east-2:444455556666:queue2",
    "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 policy denies a specific AWS account access to your queue. This example builds on the ["Example 1: Give Permission to One Account \(p. 136\)"](#) 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).

```
{
  "Version": "2012-10-17",
  "Id": "UseCase4",
  "Statement" : [
    {
      "Sid": "1",
      "Effect": "Deny",
      "Principal" : {
        "AWS": "111122223333"
      },
      "Action":["sqs:SendMessage","sqs:ReceiveMessage"],
      "Resource": "arn:aws:sqs:us-east-2:444455556666:queue2",
    }
  ]
}
```

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 "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.

When you submit the signed Query API request, don't forget to 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](#) in the *IAM User Guide*.

To call an Amazon SQS Query API action using temporary security credentials

1. Request a temporary security token using AWS Identity and Access Management. For more information, see [Creating Temporary Security Credentials to Enable Access for IAM Users](#) in the *IAM User Guide*.
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 on AUTHPARAMS in Signature Version 4, see [Examples of Signed Signature Version 4 Requests](#).

```
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=SecurityTokenValue  
&AWSAccessKeyId=Access Key ID provided by AWS Security Token Service  
&AUTHPARAMS
```

The following example uses Temporary Security Credentials to send two messages with SendMessageBatch.

```
http://sqs.us-east-2.amazonaws.com/  
?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  
&Version=2012-11-05  
&Expires=2015-12-18T22%3A52%3A43PST  
&SecurityToken=SecurityTokenValue  
&AWSAccessKeyId=Access Key ID provided by AWS Security Token Service
```

&AUTHPARAMS

Amazon SQS API Permissions: Actions and Resource Reference

When you set up [Access Control](#) (p. 119) 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`

Resource: `arn:aws:sqs:region:account_id:queue_name`

CreateQueue

Action(s): `sqs:CreateQueue`

Resource: `arn:aws:sqs:region:account_id:queue_name`

DeleteMessage

Action(s): `sqs>DeleteMessage`

Resource: `arn:aws:sqs:region:account_id:queue_name`

DeleteMessageBatch

Action(s): `sqs>DeleteMessageBatch`

Resource: `arn:aws:sqs:region:account_id:queue_name`

DeleteQueue

Action(s): sqs:DeleteQueue

Resource: arn:aws:sqs:*region*:*account_id*:*queue_name*

GetQueueAttributes

Action(s): sqs:GetQueueAttributes

Resource: arn:aws:sqs:*region*:*account_id*:*queue_name*

GetQueueUrl

Action(s): sqs:GetQueueUrl

Resource: arn:aws:sqs:*region*:*account_id*:*queue_name*

ListDeadLetterSourceQueues

Action(s): sqs:ListDeadLetterSourceQueues

Resource: arn:aws:sqs:*region*:*account_id*:*queue_name*

ListQueues

Action(s): sqs:ListQueues

Resource: arn:aws:sqs:*region*:*account_id*:*queue_name*

PurgeQueue

Action(s): sqs:PurgeQueue

Resource: arn:aws:sqs:*region*:*account_id*:*queue_name*

ReceiveMessage

Action(s): sqs:ReceiveMessage

Resource: arn:aws:sqs:*region*:*account_id*:*queue_name*

RemovePermission

Action(s): sqs:RemovePermission

Resource: arn:aws:sqs:*region*:*account_id*:*queue_name*

SendMessage

Action(s): sqs:SendMessage

Resource: arn:aws:sqs:*region*:*account_id*:*queue_name*

SendMessageBatch

Action(s): sqs:SendMessageBatch

Resource: arn:aws:sqs:*region*:*account_id*:*queue_name*

SetQueueAttributes

Action(s): sqs:SetQueueAttributes

Resource: arn:aws:sqs:*region*:*account_id*:*queue_name*

Working with Amazon SQS APIs

This section provides information on working with Amazon SQS APIs.

