
Amazon Simple Storage Service

Developer Guide

API Version 2006-03-01



Amazon Simple Storage Service: Developer Guide

Copyright © 2015 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

The following are trademarks of Amazon Web Services, Inc.: Amazon, Amazon Web Services Design, AWS, Amazon CloudFront, AWS CloudTrail, AWS CodeDeploy, Amazon Cognito, Amazon DevPay, DynamoDB, ElastiCache, Amazon EC2, Amazon Elastic Compute Cloud, Amazon Glacier, Amazon Kinesis, Kindle, Kindle Fire, AWS Marketplace Design, Mechanical Turk, Amazon Redshift, Amazon Route 53, Amazon S3, Amazon VPC, and Amazon WorkDocs. In addition, Amazon.com graphics, logos, page headers, button icons, scripts, and service names are trademarks, or trade dress of Amazon in the U.S. and/or other countries. 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 S3?	1
How Do I...?	1
Introduction	2
Overview of Amazon S3 and This Guide	2
Advantages to Amazon S3	2
Amazon S3 Concepts	3
Buckets	3
Objects	3
Keys	4
Regions	4
Amazon S3 Data Consistency Model	5
Features	7
Reduced Redundancy Storage	7
Bucket Policies	7
AWS Identity and Access Management	8
Access Control Lists	8
Versioning	8
Operations	8
Amazon S3 Application Programming Interfaces (API)	9
The REST Interface	9
The SOAP Interface	9
Paying for Amazon S3	9
Related Services	10
Making Requests	11
About Access Keys	11
AWS Account Access Keys	11
IAM User Access Keys	12
Temporary Security Credentials	12
Request Endpoints	13
Making Requests Using the AWS SDKs	14
Using AWS Account or IAM User Credentials	15
Using IAM User Temporary Credentials	20
Using Federated User Temporary Credentials	31
Making Requests Using the REST API	47
Virtual Hosting of Buckets	48
Request Redirection and the REST API	52
Buckets	55
Creating a Bucket	55
About Permissions	56
Accessing a Bucket	57
Bucket Configuration Options	57
Restrictions and Limitations	59
Rules for Naming	59
Examples of Creating a Bucket	60
Using the Amazon S3 Console	61
Using the AWS SDK for Java	61
Using the AWS SDK for .NET	62
Using Other AWS SDKs	63
Bucket Website Configuration	63
Using the AWS Management Console	64
Using the SDK for Java	64
Using the SDK for .NET	67
Using the SDK for PHP	70
Using the REST API	72
Requester Pays Buckets	73

Configure with the Console	73
Configure with the REST API	74
DevPay and Requester Pays	76
Charge Details	76
Access Control	77
Billing and Reporting	77
Cost Allocation Tagging	77
Objects	78
Object Key and Metadata	79
Object Keys	79
Object Metadata	81
Subresources	83
Versioning	84
Lifecycle Management	86
Overview	87
Lifecycle Configuration Elements	88
Object Archival	96
Object Expiration	98
Specifying a Lifecycle Configuration	98
Cross-Origin Resource Sharing	110
Cross-Origin Resource Sharing: Examples	111
How Do I Enable CORS on My Bucket?	111
How Does Amazon S3 Evaluate the CORS Configuration On a Bucket?	113
Managing Cross-Origin Resource Sharing (CORS)	114
Operations on Objects	125
Multipart Upload Overview	126
Getting Objects	130
Related Resources	130
Using the SDK for Java	131
Using the SDK for .NET	134
Using the SDK for PHP	137
Using the REST API	139
Share an Object with Others	139
Uploading Objects	145
Related Resources	145
Uploading Objects in a Single Operation	146
Uploading Objects Using Multipart Upload API	155
Uploading Objects Using Pre-Signed URLs	191
Copying Objects	199
Related Resources	200
Copying Objects in a Single Operation	200
Copying Objects Using the Multipart Upload API	210
Listing Object Keys	218
Iterating Through Multi-Page Results	219
Using a Prefix and Delimiter	219
Using the SDK for Java	220
Using the SDK for .NET	223
Using the SDK for PHP	225
Using the REST API	228
Related Resources	228
Deleting Objects	228
Deleting Objects from a Version-Enabled Bucket	228
Deleting Objects from an MFA-Enabled Bucket	229
Related Resources	229
Deleting One Object Per Request	229
Deleting Multiple Objects Per Request	239
Restoring Objects	261
Using the Console	262

Using the SDK for Java	263
Using the SDK for .NET	265
Using the REST API	268
Managing Access	269
Introduction	269
Overview	270
How Amazon S3 Authorizes a Request	275
Guidelines for Using the Available Access Policy Options	280
Example Walkthroughs: Managing Access	283
Using Bucket Policies and User Policies	312
Access Policy Language Overview	312
Bucket Policy Examples	336
User Policy Examples	342
Managing Access with ACLs	363
Access Control List (ACL) Overview	363
Managing ACLs	368
Protecting Data	380
Data Encryption	380
Server-Side Encryption	381
Client-Side Encryption	408
Reduced Redundancy Storage	419
Setting the Storage Class of an Object You Upload	420
Changing the Storage Class of an Object in Amazon S3	420
Versioning	422
How to Configure Versioning on a Bucket	423
MFA Delete	423
Related Topics	424
Examples	424
Managing Objects in a Versioning-Enabled Bucket	427
Managing Objects in a Versioning-Suspended Bucket	443
Hosting a Static Website	448
Website Endpoints	449
Key Differences Between the Amazon Website and the REST API Endpoint	450
Configure a Bucket for Website Hosting	451
Overview	451
Syntax for Specifying Routing Rules	453
Index Document Support	457
Custom Error Document Support	458
Configuring a Redirect	460
Permissions Required for Website Access	462
Example Walkthroughs	462
Example: Setting Up a Static Website	463
Example: Setting Up a Static Website Using a Custom Domain	464
Notifications	472
How to Enable Event Notifications	473
Supported Event Types	475
Supported Destinations	475
Granting Permissions to Publish Event Notification Messages to a Destination	476
Granting Permissions to Invoke an AWS Lambda Function	476
Granting Permissions to Publish Messages to an SNS Topic or an SQS Queue	476
Related Topics	477
Example Walkthrough 1	478
Walkthrough Summary	478
Step 1: Create an Amazon SNS Topic	479
Step 2: Create an Amazon SQS Queue	479
Step 3: Add a Notification Configuration to Your Bucket	480
Step 4: Test the Setup	484
Example Walkthrough 2	484

Event Message Structure	484
Cross-Region Replication	487
Use-case Scenarios	487
Requirements	487
What Is and Is Not Replicated	488
What Is Replicated	488
What Is Not Replicated	489
Related Topics	489
Related Topics	489
How to Set Up	490
Create an IAM Role	490
Add Replication Configuration	491
Walkthrough 1	494
Walkthrough 2	496
Using the Console	500
Using the AWS SDK for Java	500
Using the AWS SDK for .NET	502
Related Topics	504
Replication Status Information	504
Related Topics	506
Troubleshooting	506
Related Topics	507
Replication and Other Bucket Configurations	507
Lifecycle Configuration and Object Replicas	507
Versioning Configuration and Replication Configuration	507
Logging Configuration and Replication Configuration	507
Related Topics	508
Request Routing	509
Request Redirection and the REST API	509
Overview	509
DNS Routing	509
Temporary Request Redirection	510
Permanent Request Redirection	512
DNS Considerations	513
Performance Optimization	514
Request Rate and Performance Considerations	514
Workloads with a Mix of Request Types	515
GET-Intensive Workloads	517
TCP Window Scaling	517
TCP Selective Acknowledgement	518
BitTorrent	519
How You are Charged for BitTorrent Delivery	519
Using BitTorrent to Retrieve Objects Stored in Amazon S3	520
Publishing Content Using Amazon S3 and BitTorrent	521
Amazon DevPay	522
Amazon S3 Customer Data Isolation	522
Example	523
Amazon DevPay Token Mechanism	523
Amazon S3 and Amazon DevPay Authentication	523
Amazon S3 Bucket Limitation	524
Amazon S3 and Amazon DevPay Process	525
Additional Information	525
Error Handling	526
The REST Error Response	526
Response Headers	527
Error Response	527
The SOAP Error Response	528
Amazon S3 Error Best Practices	528

Retry InternalErrors	528
Tune Application for Repeated SlowDown errors	528
Isolate Errors	529
Server Access Logging	530
Overview	530
Log Object Key Format	531
How are Logs Delivered?	531
Best Effort Server Log Delivery	531
Bucket Logging Status Changes Take Effect Over Time	532
Related Topics	532
Enabling Logging Using the Console	532
Enabling Logging Programmatically	534
Enabling logging	534
Granting the Log Delivery Group WRITE and READ_ACP Permissions	534
Example: AWS SDK for .NET	535
Log Format	537
Custom Access Log Information	539
Programming Considerations for Extensible Server Access Log Format	539
Additional Logging for Copy Operations	539
Deleting Log Files	541
AWS SDKs and Explorers	542
Specifying Signature Version in Request Authentication	543
Using the AWS SDK for Java	544
The Java API Organization	545
Testing the Java Code Examples	545
Using the AWS SDK for .NET	546
The .NET API Organization	547
Testing the .NET Code Examples	547
Using the AWS SDK for PHP and Running PHP Examples	547
AWS SDK for PHP Levels	548
Running PHP Examples	548
Related Resources	549
Using the AWS SDK for Ruby	549
The Ruby API Organization	549
Testing the Ruby Script Examples	550
Using the AWS SDK for Python (Boto)	550
Appendices	551
Appendix A: Using the SOAP API	551
Common SOAP API Elements	551
Authenticating SOAP Requests	552
Setting Access Policy with SOAP	553
Appendix B: Authenticating Requests (AWS Signature Version 2)	554
Authenticating Requests Using the REST API	556
Signing and Authenticating REST Requests	557
Browser-Based Uploads Using POST	568
Resources	585
Document History	587
AWS Glossary	594

What Is Amazon S3?

Amazon Simple Storage Service is storage for the Internet. It is designed to make web-scale computing easier for developers.

Amazon S3 has a simple web services interface that you can use to store and retrieve any amount of data, at any time, from anywhere on the web. It gives any developer access to the same highly scalable, reliable, fast, inexpensive data storage infrastructure that Amazon uses to run its own global network of web sites. The service aims to maximize benefits of scale and to pass those benefits on to developers.

This guide explains the core concepts of Amazon S3, such as buckets and objects, and how to work with these resources using the Amazon S3 application programming interface (API).

How Do I...?

Information	Relevant Sections
General product overview and pricing	Amazon Simple Storage Service (Amazon S3)
Get a quick hands-on introduction to Amazon S3	Amazon Simple Storage Service Getting Started Guide
Learn about Amazon S3 key terminology and concepts	Introduction to Amazon S3 (p. 2)
How do I work with buckets?	Working with Amazon S3 Buckets (p. 55)
How do I work with objects?	Working with Amazon S3 Objects (p. 78)
How do I make requests?	Making Requests (p. 11)
How do I manage access to my resources?	Managing Access Permissions to Your Amazon S3 Resources (p. 269)

Introduction to Amazon S3

This introduction to Amazon Simple Storage Service is intended to give you a detailed summary of this web service. After reading this section, you should have a good idea of what it offers and how it can fit in with your business.

Topics

- [Overview of Amazon S3 and This Guide \(p. 2\)](#)
- [Advantages to Amazon S3 \(p. 2\)](#)
- [Amazon S3 Concepts \(p. 3\)](#)
- [Features \(p. 7\)](#)
- [Amazon S3 Application Programming Interfaces \(API\) \(p. 9\)](#)
- [Paying for Amazon S3 \(p. 9\)](#)
- [Related Services \(p. 10\)](#)

Overview of Amazon S3 and This Guide

Amazon S3 has a simple web services interface that you can use to store and retrieve any amount of data, at any time, from anywhere on the web.

This guide describes how you send requests to create buckets, store and retrieve your objects, and manage permissions on your resources. The guide also describes access control and the authentication process. Access control defines who can access objects and buckets within Amazon S3, and the type of access (e.g., READ and WRITE). The authentication process verifies the identity of a user who is trying to access Amazon Web Services (AWS).

Advantages to Amazon S3

Amazon S3 is intentionally built with a minimal feature set that focuses on simplicity and robustness. Following are some of advantages of the Amazon S3 service:

- **Create Buckets** – Create and name a bucket that stores data. Buckets are the fundamental container in Amazon S3 for data storage.

- **Store data in Buckets** – Store an infinite amount of data in a bucket. Upload as many objects as you like into an Amazon S3 bucket. Each object can contain up to 5 TB of data. Each object is stored and retrieved using a unique developer-assigned key.
- **Download data** – Download your data or enable others to do so. Download your data any time you like or allow others to do the same.
- **Permissions** – Grant or deny access to others who want to upload or download data into your Amazon S3 bucket. Grant upload and download permissions to three types of users. Authentication mechanisms can help keep data secure from unauthorized access.
- **Standard interfaces** – Use standards-based REST and SOAP interfaces designed to work with any Internet-development toolkit.

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

Amazon S3 Concepts

Topics

- [Buckets \(p. 3\)](#)
- [Objects \(p. 3\)](#)
- [Keys \(p. 4\)](#)
- [Regions \(p. 4\)](#)
- [Amazon S3 Data Consistency Model \(p. 5\)](#)

This section describes key concepts and terminology you need to understand to use Amazon S3 effectively. They are presented in the order you will most likely encounter them.

Buckets

A bucket is a container for objects stored in Amazon S3. Every object is contained in a bucket. For example, if the object named `photos/puppy.jpg` is stored in the `johndoe` bucket, then it is addressable using the URL `http://johndoe.s3.amazonaws.com/photos/puppy.jpg`.

Buckets serve several purposes: they organize the Amazon S3 namespace at the highest level, they identify the account responsible for storage and data transfer charges, they play a role in access control, and they serve as the unit of aggregation for usage reporting.

You can configure buckets so that they are created in a specific region. For more information, see [Buckets and Regions \(p. 57\)](#). You can also configure a bucket so that every time an object is added to it, Amazon S3 generates a unique version ID and assigns it to the object. For more information, see [Versioning \(p. 422\)](#).

For more information about buckets, see [Working with Amazon S3 Buckets \(p. 55\)](#).

Objects

Objects are the fundamental entities stored in Amazon S3. Objects consist of object data and metadata. The data portion is opaque to Amazon S3. The metadata is a set of name-value pairs that describe the object. These include some default metadata, such as the date last modified, and standard HTTP metadata, such as Content-Type. You can also specify custom metadata at the time the object is stored.

An object is uniquely identified within a bucket by a key (name) and a version ID. For more information, see [Keys \(p. 4\)](#) and [Versioning \(p. 422\)](#).

Keys

A key is the unique identifier for an object within a bucket. Every object in a bucket has exactly one key. Because the combination of a bucket, key, and version ID uniquely identify each object, Amazon S3 can be thought of as a basic data map between "bucket + key + version" and the object itself. Every object in Amazon S3 can be uniquely addressed through the combination of the web service endpoint, bucket name, key, and optionally, a version. For example, in the URL <http://doc.s3.amazonaws.com/2006-03-01/AmazonS3.wsdl>, "doc" is the name of the bucket and "2006-03-01/AmazonS3.wsdl" is the key.

Regions

You can choose the geographical region where Amazon S3 will store the buckets you create. You might choose a region to optimize latency, minimize costs, or address regulatory requirements. Amazon S3 currently supports the following regions:

- **US Standard** Uses Amazon S3 servers in the United States
 - Provides eventual consistency for all requests. This region automatically routes requests to facilities in Northern Virginia or the Pacific Northwest using network maps.
- **US West (Oregon) region** Uses Amazon S3 servers in Oregon
 - Provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and Deletes.
- **US West (N. California) region** Uses Amazon S3 servers in Northern California
 - Provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and Deletes.
- **EU (Ireland) region** Uses Amazon S3 servers in Ireland
 - Provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and Deletes.
- **EU (Frankfurt) region** Uses Amazon S3 servers in Frankfurt
 - Provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and Deletes.
- **Asia Pacific (Singapore) region** Uses Amazon S3 servers in Singapore
 - Provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and Deletes.
- **Asia Pacific (Tokyo) region** Uses Amazon S3 servers in Tokyo
 - Provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and Deletes.
- **Asia Pacific (Sydney) region** Uses Amazon S3 servers in Sydney
 - Provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and Deletes.
- **South America (Sao Paulo) region** Uses Amazon S3 servers in Sao Paulo
 - Provides read-after-write consistency for PUTS of new objects in your Amazon S3 bucket and eventual consistency for overwrite PUTS and Deletes.

Objects stored in a region never leave the region unless you explicitly transfer them to another region. For example, objects stored in the EU (Ireland) region never leave it.

Amazon S3 Data Consistency Model

Updates to a single key are atomic. For example, if you PUT to an existing key, a subsequent read might return the old data or the updated data, but it will never write corrupted or partial data.

Amazon S3 achieves high availability by replicating data across multiple servers within Amazon's data centers. If a PUT request is successful, your data is safely stored. However, information about the changes must replicate across Amazon S3, which can take some time, and so you might observe the following behaviors:

- A process writes a new object to Amazon S3 and immediately attempts to read it. Until the change is fully propagated, Amazon S3 might report "key does not exist."
- A process writes a new object to Amazon S3 and immediately lists keys within its bucket. Until the change is fully propagated, the object might not appear in the list.
- A process replaces an existing object and immediately attempts to read it. Until the change is fully propagated, Amazon S3 might return the prior data.
- A process deletes an existing object and immediately attempts to read it. Until the deletion is fully propagated, Amazon S3 might return the deleted data.
- A process deletes an existing object and immediately lists keys within its bucket. Until the deletion is fully propagated, Amazon S3 might list the deleted object.

The US Standard region provides eventual consistency for all requests. All other regions provide read-after-write consistency for PUTS of new objects and eventual consistency for overwrite PUTS and Deletes.

Note

Amazon S3 does not currently support object locking. If two PUT requests are simultaneously made to the same key, the request with the latest time stamp wins. If this is an issue, you will need to build an object-locking mechanism into your application.

Updates are key-based; there is no way to make atomic updates across keys. For example, you cannot make the update of one key dependent on the update of another key unless you design this functionality into your application.

The following table describes the characteristics of eventually consistent read and consistent read.

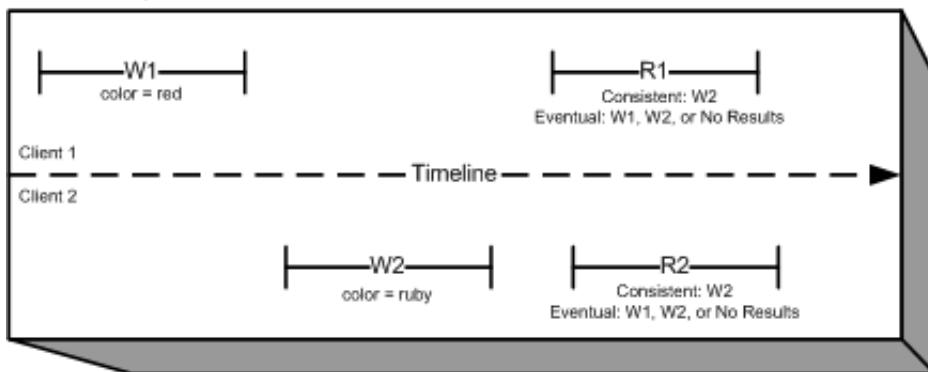
Eventually Consistent Read	Consistent Read
Stale reads possible	No stale reads
Lowest read latency	Potential higher read latency
Highest read throughput	Potential lower read throughput

Concurrent Applications

This section provides examples of eventually consistent and consistent read requests when multiple clients are writing to the same items.

In this example, both W1 (write 1) and W2 (write 2) complete before the start of R1 (read 1) and R2 (read 2). For a consistent read, R1 and R2 both return `color = ruby`. For an eventually consistent read, R1 and R2 might return `color = red`, `color = ruby`, or no results, depending on the amount of time that has elapsed.

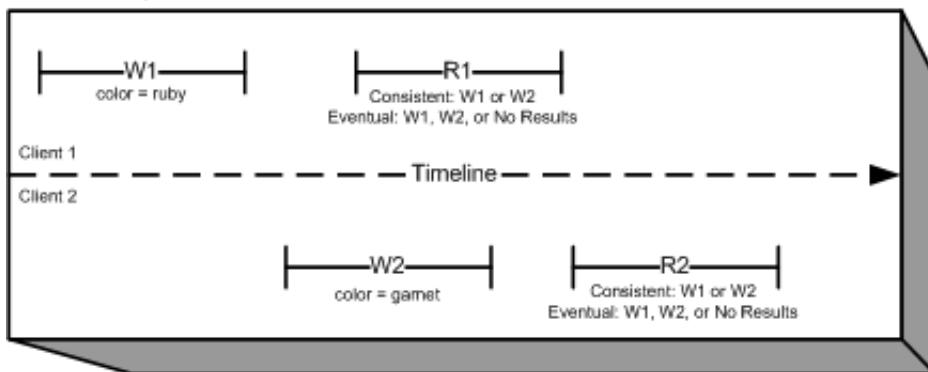
Domain = MyDomain, Item = StandardFez



In the next example, W2 does not complete before the start of R1. Therefore, R1 might return `color = ruby` or `color = garnet` for either a consistent read or an eventually consistent read. Also, depending on the amount of time that has elapsed, an eventually consistent read might return no results.

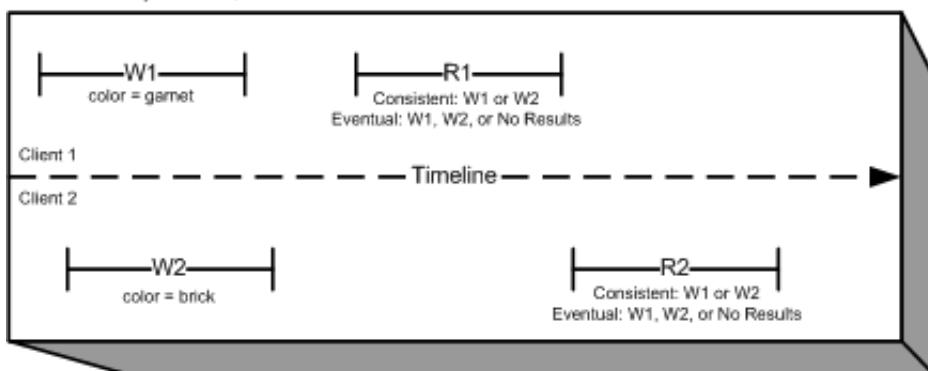
For a consistent read, R2 returns `color = garnet`. For an eventually consistent read, R2 might return `color = ruby`, `color = garnet`, or no results depending on the amount of time that has elapsed.

Domain = MyDomain, Item = StandardFez



In the last example, Client 2 performs W2 before Amazon S3 returns a success for W1, so the outcome of the final value is unknown (`color = garnet` or `color = brick`). Any subsequent reads (consistent read or eventually consistent) might return either value. Also, depending on the amount of time that has elapsed, an eventually consistent read might return no results.

Domain = MyDomain, Item = StandardFez



Features

Topics

- Reduced Redundancy Storage (p. 7)
- Bucket Policies (p. 7)
- AWS Identity and Access Management (p. 8)
- Access Control Lists (p. 8)
- Versioning (p. 8)
- Operations (p. 8)

This section describes important Amazon S3 features.

Reduced Redundancy Storage

Customers can store their data using the Amazon S3 Reduced Redundancy Storage (RRS) option. RRS enables customers to reduce their costs by storing non-critical, reproducible data at lower levels of redundancy than Amazon S3 standard storage. RRS provides a cost-effective, highly available solution for distributing or sharing content that is durably stored elsewhere, or for storing thumbnails, transcoded media, or other processed data that can be easily reproduced. The RRS option stores objects on multiple devices across multiple facilities, providing 400 times the durability of a typical disk drive, but does not replicate objects as many times as standard Amazon S3 storage, and thus is even more cost effective.

RRS provides 99.99% durability of objects over a given year. This durability level corresponds to an average expected loss of 0.01% of objects annually.

AWS charges less for using RRS than for standard Amazon S3 storage. For pricing information, go to the [Amazon S3 detail page](#).

For more information, see [Using Reduced Redundancy Storage \(p. 419\)](#).

Bucket Policies

Bucket policies provide centralized, access control to buckets and objects based on a variety of conditions, including Amazon S3 operations, requesters, resources, and aspects of the request (e.g., IP address). The policies are expressed in our *access policy language* and enable centralized management of permissions. The permissions attached to a bucket apply to all of the objects in that bucket.

Individuals as well as companies can use bucket policies. When companies register with Amazon S3 they create an *account*. Thereafter, the company becomes synonymous with the account. Accounts are financially responsible for the Amazon resources they (and their employees) create. Accounts have the power to grant bucket policy permissions and assign employees permissions based on a variety of conditions. For example, an account could create a policy that gives a user write access:

- To a particular S3 bucket
- From an account's corporate network
- During business hours
- From an account's custom application (as identified by a user agent string)

An account can grant one application limited read and write access, but allow another to create and delete buckets as well. An account could allow several field offices to store their daily reports in a single bucket, allowing each office to write only to a certain set of names (e.g. "Nevada/*" or "Utah/*") and only from the office's IP address range.

Unlike access control lists (described below), which can add (grant) permissions only on individual objects, policies can either add or deny permissions across all (or a subset) of objects within a bucket. With one request an account can set the permissions of any number of objects in a bucket. An account can use wildcards (similar to regular expression operators) on Amazon resource names (ARNs) and other values, so that an account can control access to groups of objects that begin with a common prefix or end with a given extension such as `.html`.

Only the bucket owner is allowed to associate a policy with a bucket. Policies, written in the access policy language, `allow` or `deny` requests based on:

- Amazon S3 bucket operations (such as `PUT ?acl`), and object operations (such as `PUT Object`, or `GET Object`)
- Requester
- Conditions specified in the policy

An account can control access based on specific Amazon S3 operations, such as `GetObject`, `GetObjectVersion`, `DeleteObject`, or `DeleteBucket`.

The conditions can be such things as IP addresses, IP address ranges in CIDR notation, dates, user agents, HTTP referrer and transports (HTTP and HTTPS).

For more information, see [Using Bucket Policies and User Policies \(p. 312\)](#).

AWS Identity and Access Management

For example, you can use IAM with Amazon S3 to control the type of access a user or group of users has to specific parts of an Amazon S3 bucket your AWS account owns.

For more information about IAM, see the following:

- [Identity and Access Management \(IAM\)](#)
- [IAM Getting Started Guide](#)
- [Using IAM](#)

Access Control Lists

For more information, see [Managing Access with ACLs \(p. 363\)](#)

Versioning

For more information, see [Object Versioning \(p. 84\)](#)

Operations

Following are the most common operations you'll execute through the API.

Common Operations

- **Create a Bucket** – Create and name your own bucket in which to store your objects.
- **Write an Object** – Store data by creating or overwriting an object. When you write an object, you specify a unique key in the namespace of your bucket. This is also a good time to specify any access control you want on the object.
- **Read an Object** – Read data back. You can download the data via HTTP or BitTorrent.

- **Deleting an Object** – Delete some of your data.
- **Listing Keys** – List the keys contained in one of your buckets. You can filter the key list based on a prefix.

Details on this and all other functionality are described in detail later in this guide.

Amazon S3 Application Programming Interfaces (API)

The Amazon S3 architecture is designed to be programming language-neutral, using our supported interfaces to store and retrieve objects.

Amazon S3 provides a REST and a SOAP interface. They are similar, but there are some differences. For example, in the REST interface, metadata is returned in HTTP headers. Because we only support HTTP requests of up to 4 KB (not including the body), the amount of metadata you can supply is restricted.

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

The REST Interface

The REST API is an HTTP interface to Amazon S3. Using REST, you use standard HTTP requests to create, fetch, and delete buckets and objects.

You can use any toolkit that supports HTTP to use the REST API. You can even use a browser to fetch objects, as long as they are anonymously readable.

The REST API uses the standard HTTP headers and status codes, so that standard browsers and toolkits work as expected. In some areas, we have added functionality to HTTP (for example, we added headers to support access control). In these cases, we have done our best to add the new functionality in a way that matched the style of standard HTTP usage.

The SOAP Interface

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

The SOAP API provides a SOAP 1.1 interface using document literal encoding. The most common way to use SOAP is to download the WSDL (go to <http://doc.s3.amazonaws.com/2006-03-01/AmazonS3.wsdl>), use a SOAP toolkit such as Apache Axis or Microsoft .NET to create bindings, and then write code that uses the bindings to call Amazon S3.

Paying for Amazon S3

Pricing for Amazon S3 is designed so that you don't have to plan for the storage requirements of your application. Most storage providers force you to purchase a predetermined amount of storage and network transfer capacity: If you exceed that capacity, your service is shut off or you are charged high overage fees. If you do not exceed that capacity, you pay as though you used it all.

Amazon S3 charges you only for what you actually use, with no hidden fees and no overage charges. This gives developers a variable-cost service that can grow with their business while enjoying the cost advantages of Amazon's infrastructure.

Before storing anything in Amazon S3, you need to register with the service and provide a payment instrument that will be charged at the end of each month. There are no set-up fees to begin using the service. At the end of the month, your payment instrument is automatically charged for that month's usage.

For information about paying for Amazon S3 storage, go to the [Amazon Simple Storage Service product details page](#).

Related Services

Once you load your data into Amazon S3, you can use it with other services that we provide. The following services are the ones you might use most frequently:

- **Amazon Elastic Compute Cloud** – This web service provides virtual compute resources in the cloud. For more information, go to the [Amazon EC2 product details page](#).
- **Amazon Elastic MapReduce** – This web service enables businesses, researchers, data analysts, and developers to easily and cost-effectively process vast amounts of data. It utilizes a hosted Hadoop framework running on the web-scale infrastructure of Amazon EC2 and Amazon S3. For more information, go to the [Amazon Elastic MapReduce product details page](#).
- **AWS Import/Export** – AWS Import/Export enables you to mail a storage device, such as a RAID drive, to Amazon so that we can upload your (terabytes) of data into Amazon S3. For more information, go to the [AWS Import/Export Developer Guide](#).

Making Requests

Topics

- [About Access Keys \(p. 11\)](#)
- [Request Endpoints \(p. 13\)](#)
- [Making Requests Using the AWS SDKs \(p. 14\)](#)
- [Making Requests Using the REST API \(p. 47\)](#)

Amazon S3 is a REST service. You can send requests to Amazon S3 using the REST API or the AWS SDK (see [Sample Code and Libraries](#)) wrapper libraries that wrap the underlying Amazon S3 REST API, simplifying your programming tasks.

Every interaction with Amazon S3 is either authenticated or anonymous. Authentication is a process of verifying the identity of the requester trying to access an Amazon Web Services (AWS) product. Authenticated requests must include a signature value that authenticates the request sender. The signature value is, in part, generated from the requester's AWS access keys (access key ID and secret access key). For more information about getting access keys, see [How Do I Get Security Credentials?](#) in the [AWS General Reference](#).

If you are using the AWS SDK, the libraries compute the signature from the keys you provide. However, if you make direct REST API calls in your application, you must write the code to compute the signature and add it to the request.

About Access Keys

The following sections review the types of access keys that you can use to make authenticated requests.

AWS Account Access Keys

The account access keys provide full access to the AWS resources owned by the account. The following are examples of access keys:

- Access key ID (a 20-character, alphanumeric string). For example: AKIAIOSFODNN7EXAMPLE
- Secret access key (a 40-character string). For example:
`wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`

The access key ID uniquely identifies an AWS account. You can use these access keys to send authenticated requests to Amazon S3.

IAM User Access Keys

You can create one AWS account for your company; however, there may be several employees in the organization who need access to your organization's AWS resources. Sharing your AWS account access keys reduces security, and creating individual AWS accounts for each employee might not be practical. Also, you cannot easily share resources such as buckets and objects because they are owned by different accounts. To share resources, you must grant permissions, which is additional work.

In such scenarios, you can use AWS Identity and Access Management (IAM) to create users under your AWS account with their own access keys and attach IAM user policies granting appropriate resource access permissions to them. To better manage these users, IAM enables you to create groups of users and grant group-level permissions that apply to all users in that group.

These users are referred as IAM users that you create and manage within AWS. The parent account controls a user's ability to access AWS. Any resources an IAM user creates are under the control of and paid for by the parent AWS account. These IAM users can send authenticated requests to Amazon S3 using their own security credentials. For more information about creating and managing users under your AWS account, go to the [AWS Identity and Access Management product details page](#).

Temporary Security Credentials

In addition to creating IAM users with their own access keys, IAM also enables you to grant temporary security credentials (temporary access keys and a security token) to any IAM user to enable them to access your AWS services and resources. You can also manage users in your system outside AWS. These are referred as federated users. Additionally, users can be applications that you create to access your AWS resources.

IAM provides the AWS Security Token Service API for you to request temporary security credentials. You can use either the AWS STS API or the AWS SDK to request these credentials. The API returns temporary security credentials (access key ID and secret access key), and a security token. These credentials are valid only for the duration you specify when you request them. You use the access key ID and secret key the same way you use them when sending requests using your AWS account or IAM user access keys. In addition, you must include the token in each request you send to Amazon S3.

An IAM user can request these temporary security credentials for their own use or hand them out to federated users or applications. When requesting temporary security credentials for federated users, you must provide a user name and an IAM policy defining the permissions you want to associate with these temporary security credentials. The federated user cannot get more permissions than the parent IAM user who requested the temporary credentials.

You can use these temporary security credentials in making requests to Amazon S3. The API libraries compute the necessary signature value using those credentials to authenticate your request. If you send requests using expired credentials, Amazon S3 denies the request.

For information on signing requests using temporary security credentials in your REST API requests, see [Signing and Authenticating REST Requests \(p. 557\)](#). For information about sending requests using AWS SDKs, see [Making Requests Using the AWS SDKs \(p. 14\)](#).

For more information about IAM support for temporary security credentials, see [Granting Temporary Access to Your AWS Resources](#) in [Using IAM](#).

For added security, you can require multifactor authentication (MFA) when accessing your Amazon S3 resources by configuring a bucket policy. For information, see [Adding a Bucket Policy to Require MFA Authentication \(p. 339\)](#). After you require MFA to access your Amazon S3 resources, the only way you can access these resources is by providing temporary credentials that are created with an MFA key. For

more information, see the [AWS Multi-Factor Authentication](#) detail page and [Configuring MFA-Protected API Access in Using IAM](#).

Request Endpoints

You send REST requests to the service's predefined endpoint. For a list of all AWS services and their corresponding endpoints, go to [Regions and Endpoints](#) in the [AWS General Reference](#).

Making Requests Using the AWS SDKs

Topics

- [Making Requests Using AWS Account or IAM User Credentials \(p. 15\)](#)
- [Making Requests Using IAM User Temporary Credentials \(p. 20\)](#)
- [Making Requests Using Federated User Temporary Credentials \(p. 31\)](#)

You can send authenticated requests to Amazon S3 using either the AWS SDK or by making the REST API calls directly in your application. The AWS SDK API uses the credentials that you provide to compute the signature for authentication. If you use the REST API directly in your applications, you must write the necessary code to compute the signature for authenticating your request. For a list of available AWS SDKs go to, [Sample Code and Libraries](#).

Making Requests Using AWS Account or IAM User Credentials

You can use an AWS account or IAM user security credentials to send authenticated requests to Amazon S3. This section provides examples of how you can send authenticated requests using the AWS SDK for Java, AWS SDK for .NET, and AWS SDK for PHP. For a list of available AWS SDKs go to, [Sample Code and Libraries](#).

Topics

- [Making Requests Using AWS Account or IAM User Credentials - AWS SDK for Java \(p. 15\)](#)
- [Making Requests Using AWS Account or IAM User Credentials - AWS SDK for .NET \(p. 16\)](#)
- [Making Requests Using AWS Account or IAM User Credentials - AWS SDK for PHP \(p. 18\)](#)
- [Making Requests Using AWS Account or IAM User Credentials - AWS SDK for Ruby \(p. 19\)](#)

For more information about setting up your AWS credentials for use with the AWS SDK for Java, see [Testing the Java Code Examples \(p. 545\)](#).

Making Requests Using AWS Account or IAM User Credentials - AWS SDK for Java

The following tasks guide you through using the Java classes to send authenticated requests using your AWS account credentials or IAM user credentials.

Making Requests Using Your AWS account or IAM user credentials

1	Create an instance of the <code>AmazonS3Client</code> class.
2	Execute one of the <code>AmazonS3Client</code> methods to send requests to Amazon S3. The client generates the necessary signature value from your credentials and includes it in the request it sends to Amazon S3.

The following Java code sample demonstrates the preceding tasks.

```
AmazonS3 s3client = new AmazonS3Client(new ProfileCredentialsProvider());  
  
// Send sample request (list objects in a given bucket).  
ObjectListing objectListing = s3client.listObjects(new  
    ListObjectsRequest().withBucketName(bucketName));
```

Note

You can create the `AmazonS3Client` class without providing your security credentials. Requests sent using this client are anonymous requests, without a signature. Amazon S3 returns an error if you send anonymous requests for a resource that is not publicly available.

To see how to make requests using your AWS credentials within the context of an example of listing all the object keys in your bucket, see [Listing Keys Using the AWS SDK for Java \(p. 220\)](#). For more examples, see [Working with Amazon S3 Objects \(p. 78\)](#) and [Working with Amazon S3 Buckets \(p. 55\)](#). You can test these examples using your AWS Account or IAM user credentials.

Related Resources

- Using the AWS SDKs and Explorers (p. 542)

Making Requests Using AWS Account or IAM User Credentials - AWS SDK for .NET

The following tasks guide you through using the .NET classes to send authenticated requests using your AWS account or IAM user credentials.

Making Requests Using Your AWS Account or IAM User Credentials

1	Create an instance of the <code>AmazonS3Client</code> class.
2	Execute one of the <code>AmazonS3Client</code> methods to send requests to Amazon S3. The client generates the necessary signature from your credentials and includes it in the request it sends to Amazon S3.

The following C# code sample demonstrates the preceding tasks.

For information on running the .NET examples in this guide and for instructions on how to store your credentials in a configuration file, see [Testing the .NET Code Examples \(p. 547\)](#).

```
using System;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazonaws.com.docsamples
{
    class MakeS3Request
    {
        static string bucketName      = "**** Provide bucket name ****";
        static IAmazonS3 client;

        public static void Main(string[] args)
        {
            using (client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1))

            {
                Console.WriteLine("Listing objects stored in a bucket");
                ListingObjects();
            }

            Console.WriteLine("Press any key to continue... ");
            Console.ReadKey();
        }

        static void ListingObjects()
        {
            try
            {
                ListObjectsRequest request = new ListObjectsRequest
                {
                    BucketName = bucketName,
```

```
        MaxKeys = 2
    };

    do
    {
        ListObjectsResponse response = client.ListObjects(request);

        // Process response.
        foreach (S3Object entry in response.S3Objects)
        {
            Console.WriteLine("key = {0} size = {1}",
                entry.Key, entry.Size);
        }

        // If response is truncated, set the marker to get the next
        // set of keys.
        if (response.IsTruncated)
        {
            request.Marker = response.NextMarker;
        }
        else
        {
            request = null;
        }
    } while (request != null);
}
catch (AmazonS3Exception amazonS3Exception)
{
    if (amazonS3Exception.ErrorCode != null &&
        (amazonS3Exception.ErrorCode.Equals("InvalidAccessKeyId") ||
         amazonS3Exception.ErrorCode.Equals("InvalidSecurity")))
    {
        Console.WriteLine("Check the provided AWS Credentials.");
        Console.WriteLine(
            "To sign up for service, go to http://aws.amazon.com/s3");
    }
    else
    {
        Console.WriteLine(
            "Error occurred. Message:'{0}' when listing objects",
            amazonS3Exception.Message);
    }
}
}
```

Note

You can create the `AmazonS3Client` client without providing your security credentials. Requests sent using this client are anonymous requests, without a signature. Amazon S3 returns an error if you send anonymous requests for a resource that is not publicly available.

For working examples, see [Working with Amazon S3 Objects \(p. 78\)](#) and [Working with Amazon S3 Buckets \(p. 55\)](#). You can test these examples using your AWS Account or an IAM user credentials.

For example, to list all the object keys in your bucket, see [Listing Keys Using the AWS SDK for .NET \(p. 223\)](#).

Related Resources

- [Using the AWS SDKs and Explorers \(p. 542\)](#)

Making Requests Using AWS Account or IAM User Credentials - AWS SDK for PHP

This topic guides you through using a class from the AWS SDK for PHP to send authenticated requests using your AWS account or IAM user credentials.

Note

This topic assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 547\)](#) and have the AWS SDK for PHP properly installed.

Making Requests Using Your AWS Account or IAM user Credentials

1	Create an instance of an Amazon S3 client by using the Aws\S3\S3Client class <code>factory()</code> method.
2	Execute one of the <code>Aws\S3\S3Client</code> methods to send requests to Amazon S3. For example, you can use the Aws\S3\S3Client::listBuckets() method to send a request to list all the buckets for your account. The client API generates the necessary signature using your credentials and includes it in the request it sends to Amazon S3.

The following PHP code sample demonstrates the preceding tasks and illustrates how the client makes a request using your security credentials to list all the buckets for your account.

```
use Aws\S3\S3Client;

// Instantiate the S3 client with your AWS credentials
$s3 = S3Client::factory();

$result = $s3->listBuckets();
```

For working examples, see [Working with Amazon S3 Objects \(p. 78\)](#) and [Working with Amazon S3 Buckets \(p. 55\)](#). You can test these examples using your AWS account or IAM user credentials.

For an example of listing object keys in a bucket, see [Listing Keys Using the AWS SDK for PHP \(p. 225\)](#).

Related Resources

- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::factory\(\) Method](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::listBuckets\(\) Method](#)
- [AWS SDK for PHP for Amazon S3](#)
- [AWS SDK for PHP Documentation](#)

Making Requests Using AWS Account or IAM User Credentials - AWS SDK for Ruby

The following tasks guide you through using the AWS SDK for Ruby to send authenticated requests using your AWS Account credentials or IAM user credentials.

Making Requests Using Your AWS Account or IAM user Credentials

1	Create an instance of the <code>AWS::S3</code> class.
2	Make a request to Amazon S3 by enumerating objects in a bucket using the <code>buckets</code> method of <code>AWS::S3</code> . The client generates the necessary signature value from your credentials and includes it in the request it sends to Amazon S3.

The following Ruby code sample demonstrates the preceding tasks.

```
# Get an instance of the S3 interface using the specified credentials configuration.  
s3 = AWS::S3.new()  
  
# Get a list of all object keys in a bucket.  
bucket = s3.buckets[bucket_name].objects.collect(&:key)  
puts bucket
```

Note

You can create the `AWS::S3` client without providing your security credentials. Requests sent using this client are anonymous requests, without a signature. Amazon S3 returns an error if you send anonymous requests for a resource that is not publicly available.

For working examples, see [Working with Amazon S3 Objects \(p. 78\)](#) and [Working with Amazon S3 Buckets \(p. 55\)](#). You can test these examples using your AWS Account or IAM user credentials.

Making Requests Using IAM User Temporary Credentials

Topics

- Making Requests Using IAM User Temporary Credentials - AWS SDK for Java (p. 20)
- Making Requests Using IAM User Temporary Credentials - AWS SDK for .NET (p. 23)
- Making Requests Using AWS Account or IAM User Temporary Credentials - AWS SDK for PHP (p. 26)
- Making Requests Using IAM User Temporary Credentials - AWS SDK for Ruby (p. 29)

An AWS Account or an IAM user can request temporary security credentials and use them to send authenticated requests to Amazon S3. This section provides examples of how to use the AWS SDK for Java, .NET, and PHP to obtain temporary security credentials and use them to authenticate your requests to Amazon S3.

Making Requests Using IAM User Temporary Credentials - AWS SDK for Java

An IAM user or an AWS Account can request temporary security credentials (see [Making Requests \(p. 11\)](#)) using AWS SDK for Java and use them to access Amazon S3. These credentials expire after the session duration. By default, the session duration is one hour. If you use IAM user credentials, you can specify duration, between 1 and 36 hours, when requesting the temporary security credentials.