Topics

- [Making API Requests \(p. 142\)](#)
- [Amazon SQS Batch API Actions \(p. 158\)](#)

Making API Requests

Topics

- [Endpoints \(p. 143\)](#)
- [Making Query Requests \(p. 144\)](#)
- [Request Authentication \(p. 147\)](#)
- [Responses \(p. 153\)](#)
- [Shared Queues \(p. 155\)](#)
- [Programming Languages \(p. 158\)](#)

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 \(p. 144\)](#)
- [Structure of a POST Request \(p. 145\)](#)
- [Related Topics \(p. 146\)](#)

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 on AUTHPARAMS in Signature Version 4, see [Examples of Signed Signature Version 4 Requests](#).

```
http://sqs.us-east-2.amazonaws.com/123456789012/queue1?  
Action=SendMessage&MessageBody=Your%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. 152)).

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. 45).

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 on 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'd 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
&Version=2012-11-05
&Expires=2011-10-15T12:00:00Z
&AUTHPARAMS
```

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).

```
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. 152)).

```
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 \(p. 152\)](#)).

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 \(p. 152\)](#)
- [Responses \(p. 153\)](#)

Request Authentication

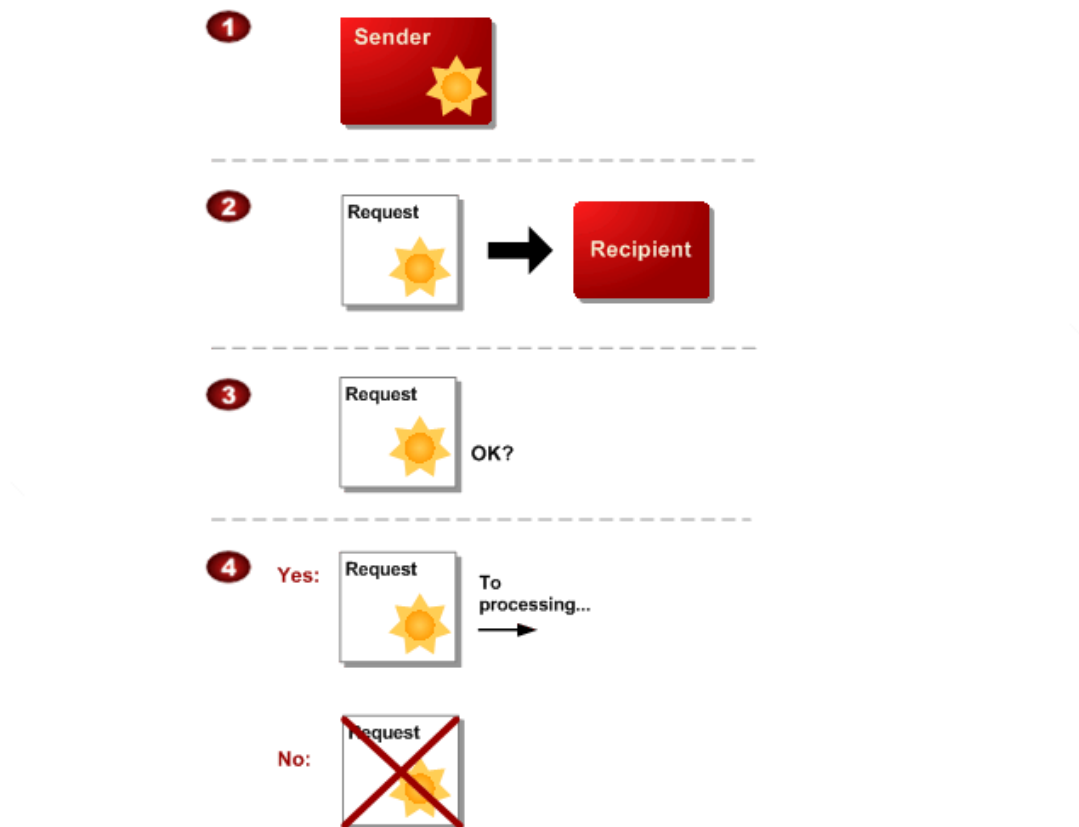
Topics

- [What Is Authentication?](#) (p. 147)
- [Your AWS Account](#) (p. 148)
- [Your Access Keys](#) (p. 148)
- [HMAC-SHA Signatures](#) (p. 148)
- [Query Request Authentication](#) (p. 152)

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 sender obtains the necessary credential.
2	The sender sends a request with the credential to the recipient.
3	The recipient uses the credential to verify the sender truly sent the request.