Making Requests Using IAM User Temporary Security Credentials

1	Create an instance of the AWS Security Token Service client <code>AWSecurityTokenServiceClient</code> .
2	Start a session by calling the <code>GetSessionToken</code> method of the STS client you created in the preceding step. You provide session information to this method using a <code>GetSessionTokenRequest</code> object. The method returns your temporary security credentials.
3	Package the temporary security credentials in an instance of the <code>BasicSessionCredentials</code> object so you can provide the credentials to your Amazon S3 client.
4	Create an instance of the <code>AmazonS3Client</code> class by passing in the temporary security credentials. You send the requests to Amazon S3 using this client. If you send requests using expired credentials, Amazon S3 returns an error.

The following Java code sample demonstrates the preceding tasks.

```
// In real applications, the following code is part of your trusted code. It
has
// your security credentials you use to obtain temporary security credentials.
AWSecurityTokenServiceClient stsClient =
        new AWSecurityTokenServiceClient(new ProfileCreden
tialsProvider());
//
// Manually start a session.
```

```
GetSessionTokenRequest getSessionTokenRequest = new GetSessionTokenRequest();
// Following duration can be set only if temporary credentials are requested
by an IAM user.
getSessionTokenRequest.setDurationSeconds(7200);

GetSessionTokenResult sessionTokenResult =
    stsClient.getSessionToken(getSessionTokenRequest);
Credentials sessionCredentials = sessionTokenResult.getCredentials();

// Package the temporary security credentials as
// a BasicSessionCredentials object, for an Amazon S3 client object to use.
BasicSessionCredentials basicSessionCredentials =
    new BasicSessionCredentials(sessionCredentials.getAccessKeyId(),
                                sessionCredentials.getSecretAccessKey(),
                                sessionCredentials.getSessionToken());

// The following will be part of your less trusted code. You provide temporary
// security
// credentials so it can send authenticated requests to Amazon S3.
// Create Amazon S3 client by passing in the basicSessionCredentials object.
AmazonS3Client s3 = new AmazonS3Client(basicSessionCredentials);

// Test. For example, get object keys in a bucket.
ObjectListing objects = s3.listObjects(bucketName);
```

Example

Note

If you obtain temporary security credentials using your AWS account credentials, the temporary security credentials are valid for only one hour. You can specify session duration only if you use IAM user credentials to request a session.

The following Java code example lists the object keys in the specified bucket. For illustration, the code example obtains temporary security credentials for a default one hour session and uses them to send an authenticated request to Amazon S3.

If you want to test the sample using IAM user credentials, you will need to create an IAM user under your AWS Account. For more information about how to create an IAM user, go to [Set Up a Group, Grant Permissions, and Add Users](#) in the *IAM Getting Started Guide*.

```
import java.io.IOException;
import com.amazonaws.auth.BasicSessionCredentials;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.securitytoken.AWSecurityTokenServiceClient;
import com.amazonaws.services.securitytoken.model.Credentials;
import com.amazonaws.services.securitytoken.model.GetSessionTokenRequest;
import com.amazonaws.services.securitytoken.model.GetSessionTokenResult;
import com.amazonaws.services.s3.model.ObjectListing;

public class S3Sample {
    private static String bucketName = "**** Provide bucket name ****";

    public static void main(String[] args) throws IOException {
        AWSecurityTokenServiceClient stsClient =
            new AWSecurityTokenServiceClient(new ProfileCre
dentialsProvider());
        //
        // Start a session.
        GetSessionTokenRequest getSessionTokenRequest =
            new GetSessionTokenRequest();

        GetSessionTokenResult sessionTokenResult =
            stsClient.getSessionToken(getSessionTokenRequest);

        Credentials sessionCredentials = sessionTokenResult.getCredentials();
        System.out.println("Session Credentials: "
            + sessionCredentials.toString());

        //
        // Package the session credentials as a BasicSessionCredentials
        // object for an S3 client object to use.
        BasicSessionCredentials basicSessionCredentials =
            new BasicSessionCredentials(sessionCredentials.getAccessKeyId(),
                sessionCredentials.getSecretAccessKey(),
                sessionCredentials.getSessionToken());
        AmazonS3Client s3 = new AmazonS3Client(basicSessionCredentials);

        //
        // Test. For example, get object keys for a given bucket.
        ObjectListing objects = s3.listObjects(bucketName);
        System.out.println("No. of Objects = " +
```

```
        objects.getObjectSummaries().size());  
    }  
}
```

Related Resources

- [Using the AWS SDKs and Explorers \(p. 542\)](#)

Making Requests Using IAM User Temporary Credentials - AWS SDK for .NET

An IAM user or an AWS Account can request temporary security credentials (see [Making Requests \(p. 11\)](#)) using the AWS SDK for .NET and use them to access Amazon S3. These credentials expire after the session duration. By default, the session duration is one hour. If you use IAM user credentials, you can specify duration, between 1 and 36 hours, when requesting the temporary security credentials.

Making Requests Using IAM User Temporary Security Credentials

1	Create an instance of the AWS Security Token Service client, <code>AmazonSecurityTokenServiceClient</code> . For information about providing credentials, see Using the AWS SDKs and Explorers (p. 542) .
2	Start a session by calling the <code>GetSessionToken</code> method of the STS client you created in the preceding step. You provide session information to this method using a <code>GetSessionTokenRequest</code> object. The method returns you temporary security credentials.
3	Package up the temporary security credentials in an instance of the <code>SessionAWSCredentials</code> object. You use this object to provide the temporary security credentials to your Amazon S3 client.
4	Create an instance of the <code>AmazonS3Client</code> class by passing in the temporary security credentials. You send requests to Amazon S3 using this client. If you send requests using expired credentials, Amazon S3 returns an error.

The following C# code sample demonstrates the preceding tasks.

```
// In real applications, the following code is part of your trusted code. It  
has  
// your security credentials you use to obtain temporary security credentials.  
AmazonSecurityTokenServiceConfig config = new AmazonSecurityTokenServiceConfig();  
  
AmazonSecurityTokenServiceClient stsClient =  
    new AmazonSecurityTokenServiceClient(config);  
  
GetSessionTokenRequest getSessionTokenRequest = new GetSessionTokenRequest();  
// Following duration can be set only if temporary credentials are requested  
by an IAM user.  
getSessionTokenRequest.DurationSeconds = 7200; // seconds.  
Credentials credentials =
```

```
stsClient.GetSessionToken(getSessionTokenRequest).GetSessionTokenResult.Credentials;

SessionAWSCredentials sessionCredentials =
    new SessionAWSCredentials(credentials.AccessKeyId,
                               credentials.SecretAccessKey,
                               credentials.SessionToken);

// The following will be part of your less trusted code. You provide temporary
// security
// credentials so it can send authenticated requests to Amazon S3.
// Create Amazon S3 client by passing in the basicSessionCredentials object.
AmazonS3Client s3Client = new AmazonS3Client(sessionCredentials);

// Test. For example, send request to list object key in a bucket.
var response = s3Client.ListObjects(bucketName);
```

Example

Note

If you obtain temporary security credentials using your AWS account security credentials, the temporary security credentials are valid for only one hour. You can specify session duration only if you use IAM user credentials to request a session.

The following C# code example lists object keys in the specified bucket. For illustration, the code example obtains temporary security credentials for a default one hour session and uses them to send authenticated request to Amazon S3.

If you want to test the sample using IAM user credentials, you will need to create an IAM user under your AWS Account. For more information about how to create an IAM user, go to [Set Up a Group, Grant Permissions, and Add Users](#) in the *IAM Getting Started Guide*.

For instructions on how to create and test a working example, see [Testing the .NET Code Examples \(p. 547\)](#).

```
using System;
using System.Configuration;
using System.Collections.Specialized;
using Amazon.S3;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;
using Amazon.Runtime;
using Amazon.S3.Model;
using System.Collections.Generic;

namespace s3.amazon.com.docsamples
{
    class TempCredExplicitSessionStart
    {
        static string bucketName = "**** Provide bucket name ****";
        static IAmazonS3 client;

        public static void Main(string[] args)
        {
            NameValueCollection appConfig = ConfigurationManager.AppSettings;
            string accessKeyID = appConfig["AWSAccessKey"];
            string secretAccessKeyID = appConfig["AWSSecretKey"];

            try
            {
                Console.WriteLine("Listing objects stored in a bucket");
                SessionAWSCredentials tempCredentials =
                    GetTemporaryCredentials(accessKeyID, secretAccessKeyID);

                // Create client by providing temporary security credentials.
                using (client = new AmazonS3Client(tempCredentials, Amazon.RegionEndpoint.USEast1))
                {
                    ListObjectsRequest listObjectRequest =
                        new ListObjectsRequest();
                    listObjectRequest.BucketName = bucketName;

                    // Send request to Amazon S3.
                    ListObjectsResponse response = client.ListObjects(listObjectRequest);
                    List<S3Object> objects = response.S3Objects;
                }
            }
        }
    }
}
```

```
        Console.WriteLine("Object count = {0}", objects.Count);

        Console.WriteLine("Press any key to continue... ");
        Console.ReadKey();
    }
}
catch (AmazonS3Exception s3Exception)
{
    Console.WriteLine(s3Exception.Message,
                      s3Exception.InnerException);
}
catch (AmazonSecurityTokenServiceException stsException)
{
    Console.WriteLine(stsException.Message,
                      stsException.InnerException);
}
}

private static SessionAWSCredentials GetTemporaryCredentials(
    string accessKeyId, string secretAccessKeyId)
{
    AmazonSecurityTokenServiceClient stsClient =
        new AmazonSecurityTokenServiceClient(accessKeyId,
                                              secretAccessKeyId);

    GetSessionTokenRequest getSessionTokenRequest =
        new GetSessionTokenRequest();
    getSessionTokenRequest.DurationSeconds = 7200; // seconds

    GetSessionTokenResponse sessionTokenResponse =
        stsClient.GetSessionToken(getSessionTokenRequest);
    Credentials credentials = sessionTokenResponse.Credentials;

    SessionAWSCredentials sessionCredentials =
        new SessionAWSCredentials(credentials.AccessKeyId,
                                  credentials.SecretAccessKey,
                                  credentials.SessionToken);

    return sessionCredentials;
}
}
```

Related Resources

- [Using the AWS SDKs and Explorers \(p. 542\)](#)

Making Requests Using AWS Account or IAM User Temporary Credentials - AWS SDK for PHP

This topic guides you through using classes from the AWS SDK for PHP to request temporary security credentials and use them to access Amazon S3.

Note

This topic assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 547\)](#) and have the AWS SDK for PHP properly installed.

An IAM user or an AWS Account can request temporary security credentials (see [Making Requests \(p. 11\)](#)) using the AWS SDK for PHP and use them to access Amazon S3. These credentials expire when the session duration expires. By default, the session duration is one hour. If you use IAM user credentials, you can specify the duration, between 1 and 36 hours, when requesting the temporary security credentials. For more information about temporary security credentials, go to [Using Temporary Security Credentials](#).

Making Requests Using AWS Account or IAM User Temporary Security Credentials

1	Create an instance of an AWS Security Token Service (AWS STS) client by using the Aws\Sts\StsClient class factory() method.
2	Execute the Aws\Sts\StsClient::getSessionToken() method to start a session. The method returns you temporary security credentials.
3	Create an instance of an Amazon S3 client by using the Aws\S3\S3Client class factory() method with the temporary security credentials you obtained in the preceding step. Any methods in the S3Client class that you call use the temporary security credentials to send authenticated requests to Amazon S3.

The following PHP code sample demonstrates how to request temporary security credentials and use them to access Amazon S3.

```
use Aws\Sts\StsClient;
use Aws\S3\S3Client;

// In real applications, the following code is part of your trusted code.
// It has your security credentials that you use to obtain temporary
// security credentials.
$sts = StsClient::factory();

$result = $sts->getSessionToken();

// The following will be part of your less trusted code. You provide temporary
// security credentials so it can send authenticated requests to Amazon S3.
// Create an Amazon S3 client using temporary security credentials.
$credentials = $result->get('Credentials');
$s3 = S3Client::factory(array(
    'key'      => $credentials['AccessKeyId'],
    'secret'   => $credentials['SecretAccessKey'],
    'token'    => $credentials['SessionToken']
));

$result = $s3->listBuckets();
```

Note

If you obtain temporary security credentials using your AWS account security credentials, the temporary security credentials are valid for only one hour. You can specify the session duration only if you use IAM user credentials to request a session.

Example of Making an Amazon S3 Request Using Temporary Security Credentials

The following PHP code example lists object keys in the specified bucket using temporary security credentials. The code example obtains temporary security credentials for a default one hour session and uses them to send authenticated request to Amazon S3. For information about running the PHP examples in this guide, go to [Running PHP Examples \(p. 548\)](#).

If you want to test the example using IAM user credentials, you will need to create an IAM user under your AWS Account. For information about how to create an IAM user, go to [Set Up a Group, Grant Permissions, and Add Users](#) in the *IAM Getting Started Guide*. For an example of setting session duration when using IAM user credentials to request a session, see [Making Requests Using Federated User Temporary Credentials - AWS SDK for PHP \(p. 40\)](#).

```
<?php

// Include the AWS SDK using the Composer autoloader.
require 'vendor/autoload.php';

use Aws\Sts\StsClient;
use Aws\S3\S3Client;
use Aws\S3\Exception\S3Exception;

$bucket = '*** Your Bucket Name ***';

$sts = StsClient::factory();

$credentials = $sts->getSessionToken()->get('Credentials');
$s3 = S3Client::factory(array(
    'key'    => $credentials['AccessKeyId'],
    'secret' => $credentials['SecretAccessKey'],
    'token'  => $credentials['SessionToken']
));

try {
    $objects = $s3->getIterator('ListObjects', array(
        'Bucket' => $bucket
    ));
    echo "Keys retrieved!\n";
    foreach ($objects as $object) {
        echo $object['Key'] . "\n";
    }
} catch (S3Exception $e) {
    echo $e->getMessage() . "\n";
}
```

Related Resources

- [AWS SDK for PHP for Amazon S3 Aws\Sts\StsClient Class](#)
- [AWS SDK for PHP for Amazon S3 Aws\Sts\StsClient::factory\(\) Method](#)
- [AWS SDK for PHP for Amazon S3 Aws\Sts\StsClient::getSessionToken\(\) Method](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::factory\(\) Method](#)
- [AWS SDK for PHP for Amazon S3](#)
- [AWS SDK for PHP Documentation](#)

Making Requests Using IAM User Temporary Credentials - AWS SDK for Ruby

An IAM user or an AWS Account can request temporary security credentials (see [Making Requests \(p. 11\)](#)) using AWS SDK for Ruby and use them to access Amazon S3. These credentials expire after the session duration. By default, the session duration is one hour. If you use IAM user credentials, you can specify the duration, between 1 and 36 hours, when requesting the temporary security credentials.

Making Requests Using IAM User Temporary Security Credentials

1	Create an instance of the AWS Security Token Service client <code>AWS::STS::Session</code> by providing your credentials.
2	Start a session by calling the <code>new_session</code> method of the STS client that you created in the preceding step. You provide session information to this method using a <code>GetSessionTokenRequest</code> object. The method returns your temporary security credentials.
3	Use the temporary credentials in a new instance of the <code>AWS::S3</code> class by passing in the temporary security credentials. You send the requests to Amazon S3 using this client. If you send requests using expired credentials, Amazon S3 returns an error.

The following Ruby code sample demonstrates the preceding tasks.

```
# Start a session.  
# In real applications, the following code is part of your trusted code. It has  
# your security credentials that you use to obtain temporary security credentials.  
  
sts = AWS::STS.new()  
  
session = sts.new_session()  
puts "Session expires at: #{session.expires_at.to_s}"  
  
# Get an instance of the S3 interface using the session credentials.  
s3 = AWS::S3.new(session.credentials)  
  
# Get a list of all object keys in a bucket.  
bucket = s3.buckets[bucket_name].objects.collect(&:key)
```

Example

Note

If you obtain temporary security credentials using your AWS account security credentials, the temporary security credentials are valid for only one hour. You can specify session duration only if you use IAM user credentials to request a session.

The following Ruby code example lists the object keys in the specified bucket. For illustration, the code example obtains temporary security credentials for a default one hour session and uses them to send an authenticated request to Amazon S3.

If you want to test the sample using IAM user credentials, you will need to create an IAM user under your AWS Account. For more information about how to create an IAM user, go to [Set Up a Group, Grant Permissions, and Add Users](#) in the *IAM Getting Started Guide*.

```
require 'rubygems'
require 'aws-sdk'

# In real applications, the following code is part of your trusted code. It has
# your security credentials you use to obtain temporary security credentials.

bucket_name = '*** Provide bucket name ***'

# Start a session.
sts = AWS::STS.new()
session = sts.new_session()
puts "Session expires at: #{session.expires_at.to_s}"

# get an instance of the S3 interface using the session credentials
s3 = AWS::S3.new(session.credentials)

# get a list of all object keys in a bucket
bucket = s3.buckets[bucket_name].objects.collect(&:key)
puts bucket
```

Making Requests Using Federated User Temporary Credentials

Topics

- Making Requests Using Federated User Temporary Credentials - AWS SDK for Java (p. 31)
- Making Requests Using Federated User Temporary Credentials - AWS SDK for .NET (p. 36)
- Making Requests Using Federated User Temporary Credentials - AWS SDK for PHP (p. 40)
- Making Requests Using Federated User Temporary Credentials - AWS SDK for Ruby (p. 45)

You can request temporary security credentials and provide them to your federated users or applications who need to access your AWS resources. This section provides examples of how you can use the AWS SDK to obtain temporary security credentials for your federated users or applications and send authenticated requests to Amazon S3 using those credentials. For a list of available AWS SDKs go to, [Sample Code and Libraries](#).

Note

Both the AWS account and an IAM user can request temporary security credentials for federated users. However, for added security, only an IAM user with the necessary permissions should request these temporary credentials to ensure that the federated user gets at most the permissions of the requesting IAM user. In some applications, you might find suitable to create an IAM user with specific permissions for the sole purpose of granting temporary security credentials to your federated users and applications.

Making Requests Using Federated User Temporary Credentials - AWS SDK for Java

You can provide temporary security credentials for your federated users and applications (see [Making Requests \(p. 11\)](#)) so they can send authenticated requests to access your AWS resources. When requesting these temporary credentials from the IAM service, you must provide a user name and an IAM policy describing the resource permissions you want to grant. By default, the session duration is one hour. However, if you are requesting temporary credentials using IAM user credentials, you can explicitly set a different duration value when requesting the temporary security credentials for federated users and applications.

Note

To request temporary security credentials for federated users and applications, for added security, you might want to use a dedicated IAM user with only the necessary access permissions. The temporary user you create can never get more permissions than the IAM user who requested the temporary security credentials. For more information, go to [AWS Identity and Access Management FAQs](#).

Making Requests Using Federated User Temporary Security Credentials

1	Create an instance of the AWS Security Token Service client <code>AWSecurityTokenServiceClient</code> .
2	Start a session by calling the <code>getFederationToken</code> method of the STS client you created in the preceding step. You will need to provide session information including the user name and an IAM policy that you want to attach to the temporary credentials. This method returns your temporary security credentials.

3	Package the temporary security credentials in an instance of the <code>BasicSessionCredentials</code> object. You use this object to provide the temporary security credentials to your Amazon S3 client.
4	Create an instance of the <code>AmazonS3Client</code> class by passing the temporary security credentials. You send requests to Amazon S3 using this client. If you send requests using expired credentials, Amazon S3 returns an error.

The following Java code sample demonstrates the preceding tasks.

```
// In real applications, the following code is part of your trusted code. It
has
// your security credentials you use to obtain temporary security credentials.
AWSecurityTokenServiceClient stsClient =
        new AWSecurityTokenServiceClient(new ProfileCreden
tialsProvider());

GetFederationTokenRequest getFederationTokenRequest =
        new GetFederationTokenRequest();
getFederationTokenRequest.setDurationSeconds(7200);
getFederationTokenRequest.setName("User1");

// Define the policy and add to the request.
Policy policy = new Policy();
// Define the policy here.
// Add the policy to the request.
getFederationTokenRequest.setPolicy(policy.toJson());

GetFederationTokenResult federationTokenResult =
        stsClient.getFederationToken(getFederationTokenRequest);
Credentials sessionCredentials = federationTokenResult.getCredentials();

// Package the session credentials as a BasicSessionCredentials object
// for an S3 client object to use.
BasicSessionCredentials basicSessionCredentials = new BasicSessionCredentials(
        sessionCredentials.getAccessKeyId(),
        sessionCredentials.getSecretAccessKey(),
        sessionCredentials.getSessionToken());

// The following will be part of your less trusted code. You provide temporary
// security
// credentials so it can send authenticated requests to Amazon S3.
// Create an Amazon S3 client by passing in the basicSessionCredentials object.
AmazonS3Client s3 = new AmazonS3Client(basicSessionCredentials);

// Test. For example, send list object keys in a bucket.
ObjectListing objects = s3.listObjects(bucketName);
```

To set a condition in the policy, create a `Condition` object and associate it with the policy. The following code sample shows a condition that allows users from a specified IP range to list objects.

```
Policy policy = new Policy();
```

```
// Allow only a specified IP range.  
Condition condition = new StringCondition(StringCondition.StringComparison  
Type.StringLike,  
    ConditionFactory.SOURCE_IP_CONDITION_KEY , "192.168.143.*");  
  
policy.withStatements(new Statement(Effect.Allow)  
    .withActions(S3Actions.ListObjects)  
    .withConditions(condition)  
    .withResources(new Resource("arn:aws:s3:::"+ bucketName)));  
  
getFederationTokenRequest.setPolicy(policy.toJson());
```

Example

The following Java code example lists keys in the specified bucket. In the code example, you first obtain temporary security credentials for a two-hour session for your federated user (User1) and use them to send authenticated requests to Amazon S3.

When requesting temporary credentials for others, for added security, you use the security credentials of an IAM user who has permissions to request temporary security credentials. You can also limit the access permissions of this IAM user to ensure that the IAM user grants only the minimum application-specific permissions when requesting temporary security credentials. This sample only lists objects in a specific bucket. Therefore, first create an IAM user with the following policy attached.

```
{  
    "Statement": [  
        {  
            "Action": ["s3>ListBucket",  
                      "sts:GetFederationToken*"  
            ],  
            "Effect": "Allow",  
            "Resource": "*"  
        }  
    ]  
}
```

The policy allows the IAM user to request temporary security credentials and access permission only to list your AWS resources. For information about how to create an IAM user, go to [Set Up a Group, Grant Permissions, and Add Users](#) in the *IAM Getting Started Guide*.

You can now use the IAM user security credentials to test the following example. The example sends authenticated request to Amazon S3 using temporary security credentials. The example specifies the following policy when requesting temporary security credentials for the federated user (User1) which restricts access to list objects in a specific bucket (YourBucketName). You must update the policy and provide your own existing bucket name.

```
{  
    "Statement": [  
        {  
            "Sid": "1",  
            "Action": ["s3>ListBucket"],  
            "Effect": "Allow",  
            "Resource": "arn:aws:s3:::YourBucketName"  
        }  
    ]  
}
```

You must update the following sample and provide the bucket name that you specified in the preceding federated user access policy.

```
import java.io.IOException;  
import com.amazonaws.auth.BasicSessionCredentials;  
import com.amazonaws.auth.PropertiesCredentials;  
import com.amazonaws.auth.policy.Policy;  
import com.amazonaws.auth.policy.Resource;  
import com.amazonaws.auth.policy.Statement;  
import com.amazonaws.auth.policy.Statement.Effect;  
import com.amazonaws.auth.policy.actions.S3Actions;  
import com.amazonaws.services.s3.AmazonS3Client;
```

```
import com.amazonaws.services.securitytoken.AWSSecurityTokenServiceClient;
import com.amazonaws.services.securitytoken.model.Credentials;
import com.amazonaws.services.securitytoken.model.GetFederationTokenRequest;
import com.amazonaws.services.securitytoken.model.GetFederationTokenResult;
import com.amazonaws.services.s3.model.ObjectListing;

public class S3Sample {
    private static String bucketName = "**** Specify bucket name ****";
    public static void main(String[] args) throws IOException {
        AWSSecurityTokenServiceClient stsClient =
            new AWSSecurityTokenServiceClient(new ProfileCreden-
tialsProvider());

        GetFederationTokenRequest getFederationTokenRequest =
            new GetFederationTokenRequest();
        getFederationTokenRequest.setDurationSeconds(7200);
        getFederationTokenRequest.setName("User1");

        // Define the policy and add to the request.
        Policy policy = new Policy();
        policy.withStatements(new Statement(Effect.Allow)
            .withActions(S3Actions.ListObjects)
            .withResources(new Resource("arn:aws:s3:::ExampleBucket")));

        getFederationTokenRequest.setPolicy(policy.toJson());

        // Get the temporary security credentials.
        GetFederationTokenResult federationTokenResult =
            stsClient.getFederationToken(getFederationTokenRequest);

        Credentials sessionCredentials = federationTokenResult.getCredentials();

        // Package the session credentials as a BasicSessionCredentials
        // object for an S3 client object to use.
        BasicSessionCredentials basicSessionCredentials =
            new BasicSessionCredentials(sessionCredentials.getAccessKeyId(),
                sessionCredentials.getSecretAccessKey(),
                sessionCredentials.getSessionToken());
        AmazonS3Client s3 = new AmazonS3Client(basicSessionCredentials);

        // Test. For example, send ListBucket request using the temporary secur-
        ity credentials.
        ObjectListing objects = s3.listObjects(bucketName);
        System.out.println("No. of Objects = " + objects.getObjectSummar-
ies().size());
    }
}
```

Related Resources

- [Using the AWS SDKs and Explorers \(p. 542\)](#)

Making Requests Using Federated User Temporary Credentials - AWS SDK for .NET

You can provide temporary security credentials for your federated users and applications (see [Making Requests \(p. 11\)](#)) so they can send authenticated requests to access your AWS resources. When requesting these temporary credentials, you must provide a user name and an IAM policy describing the resource permissions you want to grant. By default, the session duration is one hour. You can explicitly set a different duration value when requesting the temporary security credentials for federated users and applications.

Note

To request temporary security credentials for federated users and applications, for added security, you might want to use a dedicated IAM user with only the necessary access permissions. The temporary user you create can never get more permissions than the IAM user who requested the temporary security credentials. For more information, go to [AWS Identity and Access Management FAQs](#).

Making Requests Using Federated User Temporary Credentials

1	Create an instance of the AWS Security Token Service client, <code>AmazonSecurityTokenServiceClient</code> class. For information about providing credentials, see Using the AWS SDK for .NET (p. 546) .
2	Start a session by calling the <code>GetFederationToken</code> method of the STS client. You will need to provide session information including the user name and an IAM policy that you want to attach to the temporary credentials. You can provide an optional session duration. This method returns your temporary security credentials.
3	Package the temporary security credentials in an instance of the <code>SessionAWSCredentials</code> object. You use this object to provide the temporary security credentials to your Amazon S3 client.
4	Create an instance of the <code>AmazonS3Client</code> class by passing the temporary security credentials. You send requests to Amazon S3 using this client. If you send requests using expired credentials, Amazon S3 returns an error.

The following C# code sample demonstrates the preceding tasks.

```
// In real applications, the following code is part of your trusted code. It has
// your security credentials you use to obtain temporary security credentials.
AmazonSecurityTokenServiceConfig config = new AmazonSecurityTokenServiceConfig();
AmazonSecurityTokenServiceClient stsClient =
    new AmazonSecurityTokenServiceClient(config);

GetFederationTokenRequest federationTokenRequest =
    new GetFederationTokenRequest();
federationTokenRequest.Name          = "User1";
federationTokenRequest.Policy       = "**** Specify policy ***";
federationTokenRequest.DurationSeconds = 7200;

GetFederationTokenResponse federationTokenResponse = stsClient.GetFederationToken(federationTokenRequest);
```

```
GetFederationTokenResult federationTokenResult = federationTokenResponse.GetFederationTokenResult;
Credentials credentials = federationTokenResult.Credentials;

SessionAWSCredentials sessionCredentials =
    new SessionAWSCredentials(credentials.AccessKeyId,
                               credentials.SecretAccessKey,
                               credentials.SessionToken);

// The following will be part of your less trusted code. You provide temporary
// security
// credentials so it can send authenticated requests to Amazon S3.
// Create Amazon S3 client by passing in the basicSessionCredentials object.
AmazonS3Client s3Client = new AmazonS3Client(sessionCredentials);
// Test. For example, send list object keys in a bucket.
ListObjectsRequest listObjectRequest = new ListObjectsRequest();
listObjectRequest.BucketName = bucketName;
ListObjectsResponse response = s3Client.ListObjects(listObjectRequest);
```

Example

The following C# code example lists keys in the specified bucket. In the code example, you first obtain temporary security credentials for a two-hour session for your federated user (User1) and use them to send authenticated requests to Amazon S3.

When requesting temporary credentials for others, for added security, you use the security credentials of an IAM user who has permissions to request temporary security credentials. You can also limit the access permissions of this IAM user to ensure that the IAM user grants only the minimum application-specific permissions to the federated user. This sample only lists objects in a specific bucket. Therefore, first create an IAM user with the following policy attached.

```
{  
    "Statement": [  
        {  
            "Action": ["s3>ListBucket",  
                      "sts:GetFederationToken*"  
            ],  
            "Effect": "Allow",  
            "Resource": "*"  
        }  
    ]  
}
```

The policy allows the IAM user to request temporary security credentials and access permission only to list your AWS resources. For more information about how to create an IAM user, go to [Set Up a Group, Grant Permissions, and Add Users](#) in the *IAM Getting Started Guide*.

You can now use the IAM user security credentials to test the following example. The example sends authenticated request to Amazon S3 using temporary security credentials. The example specifies the following policy when requesting temporary security credentials for the federated user (User1) which restricts access to list objects in a specific bucket (YourBucketName). You must update the policy and provide your own existing bucket name.

```
{  
    "Statement": [  
        {  
            "Sid": "1",  
            "Action": ["s3>ListBucket"],  
            "Effect": "Allow",  
            "Resource": "arn:aws:s3:::YourBucketName"  
        }  
    ]  
}
```

You must update the following sample and provide the bucket name that you specified in the preceding federated user access policy. For instructions on how to create and test a working example, see [Testing the .NET Code Examples \(p. 547\)](#).

```
using System;  
using System.Configuration;  
using System.Collections.Specialized;  
using Amazon.S3;  
using Amazon.SecurityToken;  
using Amazon.SecurityToken.Model;  
using Amazon.Runtime;  
using Amazon.S3.Model;
```

```
using System.Collections.Generic;

namespace s3.amazon.com.docsamples
{
    class TempFederatedCredentials
    {
        static string bucketName = "**** Provide bucket name ****";
        static IAmazonS3 client;

        public static void Main(string[] args)
        {
            NameValueCollection appConfig = ConfigurationManager.AppSettings;
            string accessKeyID = appConfig["AWSAccessKey"];
            string secretAccessKeyID = appConfig["AWSSecretKey"];

            try
            {
                Console.WriteLine("Listing objects stored in a bucket");
                SessionAWSCredentials tempCredentials =
                    GetTemporaryFederatedCredentials(accessKeyID, secretAccess
KeyID);

                // Create client by providing temporary security credentials.
                using (client = new AmazonS3Client(tempCredentials, Amazon.Re
gionEndpoint.USEast1))
                {

                    ListObjectsRequest listObjectRequest = new ListObjects
Request();
                    listObjectRequest.BucketName = bucketName;

                    ListObjectsResponse response = client.ListObjects(listOb
jectRequest);
                    List<S3Object> objects = response.S3Objects;
                    Console.WriteLine("Object count = {0}", objects.Count);

                    Console.WriteLine("Press any key to continue...");
                    Console.ReadKey();
                }
            }
            catch (AmazonS3Exception s3Exception)
            {
                Console.WriteLine(s3Exception.Message,
                    s3Exception.InnerException);
            }
            catch (AmazonSecurityTokenServiceException stsException)
            {
                Console.WriteLine(stsException.Message,
                    stsException.InnerException);
            }
        }

        private static SessionAWSCredentials GetTemporaryFederatedCredentials(
            string accessKeyId, string secretAccessKeyId)
        {
            AmazonSecurityTokenServiceConfig config = new AmazonSecurityTokenSer
viceConfig();
            AmazonSecurityTokenServiceClient stsClient =

```

```
        new AmazonSecurityTokenServiceClient(
                        accessKeyId, secretAccessKeyId,
config);

        GetFederationTokenRequest federationTokenRequest =
                new GetFederationTokenRequest();
        federationTokenRequest.DurationSeconds = 7200;
        federationTokenRequest.Name = "User1";
        federationTokenRequest.Policy = @"
            ""Statement"":
            [
                {
                    ""Sid"": ""Stmt1311212314284"",
                    ""Action"": [ ""s3>ListBucket"" ],
                    ""Effect"": ""Allow"",
                    ""Resource"": ""arn:aws:s3:::YourBucketName"""
                }
            ]
        }
    ";
}

GetFederationTokenResponse federationTokenResponse =
        stsClient.GetFederationToken(federationTokenRequest);

Credentials credentials = federationTokenResponse.Credentials;

SessionAWSCredentials sessionCredentials =
        new SessionAWSCredentials(credentials.AccessKeyId,
                                  credentials.SecretAccessKey,
                                  credentials.SessionToken);
return sessionCredentials;
}
}
```

Related Resources

- [Using the AWS SDKs and Explorers \(p. 542\)](#)

Making Requests Using Federated User Temporary Credentials - AWS SDK for PHP

This topic guides you through using classes from the AWS SDK for PHP to request temporary security credentials for federated users and applications and use them to access Amazon S3.

Note

This topic assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 547\)](#) and have the AWS SDK for PHP properly installed.

You can provide temporary security credentials to your federated users and applications (see [Making Requests \(p. 11\)](#)) so they can send authenticated requests to access your AWS resources. When requesting these temporary credentials, you must provide a user name and an IAM policy describing the resource permissions you want to grant. These credentials expire when the session duration expires. By default, the session duration is one hour. You can explicitly set a different duration value when requesting the temporary security credentials for federated users and applications. For more information about temporary security credentials, go to [Using Temporary Security Credentials](#).

To request temporary security credentials for federated users and applications, for added security, you might want to use a dedicated IAM user with only the necessary access permissions. The temporary user you create can never get more permissions than the IAM user who requested the temporary security credentials. For information about identity federation, go to [AWS Identity and Access Management FAQs](#).

Making Requests Using Federated User Temporary Credentials

1	Create an instance of an AWS Security Token Service (AWS STS) client by using the Aws\Sts\StsClient class <code>factory()</code> method.
2	Execute the Aws\Sts\StsClient::getFederationToken() method by providing the name of the federated user in the <code>array</code> parameter's required <code>Name</code> key. You can also add the optional <code>array</code> parameter's <code>Policy</code> and <code>DurationSeconds</code> keys. The method returns temporary security credentials that you can provide to your federated users.
3	Any federated user who has the temporary security credentials can send requests to Amazon S3 by creating an instance of an Amazon S3 client by using Aws\S3\S3Client class <code>factory</code> method with the temporary security credentials. Any methods in the <code>S3Client</code> class that you call use the temporary security credentials to send authenticated requests to Amazon S3.

The following PHP code sample demonstrates obtaining temporary security credentials for a federated user and using the credentials to access Amazon S3.

```
use Aws\Sts\StsClient;
use Aws\S3\S3Client;

// In real applications, the following code is part of your trusted code. It
// has
// your security credentials that you use to obtain temporary security creden-
// tials.
$sts = StsClient::factory();

// Fetch the federated credentials.
$result = $sts->getFederationToken(array(
    'Name'          => 'User1',
    'DurationSeconds' => 3600,
    'Policy'        => json_encode(array(
        'Statement' => array(
            array(
                'Sid'      => 'randomstatementid' . time(),
                'Action'   => array('s3>ListBucket'),
                'Effect'   => 'Allow',
                'Resource' => 'arn:aws:s3:::' . YourBucketName
            )
        )
    )));
))

// The following will be part of your less trusted code. You provide temporary
// security credentials so it can send authenticated requests to Amazon S3.
$credentials = $result->get('Credentials');
$s3 = new S3Client::factory(array(
    'key'    => $credentials['AccessKeyId'],
    'secret' => $credentials['SecretAccessKey'],
    'region' => $credentials['SessionToken']
));
```

```
'secret' => $credentials['SecretAccessKey'],
'token'  => $credentials['SessionToken']
));
$result = $s3->listObjects();
```

Example of a Federated User Making an Amazon S3 Request Using Temporary Security Credentials

The following PHP code example lists keys in the specified bucket. In the code example, you first obtain temporary security credentials for an hour session for your federated user (User1) and use them to send authenticated requests to Amazon S3. For information about running the PHP examples in this guide, go to [Running PHP Examples \(p. 548\)](#).

When requesting temporary credentials for others, for added security, you use the security credentials of an IAM user who has permissions to request temporary security credentials. You can also limit the access permissions of this IAM user to ensure that the IAM user grants only the minimum application-specific permissions to the federated user. This example only lists objects in a specific bucket. Therefore, first create an IAM user with the following policy attached.

```
{  
    "Statement": [  
        {  
            "Action": [ "s3>ListBucket" ,  
                      "sts>GetFederationToken*" ] ,  
            "Effect": "Allow" ,  
            "Resource": "*" }  
    ]  
}
```

The policy allows the IAM user to request temporary security credentials and access permission only to list your AWS resources. For more information about how to create an IAM user, go to [Set Up a Group, Grant Permissions, and Add Users](#) in the *IAM Getting Started Guide*.

You can now use the IAM user security credentials to test the following example. The example sends an authenticated request to Amazon S3 using temporary security credentials. The example specifies the following policy when requesting temporary security credentials for the federated user (User1) which restricts access to list objects in a specific bucket. You must update the policy with your own existing bucket name.

```
{  
    "Statement": [  
        {  
            "Sid": "1" ,  
            "Action": [ "s3>ListBucket" ] ,  
            "Effect": "Allow" ,  
            "Resource": "arn:aws:s3:::YourBucketName" }  
    ]  
}
```

In the following example you must replace YourBucketName with your own existing bucket name when specifying the policy resource.

```
<?php  
  
// Include the AWS SDK using the Composer autoloader.  
require 'vendor/autoload.php';  
  
$bucket = '*** Your Bucket Name ***';
```

```
use Aws\Sts\StsClient;
use Aws\S3\S3Client;
use Aws\S3\Exception\S3Exception;

// Instantiate the client.
$sts = StsClient::factory();

$result = $sts->getFederationToken(array(
    'Name'          => 'User1',
    'DurationSeconds' => 3600,
    'Policy'         => json_encode(array(
        'Statement' => array(
            array(
                'Sid'      => 'randomstatementid' . time(),
                'Action'   => array('s3>ListBucket'),
                'Effect'   => 'Allow',
                'Resource' => 'arn:aws:s3:::YourBucketName'
            )
        )
    )));
);

$credentials = $result->get('Credentials');
$s3 = S3Client::factory(array(
    'key'    => $credentials['AccessKeyId'],
    'secret' => $credentials['SecretAccessKey'],
    'token'  => $credentials['SessionToken']
));

try {
    $objects = $s3->getIterator('ListObjects', array(
        'Bucket' => $bucket
    ));
    echo "Keys retrieved!\n";
    foreach ($objects as $object) {
        echo $object['Key'] . "\n";
    }
} catch (S3Exception $e) {
    echo $e->getMessage() . "\n";
}
```

Related Resources

- [AWS SDK for PHP for Amazon S3 Aws\Sts\StsClient Class](#)
- [AWS SDK for PHP for Amazon S3 Aws\Sts\StsClient::factory\(\) Method](#)
- [AWS SDK for PHP for Amazon S3 Aws\Sts\StsClient::getSessionToken\(\) Method](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::factory\(\) Method](#)
- [AWS SDK for PHP for Amazon S3](#)
- [AWS SDK for PHP Documentation](#)

Making Requests Using Federated User Temporary Credentials - AWS SDK for Ruby

You can provide temporary security credentials for your federated users and applications (see [Making Requests \(p. 11\)](#)) so that they can send authenticated requests to access your AWS resources. When requesting these temporary credentials from the IAM service, you must provide a user name and an IAM policy describing the resource permissions you want to grant. By default, the session duration is one hour. However, if you are requesting temporary credentials using IAM user credentials, you can explicitly set a different duration value when requesting the temporary security credentials for federated users and applications.

Note

To request temporary security credentials for federated users and applications, for added security, you might want to use a dedicated IAM user with only the necessary access permissions. The temporary user you create can never get more permissions than the IAM user who requested the temporary security credentials. For more information, go to [AWS Identity and Access Management FAQs](#).

Making Requests Using Federated User Temporary Security Credentials

1	Create an instance of the AWS Security Token Service client <code>AWS::STS::Session</code> .
2	<p>Start a session by calling the <code>new_federated_session</code> method of the STS client you created in the preceding step.</p> <p>You will need to provide session information including the user name and an IAM policy that you want to attach to the temporary credentials.</p> <p>This method returns your temporary security credentials.</p>
3	Create an instance of the <code>AWS::S3</code> class by passing the temporary security credentials. You send requests to Amazon S3 using this client. If you send requests using expired credentials, Amazon S3 returns an error.

The following Ruby code sample demonstrates the preceding tasks.

```
# Start a session with restricted permissions.
sts = AWS::STS.new()
policy = AWS::STS::Policy.new
policy.allow(
  :actions => [ "s3>ListBucket" ],
  :resources => "arn:aws:s3:::{bucket_name}" )

session = sts.new_federated_session(
  'User1',
  :policy => policy,
  :duration => 2*60*60)

puts "Policy: #{policy.to_json}"

# Get an instance of the S3 interface using the session credentials.
s3 = AWS::S3.new(session.credentials)

# Get a list of all object keys in a bucket.
bucket = s3.buckets[bucket_name].objects.collect(&:key)
```

Example

The following Ruby code example lists keys in the specified bucket. In the code example, you first obtain temporary security credentials for a two hour session for your federated user (User1) and use them to send authenticated requests to Amazon S3.

When requesting temporary credentials for others, for added security, you use the security credentials of an IAM user who has permissions to request temporary security credentials. You can also limit the access permissions of this IAM user to ensure that the IAM user grants only the minimum application specific permissions when requesting temporary security credentials. This sample only lists objects in a specific bucket. Therefore, first create an IAM user with the following policy attached.

```
{  
    "Statement": [  
        {  
            "Action": ["s3>ListBucket",  
                      "sts:GetFederationToken*"  
            ],  
            "Effect": "Allow",  
            "Resource": "*"  
        }  
    ]  
}
```

The policy allows the IAM user to request temporary security credentials and access permission only to list your AWS resources. For more information about how to create an IAM user, go to [Set Up a Group, Grant Permissions, and Add Users](#) in the *IAM Getting Started Guide*.

You can now use the IAM user security credentials to test the following example. The example sends an authenticated request to Amazon S3 using temporary security credentials. The example specifies the following policy when requesting temporary security credentials for the federated user (User1), which restricts access to listing objects in a specific bucket (YourBucketName). To use this example in your code, update the policy and provide your own bucket name.

```
{  
    "Statement": [  
        {  
            "Sid": "1",  
            "Action": ["s3>ListBucket"],  
            "Effect": "Allow",  
            "Resource": "arn:aws:s3:::YourBucketName"  
        }  
    ]  
}
```

To use this example in your code, provide your access key ID and secret key and the bucket name that you specified in the preceding federated user access policy.

```
require 'rubygems'  
require 'aws-sdk'  
  
# In real applications, the following code is part of your trusted code. It has  
# your security credentials that you use to obtain temporary security credentials.  
  
bucket_name = '*** Provide bucket name ***'
```

```
# Start a session with restricted permissions.  
sts = AWS::STS.new()  
policy = AWS::STS::Policy.new  
policy.allow(  
  :actions => ["s3>ListBucket"],  
  :resources => "arn:aws:s3:::#{bucket_name}")  
  
session = sts.new_federated_session(  
  'User1',  
  :policy => policy,  
  :duration => 2*60*60)  
  
puts "Policy: #{policy.to_json}"  
  
# Get an instance of the S3 interface using the session credentials.  
s3 = AWS::S3.new(session.credentials)  
  
# Get a list of all object keys in a bucket.  
bucket = s3.buckets[bucket_name].objects.collect(&:key)  
puts "No. of Objects = #{bucket.count.to_s}"  
puts bucket
```

Making Requests Using the REST API

Topics

- [Virtual Hosting of Buckets \(p. 48\)](#)
- [Request Redirection and the REST API \(p. 52\)](#)

This section contains information specific to the Amazon S3 REST API. The examples in this guide use the newer, virtual hosted-style method for accessing buckets instead of the path-style method. For more information, see [Working with Amazon S3 Buckets \(p. 55\)](#)

Following is an example of a virtual hosted-style request to delete the `puppy.jpg` file from the `mybucket` bucket.

```
DELETE /puppy.jpg HTTP/1.1  
User-Agent: dotnet  
Host: mybucket.s3.amazonaws.com  
Date: Tue, 15 Jan 2008 21:20:27 +0000  
x-amz-date: Tue, 15 Jan 2008 21:20:27 +0000  
Authorization: AWS AKIAIOSFODNN7EXAMPLE:k3nL7gH3+PadhTEVn5EXAMPLE
```

Following is an example of a path-style version of the same request.

```
DELETE /mybucket/puppy.jpg HTTP/1.1  
User-Agent: dotnet  
Host: s3.amazonaws.com  
Date: Tue, 15 Jan 2008 21:20:27 +0000  
x-amz-date: Tue, 15 Jan 2008 21:20:27 +0000  
Authorization: AWS AKIAIOSFODNN7EXAMPLE:k3nL7gH3+PadhTEVn5EXAMPLE
```

Amazon S3 supports virtual-hosted-style and path-style access in all regions. The path-style syntax, however, requires that you use the region-specific endpoint when attempting to access a bucket. For example, if you have a bucket called `mybucket` that resides in the EU, you want to use path-style syntax, and the object is named `puppy.jpg`, the correct URI is

`http://s3-eu-west-1.amazonaws.com/mybucket/puppy.jpg`. You will receive a "PermanentRedirect" error, an HTTP response code 301, and a message indicating what the correct URI is for your resource if you try to access a bucket outside the US Standard region with path-style syntax that uses either of the following:

- `http://s3.amazonaws.com`
- An endpoint for a region different from the one where the bucket resides, for example, `http://s3-eu-west-1.amazonaws.com` for a bucket that was created in the US West (Northern California) region

Virtual Hosting of Buckets

Topics

- [HTTP Host Header Bucket Specification \(p. 49\)](#)
- [Examples \(p. 49\)](#)
- [Customizing Amazon S3 URLs with CNAMEs \(p. 51\)](#)
- [Limitations \(p. 52\)](#)
- [Backward Compatibility \(p. 52\)](#)

In general, virtual hosting is the practice of serving multiple web sites from a single web server. One way to differentiate sites is by using the apparent host name of the request instead of just the path name part of the URI. An ordinary Amazon S3 REST request specifies a bucket by using the first slash-delimited component of the Request-URI path. Alternatively, you can use Amazon S3 virtual hosting to address a bucket in a REST API call by using the `HTTP Host` header. In practice, Amazon S3 interprets `Host` as meaning that most buckets are automatically accessible (for limited types of requests) at `http://bucketname.s3.amazonaws.com`. Furthermore, by naming your bucket after your registered domain name and by making that name a DNS alias for Amazon S3, you can completely customize the URL of your Amazon S3 resources, for example, `http://my.bucketname.com/`.

Besides the attractiveness of customized URLs, a second benefit of virtual hosting is the ability to publish to the "root directory" of your bucket's virtual server. This ability can be important because many existing applications search for files in this standard location. For example, `favicon.ico`, `robots.txt`, `crossdomain.xml` are all expected to be found at the root.

Important

Amazon S3 supports virtual-hosted-style and path-style access in all regions. The path-style syntax, however, requires that you use the region-specific endpoint when attempting to access a bucket. For example, if you have a bucket called `mybucket` that resides in the EU, you want to use path-style syntax, and the object is named `puppy.jpg`, the correct URI is `http://s3-eu-west-1.amazonaws.com/mybucket/puppy.jpg`. You will receive a "PermanentRedirect" error, an HTTP response code 301, and a message indicating what the correct URI is for your resource if you try to access a bucket outside the US Standard region with path-style syntax that uses either of the following:

- `http://s3.amazonaws.com`
- An endpoint for a region different from the one where the bucket resides, for example, `http://s3-eu-west-1.amazonaws.com` for a bucket that was created in the US West (Northern California) region

Note

Amazon S3 routes any virtual hosted-style requests to the US Standard Region by default if you use the US Standard endpoint (s3.amazonaws.com), instead of the region-specific endpoint (for example, s3-eu-west-1.amazonaws.com). When you create a bucket, in any region, Amazon S3 updates DNS to reroute the request to the correct location, which might take time. In the meantime, the default rule applies and your virtual hosted-style request goes to the US Standard region, and Amazon S3 redirects it with HTTP 307 redirect to the correct region. For more information, see [Request Redirection and the REST API \(p. 509\)](#).

When using virtual hosted-style buckets with SSL, the SSL wild card certificate only matches buckets that do not contain periods. To work around this, use HTTP or write your own certificate verification logic.

HTTP Host Header Bucket Specification

As long as your GET request does not use the SSL endpoint, you can specify the bucket for the request by using the `HTTP Host` header. The `Host` header in a REST request is interpreted as follows:

- If the `Host` header is omitted or its value is '`s3.amazonaws.com`', the bucket for the request will be the first slash-delimited component of the Request-URI, and the key for the request will be the rest of the Request-URI. This is the ordinary method, as illustrated by the first and second examples in this section. Omitting the `Host` header is valid only for HTTP 1.0 requests.
- Otherwise, if the value of the `Host` header ends in '`.s3.amazonaws.com`', the bucket name is the leading component of the `Host` header's value up to '`.s3.amazonaws.com`'. The key for the request is the Request-URI. This interpretation exposes buckets as subdomains of `s3.amazonaws.com`, as illustrated by the third and fourth examples in this section.
- Otherwise, the bucket for the request is the lowercase value of the `Host` header, and the key for the request is the Request-URI. This interpretation is useful when you have registered the same DNS name as your bucket name and have configured that name to be a CNAME alias for Amazon S3. The procedure for registering domain names and configuring DNS is beyond the scope of this guide, but the result is illustrated by the final example in this section.

Examples

This section provides example URLs and requests.

Example Path Style Method

This example uses `johnsmith.net` as the bucket name and `homepage.html` as the key name.

The URL is as follows:

```
http://s3.amazonaws.com/johnsmith.net/homepage.html
```

The request is as follows:

```
GET /johnsmith.net/homepage.html HTTP/1.1
Host: s3.amazonaws.com
```

The request with HTTP 1.0 and omitting the `host` header is as follows:

```
GET /johnsmith.net/homepage.html HTTP/1.0
```

For information about DNS-compatible names, see [Limitations \(p. 52\)](#). For more information about keys, see [Keys \(p. 4\)](#).

Example Virtual Hosted-Style Method

This example uses `johnsmith.net` as the bucket name and `homepage.html` as the key name.

The URL is as follows:

```
http://johnsmith.net.s3.amazonaws.com/homepage.html
```

The request is as follows:

```
GET /homepage.html HTTP/1.1
Host: johnsmith.net.s3.amazonaws.com
```

The virtual hosted-style method requires the bucket name to be DNS-compliant.

Example Virtual Hosted-Style Method for a Bucket in a Region Other Than US Standard

This example uses `johnsmith.eu` as the name for a bucket in the EU (Ireland) region and `homepage.html` as the key name.

The URL is as follows:

```
http://johnsmith.eu.s3-eu-west-1.amazonaws.com/homepage.html
```

The request is as follows:

```
GET /homepage.html HTTP/1.1
Host: johnsmith.eu.s3-eu-west-1.amazonaws.com
```

Note that, instead of using the region-specific endpoint, you can also use the US Standard endpoint no matter what region the bucket resides.

```
http://johnsmith.eu.s3.amazonaws.com/homepage.html
```

The request is as follows:

```
GET /homepage.html HTTP/1.1
Host: johnsmith.eu.s3.amazonaws.com
```

Example CNAME Method

This example uses `www.johnsmith.net` as the bucket name and `homepage.html` as the key name. To use this method, you must configure your DNS name as a CNAME alias for `bucketname.s3.amazonaws.com`.

The URL is as follows:

```
http://www.johnsmith.net/homepage.html
```

The example is as follows:

```
GET /homepage.html HTTP/1.1
Host: www.johnsmith.net
```

Customizing Amazon S3 URLs with CNAMEs

Depending on your needs, you might not want "s3.amazonaws.com" to appear on your website or service. For example, if you host your website images on Amazon S3, you might prefer `http://images.johnsmith.net/` instead of `http://johnsmith-images.s3.amazonaws.com/`.

The bucket name must be the same as the CNAME. So `http://images.johnsmith.net/filename` would be the same as `http://images.johnsmith.net.s3.amazonaws.com/filename` if a CNAME were created to map `images.johnsmith.net` to `images.johnsmith.net.s3.amazonaws.com`.

Any bucket with a DNS-compatible name can be referenced as follows:

`http://[BucketName].s3.amazonaws.com/[Filename]`, for example, `http://images.johnsmith.net.s3.amazonaws.com/mydog.jpg`. By using CNAME, you can map `images.johnsmith.net` to an Amazon S3 host name so that the previous URL could become `http://images.johnsmith.net/mydog.jpg`.

The CNAME DNS record should alias your domain name to the appropriate virtual hosted-style host name. For example, if your bucket name and domain name are `images.johnsmith.net`, the CNAME record should alias to `images.johnsmith.net.s3.amazonaws.com`.

```
images.johnsmith.net CNAME      images.johnsmith.net.s3.amazonaws.com.
```

Setting the alias target to `s3.amazonaws.com` also works, but it may result in extra HTTP redirects.

Amazon S3 uses the host name to determine the bucket name. For example, suppose that you have configured `www.example.com` as a CNAME for `www.example.com.s3.amazonaws.com`. When you access `http://www.example.com`, Amazon S3 receives a request similar to the following:

```
GET / HTTP/1.1
Host: www.example.com
Date: date
Authorization: signatureValue
```

Because Amazon S3 sees only the original host name `www.example.com` and is unaware of the CNAME mapping used to resolve the request, the CNAME and the bucket name must be the same.

Any Amazon S3 endpoint can be used in a CNAME. For example, `s3-ap-southeast-1.amazonaws.com` can be used in CNAMEs. For more information about endpoints, see [Request Endpoints \(p. 13\)](#).

To associate a host name with an Amazon S3 bucket using CNAMEs

1. Select a host name that belongs to a domain you control. This example uses the `images` subdomain of the `johnsmith.net` domain.
2. Create a bucket that matches the host name. In this example, the host and bucket names are `images.johnsmith.net`.

Note

The bucket name must exactly match the host name.

3. Create a CNAME record that defines the host name as an alias for the Amazon S3 bucket. For example:

```
images.johnsmith.net CNAME images.johnsmith.net.s3.amazonaws.com
```

Important

For request routing reasons, the CNAME record must be defined exactly as shown in the preceding example. Otherwise, it might appear to operate correctly, but will eventually result in unpredictable behavior.

Note

The procedure for configuring DNS depends on your DNS server or DNS provider. For specific information, see your server documentation or contact your provider.

Limitations

Specifying the bucket for the request by using the HTTP `Host` header is supported for non-SSL requests and when using the REST API. You cannot specify the bucket in SOAP by using a different endpoint.

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

Backward Compatibility

Early versions of Amazon S3 incorrectly ignored the HTTP `Host` header. Applications that depend on this undocumented behavior must be updated to set the `Host` header correctly. Because Amazon S3 determines the bucket name from `Host` when it is present, the most likely symptom of this problem is to receive an unexpected `NoSuchBucket` error result code.

Request Redirection and the REST API

Topics

- [Redirects and HTTP User-Agents \(p. 53\)](#)
- [Redirects and 100-Continue \(p. 53\)](#)
- [Redirect Example \(p. 53\)](#)

This section describes how to handle HTTP redirects by using the Amazon S3 REST API. For general information about Amazon S3 redirects, see [Request Redirection and the REST API \(p. 509\)](#) in the Amazon Simple Storage Service API Reference.

Redirects and HTTP User-Agents

Programs that use the Amazon S3 REST API should handle redirects either at the application layer or the HTTP layer. Many HTTP client libraries and user agents can be configured to correctly handle redirects automatically; however, many others have incorrect or incomplete redirect implementations.

Before you rely on a library to fulfill the redirect requirement, test the following cases:

- Verify all HTTP request headers are correctly included in the redirected request (the second request after receiving a redirect) including HTTP standards such as Authorization and Date.
- Verify non-GET redirects, such as PUT and DELETE, work correctly.
- Verify large PUT requests follow redirects correctly.
- Verify PUT requests follow redirects correctly if the 100-continue response takes a long time to arrive.

HTTP user-agents that strictly conform to RFC 2616 might require explicit confirmation before following a redirect when the HTTP request method is not GET or HEAD. It is generally safe to follow redirects generated by Amazon S3 automatically, as the system will issue redirects only to hosts within the amazonaws.com domain and the effect of the redirected request will be the same as that of the original request.

Redirects and 100-Continue

To simplify redirect handling, improve efficiencies, and avoid the costs associated with sending a redirected request body twice, configure your application to use 100-continues for PUT operations. When your application uses 100-continue, it does not send the request body until it receives an acknowledgement. If the message is rejected based on the headers, the body of the message is not sent. For more information about 100-continue, go to [RFC 2616 Section 8.2.3](#)

Note

According to RFC 2616, when using `Expect: Continue` with an unknown HTTP server, you should not wait an indefinite period before sending the request body. This is because some HTTP servers do not recognize 100-continue. However, Amazon S3 does recognize if your request contains an `Expect: Continue` and will respond with a provisional 100-continue status or a final status code. Additionally, no redirect error will occur after receiving the provisional 100 continue go-ahead. This will help you avoid receiving a redirect response while you are still writing the request body.

Redirect Example

This section provides an example of client-server interaction using HTTP redirects and 100-continue.

Following is a sample PUT to the quotes.s3.amazonaws.com bucket.

```
PUT /nelson.txt HTTP/1.1
Host: quotes.s3.amazonaws.com
Date: Mon, 15 Oct 2007 22:18:46 +0000

Content-Length: 6
Expect: 100-continue
```

Amazon S3 returns the following:

```
HTTP/1.1 307 Temporary Redirect
Location: http://quotes.s3-4c25d83b.amazonaws.com/nelson.txt?rk=8d47490b
Content-Type: application/xml
Transfer-Encoding: chunked
Date: Mon, 15 Oct 2007 22:18:46 GMT

Server: AmazonS3

<?xml version="1.0" encoding="UTF-8"?>
<Error>
<Code>TemporaryRedirect</Code>
<Message>Please re-send this request to the
specified temporary endpoint. Continue to use the
original request endpoint for future requests.
</Message>
<Endpoint>quotes.s3-4c25d83b.amazonaws.com</Endpoint>
<Bucket>quotes</Bucket>
</Error>
```

The client follows the redirect response and issues a new request to the quotes.s3-4c25d83b.amazonaws.com temporary endpoint.

```
PUT /nelson.txt?rk=8d47490b HTTP/1.1
Host: quotes.s3-4c25d83b.amazonaws.com
Date: Mon, 15 Oct 2007 22:18:46 +0000

Content-Length: 6
Expect: 100-continue
```

Amazon S3 returns a 100-continue indicating the client should proceed with sending the request body.

```
HTTP/1.1 100 Continue
```

The client sends the request body.

```
ha ha\n
```

Amazon S3 returns the final response.

```
HTTP/1.1 200 OK
Date: Mon, 15 Oct 2007 22:18:48 GMT

ETag: "a2c8d6b872054293afd41061e93bc289"
Content-Length: 0
Server: AmazonS3
```

Working with Amazon S3 Buckets

Amazon S3 is cloud storage for the Internet. To upload your data (photos, videos, documents etc.), you first create a bucket in one of the AWS regions. You can then upload any number of objects to the bucket.

In terms of implementation, buckets and objects are resources, and Amazon S3 provides APIs for you to manage them. For example, you can create a bucket and upload objects using the Amazon S3 API. You can also use the Amazon S3 console to perform these operations. The console internally uses the Amazon S3 APIs to send requests to Amazon S3.

In this section, we explain working with buckets. For information about working with objects, see [Working with Amazon S3 Objects \(p. 78\)](#).

Amazon S3 bucket names are globally unique, regardless of the AWS region in which you create the bucket. You specify the name at the time you create the bucket. For bucket naming guidelines, see [Bucket Restrictions and Limitations \(p. 59\)](#).

Amazon S3 creates bucket in a region you specify. You can choose any AWS region that is geographically close to you to optimize latency, minimize costs, or address regulatory requirements. For example, if you reside in Europe, you might find it advantageous to create buckets in the EU (Ireland) region. For a list of AWS region, go to [Regions and Endpoints](#) in the [AWS General Reference](#).

Note

Objects belonging to a bucket that you create in a specific AWS region never leave that region, unless you explicitly transfer them to another region. For example, objects stored in the EU (Ireland) region never leave it.

Creating a Bucket

Amazon S3 provides APIs for you to create and manage buckets. You can create up to 100 buckets for each AWS account you own. When you create a bucket, you provide a name and AWS region where you want the bucket created. The next section (see [Accessing a Bucket \(p. 57\)](#)) discusses DNS-compliant bucket names.

Within each bucket, you can store any number of objects. You can create a bucket using any of the following methods:

- Create the bucket using the console.
- Create the bucket programmatically using the AWS SDKs.

Note

If you need to, you can also make the Amazon S3 REST API calls directly from your code. However, this can be cumbersome because it requires you to write code to authenticate your requests. For more information, go to [PUT Bucket](#) in the *Amazon Simple Storage Service API Reference*.

When using AWS SDKs you first create a client and then send a request to create a bucket using the client. You can specify an AWS region when you create the client (US Standard is the default region). You can also specify a region in your create bucket request. Note the following:

- If you create a client by specifying the US Standard region, it uses the following endpoint to communicate with Amazon S3.

```
s3.amazonaws.com
```

You can use this client to create a bucket in any AWS region. In your create bucket request,

- If you don't specify a region, Amazon S3 creates the bucket in the US Standard region.
- If you specify an AWS region, Amazon S3 creates the bucket in the specified region.
- If you create a client by specifying any other AWS region, each of these regions maps to the region-specific endpoint:

```
s3-<region>.amazonaws.com
```

For example, if you create a client by specifying the eu-west-1 region, it maps to the following region-specific endpoint:

```
s3-eu-west-1.amazonaws.com
```

In this case, you can use the client to create a bucket only in the eu-west-1 region. Amazon S3 returns an error if you specify any other region in your create bucket request.

For a list of available AWS regions, go to [Regions and Endpoints](#) in the *AWS General Reference*.

For examples, see [Examples of Creating a Bucket \(p. 60\)](#).

About Permissions

You can use your AWS account root credentials to create a bucket and perform any other Amazon S3 operation. However, AWS recommends not using the root credentials of your AWS account to make requests such as create a bucket. Instead, create an IAM user, and grant that user full access (users by default have no permissions). We refer to these users as administrator users. You can use the administrator user credentials, instead of the root credentials of your account, to interact with AWS and perform tasks, such as create a bucket, create users, and grant them permissions.

For more information, go to [Root Account Credentials vs. IAM User Credentials](#) in the *AWS General Reference* and [IAM Best Practices](#) in *Using IAM*.

The AWS account that creates a resource owns that resource. For example, if you create an IAM user in your AWS account and grant the user permission to create a bucket, the user can create a bucket. But the user does not own the bucket; the AWS account to which the user belongs owns the bucket. The user will need additional permission from the resource owner to perform any other bucket operations. For more information about managing permissions for your Amazon S3 resources, see [Managing Access Permissions to Your Amazon S3 Resources \(p. 269\)](#).

Accessing a Bucket

You can access your bucket using the Amazon S3 console. Using the console UI, you can perform almost all bucket operations without having to write any code.

If you access a bucket programmatically, note that Amazon S3 supports RESTful architecture in which your buckets and objects are resources, each with a resource URI that uniquely identify the resource.

Amazon S3 supports both virtual-hosted-style and path-style URLs to access a bucket.

- In a virtual-hosted-style URL, the bucket name is part of the domain name in the URL. For example:

- `http://bucket.s3.amazonaws.com`
- `http://bucket.s3-aws-region.amazonaws.com`.

In a virtual-hosted-style URL, you can use either of these endpoints. If you make a request to the `http://bucket.s3.amazonaws.com` endpoint, the DNS has sufficient information to route your request directly to the region where your bucket resides.

- In a path-style URL, the bucket name is not part of the domain (unless you use a region-specific endpoint). For example:

- US Standard endpoint, `http://s3.amazonaws.com/bucket`
- Region-specific endpoint, `http://s3-aws-region.amazonaws.com/bucket`

In a path-style URL, the endpoint you use must match the region in which the bucket resides. For example, if your bucket is in the São Paulo region, you must use the

`http://s3-sa-east-1.amazonaws.com/bucket` endpoint. If your bucket is in the US Standard region, you must use the `http://s3.amazonaws.com/bucket` endpoint.

Important

Because buckets can be accessed using path-style and virtual-hosted-style URLs, we recommend you create buckets with DNS-compliant bucket names. For more information, see [Bucket Restrictions and Limitations \(p. 59\)](#).

For more information, see [Virtual Hosting of Buckets \(p. 48\)](#).

Bucket Configuration Options

Amazon S3 supports various options for you to configure your bucket. For example, you can configure your bucket for website hosting, add configuration to manage lifecycle of objects in the bucket, and configure the bucket to log all access to the bucket. Amazon S3 supports subresources for you store, and manage the bucket configuration information. That is, using the Amazon S3 API, you can create and manage these subresources. You can also use the console or the AWS SDKs.

Note

There are also object-level configurations. For example, you can configure object-level permissions by configuring an access control list (ACL) specific to that object.

These are referred to as subresources because they exist in the context of a specific bucket or object. The following table lists subresources that enable you to manage bucket-specific configurations.

Subresource	Description
<i>lifecycle</i>	You can define lifecycle rules for objects in your bucket that have a well-defined lifecycle. For example, you can define a rule to archive objects one year after creation, or delete an object 10 years after creation. For more information, see Object Lifecycle Management .
<i>website</i>	You can configure your bucket for static website hosting. Amazon S3 stores this configuration by creating a <i>website</i> subresource. For more information, see Hosting a Static Website on Amazon S3 .
<i>versioning</i>	Versioning helps you recover accidental overwrites and deletes. We recommend versioning as a best practice to recover objects from being deleted or overwritten by mistake. For more information, see Using Versioning (p. 422) .
<i>policy and ACL</i> (Access Control List)	All your resources (such as buckets and objects) are private by default. Amazon S3 supports both bucket policy and access control list (ACL) options for you to grant and manage bucket-level permissions. Amazon S3 stores the permission information in the <i>policy</i> and <i>acl</i> subresources. For more information, see Managing Access Permissions to Your Amazon S3 Resources (p. 269) .
<i>cors</i> (cross-origin resource sharing)	You can configure your bucket to allow cross-origin requests. For more information, see Enabling Cross-Origin Resource Sharing .
<i>logging</i>	Logging enables you to track requests for access to your bucket. Each access log record provides details about a single access request, such as the requester, bucket name, request time, request action, response status, and error code, if any. Access log information can be useful in security and access audits. It can also help you learn about your customer base and understand your Amazon S3 bill. For more information, see Server Access Logging (p. 530) .
<i>tagging</i>	You can add cost allocation tags to your bucket to categorize and track your AWS costs. Amazon S3 provides the <i>tagging</i> subresource to store and manage tags on a bucket. Using tags you apply to your bucket, AWS generates a cost allocation report with usage and costs aggregated by your tags. For more information, see Billing and Reporting of Buckets (p. 77) .
<i>location</i>	When you create a bucket, you specify the AWS region where you want Amazon S3 to create the bucket. Amazon S3 stores this information in the location subresource and provides an API for you to retrieve this information.
<i>notification</i>	You can enable your bucket to send you notifications of specified bucket events. For more information, see Configuring Amazon S3 Event Notifications (p. 472) .
<i>versions</i>	Amazon S3 stores object version information in this subresource.

Subresource	Description
<i>requestPayment</i>	By default, the AWS account that creates the bucket (the bucket owner) pays for downloads from the bucket. Using this subresource, the bucket owner can specify that the person requesting the download will be charged for the download. Amazon S3 provides an API for you to manage this subresource. For more information, see Requester Pays Buckets (p. 73).

Bucket Restrictions and Limitations

A bucket is owned by the AWS account that created it. Each AWS account can own up to 100 buckets at a time. Bucket ownership is not transferable; however, if a bucket is empty, you can delete it. After a bucket is deleted, the name becomes available to reuse, but the name might not be available for you to reuse for various reasons. For example, some other account could create a bucket with that name. Note, too, that it might take some time before the name can be reused. So if you want to use the same bucket name, don't delete the bucket.

There is no limit to the number of objects that can be stored in a bucket and no difference in performance whether you use many buckets or just a few. You can store all of your objects in a single bucket, or you can organize them across several buckets.

You cannot create a bucket within another bucket.

The high-availability engineering of Amazon S3 is focused on get, put, list, and delete operations. Because bucket operations work against a centralized, global resource space, it is not appropriate to create or delete buckets on the high-availability code path of your application. It is better to create or delete buckets in a separate initialization or setup routine that you run less often.

Note

If your application automatically creates buckets, choose a bucket naming scheme that is unlikely to cause naming conflicts. Ensure that your application logic will choose a different bucket name if a bucket name is already taken.

Rules for Bucket Naming

We recommend that all bucket names comply with DNS naming conventions. These conventions are enforced in all regions except for the US Standard region.

Note

If you use the AWS management console, bucket names must be DNS compliant in all regions.

DNS-compliant bucket names allow customers to benefit from new features and operational improvements, as well as providing support for virtual-host style access to buckets. While the US Standard region currently allows non-compliant DNS bucket naming, we are moving to the same DNS-compliant bucket naming convention for the US Standard region in the coming months. This will ensure a single, consistent naming approach for Amazon S3 buckets. The rules for DNS-compliant bucket names are:

- Bucket names must be at least 3 and no more than 63 characters long.
- Bucket names must be a series of one or more labels. Adjacent labels are separated by a single period (.). Bucket names can contain lowercase letters, numbers, and hyphens. Each label must start and end with a lowercase letter or a number.
- Bucket names must not be formatted as an IP address (e.g., 192.168.5.4).

The following examples are valid bucket names:

- myawsbucket
- my.aws.bucket
- myawsbucket.1

The following examples are invalid bucket names:

Invalid Bucket Name	Comment
.myawsbucket	Bucket name cannot start with a period (.).
myawsbucket.	Bucket name cannot end with a period (.).
my..examplebucket	There can be only one period between labels.

Challenges with Non–DNS–Compliant Bucket Names

The US Standard region currently allows more relaxed standards for bucket naming, which can result in a bucket name that is not DNS-compliant. For example, `MyAWSBucket` is a valid bucket name, even though it contains uppercase letters. If you try to access this bucket by using a virtual-hosted-style request (`http://MyAWSBucket.s3.amazonaws.com/yourobject`), the URL resolves to the bucket `myawsbucket` and not the bucket `MyAWSBucket`. In response, Amazon S3 will return a "bucket not found" error.

To avoid this problem, we recommend as a best practice that you always use DNS-compliant bucket names regardless of the region in which you create the bucket. For more information about virtual-hosted-style access to your buckets, see [Virtual Hosting of Buckets \(p. 48\)](#).

The rules for bucket names in the US Standard region allow bucket names to be as long as 255 characters, and bucket names can contain any combination of uppercase letters, lowercase letters, numbers, periods (.), hyphens (-), and underscores (_).

Examples of Creating a Bucket

Topics

- [Using the Amazon S3 Console \(p. 61\)](#)
- [Using the AWS SDK for Java \(p. 61\)](#)
- [Using the AWS SDK for .NET \(p. 62\)](#)
- [Using Other AWS SDKs \(p. 63\)](#)

This section provides code examples of creating a bucket programmatically using the AWS SDKs for Java and .NET. The code examples perform the following tasks:

- Create a bucket if it does not exist — The examples create a bucket as follows:
 - Create a client by explicitly specifying an AWS region (example uses the `s3-eu-west-1` region). Accordingly, the client communicates with Amazon S3 using the `s3-eu-west-1.amazonaws.com` endpoint. You can specify any other AWS region. For a list of available AWS regions, go to [Regions and Endpoints](#) in the [AWS General Reference](#).
 - Send a create bucket request by specifying only a bucket name. The create bucket request does not specify another AWS region; therefore, the client sends a request to Amazon S3 to create the bucket in the region you specified when creating the client.

Note

If you specify a region in your create bucket request that conflicts with the region you specify when you create the client, you might get an error. For more information, see [Creating a Bucket \(p. 55\)](#).

The SDK libraries send the PUT bucket request to Amazon S3 (see [PUT Bucket](#)) to create the bucket.

- Retrieve bucket location information — Amazon S3 stores bucket location information in the *location* subresource associated with the bucket. The SDK libraries send the GET Bucket location request (see [GET Bucket location](#)) to retrieve this information

Using the Amazon S3 Console

For creating a bucket using Amazon S3 console, go to [Creating a Bucket](#) in the *Amazon Simple Storage Service Console User Guide*.

Using the AWS SDK for Java

For instructions on how to create and test a working sample, see [Testing the Java Code Examples \(p. 545\)](#).

```
import java.io.IOException;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
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.CreateBucketRequest;
import com.amazonaws.services.s3.model.GetBucketLocationRequest;

public class CreateBucket {
    private static String bucketName      = "**** bucket name ****";

    public static void main(String[] args) throws IOException {
        AmazonS3 s3client = new AmazonS3Client(new ProfileCredentialsProvider());

        s3client.setRegion(Region.getRegion(Regions.US_WEST_1));

        try {
            if(!(s3client.doesBucketExist(bucketName)))
            {
                // Note that CreateBucketRequest does not specify region. So
                // bucket is
                // created in the region specified in the client.
                s3client.createBucket(new CreateBucketRequest(
                    bucketName));
            }
            // Get location.
            String bucketLocation = s3client.getBucketLocation(new GetBucket
                LocationRequest(bucketName));
            System.out.println("bucket location = " + bucketLocation);

        } catch (AmazonServiceException ase) {
```

```
        System.out.println("Caught an AmazonServiceException, which " +
            "means your request made it " +
            "to Amazon S3, 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 " +
            "an internal error while trying to " +
            "communicate with S3, " +
            "such as not being able to access the network.");
        System.out.println("Error Message: " + ace.getMessage());
    }
}
}
```

Using the AWS SDK for .NET

For information about how to create and test a working sample, see [Testing the .NET Code Examples \(p. 547\)](#).

```
using System;
using Amazon.S3;
using Amazon.S3.Model;
using Amazon.S3.Util;

namespace s3.amazonaws.com.docsamples
{
    class CreateBucket
    {
        static string bucketName = "**** bucket name ****";

        public static void Main(string[] args)
        {
            using (var client = new AmazonS3Client(Amazon.RegionEndpoint.EUWest1))
            {

                if (!(AmazonS3Util.DoesS3BucketExist(client, bucketName)))
                {
                    CreateABucket(client);
                }
                // Retrieve bucket location.
                string bucketLocation = FindBucketLocation(client);
            }

            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }

        static string FindBucketLocation(IAmazonS3 client)
        {
```

```
        string bucketLocation;
        GetBucketLocationRequest request = new GetBucketLocationRequest()
        {
            BucketName = bucketName
        };
        GetBucketLocationResponse response = client.GetBucketLocation(re
quest);
        bucketLocation = response.Location.ToString();
        return bucketLocation;
    }

    static void CreateABucket(IAmazonS3 client)
    {
        try
        {
            PutBucketRequest putRequest1 = new PutBucketRequest
            {
                BucketName = bucketName,
                UseClientRegion = true
            };

            PutBucketResponse response1 = client.PutBucket(putRequest1);
        }
        catch (AmazonS3Exception amazonS3Exception)
        {
            if (amazonS3Exception.ErrorCode != null &&
                (amazonS3Exception.ErrorCode.Equals("InvalidAccessKeyId") ||
                amazonS3Exception.ErrorCode.Equals("InvalidSecurity")))
            {
                Console.WriteLine("Check the provided AWS Credentials.");
                Console.WriteLine(
                    "For service sign up go to http://aws.amazon.com/s3");
            }
            else
            {
                Console.WriteLine(
                    "Error occurred. Message:{0}' when writing an object"
                    , amazonS3Exception.Message);
            }
        }
    }
}
```

Using Other AWS SDKs

For information about using other AWS SDKs, go to [Sample Code and Libraries](#).

Managing Bucket Website Configuration

Topics

- Managing Websites with the AWS Management Console (p. 64)
- Managing Websites with the AWS SDK for Java (p. 64)
- Managing Websites with the AWS SDK for .NET (p. 67)
- Managing Websites with the AWS SDK for PHP (p. 70)
- Managing Websites with the REST API (p. 72)

You can host static websites in Amazon S3 by configuring your bucket for website hosting. For more information, see [Hosting a Static Website on Amazon S3 \(p. 448\)](#). There are several ways you can manage your bucket's website configuration. You can use the AWS Management Console to manage configuration without writing any code. You can programmatically create, update, and delete the website configuration by using the AWS SDKs. The SDKs provide wrapper classes around the Amazon S3 REST API. If your application requires it, you can send REST API requests directly from your application.

Managing Websites with the AWS Management Console

For more information, see [Configure a Bucket for Website Hosting \(p. 451\)](#).

Managing Websites with the AWS SDK for Java

The following tasks guide you through using the Java classes to manage website configuration to your bucket. For more information about the Amazon S3 website feature, see [Hosting a Static Website on Amazon S3 \(p. 448\)](#).

Managing Website Configuration

1	Create an instance of the <code>AmazonS3</code> class.
2	To add website configuration to a bucket, execute the <code>AmazonS3.setBucketWebsiteConfiguration</code> method. You need to provide the bucket name and the website configuration information, including the index document and the error document names. You must provide the index document, but the error document is optional. You provide website configuration information by creating a <code>BucketWebsiteConfiguration</code> object. To retrieve website configuration, execute the <code>AmazonS3.getBucketWebsiteConfiguration</code> method by providing the bucket name. To delete your bucket website configuration, execute the <code>AmazonS3.deleteBucketWebsiteConfiguration</code> method by providing the bucket name. After you remove the website configuration, the bucket is no longer available from the website endpoint. For more information, see Website Endpoints (p. 449) .

The following Java code sample demonstrates the preceding tasks.

```
AmazonS3 s3client = new AmazonS3Client(new ProfileCredentialsProvider());
// Add website configuration.
s3Client.setBucketWebsiteConfiguration(bucketName,
    new BucketWebsiteConfiguration(indexDoc , errorDoc));

// Get website configuration.
BucketWebsiteConfiguration bucketWebsiteConfiguration =
    s3Client.getBucketWebsiteConfiguration(bucketName);
```

```
// Delete website configuration.  
s3Client.deleteBucketWebsiteConfiguration(bucketName);
```

Example

The following Java code example adds a website configuration to the specified bucket, retrieves it, and deletes the website configuration. For instructions on how to create and test a working sample, see [Testing the Java Code Examples \(p. 545\)](#).

```
import java.io.IOException;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.BucketWebsiteConfiguration;

public class WebsiteConfiguration {

    private static String bucketName = "**** bucket name ****";
    private static String indexDoc    = "**** index document name ****";
    private static String errorDoc   = "**** error document name ****";

    public static void main(String[] args) throws IOException {
        AmazonS3 s3Client = new AmazonS3Client(new ProfileCredentialsProvider());

        try {
            // Get existing website configuration, if any.
            getWebsiteConfig(s3Client);

            // Set new website configuration.
            s3Client.setBucketWebsiteConfiguration(bucketName,
                new BucketWebsiteConfiguration(indexDoc, errorDoc));

            // Verify (Get website configuration again).
            getWebsiteConfig(s3Client);

            // Delete
            s3Client.deleteBucketWebsiteConfiguration(bucketName);

            // Verify (Get website configuration again)
            getWebsiteConfig(s3Client);

        } catch (AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException, which" +
                " means your request made it " +
                "to Amazon S3, 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 Amazon S3, " +
        "such as not being able to access the network." );
    System.out.println("Error Message: " + ace.getMessage());
}
}

private static BucketWebsiteConfiguration getWebsiteConfig(
    AmazonS3 s3Client) {
System.out.println("Get website config");

// 1. Get website config.
BucketWebsiteConfiguration bucketWebsiteConfiguration =
    s3Client.getBucketWebsiteConfiguration(bucketName);
if (bucketWebsiteConfiguration == null)
{
    System.out.println("No website config.");
}
else
{
    System.out.println("Index doc:" +
        bucketWebsiteConfiguration.getIndexDocumentSuffix());
    System.out.println("Error doc:" +
        bucketWebsiteConfiguration.getErrorDocument());
}
return bucketWebsiteConfiguration;
}
}
```

Managing Websites with the AWS SDK for .NET

The following tasks guide you through using the .NET classes to manage website configuration on your bucket. For more information about the Amazon S3 website feature, see [Hosting a Static Website on Amazon S3 \(p. 448\)](#).

Managing Bucket Website Configuration

1	Create an instance of the <code>AmazonS3Client</code> class.
2	To add website configuration to a bucket, execute the <code>PutBucketWebsite</code> method. You need to provide the bucket name and the website configuration information, including the index document and the error document names. You must provide the index document, but the error document is optional. You provide this information by creating a <code>PutBucketWebsiteRequest</code> object. To retrieve website configuration, execute the <code>GetBucketWebsite</code> method by providing the bucket name. To delete your bucket website configuration, execute the <code>DeleteBucketWebsite</code> method by providing the bucket name. After you remove the website configuration, the bucket is no longer available from the website endpoint. For more information, see Website Endpoints (p. 449) .

The following C# code sample demonstrates the preceding tasks.

```
static IAmazonS3 client;
client = new AmazonS3Client(Amazon.RegionEndpoint.USWest2);

// Add website configuration.
PutBucketWebsiteRequest putRequest = new PutBucketWebsiteRequest()
{
    BucketName = bucketName,
    WebsiteConfiguration = new WebsiteConfiguration()
    {
        IndexDocumentSuffix = indexDocumentSuffix,
        ErrorDocument = errorDocument
    }
};
client.PutBucketWebsite(putRequest);

// Get bucket website configuration.
GetBucketWebsiteRequest getRequest = new GetBucketWebsiteRequest()
{
    BucketName = bucketName
};

GetBucketWebsiteResponse getResponse = client.GetBucketWebsite(getRequest);

// Print configuration data.
Console.WriteLine("Index document: {0}", getResponse.WebsiteConfiguration.Index
DocumentSuffix);
Console.WriteLine("Error document: {0}", getResponse.WebsiteConfiguration.Error
Document);

// Delete website configuration.
DeleteBucketWebsiteRequest deleteRequest = new DeleteBucketWebsiteRequest()
{
    BucketName = bucketName
};
client.DeleteBucketWebsite(deleteRequest);
```

Example

The following C# code example adds a website configuration to the specified bucket. The configuration specifies both the index document and the error document names. For instructions on how to create and test a working sample, see [Testing the .NET Code Examples \(p. 547\)](#).

```
using System;
using System.Configuration;
using System.Collections.Specialized;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples
{
    class AddWebsiteConfig
    {
        static string bucketName           = "**** Provide existing bucket name ****";
        static string indexDocumentSuffix = "**** Provide index document name ****";
        static string errorDocument       = "**** Provide error document name ****";
        static IAmazonS3 client;

        public static void Main(string[] args)
        {
            using (client = new AmazonS3Client(Amazon.RegionEndpoint.USWest2))
            {
                Console.WriteLine("Adding website configuration");
                AddWebsiteConfiguration(bucketName, indexDocumentSuffix, errorDocument);
            }

            // Get bucket website configuration.
            GetBucketWebsiteRequest getRequest = new GetBucketWebsiteRequest()
            {
                BucketName = bucketName
            };

            GetBucketWebsiteResponse getResponse = client.GetBucketWebsite(getRequest);
            // Print configuration data.
            Console.WriteLine("Index document: {0}", getResponse.WebsiteConfiguration.IndexDocumentSuffix);
            Console.WriteLine("Error document: {0}", getResponse.WebsiteConfiguration.ErrorDocument);

            Console.WriteLine("Press any key to continue... ");
            Console.ReadKey();
        }

        static void AddWebsiteConfiguration(string bucketName,
                                            string indexDocumentSuffix,
                                            string errorDocument)
        {
            try
```

```
        {
            PutBucketWebsiteRequest putRequest = new PutBucketWebsite
Request()
        {
            BucketName = bucketName,
            WebsiteConfiguration = new WebsiteConfiguration()
            {
                IndexDocumentSuffix = indexDocumentSuffix,
                ErrorDocument = errorDocument
            }
        };
        client.PutBucketWebsite(putRequest);
    }
    catch (AmazonS3Exception amazonS3Exception)
    {
        if (amazonS3Exception.ErrorCode != null &&
            (amazonS3Exception.ErrorCode.Equals("InvalidAccessKeyId") ||
             amazonS3Exception.ErrorCode.Equals("InvalidSecurity")))
        {
            Console.WriteLine("Check the provided AWS Credentials.");
            Console.WriteLine("Sign up for service at ht
tp://aws.amazon.com/s3");
        }
        else
        {
            Console.WriteLine(
                "Error:{0}, occurred when adding website configuration.
Message:{1}",
                amazonS3Exception.ErrorCode, amazonS3Exception.Message);
        }
    }
}
}
```

Managing Websites with the AWS SDK for PHP

This topic guides you through using classes from the AWS SDK for PHP to configure and manage an Amazon S3 bucket for website hosting. For more information about the Amazon S3 website feature, see [Hosting a Static Website on Amazon S3 \(p. 448\)](#).

Note

This topic assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 547\)](#) and have the AWS SDK for PHP properly installed.

The following tasks guide you through using the PHP SDK classes to configure and manage an Amazon S3 bucket for website hosting.

Configuring a Bucket for Website Hosting

- | | |
|---|--|
| 1 | Create an instance of an Amazon S3 client by using the Aws\S3\S3Client class factory() method. |
|---|--|

2	To configure a bucket as a website, execute the Aws\S3\S3Client::putBucketWebsite() method. You need to provide the bucket name and the website configuration information, including the index document and the error document names. If you don't provide these document names, this method adds the <code>index.html</code> and <code>error.html</code> default names to the website configuration. You must verify that these documents are present in the bucket.
3	To retrieve existing bucket website configuration, execute the Aws\S3\S3Client::getBucketWebsite() method.
4	To delete website configuration from a bucket, execute the Aws\S3\S3Client::deleteBucketWebsite() method, passing the bucket name as a parameter. If you remove the website configuration, the bucket is no longer accessible from the website endpoints.

The following PHP code sample demonstrates the preceding tasks.

```
use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';

// 1. Instantiate the client.
$s3 = S3Client::factory();

// 2. Add website configuration.
$result = $s3->putBucketWebsite(array(
    'Bucket'      => $bucket,
    'IndexDocument' => array('Suffix' => 'index.html'),
    'ErrorDocument' => array('Key' => 'error.html'),
));

// 3. Retrieve website configuration.
$result = $s3->getBucketWebsite(array(
    'Bucket' => $bucket,
));
echo $result->getPath('IndexDocument/Suffix');

// 4.) Delete website configuration.
$result = $s3->deleteBucketWebsite(array(
    'Bucket' => $bucket,
));
```

Example of Configuring an Bucket Amazon S3 for Website Hosting

The following PHP code example first adds a website configuration to the specified bucket. The `create_website_config` method explicitly provides the index document and error document names. The sample also retrieves the website configuration and prints the response. For more information about the Amazon S3 website feature, see [Hosting a Static Website on Amazon S3 \(p. 448\)](#).