4	If yes, the recipient processes the request. If no, the recipient rejects the request and responds accordingly.
---	---

During authentication, AWS verifies both the identity of the sender and whether the sender 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.

Alternately, you could create a new AWS-enabled Amazon.com account by using a new login and password. The e-mail address you provide as the account login must be valid. You'll be asked to 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 \(p. 148\)](#)

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 \(p. 148\)](#)
- [Query Request Authentication \(p. 152\)](#)

HMAC-SHA Signatures

Topics

- [Required Authentication Information \(p. 149\)](#)
- [Basic Authentication Process \(p. 150\)](#)
- [About the String to Sign \(p. 151\)](#)
- [About the Time Stamp \(p. 151\)](#)
- [Java Sample Code for Base64 Encoding \(p. 152\)](#)
- [Java Sample Code for Calculating SHA256 Signatures \(p. 152\)](#)

The topics in this section describe how Amazon SQS uses HMAC-SHA signatures to authenticate Query requests.

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 by 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.

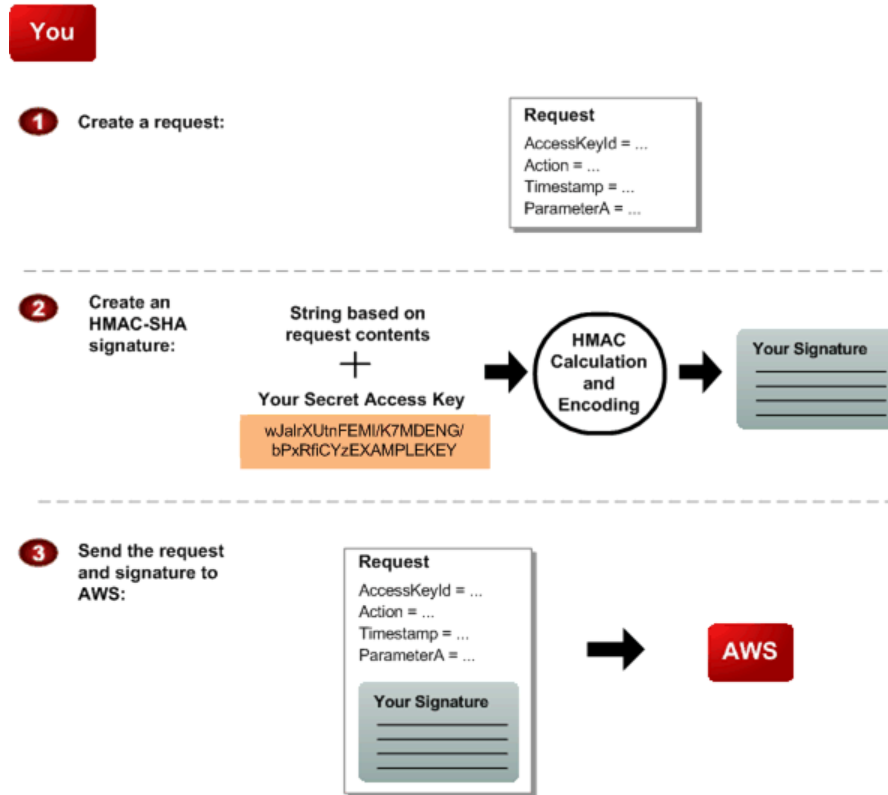
Related Topics

- [Your Access Keys \(p. 148\)](#)

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 \(p. 148\)](#) and [Your Access Keys \(p. 148\)](#).