For instructions on how to create and test a working sample, see [Using the AWS SDK for PHP and Running PHP Examples \(p. 547\)](#).

```
<?php

// Include the AWS SDK using the Composer autoloader.
require 'vendor/autoload.php';

use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';

// Instantiate the client.
$s3 = S3Client::factory();

// 1.) Add website configuration.
$result = $s3->putBucketWebsite(array(
    'Bucket'          => $bucket,
    'IndexDocument'  => array('Suffix' => 'index.html'),
    'ErrorDocument'   => array('Key' => 'error.html'),
));

// 2.) Retrieve website configuration.
$result = $s3->getBucketWebsite(array(
    'Bucket' => $bucket,
));
echo $result->getPath('IndexDocument/Suffix');

// 3.) Delete website configuration.
$result = $s3->deleteBucketWebsite(array(
    'Bucket' => $bucket,
));
```

Related Resources

- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::deleteBucketWebsite\(\) Method](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::factory\(\) Method](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::getBucketWebsite\(\) Method](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::putBucketWebsite\(\) Method](#)
- [AWS SDK for PHP for Amazon S3](#)
- [AWS SDK for PHP Documentation](#)

Managing Websites with the REST API

Topics

You can use the AWS Management Console or the AWS SDK to configure a bucket as a website. However, if your application requires it, you can send REST requests directly. For more information, see the following sections in the Amazon Simple Storage Service API Reference.

- [PUT Bucket website](#)
- [GET Bucket website](#)
- [DELETE Bucket website](#)

Requester Pays Buckets

Topics

- [Configure Requester Pays by Using the Amazon S3 Console \(p. 73\)](#)
- [Configure Requester Pays with the REST API \(p. 74\)](#)
- [DevPay and Requester Pays \(p. 76\)](#)
- [Charge Details \(p. 76\)](#)

In general, bucket owners pay for all Amazon S3 storage and data transfer costs associated with their bucket. A bucket owner, however, can configure a bucket to be a Requester Pays bucket. With Requester Pays buckets, the requester instead of the bucket owner pays the cost of the request and the data download from the bucket. The bucket owner always pays the cost of storing data.

Typically, you configure buckets to be Requester Pays when you want to share data but not incur charges associated with others accessing the data. You might, for example, use Requester Pays buckets when making available large data sets, such as zip code directories, reference data, geospatial information, or web crawling data.

Important

If you enable Requester Pays on a bucket, anonymous access to that bucket is not allowed.

You must authenticate all requests involving Requester Pays buckets. The request authentication enables Amazon S3 to identify and charge the requester for their use of the Requester Pays bucket.

After you configure a bucket to be a Requester Pays bucket, requesters must include `x-amz-request-payer` in their requests either in the header, for POST and GET requests, or as a parameter in a REST request to show that they understand that they will be charged for the request and the data download.

Requester Pays buckets do not support the following.

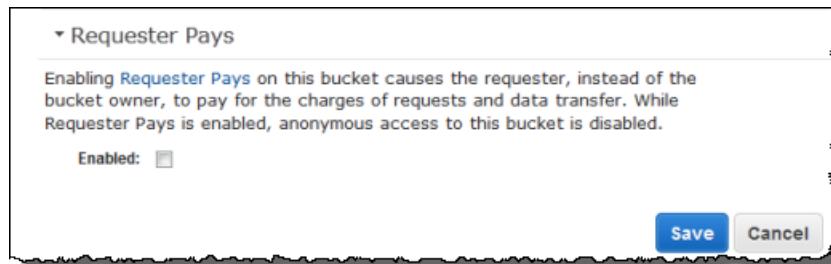
- Anonymous requests
- BitTorrent
- SOAP requests
- You cannot use a Requester Pays bucket as the target bucket for end user logging, or vice versa; however, you can turn on end user logging on a Requester Pays bucket where the target bucket is not a Requester Pays bucket.

Configure Requester Pays by Using the Amazon S3 Console

You can configure a bucket for Requester Pays by using the Amazon S3 console.

To configure a bucket for Requester Pays

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, click the details icon on the left of the bucket name and then click **Properties** to display bucket properties.
3. In the **Properties** pane, click **Requester Pays**.
4. Select the **Enabled** check box.



Configure Requester Pays with the REST API

Topics

- [Setting the requestPayment Bucket Configuration \(p. 74\)](#)
- [Retrieving the requestPayment Configuration \(p. 75\)](#)
- [Downloading Objects in Requester Pays Buckets \(p. 76\)](#)

Setting the requestPayment Bucket Configuration

Only the bucket owner can set the `RequestPaymentConfiguration.payer` configuration value of a bucket to `BucketOwner`, the default, or `Requester`. Setting the `requestPayment` resource is optional. By default, the bucket is not a Requester Pays bucket.

To revert a Requester Pays bucket to a regular bucket, you use the value `BucketOwner`. Typically, you would use `BucketOwner` when uploading data to the Amazon S3 bucket, and then you would set the value to `Requester` before publishing the objects in the bucket.

To set requestPayment

- Use a `PUT` request to set the `Payer` value to `Requester` on a specified bucket.

```
PUT ?requestPayment HTTP/1.1
Host: [BucketName].s3.amazonaws.com
Content-Length: 173
Date: Wed, 01 Mar 2009 12:00:00 GMT
Authorization: AWS [Signature]

<RequestPaymentConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
<Payer>Requester</Payer>
</RequestPaymentConfiguration>
```

If the request succeeds, Amazon S3 returns a response similar to the following.

```
HTTP/1.1 200 OK
x-amz-id-2: [id]
x-amz-request-id: [request_id]
Date: Wed, 01 Mar 2009 12:00:00 GMT
Content-Length: 0
Connection: close
Server: AmazonS3
x-amz-request-charged:requester
```

You can set Requester Pays only at the bucket level; you cannot set Requester Pays for specific objects within the bucket.

You can configure a bucket to be BucketOwner or Requester at any time. Realize, however, that there might be a small delay, on the order of minutes, before the new configuration value takes effect.

Note

Bucket owners who give out pre-signed URLs should think twice before configuring a bucket to be Requester Pays, especially if the URL has a very long lifetime. The bucket owner is charged each time the requester uses a pre-signed URL that uses the bucket owner's credentials.

Retrieving the requestPayment Configuration

You can determine the *Payer* value that is set on a bucket by requesting the resource `requestPayment`.

To return the `requestPayment` resource

- Use a GET request to obtain the `requestPayment` resource, as shown in the following request.

```
GET ?requestPayment HTTP/1.1
Host: [BucketName].s3.amazonaws.com
Date: Wed, 01 Mar 2009 12:00:00 GMT
Authorization: AWS [Signature]
```

If the request succeeds, Amazon S3 returns a response similar to the following.

```
HTTP/1.1 200 OK
x-amz-id-2: [id]
x-amz-request-id: [request_id]
Date: Wed, 01 Mar 2009 12:00:00 GMT
Content-Type: [type]
Content-Length: [length]
Connection: close
Server: AmazonS3

<?xml version="1.0" encoding="UTF-8"?>
<RequestPaymentConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
<Payer>Requester</Payer>
</RequestPaymentConfiguration>
```

This response shows that the *payer* value is set to Requester.

Downloading Objects in Requester Pays Buckets

Because requesters are charged for downloading data from Requester Pays buckets, the requests must contain a special parameter, `x-amz-request-payer`, which confirms that the requester knows he or she will be charged for the download. To access objects in Requester Pays buckets, requests must include one of the following.

- For GET and POST requests, include `x-amz-request-payer : requester` in the header
- For signed URLs, include `x-amz-request-payer=requester` in the request

If the request succeeds and the requester is charged, the response includes the header `x-amz-request-charged:requester`. If `x-amz-request-payer` is not in the request, Amazon S3 returns a 403 error and charges the bucket owner for the request.

Note

Bucket owners do not need to add `x-amz-request-payer` to their requests.

Ensure that you have included `x-amz-request-payer` and its value in your signature calculation.

For more information, see [Constructing the CanonicalizedAmzHeaders Element \(p. 561\)](#).

To download objects from a Requester Pays bucket

- Use a GET request to download an object from a Requester Pays bucket, as shown in the following request.

```
GET / [destinationObject] HTTP/1.1
Host: [BucketName].s3.amazonaws.com
x-amz-request-payer : requester
Date: Wed, 01 Mar 2009 12:00:00 GMT
Authorization: AWS [Signature]
```

If the GET request succeeds and the requester is charged, the response includes `x-amz-request-charged:requester`.

Amazon S3 can return an Access Denied error for requests that try to get objects from a Requester Pays bucket. For more information, go to [Error Responses](#).

DevPay and Requester Pays

You can use Amazon DevPay to sell content that is stored in your Requester Pays bucket. For more information, go to "Using Amazon S3 Requester Pays with DevPay," in the [Using Amazon S3 Requester Pays with DevPay](#).

Charge Details

The charge for successful Requester Pays requests is straightforward: the requester pays for the data transfer and the request; the bucket owner pays for the data storage. However, the bucket owner is charged for the request under the following conditions:

- The requester doesn't include the parameter `x-amz-request-payer` in the header (GET or POST) or as a parameter (REST) in the request (HTTP code 403).
- Request authentication fails (HTTP code 403).
- The request is anonymous (HTTP code 403).
- The request is a SOAP request.

Buckets and Access Control

Each bucket has an associated access control policy. This policy governs the creation, deletion and enumeration of objects within the bucket. For more information, see [Managing Access Permissions to Your Amazon S3 Resources \(p. 269\)](#).

Billing and Reporting of Buckets

Fees for object storage and network data transfer are always billed to the owner of the bucket that contains the object unless the bucket was created as a Requester Pays bucket.

The reporting tools available at the AWS developer portal organize your Amazon S3 usage reports by bucket. For more information, go to [Amazon S3 Pricing page](#).

Cost Allocation Tagging

You can use cost allocation tagging to label Amazon S3 buckets so that you can more easily track their cost against projects or other criteria.

Use tags to organize your AWS bill to reflect your own cost structure. To do this, sign up to get your AWS account bill with tag key values included. Then, to see the cost of combined resources, organize your billing information according to resources with the same tag key values. For example, you can tag several resources with a specific application name, and then organize your billing information to see the total cost of that application across several services. For more information, see [Cost Allocation and Tagging in About AWS Billing and Cost Management](#).

A cost allocation tag is a name-value pair that you define and associate with an Amazon S3 bucket. We recommend that you use a consistent set of tag keys to make it easier to track costs associated with your Amazon S3 buckets.

Each Amazon S3 bucket has a tag set, which contains all the tags that are assigned to that bucket. A tag set can contain as many as ten tags, or it can be empty.

If you add a tag that has the same key as an existing tag on a bucket, the new value overwrites the old value.

AWS does not apply any semantic meaning to your tags: tags are interpreted strictly as character strings. AWS does not automatically set any tags on buckets.

You can use the Amazon S3 console, the CLI, or the Amazon S3 API to add, list, edit, or delete tags. For more information about creating tags in the console, go to [Managing Cost Allocation Tagging](#) in the *Amazon Simple Storage Service Console User Guide*.

The following list describes the characteristics of a cost allocation tag.

- The tag key is the required name of the tag. The string value can contain 1 to 128 Unicode characters. It cannot be prefixed with "aws:". The string can contain only the set of Unicode letters, digits, whitespace, '_', '.', '/', '=', '+', '-' (Java regex: " $^([\backslash p\{L\}\backslash p\{Z\}\backslash p\{N\}_:=+\backslash -])$$ ").
- The tag value is a required string value of the tag. The string value can contain from 1 to 256 Unicode characters. It cannot be prefixed with "aws:". The string can contain only the set of Unicode letters, digits, whitespace, '_', '.', '/', '=', '+', '-' (Java regex: " $^([\backslash p\{L\}\backslash p\{Z\}\backslash p\{N\}_:=+\backslash -])$$ ").

Values do not have to be unique in a tag set, and they can be null. For example, you can have the same key-value pair in tag sets named project/Trinity and cost-center/Trinity.

Working with Amazon S3 Objects

Amazon S3 is a simple key, value store designed to store as many objects as you want. You store these objects in one or more buckets. An object consists of the following:

- **Key** – The name that you assign to an object. You use the object key to retrieve the object. For more information, see [Object Key and Metadata \(p. 79\)](#)
- **Version ID** – Within a bucket, a key and version ID uniquely identify an object. The version ID is a string that Amazon S3 generates when you add an object to a bucket. For more information, see [Object Versioning \(p. 84\)](#).
- **Value** – The content that you are storing. An object value can be any sequence of bytes. Objects can range in size from zero to 5 TB. For more information, see [Uploading Objects \(p. 145\)](#).
- **Metadata** – A set of name-value pairs with which you can store information regarding the object. You can assign metadata, referred to as user-defined metadata, to your objects in Amazon S3. Amazon S3 also assigns system-metadata to these objects, which it uses for managing objects. For more information, see [Object Key and Metadata \(p. 79\)](#).
- **Subresources** – Amazon S3 uses the subresource mechanism to store object-specific additional information. Because subresources are subordinates to objects, they are always associated with some other entity such as an object or a bucket. For more information, see [Object Subresources \(p. 83\)](#).
- **Access Control Information** – You can control access to the objects you store in Amazon S3. Amazon S3 supports both the resource-based access control, such as an Access Control List (ACL) and bucket policies, and user-based access control. For more information, see [Managing Access Permissions to Your Amazon S3 Resources \(p. 269\)](#).

For more information about working with objects, see the following sections. Note that your Amazon S3 resources (for example buckets and objects) are private by default. You will need to explicitly grant permission for others to access these resources. For example, you might want to share a video or a photo stored in your Amazon S3 bucket on your website. That will work only if you either make the object public or use a presigned URL on your website. For more information about sharing objects, see [Share an Object with Others \(p. 139\)](#).

Topics

- [Object Key and Metadata \(p. 79\)](#)
- [Object Subresources \(p. 83\)](#)
- [Object Versioning \(p. 84\)](#)

- Object Lifecycle Management (p. 86)
- Enabling Cross-Origin Resource Sharing (p. 110)
- Operations on Objects (p. 125)
- Getting Objects (p. 130)
- Uploading Objects (p. 145)
- Copying Objects (p. 199)
- Listing Object Keys (p. 218)
- Deleting Objects (p. 228)
- Restoring Objects (p. 261)

Object Key and Metadata

Topics

- Object Keys (p. 79)
- Object Metadata (p. 81)

Each Amazon S3 object has data, a key, and metadata. Object key (or key name) uniquely identifies the object in a bucket. Object metadata is a set of name-value pairs. You can set object metadata at the time you upload it. After you upload the object, you cannot modify object metadata. The only way to modify object metadata is to make a copy of the object and set the metadata.

Object Keys

When you create an object, you specify the key name, which uniquely identifies the object in the bucket. For example, in the Amazon S3 console (see [AWS Management Console](#)), when you highlight a bucket, a list of objects in your bucket appears. These names are the object keys. The name for a key is a sequence of Unicode characters whose UTF-8 encoding is at most 1024 bytes long.

Note

If you anticipate that your workload against Amazon S3 will exceed 100 requests per second, follow the Amazon S3 key naming guidelines for best performance. For information, see [Request Rate and Performance Considerations \(p. 514\)](#).

Object Key Naming Guidelines

Although you can use any UTF-8 characters in an object key name, the following key naming best practices help ensure maximum compatibility with other applications. Each application may parse special characters differently. The following guidelines help you maximize compliance with DNS, web safe characters, XML parsers, and other APIs.

Safe Characters

The following character sets are generally safe for use in key names:

- Alphanumeric characters [0-9a-zA-Z]
- Special characters !, -, _, ., *, ', (, and)

The following are examples of valid object key names:

- 4my-organization
- my.great_photos-2014/jan/myvacation.jpg

- videos/2014/birthday/video1.wmv

Note that the Amazon S3 data model is a flat structure: you create a bucket, and the bucket stores objects. There is no hierarchy of subbuckets or subfolders; however, you can infer logical hierarchy using keyname prefixes and delimiters as the Amazon S3 console does. The Amazon S3 console supports a concept of folders. Suppose your bucket (`companybucket`) has four objects with the following object keys:

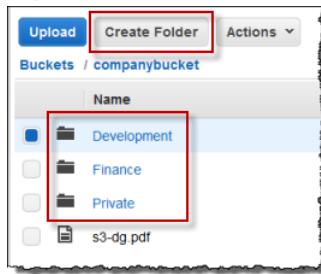
`Development/Projects1.xls`

`Finance/statement1.pdf`

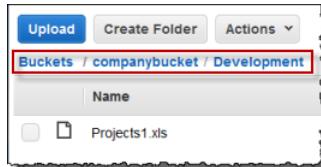
`Private/taxdocument.pdf`

`s3-dg.pdf`

The console uses the keyname prefixes (`Development/`, `Finance/`, and `Private/`) and delimiter (`/`) to present a folder structure as shown:



The `s3-dg.pdf` key does not have a prefix, so its object appears directly at the root level of the bucket. If you open the `Development/` folder, you will see the `Project1.xls` object in it.



Note

Amazon S3 supports buckets and objects, there is no hierarchy in Amazon S3. However, the prefixes and delimiters in an object key name, enables the Amazon S3 console and the AWS SDKs to infer hierarchy and introduce concept of folders.

Characters That Might Require Special Handling

The following characters in a key name may require additional code handling and will likely need to be URL encoded or referenced as HEX. Some of these are non-printable characters and your browser may not handle them, which will also require special handling:

Ampersand ("&")	Dollar ("\$")	ASCII character ranges 00–1F hex (0–31 decimal) and 7F (127 decimal.)
'At' symbol ("@")	Equals ("=")	Semicolon (";")
Colon (":")	Plus ("+")	Space – Significant sequences of spaces may be lost in some uses (especially multiple spaces)

Comma (",")	Question mark ("?")	
-------------	---------------------	--

Characters to Avoid

You should avoid the following characters in a key name because of significant special handling for consistency across all applications.

Backslash ("\")	Left curly brace ("{")	Non-printable ASCII characters (128–255 decimal characters)
Caret (^")	Right curly brace ("}")	Percent character ("%")
Grave accent / back tick (`")	Right square bracket ("]")	Quotation marks
'Greater Than' symbol (>")	Left square bracket ("[")	Tilde (~")
'Less Than' symbol (<")	'Pound' character (#")	Vertical bar / pipe (" ")

Object Metadata

There are two kinds of metadata: system metadata and user-defined metadata.

System-Defined Metadata

For each object stored in a bucket, Amazon S3 maintains a set of system metadata. Amazon S3 processes this system metadata as needed. For example, Amazon S3 maintains object creation date and size metadata and uses this information as part of object management.

There are two categories of system metadata:

- Metadata such as object creation date is system controlled where only Amazon S3 can modify the value.
- Other system metadata such as the storage class configured for the object and whether the object has server-side encryption enabled are examples of system metadata whose values you control. If you have your bucket configured as a website, sometimes you might want to redirect a page request to another page or an external URL. In this case, a web page is an object in your bucket. Amazon S3 stores the page redirect value as system metadata whose value you control.

When you create objects, you can configure values of these system metadata items or update the values when you need. For more information about storage class and server-side encryption, see [Protecting Data Using Encryption \(p. 380\)](#).

The following table provides a list of system-defined metadata and whether you can update it.

Name	Description	Can User Modify the Value?
Date	Object creation date.	No
Content-Length	Object size in bytes.	No
Last-Modified	Date the object was last modified.	No
Content-MD5	The base64-encoded 128-bit MD5 digest of the object.	No

Name	Description	Can User Modify the Value?
x-amz-server-side-encryption	Indicates whether server-side encryption is enabled for the object, and whether that encryption is from the AWS Key Management Service (SSE-KMS) or from AWS-Managed Encryption (SSE-S3). For more information, see Protecting Data Using Server-Side Encryption (p. 381) .	Yes
x-amz-version-id	Object version. When you enable versioning on a bucket, Amazon S3 assigns a version number to objects added to the bucket. For more information, see Using Versioning (p. 422) .	No
x-amz-delete-marker	In a bucket that has versioning enabled, this Boolean marker indicates whether the object is a delete marker.	No
x-amz-storage-class	Storage class used for storing the object.	Yes
x-amz-website-redirect-location	Redirects requests for the associated object to another object in the same bucket or an external URL. For more information, see Configuring a Web Page Redirect (p. 460) .	Yes
x-amz-server-side-encryption-aws-kms-key-id	If the x-amz-server-side-encryption is present and has the value of aws:kms, this indicates the ID of the Key Management Service (KMS) master encryption key that was used for the object.	Yes
x-amz-server-side-encryption-customer-algorithm	Indicates whether server-side encryption with customer-provided encryption keys (SSE-C) is enabled. For more information, see Protecting Data Using Server-Side Encryption with Customer-Provided Encryption Keys (SSE-C) (p. 394) .	Yes

Storage Classes

Each Amazon S3 object is associated with a storage class. Amazon S3 supports the following storage classes:

- Standard — The Standard storage class provides 99.99999999% durability and 99.99% availability of objects over a given year. It is designed to sustain the concurrent loss of data in two facilities.
- Reduced Redundancy Storage (RRS) — The RRS storage class reduces costs by storing noncritical, reproducible data at lower levels of redundancy than the Standard storage class. It provides 99.99% durability and 99.99% availability of objects over a given year. This durability level corresponds to an average annual expected loss of 0.01% of objects
- GLACIER — The GLACIER storage class is suitable for archiving data, where data access is infrequent and a retrieval time of several hours is acceptable. The GLACIER storage class uses the very low-cost Amazon Glacier storage service, but you still manage objects in this storage class through Amazon S3.

Note

You cannot associate an object with the GLACIER storage class as you upload it. You transition existing Amazon S3 objects to the GLACIER storage class by using lifecycle management. For more information, see [Object Lifecycle Management \(p. 86\)](#).

User-Defined Metadata

When uploading an object, you can also assign metadata to the object. You provide this optional information as a name-value pair when you send a PUT or POST request to create the object. When uploading objects using the REST API the optional user-defined metadata names must begin with "x-amz-meta-" to distinguish them from other HTTP headers. When you retrieve the object using the REST API, this prefix is returned. When uploading objects using the SOAP API, the prefix is not required. When you retrieve the object using the SOAP API, the prefix is removed, regardless of which API you used to upload the object.

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

When metadata is retrieved through the REST API, Amazon S3 combines headers that have the same name (ignoring case) into a comma-delimited list. If some metadata contains unprintable characters, it is not returned. Instead, the "x-amz-missing-meta" header is returned with a value of the number of the unprintable metadata entries.

Amazon S3 stores user-defined metadata in lowercase. Each name, value pair must conform to US-ASCII when using REST and UTF-8 when using SOAP or browser-based uploads via POST.

Note

The PUT request header is limited to 8 KB in size. Within the PUT request header, the user-defined metadata is limited to 2 KB in size. User-defined metadata is a set of key-value pairs. The size of user-defined metadata is measured by taking the sum of the number of bytes in the UTF-8 encoding of each key and value.

Object Subresources

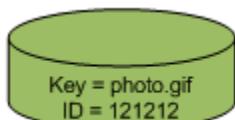
Amazon S3 defines a set of subresources associated with buckets and objects. Subresources are subordinates to objects; that is, subresources do not exist on their own, they are always associated with some other entity, such as an object or a bucket.

The following table lists the subresources associated with Amazon S3 objects.

Subre-source	Description
acl	Contains a list of grants identifying the grantees and the permissions granted. When you create an object, the acl identifies the object owner as having full control over the object. You can retrieve an object ACL or replace it with updated list of grants. Any update to an ACL requires you to replace the existing ACL. For more information about ACLs, see Managing Access with ACLs (p. 363)
torrent	Amazon S3 supports the BitTorrent protocol. Amazon S3 uses the torrent subresource to return the torrent file associated with the specific object. To retrieve a torrent file, you specify the torrent subresource in your GET request. Amazon S3 creates a torrent file and returns it. You can only retrieve the torrent subresource, you cannot create, update or delete the torrent subresource. For more information, see Using BitTorrent with Amazon S3 (p. 519) .

Object Versioning

Versioning enables you to keep multiple versions of an object in one bucket, for example, `my-image.jpg` (version 111111) and `my-image.jpg` (version 222222). You might want to enable versioning to protect yourself from unintended overwrites and deletions or to archive objects so that you can retrieve previous versions of them. Object versioning can be used in combination with [Object Lifecycle Management \(p. 86\)](#), allowing you to customize your data retention needs while controlling your related storage costs. For more information about adding lifecycle configuration to versioning-enabled buckets using the AWS Management Console, see [Lifecycle Configuration for a Bucket with Versioning](#) in the *Amazon Simple Storage Service Console User Guide*.



Note

The SOAP API does not support versioning. SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP.

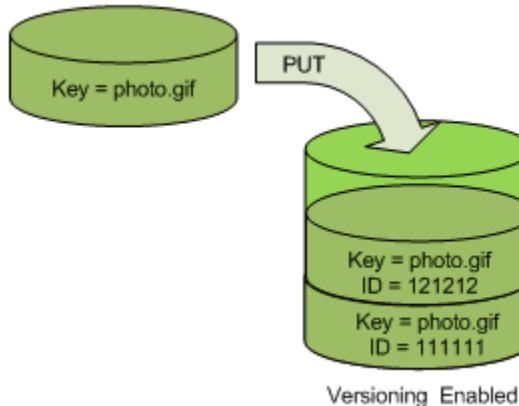
You must explicitly enable versioning on your bucket. By default, versioning is disabled. Regardless of whether you have enabled versioning, each object in your bucket has a version ID. If you have not enabled versioning, then Amazon S3 sets the version ID value to null. If you have enabled versioning, Amazon S3 assigns a unique version ID value for the object. When you enable versioning on a bucket, existing objects, if any, in the bucket are unchanged: the version IDs (null), contents, and permissions remain the same.

Enabling and suspending versioning is done at the bucket level. When you enable versioning for a bucket, all objects added to it will have a unique version ID. Unique version IDs are randomly generated, Unicode, UTF-8 encoded, URL-ready, opaque strings that are at most 1024 bytes long. An example version ID is `3/L4kqtJlcpXroDTDmJ+rmSpXd3dIbrHY+MTRCx3v+jVBH40Nr8X8gdRQBpUMLUo`. Only Amazon S3 generates version IDs. They cannot be edited.

Note

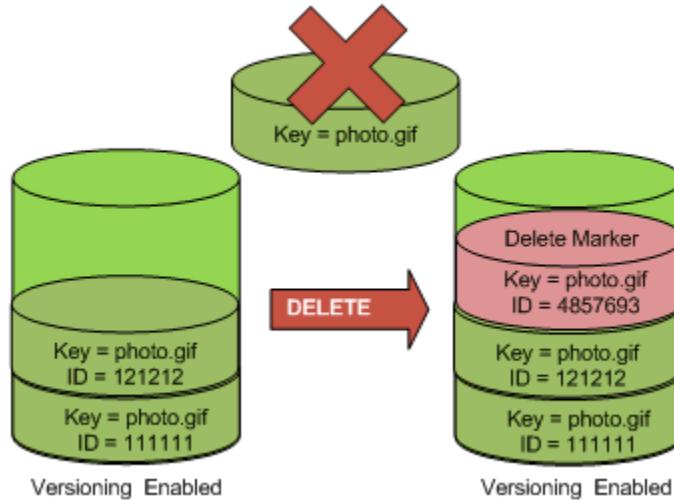
For simplicity, we will use much shorter IDs in all our examples.

When you `PUT` an object in a versioning-enabled bucket, the noncurrent version is not overwritten. The following figure shows that when a new version of `photo.gif` is `PUT` into a bucket that already contains an object with the same name, the original object (ID = 111111) remains in the bucket, Amazon S3 generates a new version ID (121212), and adds the newer version to the bucket.

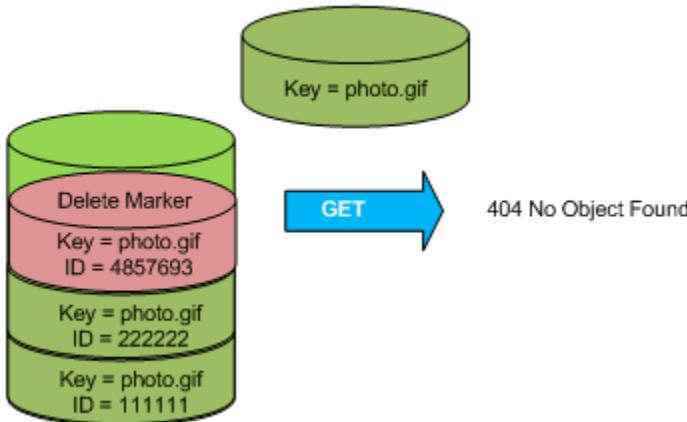


This functionality prevents you from accidentally overwriting or deleting objects and affords you the opportunity to retrieve a previous version of an object.

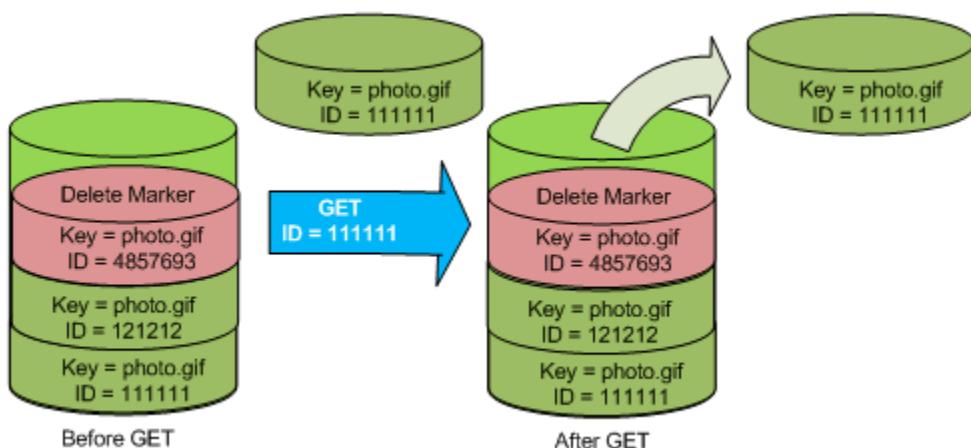
When you `DELETE` an object, all versions remain in the bucket and Amazon S3 inserts a delete marker, as shown in the following figure.



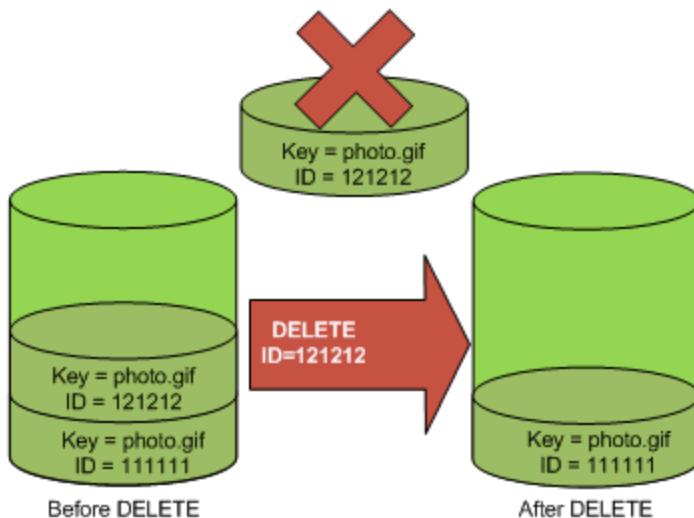
The delete marker becomes the current version of the object. By default, `GET` requests retrieve the most recently stored version. Performing a simple `GET Object` request when the current version is a delete marker returns a `404 Not Found` error, as shown in the following figure.



You can, however, `GET` a noncurrent version of an object by specifying its version ID. In the following figure, we `GET` a specific object version, 111111. Amazon S3 returns that object version even though it's not the current version.



You can permanently delete an object by specifying the version you want to delete. Only the owner of an Amazon S3 bucket can permanently delete a version. The following figure shows how `DELETE` `versionId` permanently deletes an object from a bucket and that Amazon S3 doesn't insert a delete marker.



You can add additional security by configuring a bucket to enable MFA (Multi-Factor Authentication) Delete. When you do, the bucket owner must include two forms of authentication in any request to delete a version or change the versioning state of the bucket. For more information, see [MFA Delete \(p. 423\)](#).

For more information, see [Using Versioning \(p. 422\)](#).

Object Lifecycle Management

Topics

- [Overview \(p. 87\)](#)
- [Lifecycle Configuration Elements \(p. 88\)](#)
- [Object Archival \(Transition Objects to the GLACIER Storage Class\) \(p. 96\)](#)
- [Object Expiration \(p. 98\)](#)
- [Specifying a Lifecycle Configuration \(p. 98\)](#)

Overview

What Is Lifecycle Configuration?

You manage an object's lifecycle by using a lifecycle configuration, which defines how Amazon S3 manages objects during their lifetime, from creation/initial storage to when the object is deleted or overwritten. Amazon S3 provides a number of ways to simplify the lifecycle management of your objects, including automated archival to lower cost storage in Amazon Glacier and scheduled deletions.

You can define lifecycle configuration rules for objects that have a well-defined lifecycle. Many objects that you store in an Amazon S3 bucket might have a well-defined lifecycle, for example:

- If you are uploading periodic logs to your bucket, your application might need these logs for a week or a month after creation, and after that you might want to delete them.
- Some documents are frequently accessed for a limited period of time. After that, you might not need real-time access to these objects, but your organization might require you to archive them for a longer period and then optionally delete them later. Some types of data that you might upload to Amazon S3 primarily for archival purposes include digital media archives, financial and healthcare records, raw genomics sequence data, long-term database backups, and data that must be retained for regulatory compliance.

How Do I Configure a Lifecycle?

A lifecycle configuration comprises a set of rules with predefined actions that you want Amazon S3 to perform on objects during their lifetime. You can specify a lifecycle configuration as XML. Amazon S3 stores the configuration as a "lifecycle" subresource attached to your bucket. Using the Amazon S3 API, you can [Put](#), [Get](#), or [Delete](#) a lifecycle configuration. Unless you need to make the REST API calls directly from your application, you can also configure the lifecycle by using the Amazon S3 console or programmatically by using the AWS SDK wrapper libraries.

Lifecycle Support for Versioning

You can add lifecycle configuration to nonversioned buckets and versioning-enabled buckets. By default a bucket is nonversioned and you can optionally enable versioning. A versioning-enabled bucket maintains one current and zero or more noncurrent object versions. You can use versioning and lifecycle rules together to help manage the storage costs of your objects. Using the predefined lifecycle configuration actions, you can manage lifecycle of both current and noncurrent object versions.

- The `Transition` and `Expiration` lifecycle actions enable you to manage the lifecycle of current versions of your objects.
- The `NonCurrentVersionTransition` and `NonCurrentVersionExpiration` actions enable you to manage the lifecycle of noncurrent (previous) versions of your objects.

For more information about these actions, see [Lifecycle Configuration Elements \(p. 88\)](#). For information about versioning, see [Object Versioning \(p. 84\)](#).

General Considerations

About Archiving Objects

Using lifecycle configuration, you can transition objects to the `GLACIER` storage class—that is, archive data to Amazon Glacier, a lower-cost storage solution. Before you archive objects, note the following:

- Objects in the `GLACIER` storage class are not available in real time.

Archived objects are Amazon S3 objects, but before you can access an archived object, you must first restore a temporary copy of it. The restored object copy is available only for the duration you specify in the restore request. After that, Amazon S3 deletes the temporary copy, and the object remains archived in Amazon Glacier.

Note that object restoration from an archive can take up to five hours.

You can restore an object by using the Amazon S3 console or programmatically by using the AWS SDKs wrapper libraries or the Amazon S3 REST API (see [POST Object restore](#)) in your code.

- The transition of objects to the `GLACIER` storage class is one-way.

You cannot use a lifecycle configuration rule to convert the storage class of an object from `GLACIER` to Standard or RRS. If you want to change the storage class of an already archived object to either Standard or RRS, you must use the restore operation to make a temporary copy first. Then use the copy operation to overwrite the object as a Standard or RRS object.

- `GLACIER` storage class objects are visible and available only through Amazon S3, not through Amazon Glacier.

Amazon S3 stores the archived objects in Amazon Glacier; however, these are Amazon S3 objects, and you can access them only by using the Amazon S3 console or the API. You cannot access the archived objects through the Amazon Glacier console or the API.

For more information about lifecycle configuration, see the following topics:

- [Lifecycle Configuration Elements \(p. 88\)](#) describes various elements in the lifecycle configuration XML.
- [Specifying a Lifecycle Configuration \(p. 98\)](#) describes how you can specify a lifecycle configuration on your bucket by using the Amazon S3 console and programmatically by using the AWS SDKs.

Lifecycle Configuration and MFA Enabled Buckets

Lifecycle configuration on MFA-enable buckets is not supported.

Lifecycle Configuration Elements

Topics

- [ID Element \(p. 90\)](#)
- [Status Element \(p. 90\)](#)
- [Prefix Element \(p. 90\)](#)
- [Action Elements \(p. 90\)](#)
- [Examples of Lifecycle Configuration \(p. 93\)](#)

You specify a lifecycle configuration as XML. The following are two introductory example configurations:

Example 1. Lifecycle configuration on a nonversioned bucket

By default your bucket is nonversioned. That is, there is only one version of each object in a bucket. Suppose you have a bucket that is not versioning enabled; for each object, there is only one version in the bucket. You want to transition objects with key prefix "documents/" to the GLACIER storage class one year after you create them, and then permanently remove them 10 years after you created them. You can accomplish this by attaching the following lifecycle configuration to the bucket. The lifecycle configuration defines one rule with two actions (Transition and Expiration). The rule applies to objects with key prefix "documents" specified in the Prefix element.

```
<LifecycleConfiguration>
  <Rule>
    <ID>sample-rule</ID>
    <Prefix>documents/</Prefix>
    <Status>Enabled</Status>
    <Transition>
      <Days>365</Days>
      <StorageClass>GLACIER</StorageClass>
    </Transition>
    <Expiration>
      <Days>3650</Days>
    </Expiration>
  </Rule>
</LifecycleConfiguration>
```

Example 2. Lifecycle configuration on a versioning-enabled bucket

You can optionally enable versioning on your bucket. If your bucket is versioning enabled, you have one current object version and zero or more noncurrent versions. For more information, see [Object Versioning \(p. 84\)](#). While object versioning enables you to maintain a history of object versions, a lifecycle enables you to control how long the versions exist, or how long to keep the versions live before transitioning them to the GLACIER storage class for long-term archival.

For a versioning-enabled bucket, lifecycle configuration provides additional predefined actions that you can use to manage the noncurrent object version as shown in the example. The lifecycle configuration has one rule. It specifies two actions (NoncurrentVersionTransition and NoncurrentVersionExpiration) on objects with key prefix "logs/". The NoncurrentVersionTransition action requests Amazon S3 to transition objects to the GLACIER storage class 30 days after the objects become noncurrent. The NoncurrentVersionExpiration action requests Amazon S3 to permanently remove the objects 180 days after they become noncurrent.

```
<LifecycleConfiguration>
  <Rule>
    <ID>sample-rule</ID>
    <Prefix>logs/</Prefix>
    <Status>Enabled</Status>
    <NoncurrentVersionTransition>
      <NoncurrentDays>30</NoncurrentDays>
      <StorageClass>GLACIER</StorageClass>
    </NoncurrentVersionTransition>
    <NoncurrentVersionExpiration>
      <NoncurrentDays>180</NoncurrentDays>
    </NoncurrentVersionExpiration>
  </Rule>
</LifecycleConfiguration>
```

You can use the predefined `NoncurrentVersionTransition` and `NoncurrentVersionExpiration` actions to manage noncurrent versions in your bucket. The following sections discuss the rules and actions in detail.

So in general, each lifecycle configuration rule consists of the following:

- Rule metadata that include a rule ID (the `<ID>` element), and status (the `<Status>` element) indicating whether rule is enabled or disabled. If a rule is disabled, Amazon S3 will not perform any actions specified in the rule.
- Prefix (the `<Prefix>` element) identifying objects by the key prefix to which the rule applies.
- One or more actions, for example, the `<NoncurrentVersionTransition>` and `<NoncurrentVersionExpiration>` elements in the preceding example), that you want Amazon S3 to perform on the specified objects. Each action specifies a date or a time period in object's lifetime when you want Amazon S3 to perform the specified action.

These elements of each lifecycle configuration rule are discussed in detail in the following sections.

ID Element

A lifecycle configuration can have up to 1000 rules. The ID element uniquely identifies a rule.

Status Element

The Status element value can be either Enabled or Disabled. If a rule is disabled, Amazon S3 will not perform any of the actions defined in the rule.

Prefix Element

A lifecycle configuration rule can apply to a single object or multiple objects whose key name begins with the prefix specified in the rule. For example, suppose you have the following objects:

- logs/day1
- logs/day2
- logs/day3
- ExampleObject.jpg

If you specify `ExampleObject.jpg` as the prefix, the rule applies only to that specific object. If you specify `logs/` as the prefix, the rule applies to the three objects whose key name begins with the string "logs/". If you specify an empty prefix, the rule applies to all objects in the bucket.

Action Elements

You can request Amazon S3 to perform specific actions in an object's lifetime by specifying predefined actions in a lifecycle configuration rule. The predefined actions are: `Transition`, `Expiration`, `NoncurrentVersionTransition`, and `NoncurrentVersionExpiration`. The effect of these actions depend on versioning state of your bucket.

Note

By default, buckets are nonversioned. You can optionally enable versioning on the bucket, in which case there is one current version and zero or more noncurrent versions of your objects. You can also temporarily suspend versioning. For more information about versioning states, see [Using Versioning \(p. 422\)](#).

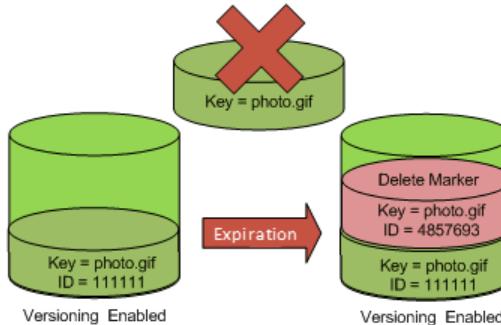
- **Transition action** – Archives an object by changing the storage class of the object to **GLACIER**. When a specified date or time period in the object's lifetime is reached, Amazon S3 transitions the applicable objects to the **GLACIER** storage class.
 - *Nonversioned bucket* – The Transition action sets the storage class of the object to **GLACIER**.

For example, you can set a rule for Amazon S3 to transition "MyArchiveObject.jpg" to the **GLACIER** storage class 30 days after creation or to transition all objects with the key prefix "glacierobjects/" to the **GLACIER** storage class a year after creation.
- **Versioning-enabled and versioning-suspended bucket** – The Transition action sets the storage class of the current version of the object to **GLACIER**. This action has no impact on the noncurrent versions of the object.

- **Expiration action** – This action expires objects identified in the rule. Amazon S3 makes all expired objects unavailable. Whether the objects are permanently removed depends on the versioning state of the bucket.
 - *Nonversioned bucket* – The Expiration action permanently deletes the object, and the deleted object cannot be recovered.
 - *Versioning-enabled bucket* – This action applies only to the current object versions; it has no impact on noncurrent versions. The Expiration action in this case does not permanently delete the current version. It retains the current version as a noncurrent version by introducing a delete marker as the new current version, only if the existing current version is not a delete marker. If the current version is already a delete marker then no action is taken. If the current version is the only object version and is a delete marker, Amazon S3 removes the current version. There can be some delay in cleaning up the delete marker as Amazon S3 ensures this is the only object version.

When you perform a `GET` request without specifying a version ID for an object whose current version is a delete marker, Amazon S3 behaves as though the object has been deleted and returns a `404 Object Not Found` error. However, you can recover the deleted object by specifying the version ID in the `GET` request.

For example, you can set a rule for Amazon S3 to expire an object called `photo.gif` five days from object creation. If `photo.gif` is created on 1/1/2014 10:30 a.m. UTC, the expiration rule executes on 1/7/2014 00:00 UTC, five days after the object was created.



The image shows that after the expiration rule executes, the `photo.gif` (version 111111) is still stored and can be accessed if necessary, but the current version of the object (version 4857693) is now a delete marker. The original `photo.gif` becomes a noncurrent version. For information about how delete markers work, see [Deleting Object Versions \(p. 436\)](#) and [Using Versioning \(p. 422\)](#).

- **Versioning-suspended bucket** – When the state of the bucket is versioning-suspended, the expiration results in the creation of a delete marker with a `null` version ID. Any existing null version is overwritten by the new null version and data associated with that version cannot be recovered. For more

information about deleting objects from a versioning-suspended bucket, see [Deleting Objects from Versioning-Suspended Buckets \(p. 445\)](#).

- Actions specific to versioning-enabled (or versioning-suspended) buckets

The following lifecycle configuration actions are applicable only when your bucket is versioning-enabled (or versioning-suspended) buckets. A versioning-enabled bucket can have many versions of an object, one current version, and zero or more noncurrent versions. Using these actions, you can request Amazon S3 to perform specific actions on noncurrent object versions. These actions have no impact on the current version of the object.

- **NoncurrentVersionTransition action** – Using this action, you can specify how long you want the noncurrent object versions to remain in the Standard storage class before being transitioned to the lower-cost `GLACIER` storage class.
- **NoncurrentVersionExpiration action** – Using this action, you can specify how long you want to retain noncurrent object versions before Amazon S3 permanently removes them. The deleted object cannot be recovered.

For example, if you want to enable a five-day window to correct any accidental deletes or overwrites, you can configure an expiration rule that deletes the noncurrent versions of your objects five days after they become noncurrent.

On 1/1/2014 10:30 a.m. UTC, you create an object called `photo.gif` (version ID 111111). On 1/2/2014 11:30 a.m. UTC, you accidentally delete `photo.gif` (version ID 111111), which creates a delete marker with a new version ID (such as version ID 4857693). You now have five days to recover the original version of `photo.gif` (version ID 111111) before the deletion is permanent.

On 1/8/2014 00:00 UTC, the lifecycle rule for expiration executes and permanently deletes `photo.gif` (version ID 111111), five days after it became a noncurrent version.

How Amazon S3 Calculates How Long an Object Has Been Noncurrent

In a versioning-enabled bucket, you can have multiple versions of an object, there is always one current version, and zero or more noncurrent versions. Each time you upload an object, the current version is retained as noncurrent version and the newly added version, the successor, become current. To determine the number of days an object is noncurrent, Amazon S3 looks at when its successor was created. Amazon S3 uses the number of days since its successor was created as the number of days an object is noncurrent.

Restoring Previous Versions of an Object When Using Lifecycle Configurations

As explained in detail in the topic [Restoring Previous Versions \(p. 441\)](#), there are two ways to retrieve previous versions of an object.

1. By copying a noncurrent version of the object into the same bucket. The copied object becomes the current version of that object, and all object versions are preserved.
2. By permanently deleting the current version of the object. When you delete the current object version, you, in effect, turn the noncurrent version into the current version of that object.

When using lifecycle configuration rules with versioning-enabled buckets, we recommend as a best practice not to use method #2, retrieving a noncurrent version by deleting the current version of the object. Instead, you should use method #1, retrieve a noncurrent version of the object by copying the noncurrent version of the object into the same bucket so that the noncurrent version becomes the current version.

Because of Amazon S3's eventual consistency semantics, a current version that you permanently deleted may not disappear until the changes propagate (Amazon S3 may be unaware of this deletion). And in the meantime, lifecycle you configured to expire noncurrent objects may

permanently remove noncurrent object, including the one you want to restore. So copying the old version, method#1, is the safer alternative.

Lifecycle Rules Based on a Specific Date

When using the Transition and Expiration actions, you can specify a date when the action will be taken.

Note

You cannot create the date-based rule using the AWS Management Console, but you can view, disable, or delete such rules.

Lifecycle Rules Based on the Object Age

When using the Transition and Expire actions, you can specify a time period in days since object creation when the action will be taken.

When you specify a number of days, Amazon S3 calculates the time by adding the number of days specified in the rule to the object creation time and rounding the resulting time to the next day midnight UTC. For example, if an object was created at 1/15/2014 10:30 a.m. UTC and you specify 3 days in a transition rule, then the transition date of the object would be calculated as 1/19/2014 00:00 UTC.

Note

Amazon S3 maintains only the last modified date for each object. For example, the Amazon S3 console shows the **Last Modified** date in the object **Properties** pane. When you initially create a new object, this date reflects the date the object is created. If you replace the object, the date will change accordingly. So when we use the term *creation date*, it is synonymous with the term *last modified date*.

When using the NoncurrentVersionTransition and NoncurrentVersionExpiration actions, you specify the number of days from when the version of the object becomes noncurrent (that is, since the object was overwritten or deleted), as the time period for when the action will be taken on the specified object or objects.

When you specify a number of days in rules that use the NoncurrentVersionTransition and NoncurrentVersionExpiration actions, Amazon S3 calculates the time by adding the number of days specified in the rule to the time when the new successor version of the object is created and rounding the resulting time to the next day midnight UTC. For example, in your bucket, you have a current version of an object that was created at 1/1/2014 10:30 am UTC, if the new successor version of the object that replaces the current version is created at 1/15/2014 10:30 a.m. UTC and you specify 3 days in a transition rule, then the transition date of the object would be calculated as 1/19/2014 00:00 UTC.

Examples of Lifecycle Configuration

This section provides examples of lifecycle configuration. Each example shows how you can specify XML in each of the example scenarios.

Example 1: Specify a Rule

The following lifecycle configuration has a rule with two actions. It requests Amazon S3 to transition objects with key prefix projectdocs/ to the GLACIER storage class 365 days after creation and delete those objects 3650 days after creation.

```
<LifecycleConfiguration>
  <Rule>
    <ID>Transition and Expiration Rule</ID>
    <Prefix>projectdocs/</Prefix>
    <Status>Enabled</Status>
    <Transition>
```

```
<Days>365</Days>
<StorageClass>GLACIER</StorageClass>
</Transition>
<Expiration>
  <Days>3650</Days>
</Expiration>
</Rule>
</LifecycleConfiguration>
```

Instead of a time period, you can specify a date for each action; however, you cannot use both a date and a time period in the same rule.

Example 2: Specify a Rule that Applies to All Objects in the Bucket

The following lifecycle configuration specifies an empty prefix and zero transition days. If you don't specify a prefix, the rule applies to all objects in the bucket. Because the Days element is set to 0, all objects are eligible for archival to Amazon Glacier at midnight UTC following creation.

```
<LifecycleConfiguration>
<Rule>
  <ID>Archive all object same-day upon creation</ID>
  <Prefix></Prefix>
  <Status>Enabled</Status>
  <Transition>
    <Days>0</Days>
    <StorageClass>GLACIER</StorageClass>
  </Transition>
</Rule>
</LifecycleConfiguration>
```

Example 3: Disabling a Rule

The following lifecycle configuration specifies two rules; however, one of them is disabled. Amazon S3 will not perform any action specified in a rule that is disabled.

```
<LifecycleConfiguration>
<Rule>
  <ID>30 days log objects expire rule</ID>
  <Prefix>logs/</Prefix>
  <Status>Enabled</Status>
  <Expiration>
    <Days>30</Days>
  </Expiration>
</Rule>
<Rule>
  <ID>1 year documents expire rule</ID>
  <Prefix>documents/</Prefix>
  <Status>Disabled</Status>
  <Expiration>
    <Days>365</Days>
  </Expiration>
</Rule>
</LifecycleConfiguration>
```

Example 4: Overlapping Prefixes Are Allowed

Rules can use overlapping prefixes as long as the rules don't share the same action. The following lifecycle configuration has a rule that sets all the noncurrent versions in a bucket (by specifying empty prefix) to expire in 10 days. This allows 10 days to revert to old versions of the objects. In another rule, the current objects with the prefix "logs/" are set to expire after 30 days.

```
<LifecycleConfiguration>
  <Rule>
    <ID>111</ID>
    <Prefix></Prefix>
    <Status>Enabled</Status>
    <NoncurrentVersionExpiration>
      <NoncurrentDays>10</Days>
    </NoncurrentVersionExpiration>
  </Rule>
  <Rule>
    <ID>222</ID>
    <Prefix>logs/</Prefix>
    <Status>Enabled</Status>
    <Expiration>
      <Days>30</Days>
    </Expiration>
  </Rule>
</LifecycleConfiguration>
```

Example 5: Specify a Rule that Includes Actions for Noncurrent Versioned Objects.

The following is a sample of a lifecycle configuration that includes actions for noncurrent versions of the objects in a versioning-enabled bucket.

```
<LifecycleConfiguration>
  <Rule>
    <ID>sample-rule</ID>
    <Prefix>key-prefix</Prefix>
    <Status>Enabled</Status>
    <Transition>
      <Days>365</Days>
      <StorageClass>GLACIER</StorageClass>
    </Transition>
    <Expiration>
      <Days>730</Days>
    </Expiration>
    <NoncurrentVersionTransition>
      <NoncurrentDays>30</NoncurrentDays>
      <StorageClass>GLACIER</StorageClass>
    </NoncurrentVersionTransition>
    <NoncurrentVersionExpiration>
      <NoncurrentDays>365</NoncurrentDays>
    </NoncurrentVersionExpiration>
  </Rule>
</LifecycleConfiguration>
```

Object Archival (Transition Objects to the GLACIER Storage Class)

Topics

- [Pricing Considerations \(p. 96\)](#)
- [Restoring Archived Objects \(p. 97\)](#)

For objects that you do not need to access in real time, Amazon S3 also offers the **GLACIER** storage class. This storage class is suitable for objects stored primarily for archival purposes.

Each object in Amazon S3 is associated with a storage class. When you upload an object, by default Amazon S3 associates it with the Standard storage class, in which Amazon S3 maintains redundant copies to ensure high durability. For noncritical, reproducible data, you can use the Reduced Redundancy Storage (RRS) option, in which Amazon S3 uses lower levels of redundancy than it does for the Standard storage class. RRS provides a cost-effective, highly available solution. You can find storage class of an object using API (see [GET Bucket \(List Objects\)](#)) or in the console. For individual objects, the **Details** section in the object **Properties** show the storage class of the object. In addition, If you list all objects in the bucket, the console shows the storage class for all the objects in the list.

All Buckets / ExampleBucket			
	Name	Storage Class	Size
<input type="checkbox"/>	ExampleArchive.pdf	Standard	6 bytes
<input type="checkbox"/>	ExampleDocument.txt	Standard	6 bytes
<input type="checkbox"/>	ExampleGuide.pdf	Standard	348.9 KB
<input type="checkbox"/>	ExampleObject.txt	Reduced Redundancy	857 bytes
<input checked="" type="checkbox"/>	ExamplePhoto1.jpg	Reduced Redundancy	21 bytes

Both the Standard and RRS objects are highly available in real-time and the **GLACIER** storage class is suitable for objects stored primarily for archival purposes.

You can specify the Standard or RRS storage class when you upload objects; however, you cannot assign the **GLACIER** storage class when uploading objects. Instead, you transition existing objects to the **GLACIER** storage class by using the [Object Lifecycle Management \(p. 86\)](#). Amazon S3 will archive objects for you and associate those objects with the **GLACIER** storage class according to rules that you define.

Amazon S3 stores the archived objects in Amazon Glacier; however, these are Amazon S3 objects, and you can access them only by using the Amazon S3 console or the API. You cannot access the archived objects through the Amazon Glacier console or the API.

The lifecycle configuration enables a one-way transition to the **GLACIER** storage class. To change the storage class from **GLACIER** to Standard or RRS, you must restore the object, as discussed in the following section, and then make a copy of the restored object.

To find when an object was transitioned to the **GLACIER** storage class by Amazon S3, you can use Amazon S3 server access logs. The value "S3.TRANSITION.OBJECT" of the operation field in the log record indicates that Amazon S3 initiated the operation. For more information, see [Server Access Logging \(p. 530\)](#).

Pricing Considerations

If you are planning to archive infrequently accessed data for a period of months or years, the **GLACIER** storage class will usually reduce your storage costs. You should, however, consider the following in order to ensure that the **GLACIER** storage class is appropriate for you:

- **Storage overhead charges** – When you transition objects to the `GLACIER` storage class, a fixed amount of storage is added to each object to accommodate metadata for managing the object.
 - For each object archived to Amazon Glacier, Amazon S3 uses 8 KB of storage for the name of the object and other metadata. Amazon S3 stores this metadata so that you can get a real-time list of your archived objects by using the Amazon S3 API (see [Get Bucket \(List Objects\)](#)). You are charged standard Amazon S3 rates for this additional storage.
 - For each archived object, Amazon Glacier adds 32 KB of storage for index and related metadata. This extra data is necessary to identify and restore your object. You are charged Amazon Glacier rates for this additional storage.

If you are archiving small objects, consider these storage charges. Also consider aggregating a large number of small objects into a smaller number of large objects in order to reduce overhead costs.

- **Number of days you plan to keep objects archived** – Amazon Glacier is a long-term archival solution. Deleting data that is archived to Amazon Glacier is free if the objects you delete are archived for three months or longer. If you delete or overwrite an object within three months of archiving it, Amazon S3 charges a prorated early deletion fee.
- **Glacier archive request charges** – Each object that you transition to the `GLACIER` storage class constitutes one archive request. There is a cost for each such request. If you plan to transition a large number of objects, consider the request costs.
- **Glacier data restore charges** – Amazon Glacier is designed for long-term archival of data that you will access infrequently. Data restore charges are based on how quickly you restore data, which is measured as your peak billable restore rate in GB/hr for the entire month. Within a month, you are charged only for the peak billable restore rate, and there is no charge for restoring data at less than the monthly peak billable restore rate. Before initiating a large restore, carefully review the [pricing FAQ](#) to determine how you will be billed for restoring data.

When you archive objects to Amazon Glacier by using object lifecycle management, Amazon S3 transitions these objects asynchronously. There may be a lag between the transition date in the lifecycle configuration rule and the date of the physical transition. You are charged Amazon Glacier prices based on the transition date specified in the rule.

The Amazon S3 product detail page provides pricing information and example calculations for archiving Amazon S3 objects. For more information, see the following topics:

- [How is my storage charge calculated for Amazon S3 objects archived to Amazon Glacier?](#)
- [How am I charged for deleting objects from Amazon Glacier that are less than 3 months old?](#)
- [Amazon S3 Pricing](#) for storage costs for the Standard and `GLACIER` storage classes. This page also provides Glacier Archive Request costs.
- [How will I be charged for restoring large amounts of data from Amazon Glacier?](#)

Restoring Archived Objects

Archived objects are not accessible in real-time. You must first initiate a restore request and then wait until a temporary copy of the object is available for the duration that you specify in the request. Restore jobs typically complete in three to five hours, so it is important that you archive only objects that you will not need to access in real time.

After you receive a temporary copy of the restored object, the object's storage class remains `GLACIER` (a GET or HEAD request will return `GLACIER` as the storage class). Note that when you restore an archive you are paying for both the archive (`GLACIER` rate) and a copy you restored temporarily (RRS rate). For information about pricing, go to the [Pricing](#) section of the [Amazon S3 product detail page](#).

You can restore an object copy programmatically or by using the Amazon S3 console. Amazon S3 will process only one restore request at a time per object. You can use both the console and the Amazon S3 API to check the restoration status and to find out when Amazon S3 will delete the restored copy.

Restoring GLACIER Objects by Using Amazon S3 Console

For information about restoring archived objects, stored using the GLACIER storage class, by using the Amazon S3 console, see [Restore an Object Using the Amazon S3 Console \(p. 262\)](#).

Restoring GLACIER Objects Programmatically

You can restore GLACIER objects programmatically directly from your application by using either the AWS SDKs or the Amazon S3 API. When you use the AWS SDKs, the Amazon S3 API provides appropriate wrapper libraries to simplify your programming tasks; however, when the request is sent over the wire, the SDK sends the preceding XML in the request body. For information about restoring objects programmatically, see [Restoring Objects \(p. 261\)](#).

Object Expiration

Topics

Some objects that you store in an Amazon S3 bucket might have well-defined lifetimes. For example, you might upload logs periodically to your bucket, and you might need to retain those logs for only a specific amount of time. You can use the [Object Lifecycle Management \(p. 86\)](#) to specify a lifetime for objects in your bucket; when the lifetime of an object expires, Amazon S3 queues the object for permanent removal or logical deletion depending on your bucket's versioning state.

Amazon S3 provides an `Expiration` action that you can specify in your lifecycle configuration to expire objects. In addition, for version-enabled buckets, Amazon S3 also provides the `NoncurrentVersionExpiration` action to expire noncurrent versions of objects.

- If the bucket is nonversioned, the `Expiration` action results in Amazon S3 permanently removing the object.

If you have server access logging enabled, Amazon S3 reports the permanent removal as operation "S3.EXPIRE.OBJECT" in the log record. For more information, see [Server Access Logging \(p. 530\)](#).

- If the bucket is versioning-enabled (or versioning is suspended), the `Expiration` action logically deletes the current version by adding a delete marker as the new current version. The `NoncurrentVersionExpiration` action permanently removes the noncurrent versions.

If you have server access logging enabled, Amazon S3 reports the logical deletion as operation "S3.CREATE.DELETEMARKER" and permanent deletion as operation "S3.EXPIRE.OBJECT" in the log record.

When an object reaches the end of its lifetime, Amazon S3 queues it for removal and removes it asynchronously. There may be a lag between the expiration date and the date at which Amazon S3 removes an object. You are not charged for storage time associated with an object that has expired.

To find when an object is scheduled to expire, you can use the GET Object or the HEAD Object APIs. These APIs return response headers that provide object expiration information. For more information, go to [HEAD Object](#) and [GET Object](#) in the *Amazon Simple Storage Service API Reference*.

For examples, see [Object Lifecycle Management \(p. 86\)](#).

Specifying a Lifecycle Configuration

Topics

- [Manage an Object's Lifecycle Using the AWS Management Console \(p. 99\)](#)
- [Manage Object Lifecycle Using the AWS SDK for Java \(p. 100\)](#)

- Manage an Object's Lifecycle Using the AWS SDK for .NET (p. 105)
- Manage an Object's Lifecycle Using the AWS SDK for Ruby (p. 110)
- Manage Object Lifecycle Using the REST API (p. 110)

You can set a lifecycle configuration on a bucket either by programmatically using the Amazon S3 API or by using the Amazon S3 console. When you add a lifecycle configuration to a bucket, there is usually some lag before a new or updated lifecycle configuration is fully propagated to all the Amazon S3 systems. Expect a delay of a few minutes before the lifecycle configuration fully takes effect. This delay can also occur when you delete a lifecycle configuration.

When you disable or delete a lifecycle rule, after a small delay Amazon S3 stops scheduling new objects for deletion or transition. Any objects that were already scheduled will be unscheduled and will not be deleted or transitioned.

Note

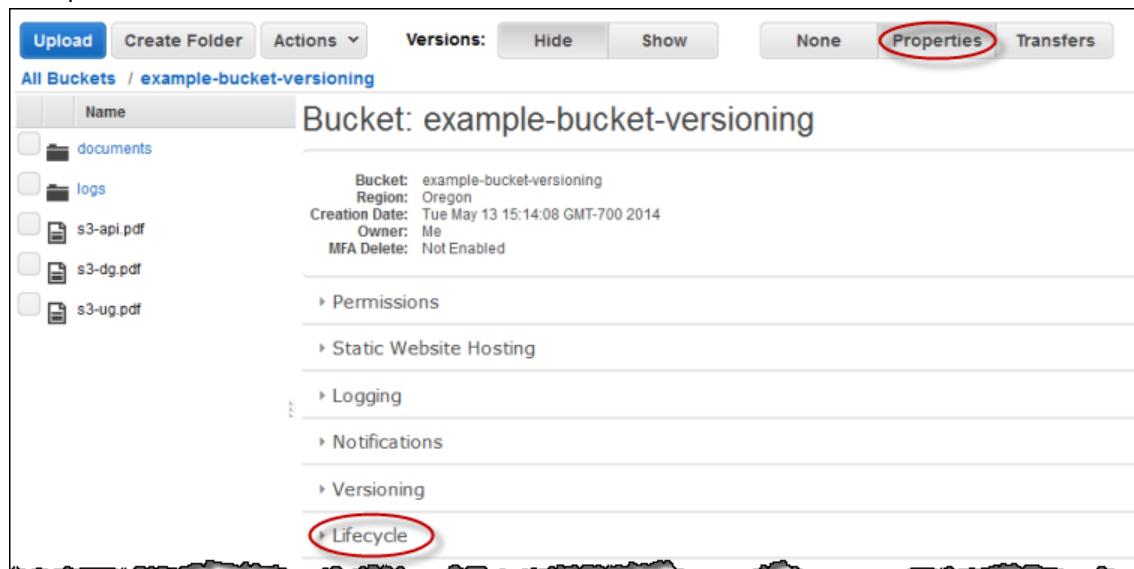
When you add a lifecycle configuration to a bucket, the configuration rules apply to both existing objects and objects that you add later. For example, if you add a lifecycle configuration rule with an expiration action today that causes objects with a specific prefix to expire 30 days after creation, Amazon S3 will queue for removal any existing objects that are more than 30 days old.

There may be a lag between when the lifecycle configuration rules are satisfied and when the action triggered by satisfying the rule is taken. However, changes in billing happen as soon as the lifecycle configuration rule is satisfied even if the action is not yet taken. One example is you are not charged for storage after the object expiration time even if the object is not deleted immediately. Another example is you are charged Amazon Glacier storage rates as soon as the object transition time elapses even if the object is not transitioned to Amazon Glacier immediately.

For information about specifying the lifecycle by using the Amazon S3 console or programmatically by using AWS SDKs, click the links provided at the beginning of this topic.

Manage an Object's Lifecycle Using the AWS Management Console

You can specify lifecycle rules (see [Object Lifecycle Management \(p. 86\)](#)) on a bucket using the Amazon S3 console. In the console, the bucket **Properties** provides a **Lifecycle** tab as shown in the following example screen shot.



Step-by-Step Instructions

For instructions on how to setup lifecycle rules using the AWS Management Console, see [Managing Lifecycle Configuration](#) in the *Amazon S3 Console User Guide*.

Manage Object Lifecycle Using the AWS SDK for Java

You can use the AWS SDK for Java to manage lifecycle configuration on a bucket. For more information about managing lifecycle configuration, see [Object Lifecycle Management \(p. 86\)](#).

This section provides sample code snippets for the tasks in the following table, followed by a complete sample program listing.

To add a lifecycle configuration to a bucket

1. Create a `BucketLifecycleConfiguration.Rule` object to specify a rule.
2. Create a `BucketLifecycleConfiguration` object using the rules.
3. Add the lifecycle configuration to the bucket by calling `s3Client.setBucketLifecycleConfiguration()`.

The following Java code snippet demonstrates the preceding tasks. The lifecycle configuration defines one rule ("Archive and delete rule") specifying two actions for Amazon S3 to perform on all objects with key prefix "projectdocs/".

```
Transition transToArchive = new Transition()
    .withDays(365)
    .withStorageClass(StorageClass.Glacier);

BucketLifecycleConfiguration.Rule ruleArchiveAndExpire = new BucketLifecycleConfiguration.Rule()
    .withId("Archive and delete rule")
    .withPrefix("projectdocs/")
    .withTransition(transToArchive)
    .withExpirationInDays(3650)
    .withStatus(BucketLifecycleConfiguration.ENABLED.toString());

List<BucketLifecycleConfiguration.Rule> rules = new ArrayList<BucketLifecycleConfiguration.Rule>();
rules.add(ruleArchiveAndExpire);

BucketLifecycleConfiguration configuration = new BucketLifecycleConfiguration()

    .withRules(rules);

// Save configuration.
s3Client.setBucketLifecycleConfiguration(bucketName, configuration);
```

If your bucket is versioning-enabled, you can specify the `NoncurrentVersionExpiration` and `NoncurrentVersionTransition` lifecycle actions for Amazon S3 to transition and delete noncurrent objects as shown in the following Java code snippet. The lifecycle rule applies to objects with the key prefix "logs/". The rule sets two actions; the `NoncurrentVersionTransition` action requests that Amazon S3 transition applicable objects to the `GLACIER` storage class 10 days after they become noncurrent. The `NoncurrentVersionExpiration` action requests that Amazon S3 expire noncurrent objects 30 days after they become noncurrent. For more information about these actions, see [Lifecycle Configuration Elements \(p. 88\)](#).

```
Transition transToArchive = new Transition()
    .withDays(365)
    .withStorageClass(StorageClass.Glacier);

BucketLifecycleConfiguration.Rule nonCurrentObjectRule =
    new BucketLifecycleConfiguration.Rule()
        .withId("test")
        .withPrefix("logs/")
        .withStatus(BucketLifecycleConfiguration.ENABLED.toString())
        .withNoncurrentVersionTransition(new NoncurrentVersionTransition()
            .withDays(10)
            .withStorageClass(StorageClass.Glacier))
        .withNoncurrentVersionExpirationInDays(30);

List<BucketLifecycleConfiguration.Rule> rules = new ArrayList<BucketLifecycleConfiguration.Rule>();
rules.add(nonCurrentObjectRule);

BucketLifecycleConfiguration configuration = new BucketLifecycleConfiguration()
    .withRules(rules);

// Save configuration.
s3Client.setBucketLifecycleConfiguration(bucketName, configuration);
```

To update an existing lifecycle configuration

- When you add a lifecycle configuration to a bucket, any existing lifecycle configuration is replaced. To update an existing lifecycle configuration, you must first retrieve the existing lifecycle configuration, make changes, and then add the revised lifecycle configuration to the bucket as shown in the following Java code snippet.

This snippet gets an existing configuration and adds a new rule with the ID NewRule.

```
// Retrieve configuration.
configuration = s3Client.getBucketLifecycleConfiguration(bucketName);

// Add a new rule.
Calendar c = Calendar.getInstance(TimeZone.getTimeZone("GMT"));
c.add(Calendar.YEAR, 10);
c.set(Calendar.HOUR_OF_DAY, 0);
c.set(Calendar.MINUTE, 0);
c.set(Calendar.SECOND, 0);
c.set(Calendar.MILLISECOND, 0);
configuration.getRules().add(
    new BucketLifecycleConfiguration.Rule()
        .withId("NewRule")
        .withPrefix("YearlyDocuments/")
        .withExpirationDate(c.getTime())
        .withStatus(BucketLifecycleConfiguration.ENABLED.toString())
);
```

```
// Save configuration.  
s3Client.setBucketLifecycleConfiguration(bucketName, configuration);
```

Example Program Listing

The following Java code example provides a complete code listing that adds, updates, and deletes a lifecycle configuration to a bucket. You will need to update the code and provide your bucket name to which the code can add the example lifecycle configuration.

For instructions on how to create and test a working sample, see [Testing the Java Code Examples \(p. 545\)](#).

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;
import java.util.TimeZone;

import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.AmazonS3Exception;
import com.amazonaws.services.s3.model.BucketLifecycleConfiguration;
import com.amazonaws.services.s3.model.BucketLifecycleConfiguration.Transition;
import com.amazonaws.services.s3.model.StorageClass;

public class S3LifecycleExample {
    public static String bucketName = "**** Provide bucket name ****";
    public static AmazonS3Client s3Client;

    public static void main(String[] args) throws IOException {
        s3Client = new AmazonS3Client(new ProfileCredentialsProvider());
        try {

            BucketLifecycleConfiguration.Rule rule1 =
                new BucketLifecycleConfiguration.Rule()
                    .withId("Archive immediately rule")
                    .withPrefix("glacierobjects/")
                    .withTransition(new Transition()
                        .withDays(0)
                        .withStorageClass(StorageClass.Glacier))
                    .withStatus(BucketLifecycleConfiguration.ENABLED.toString());

            BucketLifecycleConfiguration.Rule rule2 =
                new BucketLifecycleConfiguration.Rule()
                    .withId("Archive and then delete rule")
                    .withPrefix("projectdocs/")
                    .withTransition(new Transition()
                        .withDays(365)
                        .withStorageClass(StorageClass.Glacier))
                    .withExpirationInDays(3650)
                    .withStatus(BucketLifecycleConfiguration.ENABLED.toString());

            List<BucketLifecycleConfiguration.Rule> rules =
                new ArrayList<BucketLifecycleConfiguration.Rule>();
            rules.add(rule1);
            rules.add(rule2);

            BucketLifecycleConfiguration configuration =
                new BucketLifecycleConfiguration()
                    .withRules(rules);

            // Save configuration.
        }
    }
}
```

```
s3Client.setBucketLifecycleConfiguration(bucketName, configuration);

        // Retrieve configuration.
configuration = s3Client.getBucketLifecycleConfiguration(bucketName);

        // Add a new rule.
Calendar c = Calendar.getInstance(TimeZone.getTimeZone("GMT"));
c.add(Calendar.YEAR, 10);
c.set(Calendar.HOUR_OF_DAY, 0);
c.set(Calendar.MINUTE, 0);
c.set(Calendar.SECOND, 0);
c.set(Calendar.MILLISECOND, 0);
configuration.getRules().add(
    new BucketLifecycleConfiguration.Rule()
        .withId("NewRule")
        .withPrefix("YearlyDocuments/")
        .withExpirationDate(c.getTime())
        .withStatus(BucketLifecycleConfiguration.
            ENABLED.toString())
);
        // Save configuration.
s3Client.setBucketLifecycleConfiguration(bucketName, configuration);

        // Retrieve configuration.
configuration = s3Client.getBucketLifecycleConfiguration(bucketName);

        // Verify there are now three rules.
configuration = s3Client.getBucketLifecycleConfiguration(bucketName);

System.out.format("Expected # of rules = 3; found: %s\n",
    configuration.getRules().size());

        // Delete configuration.
s3Client.deleteBucketLifecycleConfiguration(bucketName);

        // Retrieve nonexistent configuration.
configuration = s3Client.getBucketLifecycleConfiguration(bucketName);

String s = (configuration == null) ? "No configuration found." :
"Configuration found.";
System.out.println(s);

    } catch (AmazonS3Exception amazonS3Exception) {
        System.out.format("An Amazon S3 error occurred. Exception: %s",
amazonS3Exception.toString());
    } catch (Exception ex) {
        System.out.format("Exception: %s", ex.toString());
    }
}
}
```

Manage an Object's Lifecycle Using the AWS SDK for .NET

You can use the AWS SDK for .NET to manage lifecycle configuration on a bucket. For more information, see [Object Lifecycle Management \(p. 86\)](#).

Adding a Lifecycle Configuration to a Bucket

1. Create an instance of the `LifeCycleConfiguration` class and specify a list of `LifecycleRules`.
2. Create a `PutLifecycleConfigurationRequest` object.

You will need to provide a bucket name and the `LifeCycleConfiguration` instance you created in the preceding step.

3. Execute the `client.PutLifecycleConfiguration`.

The following C# code snippet demonstrates the preceding tasks. The lifecycle configuration defines one rule ("Archive and delete rule") specifying two actions for Amazon S3 to perform on all objects with key prefix "projectdocs/".

```
string bucketName = "examplebucket";

// Add a sample configuration
using (client = new AmazonS3Client()){
    var lifeCycleConfiguration = new LifeCycleConfiguration()
    {
        Rules = new List<LifecycleRule>
        {
            new LifecycleRule
            {
                Id = "Archive and delete rule",
                Prefix = "projectdocs/",
                Status = LifecycleRuleStatus.Enabled,
                Transition = new LifecycleTransition()
                {
                    Days = 365,
                    StorageClass = S3StorageClass.Glacier
                },
                Expiration = new LifecycleRuleExpiration()
                {
                    Days = 3650
                }
            }
        };
    };

    PutLifecycleConfigurationRequest request = new PutLifecycleConfigurationRequest
    {
        BucketName = bucketName,
        Configuration = configuration
    };

    var response = client.PutLifecycleConfiguration(request);
}
```

If your bucket is versioning-enabled you can specify the `NoncurrentVersionExpiration` and `NoncurrentVersionTransition` lifecycle actions for Amazon S3 to transition and delete noncurrent

objects as shown in the following C# code snippet. The lifecycle rule applies to objects with the key prefix "logs/". The rule sets two actions; the NoncurrentVersionTransition action requests that Amazon S3 transition applicable objects to the GLACIER storage class 10 days after they become noncurrent. The NoncurrentVersionExpiration action requests that Amazon S3 expire noncurrent objects 30 days after they become noncurrent. For more information about these actions, see [Lifecycle Configuration Elements \(p. 88\)](#).

```

string bucketName = "versioningenabled-examplebucket";

// Add a sample configuration
using (client = new AmazonS3Client()){
    var lifeCycleConfiguration = new LifecycleConfiguration()
    {
        Rules = new List<LifecycleRule>
        {
            new LifecycleRule
            {
                Id = "new rule",
                Prefix = "logs/",
                Status = LifecycleRuleStatus.Enabled,
                NoncurrentVersionTransition = new LifecycleRuleNoncurrentVersion
                    Transition()
                {
                    NoncurrentDays = 10,
                    StorageClass = S3StorageClass.Glacier
                },
                NoncurrentVersionExpiration = new LifecycleRuleNoncurrentVersion
                    Expiration()
                {
                    NoncurrentDays = 30
                }
            }
        };
    }

    PutLifecycleConfigurationRequest request = new PutLifecycleConfigurationRequest
    {
        BucketName = bucketName,
        Configuration = configuration
    };

    var response = client.PutLifecycleConfiguration(request);
}

```

To update an existing lifecycle configuration

- When you add a lifecycle configuration to a bucket, any existing lifecycle configuration is replaced. To update an existing configuration, you must first retrieve it, make changes, and then add the revised configuration to the bucket as shown in the following C# code snippet.

This snippet gets an existing configuration and adds a new rule with the ID NewRule.

```

// Retrieve lifecycle configuration.
GetLifecycleConfigurationRequest request = new GetLifecycleConfigurationRequest
{

```

```
        BucketName = bucketName
    };
var response = client.GetLifecycleConfiguration(request);
var configuration = response.Configuration;

// Add new rule.
configuration.Rules.Add(new LifecycleRule
{
    Id = "NewRule",
    Prefix = "YearlyDocuments/",
    Expiration = new LifecycleRuleExpiration { Date = DateTime.Now.AddYears(10)
})
);

PutLifecycleConfigurationRequest request = new PutLifecycleConfigurationRequest
{
    BucketName = bucketName,
    Configuration = configuration
};

var response = client.PutLifecycleConfiguration(request);
```

Example Program Listing

The following C# code example provides complete code listing that adds, updates, and deletes a lifecycle configuration to a bucket. You will need to update the code and provide the bucket name to which the code can add the example lifecycle configuration.

For instructions on how to create and test a working sample, see [Using the AWS SDK for .NET Testing the .NET Code Examples](#).

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using Amazon.S3;
using Amazon.S3.Model;

namespace aws.amazon.com.s3.documentation
{
    class S3Sample
    {
        static string bucketName = "**** Provide bucket name ****";

        static AmazonS3 client;

        public static void Main(string[] args)
        {
            try
            {
                AmazonS3Config s3Config = new AmazonS3Config();
                using (client = new AmazonS3Client(s3Config))
                {
                    var lifeCycleConfiguration = new LifecycleConfiguration()
                    {
                        Rules = new List<LifecycleRule>
                        {
                            new LifecycleRule
                            {
                                Id = "Archive immediately rule",
                                Prefix = "glacierobjects/",
                                Status = LifecycleRuleStatus.Enabled,
                                Transition = new LifecycleTransition()
                                {
                                    Days = 0,
                                    StorageClass = S3StorageClass.Glacier
                                }
                            },
                            new LifecycleRule
                            {
                                Id = "Archive and then delete rule",
                                Prefix = "projectdocs/",
                                Status = LifecycleRuleStatus.Enabled,
                                Transition = new LifecycleTransition()
                                {
                                    Days = 365,
                                    StorageClass = S3StorageClass.Glacier
                                },
                                Expiration = new LifecycleRuleExpiration()
                                {
                                    Days = 3650
                                }
                            }
                        }
                    };
                    client.PutBucketLifecycle(bucketName, lifeCycleConfiguration);
                }
            }
        }
    }
}
```

```
        }
    }
};

// Add the configuration to the bucket
PutLifecycleConfiguration(lifeCycleConfiguration);

// Retrieve an existing configuration
lifeCycleConfiguration = GetLifecycleConfiguration();

// Add a new rule.
lifeCycleConfiguration.Rules.Add(new LifecycleRule
{
    Id = "NewRule",
    Prefix = "YearlyDocuments/",
    Expiration = new LifecycleRuleExpiration { Date = DateTime.Now.AddYears(10) }
});

// Add the configuration to the bucket
PutLifecycleConfiguration(lifeCycleConfiguration);

// Verify that there are now three rules
lifeCycleConfiguration = GetLifecycleConfiguration();
Console.WriteLine("Expected # of rules=3; found:{0}", lifeCycleConfiguration.Rules.Count);

// Delete the configuration
DeleteLifecycleConfiguration();

// Retrieve a nonexistent configuration
lifeCycleConfiguration = GetLifecycleConfiguration();
Debug.Assert(lifeCycleConfiguration == null);
}

Console.WriteLine("Example complete. To continue, click Enter...");  

    Console.ReadKey();
}
catch (AmazonS3Exception amazonS3Exception)
{
    Console.WriteLine("S3 error occurred. Exception: " + amazonS3Exception.ToString());
}
catch (Exception e)
{
    Console.WriteLine("Exception: " + e.ToString());
}
}

static void PutLifecycleConfiguration(LifecycleConfiguration configuration)
{
    PutLifecycleConfigurationRequest request = new PutLifecycleConfigurationRequest
    {

```

```
        BucketName = bucketName,
        Configuration = configuration
    };

    var response = client.PutLifecycleConfiguration(request);
}

static LifecycleConfiguration GetLifeCycleConfiguration()
{
    GetLifecycleConfigurationRequest request = new GetLifecycleConfig
urationRequest
    {
        BucketName = bucketName

    };
    var response = client.GetLifecycleConfiguration(request);
    var configuration = response.Configuration;
    return configuration;
}

static void DeleteLifecycleConfiguration()
{
    DeleteLifecycleConfigurationRequest request = new DeleteLifecycleCon
figurationRequest
    {
        BucketName = bucketName
    };
    client.DeleteLifecycleConfiguration(request);
}
}
```

Manage an Object's Lifecycle Using the AWS SDK for Ruby

You can use the AWS SDK for Ruby to manage lifecycle configuration on a bucket by using the class [AWS::S3::BucketLifecycleConfiguration](#). For more information about using the AWS SDK for Ruby with Amazon S3, go to [Using the AWS SDK for Ruby \(p. 549\)](#). For more information about managing lifecycle configuration, see [Object Lifecycle Management \(p. 86\)](#).

Manage Object Lifecycle Using the REST API

You can use the AWS Management Console to set the lifecycle configuration on your bucket. If your application requires it, you can also send REST requests directly. The following sections in the *Amazon Simple Storage Service API Reference* describe the REST API related to the lifecycle configuration.

- [PUT Bucket lifecycle](#)
- [GET Bucket lifecycle](#)
- [DELETE Bucket lifecycle](#)

Enabling Cross-Origin Resource Sharing

Cross-origin resource sharing (CORS) defines a way for client web applications that are loaded in one domain to interact with resources in a different domain. With CORS support in Amazon S3, you can build

rich client-side web applications with Amazon S3 and selectively allow cross-origin access to your Amazon S3 resources.

This section provides an overview of CORS. The subtopics describe how you can enable CORS using the Amazon S3 console, or programmatically using the Amazon S3 REST API and the AWS SDKs.

Topics

- [Cross-Origin Resource Sharing: Examples \(p. 111\)](#)
- [How Do I Enable CORS on My Bucket? \(p. 111\)](#)
- [How Does Amazon S3 Evaluate the CORS Configuration On a Bucket? \(p. 113\)](#)
- [Managing Cross-Origin Resource Sharing \(CORS\) \(p. 114\)](#)

Cross-Origin Resource Sharing: Examples

The following are example scenarios for using CORS:

- Example 1: Suppose you are hosting a website in an Amazon S3 bucket named `website` as described in [Hosting a Static Website on Amazon S3 \(p. 448\)](#). Your users load the website endpoint `http://website.s3-website-us-east-1.amazonaws.com`. Now you want to use JavaScript on the web pages that are stored in this bucket to be able to make authenticated GET and PUT requests against the same bucket by using the Amazon S3's API endpoint for the bucket, `website.s3.amazonaws.com`. A browser would normally block JavaScript from allowing those requests, but with CORS, you can configure your bucket to explicitly enable cross-origin requests from `website.s3-website-us-east-1.amazonaws.com`.
- Example 2: Suppose you want to host a web font from your S3 bucket. Again, browsers require a CORS check (also referred as a preflight check) for loading web fonts, so you would configure the bucket that is hosting the web font to allow any origin to make these requests.

How Do I Enable CORS on My Bucket?

To configure your bucket to allow cross-origin requests, you create a CORS configuration, an XML document with rules that identify the origins that you will allow to access your bucket, the operations (HTTP methods) will support for each origin, and other operation-specific information. You can add up to 100 rules to the configuration. You add the XML document as the `cors` subresource to the bucket.

For example, the following `cors` configuration on a bucket has three rules, which are specified as `CORSRule` elements:

- The first rule allows cross-origin PUT, POST, and DELETE requests from the `https://www.example1.com` origin. The rule also allows all headers in a preflight OPTIONS request through the `Access-Control-Request-Headers` header. In response to any preflight OPTIONS request, Amazon S3 will return any requested headers.
- The second rule allows same cross-origin requests as the first rule but the rule applies to another origin, `https://www.example2.com`.
- The third rule allows cross-origin GET requests from all origins. The '*' wildcard character refers to all origins.

```
<CORSConfiguration>
  <CORSRule>
    <AllowedOrigin>http://www.example1.com</AllowedOrigin>
    <AllowedMethod>PUT</AllowedMethod>
```

```
<AllowedMethod>POST</AllowedMethod>
<AllowedMethod>DELETE</AllowedMethod>

<AllowedHeader>*</AllowedHeader>
</CORSRule>
<CORSRule>
    <AllowedOrigin>http://www.example2.com</AllowedOrigin>

    <AllowedMethod>PUT</AllowedMethod>
    <AllowedMethod>POST</AllowedMethod>
    <AllowedMethod>DELETE</AllowedMethod>

    <AllowedHeader>*</AllowedHeader>
</CORSRule>
<CORSRule>
    <AllowedOrigin>*</AllowedOrigin>
    <AllowedMethod>GET</AllowedMethod>
</CORSRule>
</CORSConfiguration>
```

The CORS configuration also allows optional configuration parameters, as shown in the following CORS configuration. In this example, the following CORS configuration allows cross-origin PUT and POST requests from the `http://www.example.com` origin.

```
<CORSConfiguration>
    <CORSRule>
        <AllowedOrigin>http://www.example.com</AllowedOrigin>
        <AllowedMethod>PUT</AllowedMethod>
        <AllowedMethod>POST</AllowedMethod>
        <AllowedMethod>DELETE</AllowedMethod>
        <AllowedHeader>*</AllowedHeader>
        <MaxAgeSeconds>3000</MaxAgeSeconds>
        <ExposeHeader>x-amz-server-side-encryption</ExposeHeader>
        <ExposeHeader>x-amz-request-id</ExposeHeader>
        <ExposeHeader>x-amz-id-2</ExposeHeader>
    </CORSRule>
</CORSConfiguration>
```

The `CORSRule` element in the preceding configuration includes the following optional elements:

- `MaxAgeSeconds`—Specifies the amount of time in seconds (in this example, 3000) that the browser will cache an Amazon S3 response to a preflight OPTIONS request for the specified resource. By caching the response, the browser does not have to send preflight requests to Amazon S3 if the original request is to be repeated.
- `ExposeHeader`—Identifies the response headers (in this example, `x-amz-server-side-encryption`, `x-amz-request-id`, and `x-amz-id-2`) that customers will be able to access from their applications (for example, from a JavaScript XMLHttpRequest object).

AllowedMethod Element

In the CORS configuration, you can specify the following values for the `AllowedMethod` element.

- GET
- PUT
- POST

- DELETE
- HEAD

AllowedOrigin Element

In the `AllowedOrigin` element you specify the origins that you want to allow cross-domain requests from, for example, `http://www.example.com`. The origin string can contain at most one * wildcard character, such as `http://*.example.com`. You can optionally specify * as the origin to enable all the origins to send cross-origin requests. You can also specify `https` to enable only secure origins.

AllowedHeader Element

The `AllowedHeader` element specifies which headers are allowed in a preflight request through the `Access-Control-Request-Headers` header. Each header name in the `Access-Control-Request-Headers` header must match a corresponding entry in the rule. Amazon S3 will send only the allowed headers in a response that were requested. For a sample list of headers that can be used in requests to Amazon S3, go to [Common Request Headers](#) in the *Amazon Simple Storage Service API Reference* guide.

Each `AllowedHeader` string in the rule can contain at most one * wildcard character. For example, `<AllowedHeader>x-amz-*</AllowedHeader>` will enable all Amazon-specific headers.