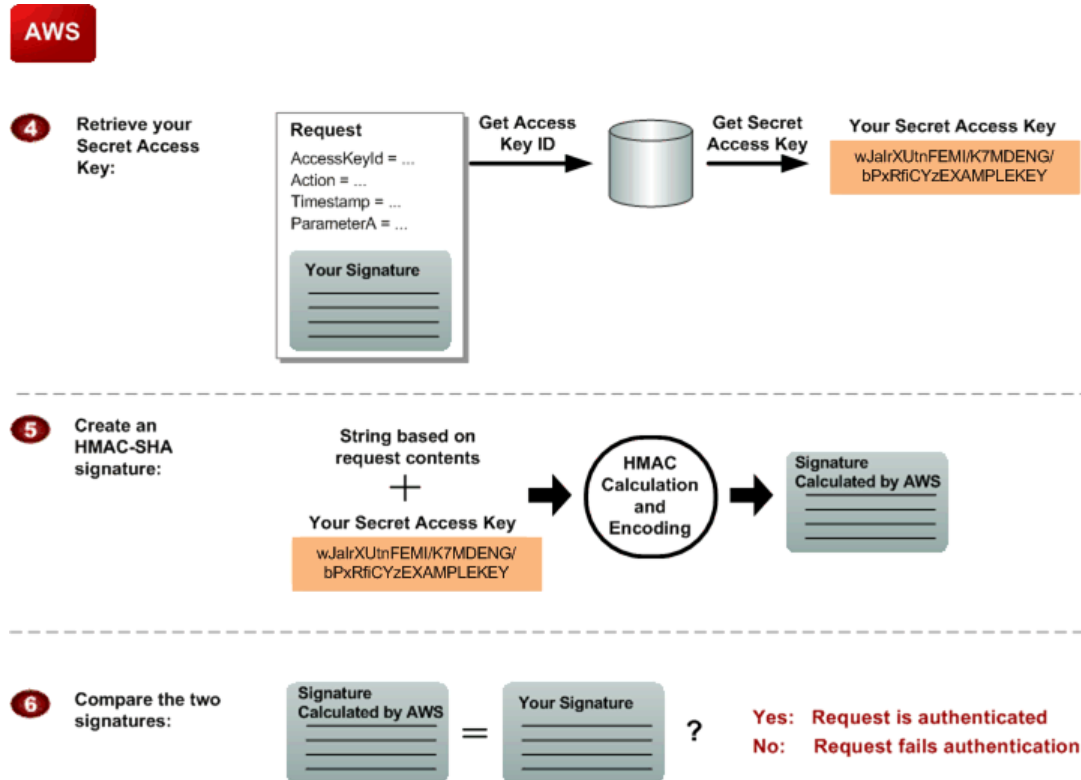
You perform the first three tasks.



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.



Process for Authentication: Tasks AWS Performs

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 \(p. 152\)](#).

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.

```
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.

For information on how to create the signature using signature version 4, see [Signature Version 4 Signing Process](#) in the AWS General Reference.

Responses

Topics

- [Structure of a Successful Response \(p. 153\)](#)
- [Structure of an Error Response \(p. 153\)](#)
- [Related Topics \(p. 154\)](#)

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 receiver or sender 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>
```

```
<Error>
  <Type>
    Sender
  </Type>
  <Code>
    InvalidParameterValue
  </Code>
  <Message>
    Value (quename_nonalpha) for parameter QueueName is invalid.
    Must be an alphanumeric String of 1 to 80 in length
  </Message>
</Error>
<RequestId>
  42d59b56-7407-4c4a-be0f-4c88daeea257
</RequestId>
</ErrorResponse>
```

Related Topics

- [Making Query Requests \(p. 144\)](#)

Shared Queues

Topics

- [Simple API for Shared Queues \(p. 155\)](#)
- [Advanced API for Shared Queues \(p. 155\)](#)
- [Understanding Resource-Level Permissions \(p. 155\)](#)
- [Granting Anonymous Access to a Queue \(p. 156\)](#)

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 access 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 Access Policy Language \(p. 130\)](#). For examples of policies, see [Access Policy Examples \(p. 136\)](#).

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.

Permission	Description
*	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.

Permission	Description
<code>ChangeMessageVisibility</code>	This grants permission to extend or terminate the read lock timeout of a specified message. <code>ChangeMessageVisibilityBatch</code> inherits permissions associated with <code>ChangeMessageVisibility</code> . For more information about visibility timeout, see Visibility Timeout (p. 47) . For more information, see the ChangeMessageVisibility operation.
<code>DeleteMessage</code>	This grants permission to delete messages from the queue. <code>DeleteMessageBatch</code> inherits permissions associated with <code>DeleteMessage</code> . For more information, see the DeleteMessage operation.
<code>GetQueueAttributes</code>	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 GetQueueAttributes operation.
<code>GetQueueUrl</code>	This grants permission to get a queue's URL. For more information, see the GetQueueUrl operation.
<code>ReceiveMessage</code>	This grants permission to receive messages in the queue. For more information, see the ReceiveMessage operation.
<code>SendMessage</code>	This grants permission to send messages to the queue. <code>SendMessageBatch</code> inherits permissions associated with <code>SendMessage</code> . For more information, see the SendMessage operation.

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.

Tip

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 Access Policy Language \(p. 130\)](#).

Caution

Keep in mind that the queue owner is responsible for all costs related to the queue. Therefore you probably want to limit anonymous access in some other way (by time or IP address, for example).

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:

- The AWS Command Line Interface (AWS CLI). For more information, see [Getting Set Up with the AWS Command Line Interface](#) and the [Amazon SQS section of the AWS CLI Reference](#).
- The AWS Tools for Windows PowerShell. For more information, see [Setting up the AWS Tools for Windows PowerShell](#) and the Amazon Simple Queue Service section of the [AWS Tools for Windows PowerShell Cmdlet Reference](#).

Amazon SQS Batch API Actions

Topics

- [Maximum Message Size for SendMessageBatch \(p. 159\)](#)
- [Client-Side Buffering and Request Batching \(p. 160\)](#)
- [Increasing Throughput with Horizontal Scaling and Batching \(p. 163\)](#)

In the 2009-02-01 API version of Amazon SQS, only the `ReceiveMessage` action supports batch processing (processing more than one message with a single call).

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 by 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/>) 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 \(p. 62\)](#) and [Increasing Throughput with Horizontal Scaling and Batching \(p. 163\)](#).

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 sample shows how to create a new `AmazonSQSBufferedAsyncClient` based on the `AmazonSQSAsyncClient`.

```
// 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 sample demonstrates.

```
CreateQueueRequest createRequest = new
    CreateQueueRequest().withQueueName("MyTestQueue");

CreateQueueResult res = bufferedSqs.createQueue(createRequest);

SendMessageRequest request = new SendMessageRequest();
String body = "test message_" + System.currentTimeMillis();
request.setRequestBody( 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 will 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.

```
// 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 for configuring `QueueBufferConfig` are as follows:

- `longPoll`—if this parameter is set to `true`, `AmazonBufferedAsyncClient` attempts to use long-polling when retrieving 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 will be batched together in a single batch request. The higher the setting, the fewer batches will be 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 `AmazonBufferedAsyncClient` 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'll sit in the buffer, which means that their visibility timeout will be expiring. If this parameter is set to zero, all pre-fetching of messages is disabled and messages are retrieved 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, are hit). Although, the higher the setting, the more threads will be consumed by the `AmazonSQSBufferedAsyncClient`. If this parameter is set to 0, all pre-fetching of messages is disabled and messages are only retrieved 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 retrieved. 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). The key to achieving this throughput is to horizontally scale message producers and consumers. In addition, you can use the batching actions in the Amazon SQS API to send, receive, or delete up to 10 messages at a time. In conjunction with horizontal scaling, batching achieves a given throughput with fewer threads, connections, and requests than would be required by individual message requests. Because Amazon SQS charges by the request instead of by the message, batching can also substantially reduce costs.

This appendix discusses horizontal scaling and batching in more detail. It then walks through a simple example that you can try out yourself. It also briefly discusses Amazon SQS throughput metrics that you can monitor by using CloudWatch.

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 will average 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'll 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 will be sending requests and receiving 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:

```
AmazonSQS sqsClient = new AmazonSQSClient(credentials,
                                           new
                                           ClientConfiguration().withMaxConnections(producerCount +
                                           consumerCount));
```

For the SDK for Java asynchronous client `AmazonSQSAsyncClient`, 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. Amazon 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 will sometimes have 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.

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:

```
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:

```
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 instance(s) 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 will be the `m1.large` instance cost. The Amazon SQS cost varies based on the API call rate for each sample, and that should range between approximately 38,000 API calls / min for the batching sample and 380,000 API calls / min for the two host single API sample. For example, a run of the single API sample on a single host should cost approximately 1 instance hour of an `m1.large` (large standard on demand instance, \$0.32 as of July 2012) and 20 min x 190,000 API calls / min x \$1 / 1,000,000 API calls = \$3.80 for Amazon SQS operations with the default 20 min duration (as of July 2012, check current pricing).

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:** The `SQS_Sample_Base_Producer_Consumer.template` sample 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.

To view the template, see https://s3.amazonaws.com/cloudformation-templates-us-east-1/SQS_Sample_Base_Producer_Consumer.template

- **Single Operation API, Two Hosts:** `SQS_Sample_Base_Producer_Consumer_x2.template` sample 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 Amazon SQS' elasticity with throughput increasing proportionally to the greater number of producers and consumers.

To view the template, see https://s3.amazonaws.com/cloudformation-templates-us-east-1/SQS_Sample_Base_Producer_Consumer_x2.template

- **Batch API, One Host:** The `SQS_Sample_Batch_Producer_Consumer.template` sample 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.

To view the template, see https://s3.amazonaws.com/cloudformation-templates-us-east-1/SQS_Sample_Batch_Producer_Consumer.template

2. If you're prompted, sign in to the AWS Management Console.
3. In the **Create Stack** wizard, on the **Select Template** page, choose **Continue**.
4. 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. Here is an example:

Create Stack [Cancel X]

SELECT TEMPLATE | **SPECIFY PARAMETERS** | REVIEW

Template Description: Single EC2 m1.large producer-consumer processing (single operation API) 1K messages for the specified duration

Specify Parameters

Below are the parameters associated with your CloudFormation template. You may review and proceed with the default parameters or make customizations as needed below.

DurationMinutes [20]
Run duration in minutes (max 60)

TerminateEC2Inst [true]
Terminate the producer-consumer EC2 instance once the run is complete?

KeyName []
Name of an existing EC2 KeyPair to enable SSH access to the producer-consumer instance

☒ I acknowledge that this template may create IAM resources

Back Continue

5. 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.
6. When all the settings are as you want them, choose **Continue**.
7. 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.
8. 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.

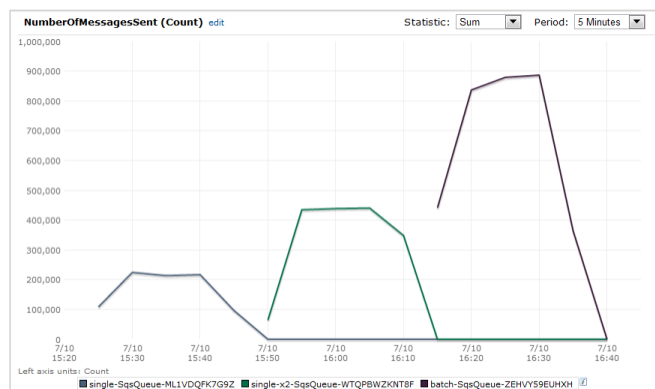
In addition, after a delay of up to 15 minutes after the queue is started, you can monitor the queue in CloudWatch as described later in this topic.

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, CloudWatch metrics will all 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) 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) 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'll find useful as you work with this service.

Resource	Description
Amazon Simple Queue Service API Reference	The API reference gives complete descriptions of API actions, parameters, and data types and a list of errors that the service returns.
Amazon SQS Release Notes	The release notes give a high-level overview of the current release. They specifically note any new features, corrections, and known issues.
Product information for Amazon SQS	The primary web page for information about Amazon SQS.
Discussion Forums	A community-based forum for developers to discuss technical questions related to Amazon SQS.
AWS Premium Support Information	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.

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:** December 16, 2016

Change	Description	Date Changed
Update	<p>The <i>Amazon Simple Queue Service Getting Started Guide</i> has been retired and some of its content has been incorporated into the following sections of this guide:</p> <ul style="list-style-type: none">• Setting Up Amazon SQS (p. 4)• Getting Started with Amazon SQS (p. 7)• Amazon SQS Tutorials (p. 15)	December 16, 2016
Update	<p>The Authentication and Access Control (p. 118) section has been reorganized and updated.</p>	December 2, 2016