ExposeHeader Element

Each `ExposeHeader` element identifies a header in the response that you want customers to be able to access from their applications (for example, from a JavaScript `XMLHttpRequest` object). For a list of common Amazon S3 response headers, go to [Common Response Headers](#) in the *Amazon Simple Storage Service API Reference* guide.

MaxAgeSeconds Element

The `MaxAgeSeconds` element specifies the time in seconds that your browser can cache the response for a preflight request as identified by the resource, the HTTP method, and the origin.

How Does Amazon S3 Evaluate the CORS Configuration On a Bucket?

When Amazon S3 receives a preflight request from a browser, it evaluates the CORS configuration for the bucket and uses the first `CORSRule` rule that matches the incoming browser request to enable a cross-origin request. For a rule to match, the following conditions must be met:

- The request's `Origin` header must match an `AllowedOrigin` element.
- The request method (for example, GET or PUT) or the `Access-Control-Request-Method` header in case of a preflight OPTIONS request must be one of the `AllowedMethod` elements.
- Every header listed in the request's `Access-Control-Request-Headers` header on the preflight request must match an `AllowedHeader` element.

Note

The ACLs and policies continue to apply when you enable CORS on the bucket.

Managing Cross-Origin Resource Sharing (CORS)

Enable cross-origin resource sharing by setting a CORS configuration on your bucket using the AWS Management Console, the REST API, or the AWS SDKs.

Topics

- [Enabling Cross-Origin Resource Sharing \(CORS\) Using the AWS Management Console \(p. 114\)](#)
- [Enabling Cross-Origin Resource Sharing \(CORS\) Using the AWS SDK for Java \(p. 114\)](#)
- [Enabling Cross-Origin Resource Sharing \(CORS\) Using the AWS SDK for .NET \(p. 119\)](#)
- [Enabling Cross-Origin Resource Sharing \(CORS\) Using the REST API \(p. 125\)](#)
- [Troubleshooting CORS Issues \(p. 125\)](#)

Enabling Cross-Origin Resource Sharing (CORS) Using the AWS Management Console

You can use the AWS Management Console to set a CORS configuration on your bucket. For instructions, see [Editing Bucket Permissions](#) in the *Amazon S3 Console User Guide*

Enabling Cross-Origin Resource Sharing (CORS) Using the AWS SDK for Java

You can use the AWS SDK for Java to manage cross-origin resource sharing (CORS) for a bucket. For more information about CORS, see [Enabling Cross-Origin Resource Sharing \(p. 110\)](#).

This section provides sample code snippets for following tasks, followed by a complete example program demonstrating all tasks.

- Creating an instance of the Amazon S3 client class
- Creating and adding a CORS configuration to a bucket
- Updating an existing CORS configuration

Cross-Origin Resource Sharing Methods

<code>AmazonS3Client()</code>	Constructs an <code>AmazonS3Client</code> object.
<code>setBucketCrossOriginConfiguration()</code>	Sets the CORS configuration that to be applied to the bucket. If a configuration already exists for the specified bucket, the new configuration will replace the existing one.
<code>getBucketCrossOriginConfiguration()</code>	Retrieves the CORS configuration for the specified bucket. If no configuration has been set for the bucket, the <code>Configuration</code> header in the response will be null.
<code>deleteBucketCrossOriginConfiguration()</code>	Deletes the CORS configuration for the specified bucket.

For more information about the AWS SDK for Java API, go to [AWS SDK for Java API Reference](#).

Creating an Instance of the Amazon S3 Client Class

The following snippet creates a new `AmazonS3Client` instance for a class called `CORS_JavaSDK`. This example retrieves the values for `accessKey` and `secretKey` from the `AwsCredentials.properties` file.

```
AmazonS3Client client;
client = new AmazonS3Client(new ProfileCredentialsProvider());
```

Creating and Adding a CORS Configuration to a Bucket

To add a CORS configuration to a bucket:

1. Create a `CORSRule` object that describes the rule.
2. Create a `BucketCrossOriginConfiguration` object, and then add the rule to the configuration object.
3. Add the CORS configuration to the bucket by calling the `client.setBucketCrossOriginConfiguration` method.

The following snippet creates two rules, `CORSRule1` and `CORSRule2`, and then adds each rule to the `rules` array. By using the `rules` array, it then adds the rules to the bucket `bucketName`.

```
// Add a sample configuration
BucketCrossOriginConfiguration configuration = new BucketCrossOriginConfiguration();

List<CORSRule> rules = new ArrayList<CORSRule>();

CORSRule rule1 = new CORSRule()
    .withId("CORSRule1")
    .withAllowedMethods(NSArray.asList(new CORSRule.AllowedMethods[] {
        CORSRule.AllowedMethods.PUT, CORSRule.AllowedMethods.POST, CORSRule.AllowedMethods.DELETE}))
    .withAllowedOrigins(NSArray.asList(new String[] {"http://*.example.com"}));

CORSRule rule2 = new CORSRule()
    .withId("CORSRule2")
    .withAllowedMethods(NSArray.asList(new CORSRule.AllowedMethods[] {
        CORSRule.AllowedMethods.GET}))
    .withAllowedOrigins(NSArray.asList(new String[] {"*"}))
    .withMaxAgeSeconds(3000)
    .withExposedHeaders(NSArray.asList(new String[] {"x-amz-server-side-encryption"}));

configuration.setRules(NSArray.asList(new CORSRule[] {rule1, rule2}));

// Save the configuration
client.setBucketCrossOriginConfiguration(bucketName, configuration);
```

Updating an Existing CORS Configuration

To update an existing CORS configuration

1. Get a CORS configuration by calling the `client.getBucketCrossOriginConfiguration` method.
2. Update the configuration information by adding or deleting rules to the list of rules.
3. Add the configuration to a bucket by calling the `client.getBucketCrossOriginConfiguration` method.

The following snippet gets an existing configuration and then adds a new rule with the ID `NewRule`.

```
// Get configuration.  
BucketCrossOriginConfiguration configuration = client.getBucketCrossOriginCon  
figuration(bucketName);  
  
// Add new rule.  
CORSRule rule3 = new CORSRule()  
.withId("CORSRule3")  
.withAllowedMethods(Arrays.asList(new CORSRule.AllowedMethods[] {  
    CORSRule.AllowedMethods.HEAD}))  
.withAllowedOrigins(Arrays.asList(new String[] {"http://www.example.com"}));  
  
List<CORSRule> rules = configuration.getRules();  
rules.add(rule3);  
configuration.setRules(rules);  
  
// Save configuration.  
client.setBucketCrossOriginConfiguration(bucketName, configuration);
```

Example Program Listing

The following Java program incorporates the preceding tasks.

For information about creating and testing a working sample, see [Testing the Java Code Examples \(p. 545\)](#).

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.BucketCrossOriginConfiguration;
import com.amazonaws.services.s3.model.CORSRule;

public class Cors {

    /**
     * @param args
     * @throws IOException
     */
    public static AmazonS3Client client;
    public static String bucketName = "****provide bucket name****";

    public static void main(String[] args) throws IOException {
        client = new AmazonS3Client(new ProfileCredentialsProvider());

        // Create a new configuration request and add two rules
        BucketCrossOriginConfiguration configuration = new BucketCrossOriginCon
figuration();

        List<CORSRule> rules = new ArrayList<CORSRule>();

        CORSRule rule1 = new CORSRule()
            .withId("CORSRule1")
            .withAllowedMethods(NSArray.asList(new CORSRule.AllowedMethods[] {
                CORSRule.AllowedMethods.PUT, CORSRule.AllowedMethods.POST,
                CORSRule.AllowedMethods.DELETE}))
            .withAllowedOrigins(NSArray.asList(new String[] {"http://*.ex
ample.com"}));

        CORSRule rule2 = new CORSRule()
            .withId("CORSRule2")
            .withAllowedMethods(NSArray.asList(new CORSRule.AllowedMethods[] {
                CORSRule.AllowedMethods.GET}))
            .withAllowedOrigins(NSArray.asList(new String[] {"*"}))
            .withMaxAgeSeconds(3000)
            .withExposedHeaders(NSArray.asList(new String[] {"x-amz-server-side-en
cryption"}));

        configuration.setRules(NSArray.asList(new CORSRule[] {rule1, rule2}));

        // Add the configuration to the bucket.
        client.setBucketCrossOriginConfiguration(bucketName, configuration);

        // Retrieve an existing configuration.
    }
}
```

```
configuration = client.getBucketCrossOriginConfiguration(bucketName);
printCORSConfiguration(configuration);

// Add a new rule.
CORSRule rule3 = new CORSRule()
.withId("CORSRule3")
.withAllowedMethods(Arrays.asList(new CORSRule.AllowedMethods[] {
    CORSRule.AllowedMethods.HEAD}))
.withAllowedOrigins(Arrays.asList(new String[] {"http://www.example.com"}));

rules = configuration.getRules();
rules.add(rule3);
configuration.setRules(rules);
client.setBucketCrossOriginConfiguration(bucketName, configuration);
System.out.format("Added another rule: %s\n", rule3.getId());

// Verify that the new rule was added.
configuration = client.getBucketCrossOriginConfiguration(bucketName);
System.out.format("Expected # of rules = 3, found %s", configuration.getRules().size());

// Delete the configuration.
client.deleteBucketCrossOriginConfiguration(bucketName);

// Try to retrieve configuration.
configuration = client.getBucketCrossOriginConfiguration(bucketName);
System.out.println("\nRemoved CORS configuration.");
printCORSConfiguration(configuration);
}

static void printCORSConfiguration(BucketCrossOriginConfiguration configuration)
{
    if (configuration == null)
    {
        System.out.println("\nConfiguration is null.");
        return;
    }

    System.out.format("\nConfiguration has %s rules:\n", configuration.getRules().size());
    for (CORSRule rule : configuration.getRules())
    {
        System.out.format("Rule ID: %s\n", rule.getId());
        System.out.format("MaxAgeSeconds: %s\n", rule.getMaxAgeSeconds());

        System.out.format("AllowedMethod: %s\n", rule.getAllowedMethods().toArray());
        System.out.format("AllowedOrigins: %s\n", rule.getAllowedOrigins());

        System.out.format("AllowedHeaders: %s\n", rule.getAllowedHeaders());

        System.out.format("ExposeHeader: %s\n", rule.getExposedHeaders());
    }
}
```

```
}
```

Enabling Cross-Origin Resource Sharing (CORS) Using the AWS SDK for .NET

You can use the AWS SDK for .NET to manage cross-origin resource sharing (CORS) for a bucket. For more information about CORS, see [Enabling Cross-Origin Resource Sharing \(p. 110\)](#).

This section provides sample code for the tasks in the following table, followed by a complete example program listing.

Managing Cross-Origin Resource Sharing

1	Create an instance of the <code>AmazonS3Client</code> class.
2	Create a new CORS configuration.
3	Retrieve and modify an existing CORS configuration.
4	Add the configuration to the bucket.

Cross-Origin Resource Sharing Methods

AmazonS3Client()	Constructs <code>AmazonS3Client</code> with the credentials defined in the <code>App.config</code> file.
PutCORSConfiguration()	Sets the CORS configuration that should be applied to the bucket. If a configuration already exists for the specified bucket, the new configuration will replace the existing one.
GetCORSConfiguration()	Retrieves the CORS configuration for the specified bucket. If no configuration has been set for the bucket, the <code>Configuration</code> header in the response will be null.
DeleteCORSConfiguration()	Deletes the CORS configuration for the specified bucket.

For more information about the AWS SDK for .NET API, go to [Using the AWS SDK for .NET \(p. 546\)](#).

Creating an Instance of the `AmazonS3` Class

The following sample creates an instance of the `AmazonS3Client` class.

```
static IAmazonS3 client;
using (client = new AmazonS3Client(Amazon.RegionEndpoint.USWest2))
```

Adding a CORS Configuration to a Bucket

To add a CORS configuration to a bucket:

1. Create a `CORSConfiguration` object describing the rule.
2. Create a `PutCORSConfigurationRequest` object that provides the bucket name and the CORS configuration.
3. Add the CORS configuration to the bucket by calling `client.PutCORSConfiguration`.

The following sample creates two rules, `CORSRule1` and `CORSRule2`, and then adds each rule to the `Rules` array. By using the `rules` array, it then adds the rules to the bucket `bucketName`.

```
// Add a sample configuration
CORSConfiguration configuration = new CORSConfiguration
{
    Rules = new System.Collections.Generic.List<CORSRule>
    {
        new CORSRule
        {
            Id = "CORSRule1",
            AllowedMethods = new List<string> { "PUT", "POST", "DELETE" },
            AllowedOrigins = new List<string> { "http://*.example.com" }
        },
        new CORSRule
        {
            Id = "CORSRule2",
            AllowedMethods = new List<string> { "GET" },
            AllowedOrigins = new List<string> { "*" },
            MaxAgeSeconds = 3000,
            ExposeHeaders = new List<string> { "x-amz-server-side-encryption" }
        }
    }
};

// Save the configuration
PutCORSConfiguration(configuration);

static void PutCORSConfiguration(CORSConfiguration configuration)
{
    PutCORSConfigurationRequest request = new PutCORSConfigurationRequest
    {
        BucketName = bucketName,
        Configuration = configuration
    };

    var response = client.PutCORSConfiguration(request);
}
```

Updating an Existing CORS Configuration

To update an existing CORS configuration

1. Get a CORS configuration by calling the `client.GetCORSConfiguration` method.
2. Update the configuration information by adding or deleting rules.
3. Add the configuration to a bucket by calling the `client.PutCORSConfiguration` method.

The following snippet gets an existing configuration and then adds a new rule with the ID `NewRule`.

```
// Get configuration.
configuration = GetCORSConfiguration();
// Add new rule.
configuration.Rules.Add(new CORSRule
{
    Id = "NewRule",
```

```
    AllowedMethods = new List<string> { "HEAD" },
    AllowedOrigins = new List<string> { "http://www.example.com" }
}) ;

// Save configuration.
PutCORSConfiguration(configuration);
```

Example Program Listing

The following C# program incorporates the preceding tasks.

For information about creating and testing a working sample, see [Testing the .NET Code Examples \(p. 547\)](#).

```
using System;
using System.Configuration;
using System.Collections.Specialized;
using System.Net;
using Amazon.S3;
using Amazon.S3.Model;
using Amazon.S3.Util;
using System.Diagnostics;
using System.Collections.Generic;

namespace s3.amazon.com.docsamples
{
    class CORS
    {
        static string bucketName = "**** Provide bucket name ****";

        static IAmazonS3 client;

        public static void Main(string[] args)
        {
            try
            {
                using (client = new AmazonS3Client(Amazon.RegionEndpoint.USWest2))
                {
                    // Create a new configuration request and add two rules

                    CORSConfiguration configuration = new CORSConfiguration
                    {
                        Rules = new System.Collections.Generic.List<CorsRule>
                        {
                            new CorsRule
                            {
                                Id = "CorsRule1",
                                AllowedMethods = new List<string> { "PUT", "POST", "DELETE" },
                                AllowedOrigins = new List<string> { "http://*.example.com" }
                            },
                            new CorsRule
                            {
                                Id = "CorsRule2",
                                AllowedMethods = new List<string> { "GET" },
                                AllowedOrigins = new List<string> { "*" },
                                MaxAgeSeconds = 3000,
                                ExposeHeaders = new List<string> { "x-amz-server-side-encryption" }
                            }
                        };
                    };

                    // Add the configuration to the bucket
                }
            }
        }
    }
}
```

```
PutCORSConfiguration(configuration);

// Retrieve an existing configuration
configuration = GetCORSConfiguration();

// Add a new rule.
configuration.Rules.Add(new CORSRule
{
    Id = "CORSRule3",
    AllowedMethods = new List<string> { "HEAD" },
    AllowedOrigins = new List<string> { "http://www.ex
ample.com" }
});

// Add the configuration to the bucket
PutCORSConfiguration(configuration);

// Verify that there are now three rules
configuration = GetCORSConfiguration();
Console.WriteLine();
Console.WriteLine("Expected # of rules=3; found:{0}", configuration.Rules.Count);
Console.WriteLine();
Console.WriteLine("Pause before configuration delete. To continue, click Enter...");
Console.ReadKey();

// Delete the configuration
DeleteCORSConfiguration();

// Retrieve a nonexistent configuration
configuration = GetCORSConfiguration();
Debug.Assert(configuration == null);
}

Console.WriteLine("Example complete.");
}
catch (AmazonS3Exception amazonS3Exception)
{
    Console.WriteLine("S3 error occurred. Exception: " +
amazonS3Exception.ToString());
    Console.ReadKey();
}
catch (Exception e)
{
    Console.WriteLine("Exception: " + e.ToString());
    Console.ReadKey();
}

Console.WriteLine("Press any key to continue...");
Console.ReadKey();
}

static void PutCORSConfiguration(CORSConfiguration configuration)
{

PutCORSConfigurationRequest request = new PutCORSConfigurationRequest
```

```
        {
            BucketName = bucketName,
            Configuration = configuration
        };

        var response = client.PutCORSConfiguration(request);
    }

    static CORSConfiguration GetCORSConfiguration()
    {
        GetCORSConfigurationRequest request = new GetCORSConfigurationRequest
        {
            BucketName = bucketName
        };

        var response = client.GetCORSConfiguration(request);
        var configuration = response.Configuration;
        PrintCORSRules(configuration);
        return configuration;
    }

    static void DeleteCORSConfiguration()
    {
        DeleteCORSConfigurationRequest request = new DeleteCORSConfigurationRequest
        {
            BucketName = bucketName
        };
        client.DeleteCORSConfiguration(request);
    }

    static void PrintCORSRules(CORSConfiguration configuration)
    {
        Console.WriteLine();

        if (configuration == null)
        {
            Console.WriteLine("\nConfiguration is null");
            return;
        }

        Console.WriteLine("Configuration has {0} rules:", configuration.Rules.Count);
        foreach (CORSRule rule in configuration.Rules)
        {
            Console.WriteLine("Rule ID: {0}", rule.Id);
            Console.WriteLine("MaxAgeSeconds: {0}", rule.MaxAgeSeconds);
            Console.WriteLine("AllowedMethod: {0}", string.Join(", ", rule.AllowedMethods.ToArray()));
            Console.WriteLine("AllowedOrigins: {0}", string.Join(", ", rule.AllowedOrigins.ToArray()));
            Console.WriteLine("AllowedHeaders: {0}", string.Join(", ", rule.AllowedHeaders.ToArray()));
            Console.WriteLine("ExposeHeader: {0}", string.Join(", ", rule.ExposeHeaders.ToArray()));
        }
    }
}
```

```
}
```

Enabling Cross-Origin Resource Sharing (CORS) Using the REST API

You can use the AWS Management Console to set CORS configuration on your bucket. If your application requires it, you can also send REST requests directly. The following sections in the *Amazon Simple Storage Service API Reference* describe the REST API actions related to the CORS configuration:

- [PUT Bucket cors](#)
- [GET Bucket cors](#)
- [DELETE Bucket cors](#)
- [OPTIONS object](#)

Troubleshooting CORS Issues

When you are accessing buckets set with the CORS configuration, if you encounter unexpected behavior the following are some troubleshooting actions you can take:

1. Verify that the CORS configuration is set on the bucket.

For instructions, go to [Editing Bucket Permissions](#) in the *Amazon Simple Storage Service Console User Guide*. If you have the CORS configuration set, the console displays an **Edit CORS Configuration** link in the **Permissions** section of the **Properties** bucket.

2. Capture the complete request and response using a tool of your choice. For each request Amazon S3 receives, there must exist one CORS rule matching the data in your request, as follows:

- a. Verify the request has the Origin header.

If the header is missing, Amazon S3 does not treat the request as a cross-origin request and does not send CORS response headers back in the response.

- b. Verify that the Origin header in your request matches at least one of the AllowedOrigin elements in the specific CORSRule.

The scheme, the host, and the port values in the Origin request header must match the AllowedOrigin in the CORSRule. For example, if you set the CORSRule to allow the origin `http://www.example.com`, then both `https://www.example.com` and `http://www.example.com:80` origins in your request do not match the allowed origin in your configuration.

- c. Verify that the Method in your request (or the method specified in the `Access-Control-Request-Method` in case of a preflight request) is one of the AllowedMethod elements in the same CORSRule.

- d. For a preflight request, if the request includes an `Access-Control-Request-Headers` header, verify that the CORSRule includes the AllowedHeader entries for each value in the `Access-Control-Request-Headers` header.

Operations on Objects

Amazon S3 enables you to store, retrieve, and delete objects. You can retrieve an entire object or a portion of an object. If you have enabled versioning on your bucket, you can retrieve a specific version of the object. You can also retrieve a subresource associated with your object and update it where

applicable. You can make a copy of your existing object. Depending on the object size, the following upload and copy related considerations apply:

- **Uploading objects**—You can upload objects of up to 5 GB in size in a single operation. For objects greater than 5 GB you must use the multipart upload API.
Using the multipart upload API you can upload objects up to 5 TB each. For more information, see [Uploading Objects Using Multipart Upload API \(p. 155\)](#).
- **Copying objects**—The copy operation creates a copy of an object that is already stored in Amazon S3.
You can create a copy of your object up to 5 GB in size in a single atomic operation. However, for copying an object greater than 5 GB, you must use the multipart upload API. For more information, see [Copying Objects \(p. 199\)](#).

You can use the REST API (see [Making Requests Using the REST API \(p. 47\)](#)) to work with objects or use one of the following AWS SDK libraries:

- [AWS SDK for Java](#)
- [AWS SDK for .NET](#)
- [AWS SDK for PHP](#)

These libraries provide a high-level abstraction that makes working with objects easy. However, if your application requires, you can use the REST API directly.

Multipart Upload Overview

Topics

- [Concurrent Multipart Upload Operations \(p. 127\)](#)
- [Multipart Upload and Pricing \(p. 128\)](#)
- [Quick Facts \(p. 128\)](#)
- [API Support for Multipart Upload \(p. 128\)](#)
- [Multipart Upload API and Permissions \(p. 129\)](#)

The Multipart upload API enables you to upload large objects in parts. You can use this API to upload new large objects or make a copy of an existing object (see [Operations on Objects \(p. 125\)](#)).

Multipart uploading is a three-step process: You initiate the upload, you upload the object parts, and after you have uploaded all the parts, you complete the multipart upload. Upon receiving the complete multipart upload request, Amazon S3 constructs the object from the uploaded parts, and you can then access the object just as you would any other object in your bucket.

You can list of all your in-progress multipart uploads or get a list of the parts that you have uploaded for a specific multipart upload. Each of these operations is explained in this section.

Multipart Upload Initiation

When you send a request to initiate a multipart upload, Amazon S3 returns a response with an upload ID, which is a unique identifier for your multipart upload. You must include this upload ID whenever you upload parts, list the parts, complete an upload, or abort an upload. If you want to provide any metadata describing the object being uploaded, you must provide it in the request to initiate multipart upload.

Parts Upload

When uploading a part, in addition to the upload ID, you must specify a part number. You can choose any part number between 1 and 10,000. A part number uniquely identifies a part and its position in the

object you are uploading. If you upload a new part using the same part number as a previously uploaded part, the previously uploaded part is overwritten. Whenever you upload a part, Amazon S3 returns an *ETag* header in its response. For each part upload, you must record the part number and the ETag value. You need to include these values in the subsequent request to complete the multipart upload.

Note

After you initiate multipart upload and upload one or more parts, you must either complete or abort multipart upload in order to stop getting charged for storage of the uploaded parts. Only after you either complete or abort multipart upload, Amazon S3 frees up the parts storage and stops charging you for the parts storage.

Multipart Upload Completion (or Abort)

When you complete a multipart upload, Amazon S3 creates an object by concatenating parts in ascending order based on the part number. If any object metadata was provided in the *initiate multipart upload* request, Amazon S3 associates that metadata with the object. After a successful *complete* request, the parts no longer exist. Your *complete multipart upload* request must include the upload ID and a list of both part numbers and corresponding ETag values. Amazon S3 response includes an ETag that uniquely identifies the combined object data. This ETag will not necessarily be an MD5 hash of the object data. You can optionally abort the multipart upload. After aborting a multipart upload, you cannot upload any part using that upload ID again. All storage that any parts from the aborted multipart upload consumed is then freed. If any part uploads were in-progress, they can still succeed or fail even after you aborted. To free all storage consumed by all parts, you must abort a multipart upload only after all part uploads have completed.

Multipart Upload Listings

You can list the parts of a specific multipart upload or all in-progress multipart uploads. The list parts operation returns the parts information that you have uploaded for a specific multipart upload. For each list parts request, Amazon S3 returns the parts information for the specified multipart upload, up to a maximum of 1000 parts. If there are more than 1000 parts in the multipart upload, you must send a series of list part requests to retrieve all the parts. Note that the returned list of parts doesn't include parts that haven't completed uploading.

Note

Only use the returned listing for verification. You should not use the result of this listing when sending a *complete multipart upload* request. Instead, maintain your own list of the part numbers you specified when uploading parts and the corresponding ETag values that Amazon S3 returns.

Using the *list multipart uploads* operation, you can obtain a list of multipart uploads in progress. An in-progress multipart upload is an upload that you have initiated, but have not yet completed or aborted. Each request returns at most 1000 multipart uploads. If there are more than 1000 multipart uploads in progress, you need to send additional requests to retrieve the remaining multipart uploads.

Concurrent Multipart Upload Operations

In a distributed development environment, it is possible for your application to initiate several updates on the same object at the same time. Your application might initiate several multipart uploads using the same object key. For each of these uploads, your application can then upload parts and send a complete upload request to Amazon S3 to create the object. When the buckets have versioning enabled, completing a multipart upload always creates a new version. For buckets that do not have versioning enabled, it is possible that some other request received between the time when a multipart upload is initiated and when it is completed might take precedence.

Note

It is possible for some other request received between the time you initiated a multipart upload and completed it to take precedence. For example, if another operation deletes a key after you initiate a multipart upload with that key, but before you complete it, the complete multipart upload response might indicate a successful object creation without you ever seeing the object.

Multipart Upload and Pricing

Once you initiate a multipart upload, Amazon S3 retains all the parts until you either complete or abort the upload. Throughout its lifetime, you are billed for all storage, bandwidth, and requests for this multipart upload and its associated parts. If you abort the multipart upload, Amazon S3 deletes upload artifacts and any parts you have uploaded, and you are no longer billed for them. For more information on pricing, go to the [Amazon S3 Pricing page](#).

Quick Facts

The following table provides multipart upload (see [Multipart Upload Overview \(p. 126\)](#)) core specifications.

Item	Specification
Maximum object size	5 TB
Maximum number of parts per upload	10,000
Part numbers	1 to 10,000 (inclusive)
Part size	5 MB to 5 GB, last part can be < 5 MB
Maximum number of parts returned for a list parts request	1000
Maximum number of multipart uploads returned in a list multipart uploads request	1000

API Support for Multipart Upload

You can use an AWS SDK to upload an object in parts. The following AWS SDK libraries support multipart upload:

- [AWS SDK for Java](#)
- [AWS SDK for .NET](#)
- [AWS SDK for PHP](#)

These libraries provide a high-level abstraction that makes uploading multipart objects easy. However, if your application requires, you can use the REST API directly. The following sections in the Amazon Simple Storage Service API Reference describe the REST API for multipart upload.

- [Initiate Multipart Upload](#)
- [Upload Part](#)
- [Upload Part \(Copy\)](#)
- [Complete Multipart Upload](#)
- [Abort Multipart Upload](#)
- [List Parts](#)
- [List Multipart Uploads](#)

Multipart Upload API and Permissions

An individual must have the necessary permissions to use the multipart upload operations. You can use ACLs, the bucket policy, or the user policy to grant individuals permissions to perform these operations. The following table lists the required permissions for various multipart upload operations when using ACLs, bucket policy, or the user policy.

Action	Required Permissions
Initiate Multipart Upload	You must be allowed to perform the <code>s3:PutObject</code> action on an object to initiate multipart upload. The bucket owner can allow other principals to perform the <code>s3:PutObject</code> action.
Upload Part	You must be allowed to perform the <code>s3:PutObject</code> action on an object to upload a part. Only the initiator of a multipart upload can upload parts. The bucket owner must allow the initiator to perform the <code>s3:PutObject</code> action on an object in order for the initiator to upload a part for that object.
Upload Part (Copy)	You must be allowed to perform the <code>s3:PutObject</code> action on an object to upload a part. Because you are uploading a part from an existing object, you must be allowed <code>s3:GetObject</code> on the source object. Only the initiator of a multipart upload can upload parts. The bucket owner must allow the initiator to perform the <code>s3:PutObject</code> action on an object in order for the initiator to upload a part for that object.
Complete Multipart Upload	You must be allowed to perform the <code>s3:PutObject</code> action on an object to complete a multipart upload. Only the initiator of a multipart upload can complete that multipart upload. The bucket owner must allow the initiator to perform the <code>s3:PutObject</code> action on an object in order for the initiator to complete a multipart upload for that object.
Abort Multipart Upload	You must be allowed to perform the <code>s3:AbortMultipartUpload</code> action to abort a multipart upload. By default, the bucket owner and the initiator of the multipart upload are allowed to perform this action. If the initiator is an IAM user, that user's AWS account is also allowed to abort that multipart upload. In addition to these defaults, the bucket owner can allow other principals to perform the <code>s3:AbortMultipartUpload</code> action on an object. The bucket owner can deny any principal the ability to perform the <code>s3:AbortMultipartUpload</code> action.
List Parts	You must be allowed to perform the <code>s3>ListMultipartUploadParts</code> action to list parts in a multipart upload. By default, the bucket owner has permission to list parts for any multipart upload to the bucket. The initiator of the multipart upload has the permission to list parts of the specific multipart upload. If the multipart upload initiator is an IAM user, the AWS account controlling that IAM user also has permission to list parts of that upload. In addition to these defaults, the bucket owner can allow other principals to perform the <code>s3>ListMultipartUploadParts</code> action on an object. The bucket owner can also deny any principal the ability to perform the <code>s3>ListMultipartUploadParts</code> action.

Action	Required Permissions
List Multipart Uploads	You must be allowed to perform the <code>s3:ListBucketMultipartUploads</code> action on a bucket to list multipart uploads in progress to that bucket. In addition to the default, the bucket owner can allow other principals to perform the <code>s3:ListBucketMultipartUploads</code> action on the bucket.

For information on the relationship between ACL permissions and permissions in access policies, see [Mapping of ACL Permissions and Access Policy Permissions \(p. 365\)](#). For information on IAM users, go to [Working with Users and Groups](#).

Getting Objects

Topics

- [Related Resources \(p. 130\)](#)
- [Get an Object Using the AWS SDK for Java \(p. 131\)](#)
- [Get an Object Using the AWS SDK for .NET \(p. 134\)](#)
- [Get an Object Using the AWS SDK for PHP \(p. 137\)](#)
- [Get an Object Using the REST API \(p. 139\)](#)
- [Share an Object with Others \(p. 139\)](#)

You can retrieve objects directly from Amazon S3. You have the following options when retrieving an object:

- **Retrieve an entire object**—A single GET operation can return you the entire object stored in Amazon S3.
- **Retrieve object in parts**—Using the `Range` HTTP header in a GET request, you can retrieve a specific range of bytes in an object stored in Amazon S3.
You resume fetching other parts of the object whenever your application is ready. This resumable download is useful when you need only portions of your object data. It is also useful where network connectivity is poor and you need to react to failures.

When you retrieve an object, its metadata is returned in the response headers. There are times when you want to override certain response header values returned in a GET response. For example, you might override the `Content-Disposition` response header value in your GET request. The REST GET Object API (see [GetObject](#)) allows you to specify query string parameters in your GET request to override these values.

The AWS SDK for Java, .NET and PHP also provide necessary objects you can use to specify values for these response headers in your GET request.

When retrieving objects that are stored encrypted using server-side encryption you will need to provide appropriate request headers. For more information, see [Protecting Data Using Encryption \(p. 380\)](#).

Related Resources

- [Using the AWS SDKs and Explorers \(p. 542\)](#)

Get an Object Using the AWS SDK for Java

When you download an object, you get all of object's metadata and a stream from which to read the contents. You should read the content of the stream as quickly as possible because the data is streamed directly from Amazon S3 and your network connection will remain open until you read all the data or close the input stream.

Downloading Objects

1	Create an instance of the <code>AmazonS3Client</code> class.
2	Execute one of the <code>AmazonS3Client.getObject()</code> method. You need to provide the request information, such as bucket name, and key name. You provide this information by creating an instance of the <code>GetObjectRequest</code> class.
3	Execute one of the <code>getObjectContent()</code> methods on the object returned to get a stream on the object data and process the response.

The following Java code sample demonstrates the preceding tasks.

```
AmazonS3 s3Client = new AmazonS3Client(new ProfileCredentialsProvider());  
  
S3Object object = s3Client.getObject(  
    new GetObjectRequest(bucketName, key));  
InputStream objectData = object.getObjectContent();  
// Process the objectData stream.  
objectData.close();
```

The `GetObjectRequest` object provides several options, including conditional downloading of objects based on modification times, ETags, and selectively downloading a range of an object. The following Java code sample demonstrates how you can specify a range of data bytes to retrieve from an object.

```
AmazonS3 s3Client = new AmazonS3Client(new ProfileCredentialsProvider());  
  
GetObjectRequest rangeObjectRequest = new GetObjectRequest(  
    bucketName, key);  
rangeObjectRequest.setRange(0, 10); // retrieve 1st 10 bytes.  
S3Object objectPortion = s3Client.getObject(rangeObjectRequest);  
  
InputStream objectData = objectPortion.getObjectContent();  
// Process the objectData stream.  
objectData.close();
```

When retrieving an object, you can optionally override the response header values (see [Getting Objects \(p. 130\)](#)) by using the `ResponseHeaderOverrides` object and setting the corresponding request property, as shown in the following Java code sample.

```
GetObjectRequest request = new GetObjectRequest(bucketName, key);  
  
ResponseHeaderOverrides responseHeaders = new ResponseHeaderOverrides();  
responseHeaders.setCacheControl("No-cache");  
responseHeaders.setContentDisposition("attachment; filename=testing.txt");
```

```
// Add the ResponseHeaderOverrides to the request.  
request.setResponseHeaders(responseHeaders);
```

Example

The following Java code example retrieves an object from a specified Amazon S3 bucket. For instructions on how to create and test a working sample, see [Testing the Java Code Examples \(p. 545\)](#).

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.GetObjectRequest;
import com.amazonaws.services.s3.model.S3Object;

public class GetObject {
    private static String bucketName = "**** provide bucket name ****";
    private static String key      = "**** provide object key ****";

    public static void main(String[] args) throws IOException {
        AmazonS3 s3Client = new AmazonS3Client(new ProfileCredentialsProvider());

        try {
            System.out.println("Downloading an object");
            S3Object s3object = s3Client.getObject(new GetObjectRequest(
                bucketName, key));
            System.out.println("Content-Type: " +
                s3object.getObjectMetadata().getContentType());
            displayTextInputStream(s3object.getObjectContent());

            // Get a range of bytes from an object.

            GetObjectRequest rangeObjectRequest = new GetObjectRequest(
                bucketName, key);
            rangeObjectRequest.setRange(0, 10);
            S3Object objectPortion = s3Client.getObject(rangeObjectRequest);

            System.out.println("Printing bytes retrieved.");
            displayTextInputStream(objectPortion.getObjectContent());
        } catch (AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException, which" +
                " means your request made it " +
                "to Amazon S3, but was rejected with an error response" +
                " for some reason.");
            System.out.println("Error Message: " + ase.getMessage());
            System.out.println("HTTP Status Code: " + ase.getStatusCode());
            System.out.println("AWS Error Code: " + ase.getErrorCode());
            System.out.println("Error Type: " + ase.getErrorCode());
            System.out.println("Request ID: " + ase.getRequestId());
        } catch (AmazonClientException ace) {
            System.out.println("Caught an AmazonClientException, which means" +
                " the client encountered " +

```

```
        "an internal error while trying to " +
        "communicate with S3, " +
        "such as not being able to access the network.");
    System.out.println("Error Message: " + ace.getMessage());
}
}

private static void displayTextInputStream(InputStream input)
throws IOException {
// Read one text line at a time and display.
BufferedReader reader = new BufferedReader(new
    InputStreamReader(input));
while (true) {
    String line = reader.readLine();
    if (line == null) break;

    System.out.println("      " + line);
}
System.out.println();
}
```

Get an Object Using the AWS SDK for .NET

The following tasks guide you through using the .NET classes to retrieve an object or a portion of the object, and save it locally to a file.

Downloading Objects

1	Create an instance of the <code>AmazonS3</code> class.
2	Execute one of the <code>AmazonS3.GetObject</code> methods. You need to provide information such as bucket name, file path, or a stream. You provide this information by creating an instance of the <code>GetObjectRequest</code> class.
3	Execute one of the <code>GetObjectResponse.WriteResponseStreamToFile</code> methods to save the stream to a file.

The following C# code sample demonstrates the preceding tasks. The examples saves the object to a file on your desktop.

```
static IAmazonS3 client;
using (client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1))
{
    GetObjectRequest request = new GetObjectRequest
    {
        BucketName = bucketName,
        Key = keyName
    };

    using (GetObjectResponse response = client.GetObject(request))
    {
        string dest = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.Desktop), keyName);
```

```
        if (!File.Exists(dest))
    {
        response.WriteResponseStreamToFile(dest);
    }
}
```

Instead of reading the entire object you can read only the portion of the object data by specifying the byte range in the request, as shown in the following C# code sample.

```
GetObjectRequest request = new GetObjectRequest
{
    BucketName = bucketName,
    Key = keyName,
    ByteRange = new ByteRange(0, 10)
};
```

When retrieving an object, you can optionally override the response header values (see [Getting Objects \(p. 130\)](#)) by using the `ResponseHeaderOverrides` object and setting the corresponding request property, as shown in the following C# code sample. You can use this feature to indicate the object should be downloaded into a different filename than the object key name.

```
GetObjectRequest request = new GetObjectRequest
{
    BucketName = bucketName,
    Key = keyName
};

ResponseHeaderOverrides responseHeaders = new ResponseHeaderOverrides();
responseHeaders.CacheControl = "No-cache";
responseHeaders.ContentDisposition = "attachment; filename=testing.txt";

request.ResponseHeaderOverrides = responseHeaders;
```

Example

The following C# code example retrieves an object from an Amazon S3 bucket. From the response, the example reads the object data using the `GetObjectResponse.ResponseStream` property. The example also shows how you can use the `GetObjectResponse.Metadata` collection to read object metadata. If the object you retrieve has the `x-amz-meta-title` metadata, the code will print the metadata value.

For instructions on how to create and test a working sample, see [Testing the .NET Code Examples \(p. 547\)](#).

```
using System;
using System.IO;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples
{
    class GetObject
    {
        static string bucketName = "**** bucket name ****";
        static string keyName    = "**** object key ****";
        static IAmazonS3 client;

        public static void Main(string[] args)
        {
            try
            {
                Console.WriteLine("Retrieving (GET) an object");
                string data = ReadObjectData();
            }
            catch (AmazonS3Exception s3Exception)
            {
                Console.WriteLine(s3Exception.Message,
                                  s3Exception.InnerException);
            }
            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }

        static string ReadObjectData()
        {
            string responseBody = "";

            using (client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1))
            {
                GetObjectRequest request = new GetObjectRequest
                {
                    BucketName = bucketName,
                    Key = keyName
                };

                using (GetObjectResponse response = client.GetObject(request))

                using (Stream responseStream = response.ResponseStream)
                using (StreamReader reader = new StreamReader(responseStream))

                {
                    string title = response.Metadata["x-amz-meta-title"];
                }
            }
        }
    }
}
```

```
        Console.WriteLine("The object's title is {0}", title);

        responseBody = reader.ReadToEnd();
    }
}
return responseBody;
}
}
```

Get an Object Using the AWS SDK for PHP

This topic guides you through using a class from the AWS SDK for PHP to retrieve an object. You can retrieve an entire object or specify a byte range to retrieve from the object.

Note

This topic assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 547\)](#) and have the AWS SDK for PHP properly installed.

Downloading an Object

1	Create an instance of an Amazon S3 client by using the Aws\S3\S3Client class <code>factory()</code> method.
2	Execute the Aws\S3\S3Client::getObject() method. You must provide a bucket name and a key name in the <code>array</code> parameter's required keys, <code>Bucket</code> and <code>Key</code> . Instead of retrieving the entire object you can retrieve a specific byte range from the object data. You provide the range value by specifying the <code>array</code> parameter's <code>Range</code> key in addition to the required keys. You can save the object you retrieved from Amazon S3 to a file in your local file system by specifying a file path to where to save the file in the <code>array</code> parameter's <code>SaveAs</code> key, in addition to the required keys, <code>Bucket</code> and <code>Key</code> .

The following PHP code sample demonstrates the preceding tasks for downloading an object.

```
use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';
$filepath = '*** Your File Path ***';

// Instantiate the client.
$s3 = S3Client::factory();

// Get an object.
$result = $s3->getObject(array(
    'Bucket' => $bucket,
    'Key'     => $keyname
));

// Get a range of bytes from an object.
$result = $s3->getObject(array(
    'Bucket' => $bucket,
```

```
'Key'      => $keyname,
'Range'    => 'bytes=0-99'
)) ;

// Save object to a file.
$result = $s3->getObject(array(
    'Bucket'  => $bucket,
    'Key'      => $keyname,
    'SaveAs'   => $filepath
));
```

When retrieving an object, you can optionally override the response header values (see [Getting Objects \(p. 130\)](#)) by adding the array parameter's response keys, ResponseContentType, ResponseContentLanguage, ResponseContentDisposition, ResponseCacheControl, and ResponseExpires, to the `getObject()` method, as shown in the following PHP code sample.

```
$result = $s3->getObject(array(
    'Bucket'          => $bucket,
    'Key'             => $keyname,
    'ResponseContentType' => 'text/plain',
    'ResponseContentLanguage' => 'en-US',
    'ResponseContentDisposition' => 'attachment; filename=testing.txt',
    'ResponseCacheControl'     => 'No-cache',
    'ResponseExpires'         => gmdate(DATE_RFC2822, time() + 3600),
));
```

Example of Downloading an Object Using PHP

The following PHP example retrieves an object and displays object content in the browser. The example illustrates the use of the `getObject()` method. For information about running the PHP examples in this guide, go to [Running PHP Examples \(p. 548\)](#).

```
<?php

// Include the AWS SDK using the Composer autoloader.
require 'vendor/autoload.php';

use Aws\S3\S3Client;
use Aws\S3\Exception\S3Exception;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

// Instantiate the client.
$s3 = S3Client::factory();

try {
    // Get the object
    $result = $s3->getObject(array(
        'Bucket' => $bucket,
        'Key'     => $keyname
    ));

    // Display the object in the browser
    header("Content-Type: {$result['ContentType']}");
    echo $result['Body'];
} catch (S3Exception $e) {
    echo $e->getMessage() . "\n";
}
```

Related Resources

- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::factory\(\) Method](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::getObject\(\) Method](#)
- [AWS SDK for PHP for Amazon S3 - Downloading Objects](#)
- [AWS SDK for PHP for Amazon S3](#)
- [AWS SDK for PHP Documentation](#)

Get an Object Using the REST API

You can use AWS SDK to list object keys in a bucket. However, if your application requires it, you can send REST requests directly. You can send a GET request to retrieve object keys. For more information about the request and response format, go to [Get Object](#).

Share an Object with Others

Topics

- [Generate a Pre-signed Object URL using AWS Explorer for Visual Studio \(p. 140\)](#)

- [Generate a Pre-signed Object URL using AWS SDK for Java \(p. 140\)](#)
- [Generate a Pre-signed Object URL using AWS SDK for .NET \(p. 143\)](#)

All objects by default are private. Only the object owner has permission to access these objects. However, the object owner can optionally share objects with others by creating a pre-signed URL, using their own security credentials, to grant time-limited permission to download the objects.

When you create a pre-signed URL for your object, you must provide your security credentials, specify a bucket name, an object key, specify the HTTP method (GET to download the object) and expiration date and time. The pre-signed URLs are valid only for the specified duration.

Anyone who receives the pre-signed URL can then access the object. For example, if you have a video in your bucket and both the bucket and the object are private, you can share the video with others by generating a pre-signed URL.