New feature	<p>You can use Amazon SQS to create <i>FIFO (First-In-First-Out) queues</i> or <i>standard queues</i> (the new name for existing queues) in US West (Oregon) and US East (Ohio). For more information on how FIFO queues work and how to get started using them, see the following:</p> <ul style="list-style-type: none">• FIFO (First-In-First-Out) Queues (p. 39)• Moving From a Standard Queue to a FIFO Queue (p. 44)• Recommendations for FIFO (First-In-First-Out) Queues (p. 99) <p>For revised Amazon SQS tutorials, see the following:</p> <ul style="list-style-type: none">• Creating an Amazon SQS Queue (p. 15)• Sending a Message to an Amazon SQS Queue (p. 21)• Receiving and Deleting a Message from an Amazon SQS Queue (p. 24) <p>FIFO queues add the following API functionality:</p>	November 17, 2016

Change	Description	Date Changed
	<ul style="list-style-type: none"> The <code>FifoQueue</code> and <code>ContentBasedDeduplication</code> attributes for the CreateQueue, GetQueueAttributes, and SetQueueAttributes actions. The <code>MessageDeduplicationId</code> and <code>MessageGroupId</code> request parameters for the SendMessage and SendMessageBatch actions and attributes for the ReceiveMessage action. The <code>ReceiveRequestAttemptId</code> request parameter for the ReceiveMessage action. The <code>SequenceNumber</code> response parameter for the SendMessage and SendMessageBatch actions and the <code>SequenceNumber</code> attribute for the ReceiveMessage action. <p>Important As of November 17, 2016, Amazon SQS no longer publishes a WSDL. The following clients don't currently support FIFO queues:</p> <ul style="list-style-type: none"> Amazon SQS Buffered Asynchronous Client Amazon SQS Extended Client Library for Java Amazon SQS Java Message Service (JMS) Client <p>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:</p> <ul style="list-style-type: none"> Amazon CloudWatch Events Amazon S3 Event Notifications Amazon SNS Topic Subscriptions <p>Note You can use Amazon SNS to forward messages to Amazon SQS standard queues.</p> <ul style="list-style-type: none"> Auto Scaling Lifecycle Hooks AWS IoT Rule Actions AWS Lambda Dead Letter Queues <p>For information about compatibility of other services with FIFO queues, see your service documentation. FIFO queues don't support timers on individual messages.</p>	
Update	The Walkthroughs section has been renamed to Amazon SQS Tutorials (p. 15) .	November 2, 2016
New feature	You can use the <code>ApproximateAgeOfOldestMessage</code> CloudWatch metric to find the approximate age of the oldest non-deleted message in the queue. For more information, see Available CloudWatch Metrics for Amazon SQS (p. 110) .	August 31, 2016

Change	Description	Date Changed
Update	Added the Best Practices for Amazon SQS (p. 98) section.	May 27, 2016
Update	Added the Limits in Amazon SQS (p. 102) section.	May 12, 2016
New feature	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. 104) .	February 12, 2016
Update	Updated Amazon SQS console screenshots.	December 7, 2015
New feature	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. 73) in the <i>Amazon Simple Queue Service Developer Guide</i> .	October 27, 2015
New feature	Amazon SQS lets you use JMS (Java Message Service) with Amazon SQS queues. For more information, see Using JMS with Amazon SQS (p. 77) in the <i>Amazon Simple Queue Service Developer Guide</i> .	December 29, 2014
New feature	Amazon SQS lets you delete the messages in a queue by using the <code>PurgeQueue</code> API. For more information, see PurgeQueue in the <i>Amazon SQS API Reference</i> .	December 8, 2014
Update	Updated information about access keys. For more information, see Your Access Keys (p. 148) .	August 4, 2014
New feature	Amazon SQS lets you log API calls by using AWS CloudTrail. For more information, see Logging Amazon SQS API Actions Using AWS CloudTrail (p. 113) .	July 16, 2014
New feature	Amazon SQS provides support for message attributes. For more information, see Using Amazon SQS Message Attributes (p. 54) .	May 6, 2014
New feature	Amazon SQS provides support for dead letter queues. For more information, see Using Amazon SQS Dead Letter Queues (p. 51) .	January 29, 2014
New feature	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. 31) .	November 21, 2012
New feature	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. 152) .	November 5, 2012
New feature	The AWS SDK for Java includes a buffered asynchronous client, <code>AmazonSQSBufferedAsyncClient</code> , 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 Client-Side Buffering and Request Batching (p. 160) .	November 5, 2012

Change	Description	Date Changed
New feature	The 2012-11-05 API version of Amazon SQS adds long polling support. Long polling allows for 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 Amazon SQS Long Polling (p. 62) .	November 5, 2012