Note

Anyone with valid security credentials can create a pre-signed URL. However, in order to successfully access an object, the pre-signed URL must be created by someone who has permission to perform the operation that the pre-signed URL is based upon.

You can generate pre-signed URL programmatically using the AWS SDK for Java and .NET.

Generate a Pre-signed Object URL using AWS Explorer for Visual Studio

If you are using Visual Studio, you can generate a pre-signed URL for an object without writing any code by using AWS Explorer for Visual Studio. Anyone with this URL can download the object. For more information, go to [Using Amazon S3 from AWS Explorer](#).

For instructions about how to install the AWS Explorer, see [Using the AWS SDKs and Explorers \(p. 542\)](#).

Generate a Pre-signed Object URL using AWS SDK for Java

The following tasks guide you through using the Java classes to generate a pre-signed URL.

Downloading Objects

1	Create an instance of the <code>AmazonS3</code> class. For information about providing credentials, see Using the AWS SDK for Java (p. 544) . These credentials are used in creating a signature for authentication when you generate a pre-signed URL.
2	Execute the <code>AmazonS3.generatePresignedUrl</code> method to generate a pre-signed URL. You provide information including a bucket name, an object key, and an expiration date by creating an instance of the <code>GeneratePresignedUrlRequest</code> class. The request by default sets the verb to GET. To use the pre-signed URL for other operations, for example PUT, you must explicitly set the <code>verb</code> when you create the request.

The following Java code sample demonstrates the preceding tasks.

```
AmazonS3 s3Client = new AmazonS3Client(new ProfileCredentialsProvider());  
  
java.util.Date expiration = new java.util.Date();  
long msec = expiration.getTime();  
msec += 1000 * 60 * 60; // 1 hour.
```

```
expiration.setTime(msec);

GeneratePresignedUrlRequest generatePresignedUrlRequest =
    new GeneratePresignedUrlRequest(bucketName, objectKey);
generatePresignedUrlRequest.setMethod(HttpMethod.GET); // Default.
generatePresignedUrlRequest.setExpiration(expiration);

URL s = s3client.generatePresignedUrl(generatePresignedUrlRequest);
```

Example

The following Java code example generates a pre-signed URL that you can give to others so that they can retrieve the object. You can use the generated pre-signed URL to retrieve the object. To use the pre-signed URL for other operations, such as put an object, you must explicitly set the verb in the `GetPreSignedUrlRequest`. For instructions about how to create and test a working sample, see [Testing the Java Code Examples \(p. 545\)](#).

```
import java.io.IOException;
import java.net.URL;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.HttpMethod;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.GeneratePresignedUrlRequest;

public class GeneratePreSignedUrl {
    private static String bucketName = "**** Provide a bucket name ****";
    private static String objectKey = "**** Provide an object key ****";

    public static void main(String[] args) throws IOException {
        AmazonS3 s3client = new AmazonS3Client(new ProfileCredentialsProvider());

        try {
            System.out.println("Generating pre-signed URL.");
            java.util.Date expiration = new java.util.Date();
            long milliSeconds = expiration.getTime();
            milliSeconds += 1000 * 60 * 60; // Add 1 hour.
            expiration.setTime(milliSeconds);

            GeneratePresignedUrlRequest generatePresignedUrlRequest =
                new GeneratePresignedUrlRequest(bucketName, objectKey);
            generatePresignedUrlRequest.setMethod(HttpMethod.GET);
            generatePresignedUrlRequest.setExpiration(expiration);

            URL url = s3client.generatePresignedUrl(generatePresignedUrlRequest);

            System.out.println("Pre-Signed URL = " + url.toString());
        } catch (AmazonServiceException exception) {
            System.out.println("Caught an AmazonServiceException, " +
                "which means your request made it " +
                "to Amazon S3, but was rejected with an error response " +
                "for some reason.");
            System.out.println("Error Message: " + exception.getMessage());
            System.out.println("HTTP Code: " + exception.getStatusCode());
            System.out.println("AWS Error Code:" + exception.getErrorCode());
            System.out.println("Error Type: " + exception.getErrorType());
            System.out.println("Request ID: " + exception.getRequestId());
        } catch (AmazonClientException ace) {
            System.out.println("Caught an AmazonClientException, " +
                "which means the client encountered " +
                "an internal error while trying to communicate" +
                " with S3, " +
                "such as not being able to access the network.");
            System.out.println("Error Message: " + ace.getMessage());
        }
    }
}
```

```
    }  
}  
}
```

Generate a Pre-signed Object URL using AWS SDK for .NET

The following tasks guide you through using the .NET classes to generate a pre-signed URL.

Downloading Objects

1	Create an instance of the <code>AmazonS3</code> class. For information about providing your credentials see Using the AWS SDK for .NET (p. 546) . These credentials are used in creating a signature for authentication when you generate a pre-signed URL.
2	Execute the <code>AmazonS3.GetPreSignedURL</code> method to generate a pre-signed URL. You provide information including a bucket name, an object key, and an expiration date by creating an instance of the <code>GetPreSignedUrlRequest</code> class.

The following C# code sample demonstrates the preceding tasks.

```
static IAmazonS3 s3Client;  
s3Client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1)  
  
GetPreSignedUrlRequest request1 = new GetPreSignedUrlRequest()  
{  
    BucketName = bucketName,  
    Key = objectKey,  
    Expires = DateTime.Now.AddMinutes(5)  
};  
  
string url = s3Client.GetPreSignedURL(request);
```

Example

The following C# code example generates a pre-signed URL for a specific object. For instructions about how to create and test a working sample, see [Testing the .NET Code Examples \(p. 547\)](#).

```
using System;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples
{
    class GeneratePresignedURL
    {
        static string bucketName = "**** Provide a bucket name ****";
        static string objectKey = "**** Provide an object name ****";
        static IAmazonS3 s3Client;

        public static void Main(string[] args)
        {

            using (s3Client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1))
            {

                string urlString = GeneratePreSignedURL();
            }

            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }

        static string GeneratePreSignedURL()
        {
            string urlString = "";
            GetPreSignedUrlRequest request1 = new GetPreSignedUrlRequest
            {
                BucketName = bucketName,
                Key = objectKey,
                Expires = DateTime.Now.AddMinutes(5)
            };

            try
            {
                urlString = s3Client.GetPreSignedURL(request1);
                //string url = s3Client.GetPreSignedURL(request1);
            }
            catch (AmazonS3Exception amazonS3Exception)
            {
                if (amazonS3Exception.ErrorCode != null &&
                    (amazonS3Exception.ErrorCode.Equals("InvalidAccessKeyId")
                     ||
                     amazonS3Exception.ErrorCode.Equals("InvalidSecurity")))
                {
                    Console.WriteLine("Check the provided AWS Credentials.");
                    Console.WriteLine(
                        "To sign up for service, go to http://aws.amazon.com/s3");
                }
            }
        }
    }
}
```

```
        }
    else
    {
        Console.WriteLine(
            "Error occurred. Message:{0}' when listing objects",
            amazonS3Exception.Message);
    }
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}

return urlString;
}
}
```

Uploading Objects

Topics

- [Related Resources \(p. 145\)](#)
- [Uploading Objects in a Single Operation \(p. 146\)](#)
- [Uploading Objects Using Multipart Upload API \(p. 155\)](#)
- [Uploading Objects Using Pre-Signed URLs \(p. 191\)](#)

Depending on the size of the data you are uploading, Amazon S3 offers the following options:

- **Upload objects in a single operation**—With a single PUT operation you can upload objects up to 5 GB in size.
For more information, see [Uploading Objects in a Single Operation \(p. 146\)](#).
- **Upload objects in parts**—Using the Multipart upload API you can upload large objects, up to 5 TB. The Multipart Upload API is designed to improve the upload experience for larger objects. You can upload objects in parts. These object parts can be uploaded independently, in any order, and in parallel. You can use a Multipart Upload for objects from 5 MB to 5 TB in size. For more information, see [Uploading Objects Using Multipart Upload](#). For more information, see [Uploading Objects Using Multipart Upload API \(p. 155\)](#).

We encourage Amazon S3 customers to use Multipart Upload for objects greater than 100 MB.

When uploading objects you optionally request Amazon S3 to encrypt your object before saving it on disks in its data centers and decrypt it when you download the objects. For more information, see [Protecting Data Using Encryption \(p. 380\)](#).

Related Resources

- [Using the AWS SDKs and Explorers \(p. 542\)](#)

Uploading Objects in a Single Operation

Topics

- [Upload an Object Using the AWS SDK for Java \(p. 146\)](#)
- [Upload an Object Using the AWS SDK for .NET \(p. 147\)](#)
- [Upload an Object Using the AWS SDK for PHP \(p. 150\)](#)
- [Upload an Object Using the AWS SDK for Ruby \(p. 152\)](#)

You can use the AWS SDK to upload objects. The SDK provides wrapper libraries for you to upload data easily. However, if your application requires it, you can use the REST API directly in your application.

Upload an Object Using the AWS SDK for Java

The following tasks guide you through using the Java classes to upload a file. The API provides several variations, called *overloads*, of the `putObject` method to easily upload your data.

Uploading Objects

1	Create an instance of the <code>AmazonS3Client</code> .
2	Execute one of the <code>AmazonS3Client.putObject</code> overloads depending on whether you are uploading data from a file, or a stream.

The following Java code sample demonstrates the preceding tasks.

```
AmazonS3 s3client = new AmazonS3Client(new ProfileCredentialsProvider());  
s3client.putObject(new PutObjectRequest(bucketName, keyName, file));
```

Example

The following Java code example uploads a file to an Amazon S3 bucket. For instructions on how to create and test a working sample, see [Testing the Java Code Examples \(p. 545\)](#).

```
import java.io.File;
import java.io.IOException;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.PutObjectRequest;

public class UploadObjectSingleOperation {
    private static String bucketName      = "**** Provide bucket name ***";
    private static String keyName         = "**** Provide key ***";
    private static String uploadFileName = "**** Provide file name ***";

    public static void main(String[] args) throws IOException {
        AmazonS3 s3client = new AmazonS3Client(new ProfileCredentialsProvider());

        try {
            System.out.println("Uploading a new object to S3 from a file\n");
            File file = new File(uploadFileName);
            s3client.putObject(new PutObjectRequest(
                bucketName, keyName, file));

        } catch (AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException, which " +
                "means your request made it " +
                "to Amazon S3, 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 " +
                "an internal error while trying to " +
                "communicate with S3, " +
                "such as not being able to access the network.");
            System.out.println("Error Message: " + ace.getMessage());
        }
    }
}
```

Upload an Object Using the AWS SDK for .NET

The tasks in the following process guide you through using the .NET classes to upload an object. The API provides several variations, overloads, of the `PutObject` method to easily upload your data.

Uploading Objects

1	Create an instance of the <code>AmazonS3</code> class.
2	Execute one of the <code>AmazonS3.PutObject</code> . You need to provide information such as a bucket name, file path, or a stream. You provide this information by creating an instance of the <code>PutObjectRequest</code> class.

The following C# code sample demonstrates the preceding tasks.

```
static IAmazonS3 client;
client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1);
PutObjectRequest request = new PutObjectRequest()
{
    BucketName = bucketName,
    Key = keyName,
    FilePath = filePath
};
PutObjectResponse response2 = client.PutObject(request);
```

Example

The following C# code example uploads an object. The object data is provided as a text string in the code. The example uploads the object twice.

- The first `PutObjectRequest` specifies only the bucket name, key name, and a text string embedded in the code as sample object data.
- The second `PutObjectRequest` provides additional information including the optional object metadata and a `ContentType` header. The request specifies a file name to upload.

Each successive call to `AmazonS3.PutObject` replaces the previous upload. For instructions on how to create and test a working sample, see [Testing the .NET Code Examples \(p. 547\)](#).

```
using System;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples
{
    class UploadObject
    {
        static string bucketName = "**** bucket name ****";
        static string keyName    = "**** key name when object is created ****";
        static string filePath   = "**** absolute path to a sample file to upload
****";

        static IAmazonS3 client;

        public static void Main(string[] args)
        {
            using (client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1))

            {
                Console.WriteLine("Uploading an object");
                WritingAnObject();
            }

            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }

        static void WritingAnObject()
        {
            try
            {
                PutObjectRequest putRequest1 = new PutObjectRequest
                {
                    BucketName = bucketName,
                    Key = keyName,
                    ContentBody = "sample text"
                };

                PutObjectResponse response1 = client.PutObject(putRequest1);

                // 2. Put object-set ContentType and add metadata.
                PutObjectRequest putRequest2 = new PutObjectRequest
                {
```

```
        BucketName = bucketName,
        Key = keyName,
        FilePath = filePath,
        ContentType = "text/plain"
    };
    putRequest2.Metadata.Add("x-amz-meta-title", "someTitle");

    PutObjectResponse response2 = client.PutObject(putRequest2);

}
catch (AmazonS3Exception amazonS3Exception)
{
    if (amazonS3Exception.ErrorCode != null &&
        (amazonS3Exception.ErrorCode.Equals("InvalidAccessKeyId")
        ||
        amazonS3Exception.ErrorCode.Equals("InvalidSecurity")))
    {
        Console.WriteLine("Check the provided AWS Credentials.");
        Console.WriteLine(
            "For service sign up go to http://aws.amazon.com/s3");
    }
    else
    {
        Console.WriteLine(
            "Error occurred. Message:'{0}' when writing an object"
            , amazonS3Exception.Message);
    }
}
}
```

Upload an Object Using the AWS SDK for PHP

This topic guides you through using classes from the AWS SDK for PHP to upload an object of up to 5 GB in size. For larger files you must use multipart upload API. For more information, see [Uploading Objects Using Multipart Upload API \(p. 155\)](#).

Note

This topic assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 547\)](#) and have the AWS SDK for PHP properly installed.

Uploading Objects

1	Create an instance of an Amazon S3 client by using the Aws\S3\S3Client class factory() method.
2	Execute the Aws\S3\S3Client::putObject() method. You must provide a bucket name and a key name in the array parameter's required keys, Bucket and Key. If you are uploading a file, you specify the file name by adding the array parameter with the SourceFile key. You can also provide the optional object metadata using the array parameter.

The following PHP code sample demonstrates how to create an object by uploading a file specified in the `SourceFile` key in the `putObject` method's array parameter.

```
use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';
// $filepath should be absolute path to a file on disk
$filepath = '*** Your File Path ***';

// Instantiate the client.
$s3 = S3Client::factory();

// Upload a file.
$result = $s3->putObject(array(
    'Bucket'      => $bucket,
    'Key'         => $keyname,
    'SourceFile'   => $filepath,
    'ContentType' => 'text/plain',
    'ACL'          => 'public-read',
    'StorageClass' => 'REDUCED_REDUNDANCY',
    'Metadata'     => array(
        'param1' => 'value 1',
        'param2' => 'value 2'
    )
));
echo $result['ObjectURL'];
```

Instead of specifying a file name, you can provide object data inline by specifying the array parameter with the `Body` key, as shown in the following PHP code example.

```
use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

// Instantiate the client.
$s3 = S3Client::factory();

// Upload data.
$result = $s3->putObject(array(
    'Bucket' => $bucket,
    'Key'     => $keyname,
    'Body'    => 'Hello, world!'
));
echo $result['ObjectURL'];
```

Example of Creating an Object in an Amazon S3 bucket by Uploading Data

The following PHP example creates an object in a specified bucket by uploading data using the `putObject()` method. For information about running the PHP examples in this guide, go to [Running PHP Examples \(p. 548\)](#).

```
<?php

// Include the AWS SDK using the Composer autoloader.
require 'vendor/autoload.php';

use Aws\S3\S3Client;
use Aws\S3\Exception\S3Exception;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

// Instantiate the client.
$s3 = S3Client::factory();

try {
    // Upload data.
    $result = $s3->putObject(array(
        'Bucket' => $bucket,
        'Key'     => $keyname,
        'Body'    => 'Hello, world!',
        'ACL'     => 'public-read'
    ));

    // Print the URL to the object.
    echo $result['ObjectURL'] . "\n";
} catch (S3Exception $e) {
    echo $e->getMessage() . "\n";
}
```

Related Resources

- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::factory\(\) Method](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::putObject\(\) Method](#)
- [AWS SDK for PHP for Amazon S3](#)
- [AWS SDK for PHP Documentation](#)

Upload an Object Using the AWS SDK for Ruby

The following tasks guide you through using a Ruby script to upload an object for either version of the SDK for Ruby.

Using AWS SDK for Ruby - Version 2

The AWS SDK for Ruby - Version 2 has two ways of uploading an object to Amazon S3. The first is a managed file uploader, which makes it easy to upload files of any size from disk.

Uploading a File

1	Create an instance of the <code>Aws::S3::Resource</code> class.
2	Reference the target object by bucket name and key.
2	Call <code>#upload_file</code> on the object.

```
require 'aws-sdk'

s3 = Aws::S3::Resource.new(region:'us-west-2')
obj = s3.bucket('bucket-name').object('key')
obj.upload_file('/path/to/source/file')
```

The second way that SDK for Ruby - Version 2 can upload an object is to use the `#put` method of `Aws::S3::Object`. This is useful if the object is a string or an IO object that is not a file on disk.

Put Object

1	Create an instance of the <code>Aws::S3::Resource</code> class.
2	Reference the target object by bucket name and key.
2	Call <code>#put</code> passing in teh string or IO object.

```
require 'aws-sdk'

s3 = Aws::S3::Resource.new(region:'us-west-2')
obj = s3.bucket('bucket-name').object('key')

# string data
obj.put(body: 'Hello World!')

# IO object
File.open('source', 'rb') do |file|
  obj.put(body: file)
end
```

Using AWS SDK for Ruby - Version 1

The API provides a `#write` method that can take options that you can use to specify how to upload your data.

Uploading Objects - SDK for Ruby - Version 1

1	Create an instance of the <code>AWS::S3</code> class by providing your AWS credentials.
2	Use the <code>AWS::S3::S3Object#write</code> method which takes a <code>data</code> parameter and <code>options</code> hash which allow you to upload data from a file, or a stream.

The following code sample for the SDK for Ruby - Version 1 demonstrates the preceding tasks and uses the `options` hash `:file` to specify the path to the file to upload.

```
s3 = AWS::S3.new

# Upload a file.
key = File.basename(file_name)
s3.buckets[bucket_name].objects[key].write(:file => file_name)
```

Example

The following SDK for Ruby - Version 1 script example uploads a file to an Amazon S3 bucket. For instructions about how to create and test a working sample, see [Using the AWS SDK for Ruby \(p. 549\)](#).

```
#!/usr/bin/env ruby

require 'rubygems'
require 'aws-sdk'

bucket_name = '*** Provide bucket name ***'
file_name = '*** Provide file name ***'

# Get an instance of the S3 interface.
s3 = AWS::S3.new

# Upload a file.
key = File.basename(file_name)
s3.buckets[bucket_name].objects[key].write(:file => file_name)
puts "Uploading file #{file_name} to bucket #{bucket_name}."
```

Upload an Object Using the REST API

You can use AWS SDK to upload an object. However, if your application requires it, you can send REST requests directly. You can send a PUT request to upload data in a single operation. For more information, go to [PUT Object](#).

Uploading Objects Using Multipart Upload API

Topics

- [Using the AWS SDK for Java for Multipart Upload \(p. 156\)](#)
- [Using the AWS SDK for .NET for Multipart Upload \(p. 166\)](#)
- [Using the AWS SDK for PHP for Multipart Upload \(p. 181\)](#)
- [Using the AWS SDK for Ruby for Multipart Upload \(p. 190\)](#)
- [Using the REST API for Multipart Upload \(p. 190\)](#)

Multipart upload allows you to upload a single object as a set of parts. Each part is a contiguous portion of the object's data. You can upload these object parts independently and in any order. If transmission of any part fails, you can retransmit that part without affecting other parts. After all parts of your object are uploaded, Amazon S3 assembles these parts and creates the object. In general, when your object size reaches 100 MB, you should consider using multipart uploads instead of uploading the object in a single operation.

Using multipart upload provides the following advantages:

- **Improved throughput**—You can upload parts in parallel to improve throughput.
- **Quick recovery from any network issues**—Smaller part size minimizes the impact of restarting a failed upload due to a network error.
- **Pause and resume object uploads**—You can upload object parts over time. Once you initiate a multipart upload there is no expiry; you must explicitly complete or abort the multipart upload.
- **Begin an upload before you know the final object size**—You can upload an object as you are creating it.

For more information, see [Multipart Upload Overview \(p. 126\)](#).

Using the AWS SDK for Java for Multipart Upload

Topics

- [Using the High-Level Java API for Multipart Upload \(p. 156\)](#)
- [Using the Low-Level Java API for Multipart Upload \(p. 161\)](#)

As described in Using the AWS SDK (see [Using the AWS SDKs and Explorers \(p. 542\)](#)), the SDK for Java provides support for the multipart upload API (see [Uploading Objects Using Multipart Upload API \(p. 155\)](#)). This section provides examples of using both the high-level and low-level SDK for Java API for uploading objects in parts.

Using the High-Level Java API for Multipart Upload

Topics

- [Upload a File \(p. 156\)](#)
- [Abort Multipart Uploads \(p. 157\)](#)
- [Track Multipart Upload Progress \(p. 158\)](#)

The AWS SDK for Java exposes a high-level API that simplifies multipart upload (see [Uploading Objects Using Multipart Upload API \(p. 155\)](#)). You can upload data from a file or a stream. You can also set advanced options, such as the part size you want to use for the multipart upload, or the number of threads you want to use when uploading the parts concurrently. You can also set optional object properties, the storage class, or ACL. You use the `PutObjectRequest` and the `TransferManagerConfiguration` classes to set these advanced options. The `TransferManager` class of the Java API provides the high-level API for you to upload data.

When possible, `TransferManager` attempts to use multiple threads to upload multiple parts of a single upload at once. When dealing with large content sizes and high bandwidth, this can have a significant increase on throughput.

In addition to file upload functionality, the `TransferManager` class provides a method for you to abort multipart upload in progress. You must provide a `Date` value, and then the API aborts all the multipart uploads that were initiated before the specified date.

Upload a File

The following tasks guide you through using the high-level Java classes to upload a file. The API provides several variations, called *overloads*, of the `upload` method to easily upload your data.

High-Level API File Uploading Process

1	Create an instance of the <code>TransferManager</code> class.
2	Execute one of the <code>TransferManager.upload</code> overloads depending on whether you are uploading data from a file, or a stream.

The following Java code example demonstrates the preceding tasks.

Example

The following Java code example uploads a file to an Amazon S3 bucket. For instructions on how to create and test a working sample, see [Testing the Java Code Examples \(p. 545\)](#).

```
import java.io.File;

import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.Upload;

public class UploadObjectMultipartUploadUsingHighLevelAPI {

    public static void main(String[] args) throws Exception {
        String existingBucketName = "**** Provide existing bucket name ****";
        String keyName           = "**** Provide object key ****";
        String filePath          = "**** Path to and name of the file to upload
****";

        TransferManager tm = new TransferManager(new ProfileCredentialsProvider());
        System.out.println("Hello");
        // TransferManager processes all transfers asynchronously,
        // so this call will return immediately.
        Upload upload = tm.upload(
            existingBucketName, keyName, new File(filePath));
        System.out.println("Hello2");

        try {
            // Or you can block and wait for the upload to finish
            upload.waitForCompletion();
            System.out.println("Upload complete.");
        } catch (AmazonClientException amazonClientException) {
            System.out.println("Unable to upload file, upload was aborted.");
            amazonClientException.printStackTrace();
        }
    }
}
```

Abort Multipart Uploads

The `TransferManager` class provides a method, `abortMultipartUploads`, to abort multipart uploads in progress. An upload is considered to be in progress once you initiate it and until you complete it or abort it. You provide a `Date` value and this API aborts all the multipart uploads, on that bucket, that were initiated before the specified `Date` and are still in progress.

Because you are billed for all storage associated with uploaded parts (see [Multipart Upload and Pricing \(p. 128\)](#)), it is important that you either complete the multipart upload to have the object created or abort the multipart upload to remove any uploaded parts.

The following tasks guide you through using the high-level Java classes to abort multipart uploads.

High-Level API Multipart Uploads Aborting Process

1	Create an instance of the <code>TransferManager</code> class.
---	---

- | | |
|---|---|
| 2 | Execute the <code>TransferManager.abortMultipartUploads</code> method by passing the bucket name and a <code>Date</code> value. |
|---|---|

The following Java code example demonstrates the preceding tasks.

Example

The following Java code aborts all multipart uploads in progress that were initiated on a specific bucket over a week ago. For instructions on how to create and test a working sample, see [Testing the Java Code Examples \(p. 545\)](#).

```
import java.util.Date;

import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.transfer.TransferManager;

public class AbortMPUUsingHighLevelAPI {

    public static void main(String[] args) throws Exception {
        String existingBucketName = "**** Provide existing bucket name ***";

        TransferManager tm = new TransferManager(new ProfileCredentialsProvider());

        int sevenDays = 1000 * 60 * 60 * 24 * 7;
        Date oneWeekAgo = new Date(System.currentTimeMillis() - sevenDays);

        try {
            tm.abortMultipartUploads(existingBucketName, oneWeekAgo);
        } catch (AmazonClientException amazonClientException) {
            System.out.println("Unable to upload file, upload was aborted.");
            amazonClientException.printStackTrace();
        }
    }
}
```

Note

You can also abort a specific multipart upload. For more information, see [Abort a Multipart Upload \(p. 164\)](#).

Track Multipart Upload Progress

The high-level multipart upload API provides a listen interface, `ProgressListener`, to track the upload progress when uploading data using the `TransferManager` class. To use the event in your code, you must import the `com.amazonaws.services.s3.model.ProgressEvent` and `com.amazonaws.services.s3.model.ProgressListener` types.

Progress events occurs periodically and notify the listener that bytes have been transferred.

The following Java code sample demonstrates how you can subscribe to the `ProgressEvent` event and write a handler.

```
TransferManager tm = new TransferManager(new ProfileCredentialsProvider());
```

```
PutObjectRequest request = new PutObjectRequest(  
    existingBucketName, keyName, new File(filePath));  
  
// Subscribe to the event and provide event handler.  
request.setProgressListener(new ProgressListener() {  
    public void progressChanged(ProgressEvent event) {  
        System.out.println("Transferred bytes: " +  
            event.getBytesTransferred());  
    }  
});
```

Example

The following Java code uploads a file and uses the `ProgressListener` to track the upload progress. For instructions on how to create and test a working sample, see [Testing the Java Code Examples \(p. 545\)](#).

```
import java.io.File;

import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.event.ProgressEvent;
import com.amazonaws.event.ProgressListener;
import com.amazonaws.services.s3.model.PutObjectRequest;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.Upload;

public class TrackMPUProgressUsingHighLevelAPI {

    public static void main(String[] args) throws Exception {
        String existingBucketName = "**** Provide bucket name ****";
        String keyName           = "**** Provide object key ****";
        String filePath          = "**** file to upload ***";

        TransferManager tm = new TransferManager(new ProfileCredentialsProvider());

        // For more advanced uploads, you can create a request object
        // and supply additional request parameters (ex: progress listeners,
        // canned ACLs, etc.)
        PutObjectRequest request = new PutObjectRequest(
            existingBucketName, keyName, new File(filePath));

        // You can ask the upload for its progress, or you can
        // add a ProgressListener to your request to receive notifications
        // when bytes are transferred.
        request.setGeneralProgressListener(new ProgressListener() {
            @Override
            public void progressChanged(ProgressEvent progressEvent) {
                System.out.println("Transferred bytes: " +
                    progressEvent.getBytesTransferred());
            }
        });

        // TransferManager processes all transfers asynchronously,
        // so this call will return immediately.
        Upload upload = tm.upload(request);

        try {
            // You can block and wait for the upload to finish
            upload.waitForCompletion();
        } catch (AmazonClientException amazonClientException) {
            System.out.println("Unable to upload file, upload aborted.");
            amazonClientException.printStackTrace();
        }
    }
}
```

Using the Low-Level Java API for Multipart Upload

Topics

- [Upload a File \(p. 161\)](#)
- [List Multipart Uploads \(p. 164\)](#)
- [Abort a Multipart Upload \(p. 164\)](#)

The AWS SDK for Java exposes a low-level API that closely resembles the Amazon S3 REST API for multipart upload (see [Uploading Objects Using Multipart Upload API \(p. 155\)](#)). Use the low-level API when you need to pause and resume multipart uploads, vary part sizes during the upload, or do not know the size of the data in advance. Use the high-level API (see [Using the High-Level Java API for Multipart Upload \(p. 156\)](#)) whenever you don't have these requirements.

Upload a File

The following tasks guide you through using the low-level Java classes to upload a file.

Low-Level API File Uploading Process

1	Create an instance of the <code>AmazonS3Client</code> class.
2	Initiate multipart upload by executing the <code>AmazonS3Client.initiateMultipartUpload</code> method. You will need to provide the required information, i.e., bucket name and key name, to initiate the multipart upload by creating an instance of the <code>InitiateMultipartUploadRequest</code> class.
3	Save the upload ID that the <code>AmazonS3Client.initiateMultipartUpload</code> method returns. You will need to provide this upload ID for each subsequent multipart upload operation.
4	Upload parts. For each part upload, execute the <code>AmazonS3Client.uploadPart</code> method. You need to provide part upload information, such as upload ID, bucket name, and the part number. You provide this information by creating an instance of the <code>UploadPartRequest</code> class.
5	Save the response of the <code>AmazonS3Client.uploadPart</code> method in a list. This response includes the ETag value and the part number you will need to complete the multipart upload.
6	Repeat tasks 4 and 5 for each part.
7	Execute the <code>AmazonS3Client.completeMultipartUpload</code> method to complete the multipart upload.

The following Java code sample demonstrates the preceding tasks.

```
AmazonS3 s3Client = new AmazonS3Client(new ProfileCredentialsProvider());  
  
// Create a list of UploadPartResponse objects. You get one of these for  
// each part upload.  
List<PartETag> partETags = new ArrayList<PartETag>();  
  
// Step 1: Initialize.  
InitiateMultipartUploadRequest initRequest = new InitiateMultipartUploadRequest(  
    existingBucketName, keyName);  
InitiateMultipartUploadResult initResponse =
```

```
s3Client.initiateMultipartUpload(initRequest);

File file = new File(filePath);
long contentLength = file.length();
long partSize = 5 * 1024 * 1024; // Set part size to 5 MB.

try {
    // Step 2: Upload parts.
    long filePosition = 0;
    for (int i = 1; filePosition < contentLength; i++) {
        // Last part can be less than 5 MB. Adjust part size.
        partSize = Math.min(partSize, (contentLength - filePosition));

        // Create request to upload a part.
        UploadPartRequest uploadRequest = new UploadPartRequest()
            .withBucketName(existingBucketName).withKey(keyName)
            .withUploadId(initResponse.getUploadId()).withPartNumber(i)
            .withFileOffset(filePosition)
            .withFile(file)
            .withPartSize(partSize);

        // Upload part and add response to our list.
        partETags.add(s3Client.uploadPart(uploadRequest).getPartETag());

        filePosition += partSize;
    }

    // Step 3: Complete.
    CompleteMultipartUploadRequest compRequest = new
        CompleteMultipartUploadRequest(existingBucketName,
            keyName,
            initResponse.getUploadId(),
            partETags);

    s3Client.completeMultipartUpload(compRequest);
} catch (Exception e) {
    s3Client.abortMultipartUpload(new AbortMultipartUploadRequest(
        existingBucketName, keyName, initResponse.getUploadId()));
}
```

Example

The following Java code example uploads a file to an Amazon S3 bucket. For instructions on how to create and test a working sample, see [Testing the Java Code Examples \(p. 545\)](#).

```
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.AbortMultipartUploadRequest;
import com.amazonaws.services.s3.model.CompleteMultipartUploadRequest;
import com.amazonaws.services.s3.model.InitiateMultipartUploadRequest;
import com.amazonaws.services.s3.model.InitiateMultipartUploadResult;
import com.amazonaws.services.s3.model.PartETag;
import com.amazonaws.services.s3.model.UploadPartRequest;

public class UploadObjectMPULowLevelAPI {

    public static void main(String[] args) throws IOException {
        String existingBucketName = "**** Provide-Your-Existing-BucketName ***";

        String keyName           = "**** Provide-Key-Name ***";
        String filePath          = "**** Provide-File-Path ***";

        AmazonS3 s3Client = new AmazonS3Client(new ProfileCredentialsProvider());

        // Create a list of UploadPartResponse objects. You get one of these
        // for each part upload.
        List<PartETag> partETags = new ArrayList<PartETag>();

        // Step 1: Initialize.
        InitiateMultipartUploadRequest initRequest = new
            InitiateMultipartUploadRequest(existingBucketName, keyName);
        InitiateMultipartUploadResult initResponse =
            s3Client.initiateMultipartUpload(initRequest);

        File file = new File(filePath);
        long contentLength = file.length();
        long partSize = 5242880; // Set part size to 5 MB.

        try {
            // Step 2: Upload parts.
            long filePosition = 0;
            for (int i = 1; filePosition < contentLength; i++) {
                // Last part can be less than 5 MB. Adjust part size.
                partSize = Math.min(partSize, (contentLength - filePosition));

                // Create request to upload a part.
                UploadPartRequest uploadRequest = new UploadPartRequest()
                    .withBucketName(existingBucketName).withKey(keyName)
                    .withUploadId(initResponse.getUploadId()).withPartNumber(i)

                    .withFileOffset(filePosition)
        }
    }
}
```

```
        .withFile(file)
        .withPartSize(partSize);

    // Upload part and add response to our list.
    partETags.add(
        s3Client.uploadPart(uploadRequest).getPartETag() );

    filePosition += partSize;
}

// Step 3: Complete.
CompleteMultipartUploadRequest compRequest = new
    CompleteMultipartUploadRequest(
        existingBucketName,
        keyName,
        initResponse.getUploadId(),
        partETags);

    s3Client.completeMultipartUpload(compRequest);
} catch (Exception e) {
    s3Client.abortMultipartUpload(new AbortMultipartUploadRequest(
        existingBucketName, keyName, initResponse.getUploadId()));
}

}
```

List Multipart Uploads

The following tasks guide you through using the low-level Java classes to list all in-progress multipart uploads on a bucket.

Low-Level API Multipart Uploads Listing Process

1	Create an instance of the <code>ListMultipartUploadsRequest</code> class and provide the bucket name.
2	Execute the <code>AmazonS3Client.listMultipartUploads</code> method. The method returns an instance of the <code>MultipartUploadListing</code> class that gives you information about the multipart uploads in progress.

The following Java code sample demonstrates the preceding tasks.

```
ListMultipartUploadsRequest allMultipartUploadsRequest =
    new ListMultipartUploadsRequest(existingBucketName);
MultipartUploadListing multipartUploadListing =
    s3Client.listMultipartUploads(allMultipartUploadsRequest);
```

Abort a Multipart Upload

You can abort an in-progress multipart upload by calling the `AmazonS3.abortMultipartUpload` method. This method deletes any parts that were uploaded to Amazon S3 and frees up the resources. You must provide the upload ID, bucket name, and key name. The following Java code sample demonstrates how to abort an in-progress multipart upload.

```
InitiateMultipartUploadRequest initRequest =  
    new InitiateMultipartUploadRequest(existingBucketName, keyName);  
InitiateMultipartUploadResult initResponse =  
    s3Client.initiateMultipartUpload(initRequest);  
  
AmazonS3 s3Client = new AmazonS3Client(new ProfileCredentialsProvider());  
s3Client.abortMultipartUpload(new AbortMultipartUploadRequest(  
    existingBucketName, keyName, initResponse.getUploadId()));
```

Note

Instead of a specific multipart upload, you can abort all your multipart uploads initiated before a specific time that are still in progress. This clean-up operation is useful to abort old multipart uploads that you initiated but neither completed nor aborted. For more information, see [Abort Multipart Uploads \(p. 157\)](#).

Using the AWS SDK for .NET for Multipart Upload

Topics

- [Using the High-Level .NET API for Multipart Upload \(p. 166\)](#)
- [Using the Low-Level .NET API for Multipart Upload \(p. 175\)](#)

As described in Using the AWS SDK (see [Using the AWS SDKs and Explorers \(p. 542\)](#)), the SDK for .NET provides support for the multipart upload API (see [Uploading Objects Using Multipart Upload API \(p. 155\)](#)). This section provides examples of using the high-level and the low-level .NET API for uploading objects in parts.

Using the High-Level .NET API for Multipart Upload

Topics

- [Upload a File \(p. 166\)](#)
- [Upload a Directory \(p. 169\)](#)
- [Abort Multipart Uploads \(p. 171\)](#)
- [Track Multipart Upload Progress \(p. 172\)](#)

The AWS SDK for .NET exposes a high-level API that simplifies multipart upload (see [Uploading Objects Using Multipart Upload API \(p. 155\)](#)). You can upload data from a file, directory, or a stream. When uploading data from a file, if you don't provide the object's key name, the API uses the file name for the object's key name. You must provide the object's key name if you are uploading data from a stream. You can optionally set advanced options such as the part size you want to use for the multipart upload, number of threads you want to use when uploading the parts concurrently, optional file metadata, the storage class (STANDARD or REDUCED_REDUNDANCY), or ACL. The high-level API provides the `TransferUtilityUploadRequest` class to set these advanced options.

The `TransferUtility` class provides a method for you to abort multipart uploads in progress. You must provide a `DateTime` value, and then the API aborts all the multipart uploads that were initiated before the specified date and time.

Upload a File

The following tasks guide you through using the high-level .NET classes to upload a file. The API provides several variations, *overloads*, of the `Upload` method to easily upload your data.

High-Level API File Uploading Process

1	Create an instance of the <code>TransferUtility</code> class by providing your AWS credentials.
2	Execute one of the <code>TransferUtility.Upload</code> overloads depending on whether you are uploading data from a file, a stream, or a directory.

The following C# code sample demonstrates the preceding tasks.

```
TransferUtility utility = new TransferUtility();
utility.Upload(filePath, existingBucketName);
```

When uploading large files using the .NET API, timeout might occur even while data is being written to the request stream. You can set explicit timeout using the `TransferUtilityConfig.DefaultTimeout` as demonstrated in the following C# code sample.

```
TransferUtilityConfig config = new TransferUtilityConfig();
config.DefaultTimeout = 11111;
TransferUtility utility = new TransferUtility(config);
```

Example

The following C# code example uploads a file to an Amazon S3 bucket. The example illustrates the use of various `TransferUtility.Upload` overloads to upload a file; each successive call to upload replaces the previous upload. For instructions on how to create and test a working sample, see [Testing the .NET Code Examples \(p. 547\)](#)

```
using System;
using System.IO;
using Amazon.S3;
using Amazon.S3.Transfer;

namespace s3.amazon.com.docsamples
{
    class UploadFileMPUHighLevelAPI
    {
        static string existingBucketName = "**** Provide bucket name ****";
        static string keyName           = "**** Provide your object key ****";
        static string filePath          = "**** Provide file name ****";

        static void Main(string[] args)
        {
            try
            {
                TransferUtility fileTransferUtility = new
                    TransferUtility(new AmazonS3Client(Amazon.RegionEndpoint.USEast1));

                // 1. Upload a file, file name is used as the object key name.

                fileTransferUtility.Upload(filePath, existingBucketName);
                Console.WriteLine("Upload 1 completed");

                // 2. Specify object key name explicitly.
                fileTransferUtility.Upload(filePath,
                                           existingBucketName, keyName);
                Console.WriteLine("Upload 2 completed");

                // 3. Upload data from a type of System.IO.Stream.
                using (FileStream fileToUpload =
                    new FileStream(filePath, FileMode.Open, FileAccess.Read))
                {
                    fileTransferUtility.Upload(fileToUpload,
                                               existingBucketName, keyName);
                }
                Console.WriteLine("Upload 3 completed");

                // 4.Specify advanced settings/options.
                TransferUtilityUploadRequest fileTransferUtilityRequest = new
TransferUtilityUploadRequest
                {
                    BucketName = existingBucketName,
                    FilePath = filePath,
                    StorageClass = S3StorageClass.ReducedRedundancy,
                    PartSize = 6291456, // 6 MB.
                    Key = keyName,
                    CannedACL = S3CannedACL.PublicRead
                };
            }
        }
    }
}
```

```
        fileTransferUtilityRequest.Metadata.Add("param1", "Value1");
        fileTransferUtilityRequest.Metadata.Add("param2", "Value2");
        fileTransferUtility.Upload(fileTransferUtilityRequest);
        Console.WriteLine("Upload 4 completed");
    }
    catch (AmazonS3Exception s3Exception)
    {
        Console.WriteLine(s3Exception.Message,
                          s3Exception.InnerException);
    }
}
}
```

Upload a Directory

Using the `TransferUtility` class you can also upload an entire directory. By default, Amazon S3 only uploads the files at the root of the specified directory. You can, however, specify to recursively upload files in all the subdirectories.

You can also specify filtering expressions to select files, in the specified directory, based on some filtering criteria. For example, to upload only the `.pdf` files from a directory you specify a `"*.pdf"` filter expression.

When uploading files from a directory you cannot specify the object's key name. It is constructed from the file's location in the directory as well as its name. For example, assume you have a directory, `c:\myfolder`, with the following structure:

```
C:\myfolder
  \a.txt
  \b.pdf
  \media\
      An.mp3
```

When you upload this directory, Amazon S3 uses the following key names:

```
a.txt
b.pdf
media/An.mp3
```

The following tasks guide you through using the high-level .NET classes to upload a directory.

High-Level API Directory Uploading Process

1	Create an instance of the <code>TransferUtility</code> class by providing your AWS credentials.
2	Execute one of the <code>TransferUtility.UploadDirectory</code> overloads.

The following C# code sample demonstrates the preceding tasks.

```
TransferUtility utility = new TransferUtility();
utility.UploadDirectory(directoryPath, existingBucketName);
```

Example

The following C# code example uploads a directory to an Amazon S3 bucket. The example illustrates the use of various `TransferUtility.UploadDirectory` overloads to upload a directory, each successive call to upload replaces the previous upload. For instructions on how to create and test a working sample, see [Testing the .NET Code Examples \(p. 547\)](#).

```
using System;
using System.IO;
using Amazon.S3;
using Amazon.S3.Transfer;

namespace s3.amazon.com.docsamples
{
    class UploadDirectoryMPUHighLevelAPI
    {
        static string existingBucketName = "**** Provide bucket name ****";
        static string directoryPath      = "**** Provide directory name ****";

        static void Main(string[] args)
        {
            try
            {
                TransferUtility directoryTransferUtility =
                    new TransferUtility(new AmazonS3Client(Amazon.RegionEndpoint.USEast1));

                // 1. Upload a directory.
                directoryTransferUtility.UploadDirectory(directoryPath,
                                                existingBucketName);
                Console.WriteLine("Upload statement 1 completed");

                // 2. Upload only the .txt files from a directory.
                //     Also, search recursively.
                directoryTransferUtility.UploadDirectory(
                    directoryPath,
                    existingBucketName,
                    ".txt",
                    SearchOption.AllDirectories);
                Console.WriteLine("Upload statement 2 completed");

                // 3. Same as 2 and some optional configuration
                //     Search recursively for .txt files to upload).
                TransferUtilityUploadDirectoryRequest request =
                    new TransferUtilityUploadDirectoryRequest
                    {
                        BucketName = existingBucketName,
                        Directory = directoryPath,
                        SearchOption = SearchOption.AllDirectories,
                        SearchPattern = ".txt"
                    };

                directoryTransferUtility.UploadDirectory(request);
                Console.WriteLine("Upload statement 3 completed");
            }

            catch (AmazonS3Exception e)
            {

```

```
        Console.WriteLine(e.Message, e.InnerException);
    }
}
}
```

Abort Multipart Uploads

The `TransferUtility` class provides a method, `AbortMultipartUploads`, to abort multipart uploads in progress. An upload is considered to be in-progress once you initiate it and until you complete it or abort it. You provide a `DateTime` value and this API aborts all the multipart uploads, on that bucket, that were initiated before the specified `DateTime` and in progress.

Because you are billed for all storage associated with uploaded parts (see [Multipart Upload and Pricing \(p. 128\)](#)), it is important that you either complete the multipart upload to have the object created or abort the multipart upload to remove any uploaded parts.

The following tasks guide you through using the high-level .NET classes to abort multipart uploads.

High-Level API Multipart Uploads Aborting Process

1	Create an instance of the <code>TransferUtility</code> class by providing your AWS credentials.
2	Execute the <code>TransferUtility.AbandonMultipartUploads</code> method by passing the bucket name and a <code>DateTime</code> value.

The following C# code sample demonstrates the preceding tasks.

```
TransferUtility utility = new TransferUtility();
utility.AbandonMultipartUploads(existingBucketName, DateTime.Now.AddDays(-7));
```

Example

The following C# code aborts all multipart uploads in progress that were initiated on a specific bucket over a week ago. For instructions on how to create and test a working sample, see [Testing the .NET Code Examples \(p. 547\)](#).

```
using System;
using Amazon.S3;
using Amazon.S3.Transfer;

namespace s3.amazonaws.com.docsamples
{
    class AbortMPUUsingHighLevelAPI
    {
        static string existingBucketName = "****Provide bucket name****";

        static void Main(string[] args)
        {
            try
            {
                TransferUtility transferUtility =
                    new TransferUtility(new AmazonS3Client(Amazon.RegionEndpoint.USEast1));
                // Aborting uploads that were initiated over a week ago.
                transferUtility.AbortMultipartUploads(
                    existingBucketName, DateTime.Now.AddDays(-7));
            }

            catch (AmazonS3Exception e)
            {
                Console.WriteLine(e.Message, e.InnerException);
            }
        }
    }
}
```

Note

You can also abort a specific multipart upload. For more information, see [List Multipart Uploads \(p. 179\)](#).

Track Multipart Upload Progress

The high-level multipart upload API provides an event, `TransferUtilityUploadRequest.UploadProgressEvent`, to track the upload progress when uploading data using the `TransferUtility` class.

The event occurs periodically and returns multipart upload progress information such as the total number of bytes to transfer, and the number of bytes transferred at the time event occurred.

The following C# code sample demonstrates how you can subscribe to the `UploadProgressEvent` event and write a handler.

```
TransferUtility fileTransferUtility =
    new TransferUtility(new AmazonS3Client(Amazon.RegionEndpoint.USEast1));

// Use TransferUtilityUploadRequest to configure options.
// In this example we subscribe to an event.
```

```
TransferUtilityUploadRequest uploadRequest =
    new TransferUtilityUploadRequest
    {
        BucketName = existingBucketName,
        FilePath = filePath,
        Key = keyName
    };

uploadRequest.UploadProgressEvent +=
    new EventHandler<UploadProgressArgs>
    (uploadRequest_UploadPartProgressEvent);

fileTransferUtility.Upload(uploadRequest);

static void uploadRequest_UploadPartProgressEvent(object sender, UploadProgressArgs e)
{
    // Process event.
    Console.WriteLine("{0}/{1}", e.TransferredBytes, e.TotalBytes);
}
```

Example

The following C# code example uploads a file to an Amazon S3 bucket and tracks the progress by subscribing to the `TransferUtilityUploadRequest.UploadProgressEvent` event. For instructions on how to create and test a working sample, see [Testing the .NET Code Examples \(p. 547\)](#).

```
using System;
using System.Collections.Specialized;
using System.Configuration;
using Amazon.S3;
using Amazon.S3.Transfer;

namespace s3.amazon.com.docsamples
{
    class TrackMPUUUsingHighLevelAPI
    {
        static string existingBucketName = "**** Provide bucket name ***";
        static string keyName           = "**** Provide key name ***";
        static string filePath          = "**** Provide file to upload ***";

        static void Main(string[] args)
        {
            try
            {
                TransferUtility fileTransferUtility =
                    new TransferUtility(new AmazonS3Client(Amazon.RegionEndpoint.USEast1));

                // Use TransferUtilityUploadRequest to configure options.
                // In this example we subscribe to an event.
                TransferUtilityUploadRequest uploadRequest =
                    new TransferUtilityUploadRequest
                    {
                        BucketName = existingBucketName,
                        FilePath = filePath,
                        Key = keyName
                    };

                uploadRequest.UploadProgressEvent +=
                    new EventHandler<UploadProgressArgs>
                    (uploadRequest_UploadPartProgressEvent);

                fileTransferUtility.Upload(uploadRequest);
                Console.WriteLine("Upload completed");
            }

            catch (AmazonS3Exception e)
            {
                Console.WriteLine(e.Message, e.InnerException);
            }
        }

        static void uploadRequest_UploadPartProgressEvent(
            object sender, UploadProgressArgs e)
        {
            // Process event.
            Console.WriteLine("{0}/{1}", e.TransferredBytes, e.TotalBytes);
        }
    }
}
```

```
    }  
}
```

Using the Low-Level .NET API for Multipart Upload

Topics

- [Upload a File \(p. 175\)](#)
- [List Multipart Uploads \(p. 179\)](#)
- [Track Multipart Upload Progress \(p. 180\)](#)
- [Abort a Multipart Upload \(p. 180\)](#)

The AWS SDK for .NET exposes a low-level API that closely resembles the Amazon S3 REST API for multipart upload (see [Using the REST API for Multipart Upload \(p. 190\)](#)). Use the low-level API when you need to pause and resume multipart uploads, vary part sizes during the upload, or do not know the size of the data in advance. Use the high-level API (see [Using the High-Level .NET API for Multipart Upload \(p. 166\)](#)), whenever you don't have these requirements.

Upload a File

The following tasks guide you through using the low-level .NET classes to upload a file.

Low-Level API File Uploading Process

1	Create an instance of the <code>AmazonS3Client</code> class, by providing your AWS credentials.
2	Initiate multipart upload by executing the <code>AmazonS3Client.InitiateMultipartUpload</code> method. You will need to provide information required to initiate the multipart upload by creating an instance of the <code>InitiateMultipartUploadRequest</code> class.
3	Save the Upload ID that the <code>AmazonS3Client.InitiateMultipartUpload</code> method returns. You will need to provide this upload ID for each subsequent multipart upload operation.
4	Upload the parts. For each part upload, execute the <code>AmazonS3Client.UploadPart</code> method. You will need to provide part upload information such as upload ID, bucket name, and the part number. You provide this information by creating an instance of the <code>UploadPartRequest</code> class.
5	Save the response of the <code>AmazonS3Client.UploadPart</code> method in a list. This response includes the ETag value and the part number you will later need to complete the multipart upload.
6	Repeat tasks 4 and 5 for each part.
7	Execute the <code>AmazonS3Client.CompleteMultipartUpload</code> method to complete the multipart upload.

The following C# code sample demonstrates the preceding tasks.

```
IAmazonS3 s3Client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1);  
  
// List to store upload part responses.  
List<UploadPartResponse> uploadResponses = new List<UploadPartResponse>();
```

```
// 1. Initialize.  
InitiateMultipartUploadRequest initiateRequest = new InitiateMultipartUploadRe  
quest  
{  
    BucketName = existingBucketName,  
    Key = keyName  
};  
  
InitiateMultipartUploadResponse initResponse =  
    s3Client.InitiateMultipartUpload(initRequest);  
  
// 2. Upload Parts.  
long contentLength = new FileInfo(filePath).Length;  
long partSize = 5242880; // 5 MB  
  
try  
{  
    long filePosition = 0;  
    for (int i = 1; filePosition < contentLength; i++)  
    {  
  
        // Create request to upload a part.  
        UploadPartRequest uploadRequest = new UploadPartRequest  
        {  
            BucketName = existingBucketName,  
            Key = keyName,  
            UploadId = initResponse.UploadId,  
            PartNumber = i,  
            PartSize = partSize,  
            FilePosition = filePosition,  
            FilePath = filePath  
        };  
  
        // Upload part and add response to our list.  
        uploadResponses.Add(s3Client.UploadPart(uploadRequest));  
  
        filePosition += partSize;  
    }  
  
    // Step 3: complete.  
    CompleteMultipartUploadRequest completeRequest = new CompleteMultipartUp  
loadRequest  
    {  
        BucketName = existingBucketName,  
        Key = keyName,  
        UploadId = initResponse.UploadId,  
    };  
  
    CompleteMultipartUploadResponse completeUploadResponse =  
        s3Client.CompleteMultipartUpload(completeRequest);  
}  
catch (Exception exception)  
{  
    Console.WriteLine("Exception occurred: {0}", exception.Message);  
    AbortMultipartUploadRequest abortMPURequest = new AbortMultipartUploadRequest
```

```
{  
    BucketName = existingBucketName,  
    Key = keyName,  
    UploadId = initResponse.UploadId  
};  
s3Client.AbortMultipartUpload(abortMPURequest);  
}
```

Note

When uploading large objects using the .NET API, timeout might occur even while data is being written to the request stream. You can set explicit timeout using the `UploadPartRequest`.

Example

The following C# code example uploads a file to an Amazon S3 bucket. For instructions on how to create and test a working sample, see [Testing the .NET Code Examples \(p. 547\)](#).

```
using System;
using System.Collections.Generic;
using System.IO;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazonaws.com.docsamples
{
    class UploadFileMPULowLevelAPI
    {
        static string existingBucketName = "**** bucket name ****";
        static string keyName           = "**** key name ****";
        static string filePath          = "**** file path ****";

        static void Main(string[] args)
        {
            IAmazonS3 s3Client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1);

            // List to store upload part responses.
            List<UploadPartResponse> uploadResponses = new List<UploadPartResponse>();

            // 1. Initialize.
            InitiateMultipartUploadRequest initiateRequest = new InitiateMultipartUploadRequest
            {
                BucketName = existingBucketName,
                Key = keyName
            };

            InitiateMultipartUploadResponse initResponse =
                s3Client.InitiateMultipartUpload(initiateRequest);

            // 2. Upload Parts.
            long contentLength = new FileInfo(filePath).Length;
            long partSize = 5 * (long) Math.Pow(2, 20); // 5 MB

            try
            {
                long filePosition = 0;
                for (int i = 1; filePosition < contentLength; i++)
                {
                    UploadPartRequest uploadRequest = new UploadPartRequest
                    {
                        BucketName = existingBucketName,
                        Key = keyName,
                        UploadId = initResponse.UploadId,
                        PartNumber = i,
                        PartSize = partSize,
                        FilePosition = filePosition,
                        FilePath = filePath
                    };
                }
            }
        }
    }
}
```

List Multipart Uploads

The following tasks guide you through using the low-level .NET classes to list all in-progress multipart uploads on a bucket.

Low-Level API Multipart Uploads Listing Process

- 1 Create an instance of the `ListMultipartUploadsRequest` class and provide the bucket name.
- 2 Execute the `AmazonS3Client.ListMultipartUploads` method. The method returns an instance of the `ListMultipartUploadsResponse` class, providing you the information about the in-progress multipart uploads.

The following C# code sample demonstrates the preceding tasks.

```
ListMultipartUploadsRequest request = new ListMultipartUploadsRequest
{
    BucketName = existingBucketName
};
```

Track Multipart Upload Progress

The low-level multipart upload API provides an event, `UploadPartRequest.StreamTransferProgress`, to track the upload progress.

The event occurs periodically and returns multipart upload progress information such as the total number of bytes to transfer, and the number of bytes transferred at the time event occurred.

The following C# code sample demonstrates how you can subscribe to the `StreamTransferProgress` event and write a handler.

```
UploadPartRequest uploadRequest = new UploadPartRequest
{
    // provide request data.
};

uploadRequest.StreamTransferProgress +=
    new EventHandler<StreamTransferProgressArgs>(UploadPartProgressEventCallback);

...
public static void UploadPartProgressEventCallback(object sender, StreamTransferProgressArgs e)
{
    // Process event.
    Console.WriteLine("{0}/{1}", e.TransferredBytes, e.TotalBytes);
}
```

Abort a Multipart Upload

You can abort an in-progress multipart upload by calling the `AmazonS3Client.AbortMultipartUpload` method. This method deletes any parts that were uploaded to S3 and free up the resources. You must provide the upload ID, bucket name and the key name. The following C# code sample demonstrates how you can abort a multipart upload in progress.

```
s3Client.AbortMultipartUpload(new AbortMultipartUploadRequest
{
    BucketName = existingBucketName,
    Key = keyName,
    UploadId = uploadID
});
```

Note

Instead of a specific multipart upload, you can abort all your in-progress multipart uploads initiated prior to a specific time. This clean up operation is useful to abort old multipart uploads that you initiated but neither completed or aborted. For more information, see [Abort Multipart Uploads \(p. 171\)](#).

Using the AWS SDK for PHP for Multipart Upload

Topics

- [Using the AWS SDK for PHP High-Level Abstractions for Multipart Upload \(p. 181\)](#)
- [Using the AWS SDK for PHP Low-Level API for Multipart Upload \(p. 184\)](#)

The AWS SDK for PHP provides support for the multipart upload API (see [Uploading Objects Using Multipart Upload API \(p. 155\)](#)).

High-level Abstractions

The AWS SDK for PHP high-level abstractions simplify the multipart upload flow. In just a few lines of code you can upload files to Amazon S3. This is recommended for simple file uploads.

Low-level API

The AWS SDK for PHP low-level APIs correspond to the multipart upload REST operations (see [Using the REST API for Multipart Upload \(p. 190\)](#)). Use the low-level API when you need to pause and resume multipart uploads, vary part sizes during the upload, or do not know the size of the data in advance. If you do not have these requirements, use the high-level API.

Using the AWS SDK for PHP High-Level Abstractions for Multipart Upload

Amazon S3 allows you to upload large files in multiple parts. You must use a multipart upload for files larger than 5 GB. The AWS SDK for PHP exposes the high-level `Aws\S3\Model\MultipartUpload\UploadBuilder` class that simplifies multipart uploads.

The `Aws\S3\Model\MultipartUpload\UploadBuilder` class is best used for a simple multipart upload. If you need to pause and resume multipart uploads, vary part sizes during the upload, or do not know the size of the data in advance, you should use the low-Level PHP API. For more information, see [Using the AWS SDK for PHP Low-Level API for Multipart Upload \(p. 184\)](#).

For more information about multipart uploads, see [Uploading Objects Using Multipart Upload API \(p. 155\)](#). For information on uploading files that are less than 5GB in size, see [Upload an Object Using the AWS SDK for PHP \(p. 150\)](#).

Upload a File Using the High-Level Multipart Upload

This topic guides you through using the high-level `Aws\S3\Model\MultipartUpload\UploadBuilder` class from the AWS SDK for PHP for multipart file uploads.

Note

This topic assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 547\)](#) and have the AWS SDK for PHP properly installed.

High-Level Multipart File Upload Process

1	Create an instance of an Amazon S3 client by using the <code>Aws\S3\S3Client</code> class <code>factory()</code> method.
2	Create an instance of the <code>UploadBuilder</code> using the Amazon S3 <code>Aws\S3\Model\MultipartUpload\UploadBuilder</code> class <code>newInstance()</code> method, which is inherited from the <code>Aws\Common\Model\MultipartUpload\AbstractUploadBuilder</code> class. For the <code>UploadBuilder</code> object set the client, the bucket name, and the key name using the <code>setClient()</code> , <code>setBucket()</code> , and <code>setKey()</code> methods. Set the path and name of the file you want to upload with the <code>setSource()</code> method.

3	Execute the <code>UploadBuilder</code> object's <code>build()</code> method to build the appropriate uploader transfer object based on the builder options you set. (The transfer object is of a subclass of the <code>Aws\S3\Model\MultipartUpload\AbstractTransfer</code> class.)
4	Execute the <code>upload()</code> method of the built transfer object to perform the upload.

The following PHP code sample demonstrates how to upload a file using the high-level `UploadBuilder` object.

```
use Aws\Common\Exception\MultipartUploadException;
use Aws\S3\Model\MultipartUpload\UploadBuilder;
use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

// Instantiate the client.
$s3 = S3Client::factory();

// Prepare the upload parameters.
uploader = UploadBuilder::newInstance()
    ->setClient($s3)
    ->setSource('/path/to/large/file.mov')
    ->setBucket($bucket)
    ->setKey($keyname)
    ->build();

// Perform the upload. Abort the upload if something goes wrong.
try {
    $uploader->upload();
    echo "Upload complete.\n";
} catch (MultipartUploadException $e) {
    $uploader->abort();
    echo "Upload failed.\n";
    echo $e->getMessage() . "\n";
}
```

Example of a Multipart Upload of a File to an Amazon S3 Bucket Using the High-level UploadBuilder

The following PHP example uploads a file to an Amazon S3 bucket. The example demonstrates how to set advanced options for the UploadBuilder object. For example, you can use the [setMinPartSize\(\)](#) method to set the part size you want to use for the multipart upload and the [setOption\(\)](#) method to set optional file metadata or an access control list (ACL).

The example also demonstrates how to upload file parts in parallel by setting the concurrency option using the [setConcurrency\(\)](#) method for the UploadBuilder object. The example creates a transfer object that will attempt to upload three parts in parallel until the entire file has been uploaded. For information about running the PHP examples in this guide, go to [Running PHP Examples \(p. 548\)](#).

```
<?php

// Include the AWS SDK using the Composer autoloader.
require 'vendor/autoload.php';

use Aws\Common\Exception\MultipartUploadException;
use Aws\S3\Model\MultipartUpload\UploadBuilder;
use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

// Instantiate the client.
$s3 = S3Client::factory();

// Prepare the upload parameters.
$uploader = UploadBuilder::newInstance()
    ->setClient($s3)
    ->setSource('/path/to/large/file.mov')
    ->setBucket($bucket)
    ->setKey($keyname)
    ->setMinPartSize(25 * 1024 * 1024)
    ->setOption('Metadata', array(
        'param1' => 'value1',
        'param2' => 'value2'
    ))
    ->setOption('ACL', 'public-read')
    ->setConcurrency(3)
    ->build();

// Perform the upload. Abort the upload if something goes wrong.
try {
    $uploader->upload();
    echo "Upload complete.\n";
} catch (MultipartUploadException $e) {
    $uploader->abort();
    echo "Upload failed.\n";
    echo $e->getMessage() . "\n";
}
```

Related Resources

- [AWS SDK for PHP Aws\Common\Model\MultipartUpload\AbstractUploadBuilder Class](#)
- [AWS SDK for PHP Aws\Common\Model\MultipartUpload\AbstractUploadBuilder::newInstance\(\) Method](#)

- AWS SDK for PHP Aws\Common\Model\MultipartUpload\AbstractUploadBuilder::SetSource() Method
- AWS SDK for PHP for Amazon S3 Aws\S3\Model\MultipartUpload\UploadBuilder Class
- AWS SDK for PHP for Amazon S3 Aws\S3\Model\MultipartUpload\UploadBuilder::build() Method
- AWS SDK for PHP for Amazon S3 Aws\S3\Model\MultipartUpload\UploadBuilder::setMinPartSize() Method
- AWS SDK for PHP for Amazon S3 Aws\S3\Model\MultipartUpload\UploadBuilder::setOption() Method
- AWS SDK for PHP for Amazon S3 Aws\S3\Model\MultipartUpload\UploadBuilder::setConcurrency() Method
- AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class
- AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::factory() Method
- AWS SDK for PHP for Amazon S3 - Uploading Large Files Using Multipart Uploads
- AWS SDK for PHP for Amazon S3
- AWS SDK for PHP Documentation

Using the AWS SDK for PHP Low-Level API for Multipart Upload

Topics

- [Upload a File in Multiple Parts Using the PHP SDK Low-Level API \(p. 184\)](#)
- [List Multipart Uploads Using the PHP SDK Low-Level API \(p. 188\)](#)
- [Abort a Multipart Upload \(p. 189\)](#)

The AWS SDK for PHP exposes a low-level API that closely resembles the Amazon S3 REST API for multipart upload (see [Using the REST API for Multipart Upload \(p. 190\)](#)). Use the low-level API when you need to pause and resume multipart uploads, vary part sizes during the upload, or do not know the size of the data in advance. Use the AWS SDK for PHP high-level abstractions (see [Using the AWS SDK for PHP High-Level Abstractions for Multipart Upload \(p. 181\)](#)) whenever you don't have these requirements.

Upload a File in Multiple Parts Using the PHP SDK Low-Level API

This topic guides you through using low-level multipart upload classes from the AWS SDK for PHP to upload a file in multiple parts.

Note

This topic assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 547\)](#) and have the AWS SDK for PHP properly installed.

PHP SDK Low-Level API Multipart File Upload Process

1	Create an instance of an Amazon S3 client by using the Aws\S3\S3Client class <code>factory()</code> method.
2	Initiate multipart upload by executing the Aws\S3\S3Client::createMultipartUpload() method. You must provide a bucket name and a key name in the <code>array</code> parameter's required keys, <code>Bucket</code> and <code>Key</code> . Retrieve and save the <code>UploadID</code> from the response body. The <code>UploadID</code> is used in each subsequent multipart upload operation.

3	<p>Upload the file in parts by executing the Aws\S3\S3Client::uploadPart() method for each file part until the end of the file is reached. The required array parameter keys for <code>upload_part()</code> are <code>Bucket</code>, <code>Key</code>, <code>UploadId</code>, and <code>PartNumber</code>. You must increment the value passed as the argument for the <code>PartNumber</code> key for each subsequent call to <code>upload_part()</code> to upload each successive file part.</p> <p>Save the response of each of the <code>upload_part()</code> methods calls in an array. Each response includes the <code>ETag</code> value you will later need to complete the multipart upload.</p>
4	<p>Execute the Aws\S3\S3Client::completeMultipartUpload() method to complete the multipart upload. The required array parameters for <code>completeMultipartUpload()</code> are <code>Bucket</code>, <code>Key</code>, and <code>UploadId</code>.</p>

The following PHP code example demonstrates uploading a file in multiple parts using the PHP SDK low-level API.

```

use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';
$filename = '*** Path to and Name of the File to Upload ***';

// 1. Instantiate the client.
$s3 = S3Client::factory();

// 2. Create a new multipart upload and get the upload ID.
$response = $s3->createMultipartUpload(array(
    'Bucket' => $bucket,
    'Key'     => $keyname
));
$uploadId = $response['UploadId'];

// 3. Upload the file in parts.
$file = fopen($filename, 'r');
$parts = array();
$partNumber = 1;
while (!feof($file)) {
    $result = $s3->uploadPart(array(
        'Bucket'      => $bucket,
        'Key'         => $key,
        'UploadId'   => $uploadId,
        'PartNumber'  => $partNumber,
        'Body'        => fread($file, 5 * 1024 * 1024),
    ));
    $parts[] = array(
        'PartNumber' => $partNumber++,
        'ETag'        => $result['ETag'],
    );
}
}

// 4. Complete multipart upload.
$result = $s3->completeMultipartUpload(array(
    'Bucket'      => $bucket,
    'Key'         => $key,
    'UploadId'   => $uploadId,
    'Parts'       => $parts,
));

```

```
) );  
$url = $result['Location'];  
  
fclose($file);
```

Example of Uploading a File to an Amazon S3 Bucket Using the Low-level Multipart Upload PHP SDK API

The following PHP code example uploads a file to an Amazon S3 bucket using the low-level PHP API multipart upload. For information about running the PHP examples in this guide, go to [Running PHP Examples \(p. 548\)](#).

```
<?php

// Include the AWS SDK using the Composer autoloader
require 'vendor/autoload.php';

use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';
$filename = '*** Path to and Name of the File to Upload ***';

// 1. Instantiate the client.
$s3 = S3Client::factory();

// 2. Create a new multipart upload and get the upload ID.
$result = $s3->createMultipartUpload(array(
    'Bucket'      => $bucket,
    'Key'         => $keyname,
    'StorageClass' => 'REDUCED_REDUNDANCY',
    'ACL'          => 'public-read',
    'Metadata'     => array(
        'param1' => 'value 1',
        'param2' => 'value 2',
        'param3' => 'value 3'
    )
));
$uploadId = $result['UploadId'];

// 3. Upload the file in parts.
try {
    $file = fopen($filename, 'r');
    $parts = array();
    $partNumber = 1;
    while (!feof($file)) {
        $result = $s3->uploadPart(array(
            'Bucket'      => $bucket,
            'Key'         => $keyname,
            'UploadId'   => $uploadId,
            'PartNumber'  => $partNumber,
            'Body'        => fread($file, 5 * 1024 * 1024),
        ));
        $parts[] = array(
            'PartNumber' => $partNumber++,
            'ETag'        => $result['ETag'],
        );
        echo "Uploading part {$partNumber} of {$filename}.\n";
    }
    fclose($file);
} catch (S3Exception $e) {
    $result = $s3->abortMultipartUpload(array(

```

```

        'Bucket'    => $bucket,
        'Key'       => $keyname,
        'UploadId'  => $uploadId
    ) );
}

echo "Upload of {$filename} failed.\n";
}

// 4. Complete multipart upload.
$result = $s3->completeMultipartUpload(array(
    'Bucket'    => $bucket,
    'Key'       => $keyname,
    'UploadId'  => $uploadId,
    'Parts'     => $parts,
));
$url = $result['Location'];

echo "Uploaded {$filename} to {$url}.\n";

```

Related Resources

- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::factory\(\) Method](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::createMultipartUpload\(\) Method](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::uploadPart\(\) Method](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::completeMultipartUpload\(\) Method](#)
- [AWS SDK for PHP for Amazon S3](#)
- [AWS SDK for PHP Documentation](#)

List Multipart Uploads Using the PHP SDK Low-Level API

This topic guides you through using the low-level API classes from the AWS SDK for PHP to list all in-progress multipart uploads on a bucket.

Note

This topic assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 547\)](#) and have the AWS SDK for PHP properly installed.

PHP SDK Low-Level API Multipart Uploads Listing Process

1	Create an instance of an Amazon S3 client by using the Aws\S3\S3Client class factory() method.
2	Execute the Aws\S3\S3Client::listMultipartUploads() method by providing a bucket name. The method returns all of the in-progress multipart uploads on the specified bucket.

The following PHP code sample demonstrates listing all in-progress multipart uploads on a bucket.

```

use Aws\S3\S3Client;

$s3 = S3Client::factory();

$bucket = '*** Your Bucket Name ***';

```

```
$result = $s3->listMultipartUploads(array('Bucket' => $bucket));  
  
print_r($result->toArray());
```

Related Resources

- AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class
- AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::factory() Method
- AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::listMultipartUploads() Method
- AWS SDK for PHP for Amazon S3
- AWS SDK for PHP Documentation

Abort a Multipart Upload

This topic describes how to use a class from the AWS SDK for PHP to abort a multipart upload that is in progress.

Note

This topic assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 547\)](#) and have the AWS SDK for PHP properly installed.

Aborting a Multipart Upload

1	Create an instance of an Amazon S3 client by using the Aws\S3\S3Client class factory() method.
2	Execute the Aws\S3\S3Client::abortMultipartUpload() method. You must provide a bucket name, a key name, and the upload ID, in the array parameter's required keys, Bucket, Key, and UploadId. The <code>abortMultipartUpload()</code> method deletes any parts that were uploaded to Amazon S3 and frees up the resources.

Example of Aborting a Multipart Upload

The following PHP code example demonstrates how you can abort a multipart upload in progress. The example illustrates the use of the `abortMultipartUpload()` method. For information about running the PHP examples in this guide, go to [Running PHP Examples \(p. 548\)](#).

```
<?php

// Include the AWS SDK using the Composer autoloader.
require 'vendor/autoload.php';

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

// Instantiate the client.
$s3 = S3Client::factory();

// Abort the multipart upload.
$s3->abortMultipartUpload(array(
    'Bucket'      => $bucket,
    'Key'         => $keyname,
    'UploadId'   => 'VXBsb2FkIE1ExampleB1bHZpbmcncyBtExamplepZS5tMnRzIHVwbG9hz',
));
)
```

Related Resources

- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::factory\(\) Method](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::abortMultipartUpload\(\) Method](#)
- [AWS SDK for PHP for Amazon S3](#)
- [AWS SDK for PHP Documentation](#)

Using the AWS SDK for Ruby for Multipart Upload

The AWS SDK for Ruby supports Amazon S3 multipart uploads by using the class `AWS::S3::MultipartUpload`. For more information about using the AWS SDK for Ruby with Amazon S3, go to [Using the AWS SDK for Ruby \(p. 549\)](#).

Using the REST API for Multipart Upload

The following sections in the Amazon Simple Storage Service API Reference describe the REST API for multipart upload.

- [Initiate Multipart Upload](#)
- [Upload Part](#)
- [Complete Multipart Upload](#)
- [Abort Multipart Upload](#)
- [List Parts](#)
- [List Multipart Uploads](#)

You can use these APIs to make your own REST requests, or you can use one the SDKs we provide. For more information about the SDKs, see [API Support for Multipart Upload \(p. 128\)](#).

Uploading Objects Using Pre-Signed URLs

Topics

- [Upload an Object Using a Pre-Signed URL \(AWS SDK for Java\) \(p. 191\)](#)
- [Upload an Object Using a Pre-Signed URL \(AWS SDK for .NET\) \(p. 194\)](#)
- [Upload an Object Using a Pre-Signed URL \(AWS SDK for Ruby\) \(p. 197\)](#)

A pre-signed URL gives you access to the object identified in the URL, provided that the creator of the pre-signed URL has permissions to access that object. That is, if you receive a pre-signed URL to upload an object, you can upload the object only if the creator of the pre-signed URL has the necessary permissions to upload that object.

All objects and buckets by default are private. The pre-signed URLs are useful if you want your user/customer to be able upload a specific object to your bucket, but you don't require them to have AWS security credentials or permissions. When you create a pre-signed URL, you must provide your security credentials, specify a bucket name an object key, an HTTP method (PUT of uploading objects), and an expiration date and time. The pre-signed URLs are valid only for the specified duration.

You can generate a pre-signed URL programmatically using the AWS SDK for Java or the AWS SDK for .NET. If you are using Visual Studio, you can also use AWS Explorer to generate a pre-signed object URL without writing any code. Anyone who receives a valid pre-signed URL can then programmatically upload an object.

For more information, go to [Using Amazon S3 from AWS Explorer](#).

For instructions about how to install AWS Explorer, see [Using the AWS SDKs and Explorers \(p. 542\)](#).

Note

Anyone with valid security credentials can create a pre-signed URL. However, in order to successfully upload an object, the pre-signed URL must be created by someone who has permission to perform the operation that the pre-signed URL is based upon.

Upload an Object Using a Pre-Signed URL (AWS SDK for Java)

The following tasks guide you through using the Java classes to upload an object using a pre-signed URL.

Uploading Objects

1	Create an instance of the <code>AmazonS3</code> class.
2	Generate a pre-signed URL by executing the <code>AmazonS3.generatePresignedUrl</code> method. You provide a bucket name, an object key, and an expiration date by creating an instance of the <code>GeneratePresignedUrlRequest</code> class. You must specify the HTTP verb PUT when creating this URL if you want to use it to upload an object.
3	Anyone with the pre-signed URL can upload an object. The upload creates an object or replaces any existing object with the same key that is specified in the pre-signed URL.

The following Java code sample demonstrates the preceding tasks.

```
AmazonS3 s3Client = new AmazonS3Client(new ProfileCredentialsProvider());

java.util.Date expiration = new java.util.Date();
long msec = expiration.getTime();
msec += 1000 * 60 * 60; // Add 1 hour.
expiration.setTime(msec);

GeneratePresignedUrlRequest generatePresignedUrlRequest = new GeneratePresigned
UrlRequest(bucketName, objectKey);
generatePresignedUrlRequest.setMethod(HttpMethod.PUT);
generatePresignedUrlRequest.setExpiration(expiration);

URL s = s3client.generatePresignedUrl(generatePresignedUrlRequest);

// Use the pre-signed URL to upload an object.
```

Example

The following Java code example generates a pre-signed URL. The example code then uses the pre-signed URL to upload sample data as an object. For instructions about how to create and test a working sample, see [Testing the Java Code Examples \(p. 545\)](#).

```
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.net.HttpURLConnection;
import java.net.URL;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.HttpMethod;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.GeneratePresignedUrlRequest;

public class GeneratePresignedUrlAndUploadObject {
    private static String bucketName = "*** bucket name ***";
    private static String objectKey = "*** object key ***";

    public static void main(String[] args) throws IOException {
        AmazonS3 s3client = new AmazonS3Client(new ProfileCredentialsProvider());

        try {
            System.out.println("Generating pre-signed URL.");
            java.util.Date expiration = new java.util.Date();
            long milliSeconds = expiration.getTime();
            milliSeconds += 1000 * 60 * 60; // Add 1 hour.
            expiration.setTime(milliSeconds);

            GeneratePresignedUrlRequest generatePresignedUrlRequest =
                new GeneratePresignedUrlRequest(bucketName, objectKey);
            generatePresignedUrlRequest.setMethod(HttpMethod.PUT);
            generatePresignedUrlRequest.setExpiration(expiration);

            URL url = s3client.generatePresignedUrl(generatePresignedUrlRequest);

            UploadObject(url);

            System.out.println("Pre-Signed URL = " + url.toString());
        } catch (AmazonServiceException exception) {
            System.out.println("Caught an AmazonServiceException, " +
                "which means your request made it " +
                "to Amazon S3, but was rejected with an error response " +
                "for some reason.");
            System.out.println("Error Message: " + exception.getMessage());
            System.out.println("HTTP Code: " + exception.getStatusCode());
            System.out.println("AWS Error Code:" + exception.getErrorCode());
            System.out.println("Error Type: " + exception.getErrorType());
            System.out.println("Request ID: " + exception.getRequestId());
        } catch (AmazonClientException ace) {
            System.out.println("Caught an AmazonClientException, " +
                "which means the client encountered " +
                "an internal error while trying to communicate" +
                " with S3, " +

```

```

        "such as not being able to access the network.");
        System.out.println("Error Message: " + ace.getMessage());
    }

}

public static void UploadObject(URL url) throws IOException
{
    HttpURLConnection connection=(HttpURLConnection) url.openConnection();
    connection.setDoOutput(true);
    connection.setRequestMethod("PUT");
    OutputStreamWriter out = new OutputStreamWriter(
        connection.getOutputStream());
    out.write("This text uploaded as object.");
    out.close();
    int responseCode = connection.getResponseCode();
    System.out.println("Service returned response code " + responseCode);
}
}

```

Upload an Object Using a Pre-Signed URL (AWS SDK for .NET)

The following tasks guide you through using the .NET classes to upload an object using a pre-signed URL.

Uploading Objects

1	Create an instance of the <code>AmazonS3</code> class. These credentials are used in creating a signature for authentication when you generate a pre-signed URL.
2	Generate a pre-signed URL by executing the <code>AmazonS3.GetPreSignedURL</code> method. You provide a bucket name, an object key, and an expiration date by creating an instance of the <code>GetPreSignedUrlRequest</code> class. You must specify the HTTP verb PUT when creating this URL if you plan to use it to upload an object.
3	Anyone with the pre-signed URL can upload an object. You can create an instance of the <code>HttpWebRequest</code> class by providing the pre-signed URL and uploading the object.

The following C# code sample demonstrates the preceding tasks.

```

IAmazonS3 client;
client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1);
// Generate a pre-signed URL.
GetPreSignedUrlRequest request = new GetPreSignedUrlRequest
{
    BucketName = bucketName,
    Key        = objectKey,
    Verb       = HttpVerb.PUT,
    Expires    = DateTime.Now.AddMinutes(5)
};
string url = null;
url = s3Client.GetPreSignedURL(request);

```

```
// Upload a file using the pre-signed URL.  
HttpWebRequest httpRequest = WebRequest.Create(url) as HttpWebRequest;  
httpRequest.Method = "PUT";  
using (Stream dataStream = httpRequest.GetRequestStream())  
{  
    // Upload object.  
}  
  
HttpWebResponse response = httpRequest.GetResponse() as HttpWebResponse;
```

Example

The following C# code example generates a pre-signed URL for a specific object and uses it to upload a file. For instructions about how to create and test a working sample, see [Testing the .NET Code Examples \(p. 547\)](#).

```
using System;
using System.IO;
using System.Net;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazonaws.com.docsamples
{
    class UploadObjectUsingPresignedURL
    {
        static IAmazonS3 s3Client;
        // File to upload.
        static string filePath = "**** Specify file to upload ****";
        // Information to generate pre-signed object URL.
        static string bucketName = "**** Provide bucket name ****";
        static string objectKey = "**** Provide object key for the new object
****";

        public static void Main(string[] args)
        {
            try
            {
                using (s3Client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1))
                {
                    string url = GeneratePreSignedURL();
                    UploadObject(url);

                }
            }
            catch (AmazonS3Exception amazonS3Exception)
            {
                if (amazonS3Exception.ErrorCode != null &&
                    (amazonS3Exception.ErrorCode.Equals("InvalidAccessKeyId") ||
                     amazonS3Exception.ErrorCode.Equals("InvalidSecurity")))
                {
                    Console.WriteLine("Check the provided AWS Credentials.");
                    Console.WriteLine(
                        "To sign up for service, go to http://aws.amazon.com/s3");
                }
                else
                {
                    Console.WriteLine(
                        "Error occurred. Message:'{0}' when listing objects",
                        amazonS3Exception.Message);
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

```
        }
        Console.WriteLine("Press any key to continue... ");
        Console.ReadKey();
    }

    static void UploadObject(string url)
    {
        HttpWebRequest httpRequest = WebRequest.Create(url) as HttpWebRe
quest;
        httpRequest.Method = "PUT";
        using (Stream dataStream = httpRequest.GetRequestStream())
        {
            byte[] buffer = new byte[8000];
            using (FileStream fileStream = new FileStream(filePath,
 FileMode.Open, FileAccess.Read))
            {
                int bytesRead = 0;
                while ((bytesRead = fileStream.Read(buffer, 0, buf
fer.Length)) > 0)
                {
                    dataStream.Write(buffer, 0, bytesRead);
                }
            }
        }

        HttpWebResponse response = httpRequest.GetResponse() as HttpWebRe
sponse;
    }

    static string GeneratePreSignedURL()
    {
        GetPreSignedUrlRequest request = new GetPreSignedUrlRequest
        {
            BucketName = bucketName,
            Key        = objectKey,
            Verb       = HttpVerb.PUT,
            Expires    = DateTime.Now.AddMinutes(5)
        };

        string url = null;
        url = s3Client.GetPreSignedURL(request);
        return url;
    }
}
```

Upload an Object Using a Pre-Signed URL (AWS SDK for Ruby)

The following tasks guide you through using a Ruby script to upload an object using a pre-signed URL for either version of the SDK for Ruby.

Topics

- [Using AWS SDK for Ruby - Version 2 \(p. 198\)](#)
- [Using AWS SDK for Ruby - Version 1 \(p. 198\)](#)

Using AWS SDK for Ruby - Version 2

The following tasks guide you through using a Ruby script to upload an object using a pre-signed URL for SDK for Ruby - Version 2.

Uploading Objects - SDK for Ruby - Version 2

1	Create an instance of the <code>Aws::S3::Resource</code> class.
2	You provide a bucket name and an object key by calling the <code>#bucket[]</code> and the <code>#object[]</code> methods of your <code>Aws::S3::Resource</code> class instance. Generate a pre-signed URL by creating an instance of the <code>URI</code> class and use it to parse the <code>.presigned_url</code> method of your <code>Aws::S3::Resource</code> class instance. You must specify <code>:put</code> as an argument to <code>.presigned_url</code> , and you must specify <code>PUT</code> to <code>Net::HTTP::Session#send_request</code> if you want to upload an object.
3	Anyone with the pre-signed URL can upload an object. The upload creates an object or replaces any existing object with the same key that is specified in the pre-signed URL.

The following Ruby code sample demonstrates the preceding tasks for SDK for Ruby - Version 2.

```
#Uploading an object using a pre-signed URL for SDK for Ruby - Version 2.

require 'aws-sdk-resources'
require 'net/http'

s3 = Aws::S3::Resource.new(region:'us-west-2')

obj = s3.bucket('BucketName').object('KeyName')
# Replace BucketName with the name of your bucket.
# Replace KeyName with the name of the object you are creating or replacing.

url = URI.parse(obj.presigned_url(:put))

body = "Hello World!"
# This is the contents of your object. In this case, it's a simple string.

Net::HTTP.start(url.host) do |http|
  http.send_request("PUT", url.request_uri, body, {
    # This is required, or Net::HTTP will add a default unsigned content-type.
    "content-type" => "",
  })
end

puts obj.get.body.read
# This will print out the contents of your object to the terminal window.
```

Using AWS SDK for Ruby - Version 1

Uploading Objects - SDK for Ruby - Version 1

1	Create an instance of the <code>AWS::S3</code> class.
---	---

2	<p>You provide a bucket name and an object key by calling the <code>#bucket[]</code> and the <code>#object[]</code> methods of your <code>AWS::S3::S3Object</code> class instance.</p> <p>Generate a pre-signed URL by calling the <code>.url_for</code> method of your <code>AWS::S3</code> class instance. You must specify <code>:put</code> as an argument to <code>.url_for</code>, and you must specify <code>PUT</code> to <code>Net::HTTP::Session#send_request</code> if you want to upload an object.</p>
3	<p>Anyone with the pre-signed URL can upload an object.</p> <p>The upload creates an object or replaces any existing object with the same key that is specified in the pre-signed URL.</p>

The following Ruby code sample demonstrates the preceding tasks for AWS SDK for Ruby - Version 1.

```
#Uploading an object using a pre-signed URL for SDK for Ruby - Version 1.

require 'aws-sdk-v1'
require 'net/http'

s3 = AWS::S3.new(region:'us-west-2')

obj = s3.buckets['BucketName'].objects['KeyName']
# Replace BucketName with the name of your bucket.
# Replace KeyName with the name of the object you are creating or replacing.

url = obj.url_for(:write, :content_type => "text/plain")

body = "Hello World!"
# This is the contents of your object. In this case, it's a simple string.

Net::HTTP.start(url.host) do |http|
  http.send_request("PUT", url.request_uri, body, {"content-type" =>
"text/plain",})
# The content-type must be specified in the pre-signed url.
end

puts obj.read
# This will print out the contents of your object to the terminal window.

puts obj.content_type
# This will print out the content type of your object to the terminal window.
```

Copying Objects

Topics

- [Related Resources \(p. 200\)](#)
- [Copying Objects in a Single Operation \(p. 200\)](#)
- [Copying Objects Using the Multipart Upload API \(p. 210\)](#)

The copy operation creates a copy of an object that is already stored in Amazon S3. You can create a copy of your object up to 5 GB in a single atomic operation. However, for copying an object that is greater than 5 GB, you must use the multipart upload API. Using the `copy` operation, you can:

- Create additional copies of objects
- Rename objects by copying them and deleting the original ones
- Move objects across Amazon S3 locations (e.g., us-west-1 and EU)
- Change object metadata

Each Amazon S3 object has metadata. It is a set of name-value pairs. You can set object metadata at the time you upload it. After you upload the object, you cannot modify object metadata. The only way to modify object metadata is to make copy of the object and set the metadata. In the copy operation you set the same object as the source and target.

Each object has metadata. Some of it is system metadata and other user-defined. Users control some of the system metadata such as storage class configuration to use for the object, and configure server-side encryption. When you copy an object, user-controlled system metadata and user-defined metadata are also copied. Amazon S3 resets the system controlled metadata. For example, when you copy an object, Amazon S3 resets creation date of copied object. You don't need to set any of these values in your copy request.

When copying an object, you might decide to update some of the metadata values. For example, if your source object is configured to use standard storage, you might choose to use reduced redundancy storage for the object copy. You might also decide to alter some of the user-defined metadata values present on the source object. Note that if you choose to update any of the object's user configurable metadata (system or user-defined) during the copy, then you must explicitly specify all the user configurable metadata, even if you are only changing only one of the metadata values, present on the source object in your request.

For more information about the object metadata, see [Object Key and Metadata \(p. 79\)](#).

Note

Copying objects across locations incurs bandwidth charges.

Note

If the source object is archived in Amazon Glacier (the storage class of the object is `GLACIER`), you must first restore a temporary copy before you can copy the object to another bucket. For information about archiving objects, see [Object Archival \(p. 96\)](#).

When copying objects you can request Amazon S3 to save the target object encrypted using an AWS Key Management Service (KMS) encryption key, an Amazon S3-managed encryption key, or a customer-provided encryption key. Accordingly you must specify encryption information in your request. If the copy source is an object that stored in Amazon S3 using server-side encryption with customer provided key, you will need to provide encryption information in your request so Amazon S3 can decrypt the object for copying. For more information, see [Protecting Data Using Encryption \(p. 380\)](#).

Related Resources

- [Using the AWS SDKs and Explorers \(p. 542\)](#)

Copying Objects in a Single Operation

Topics

- [Copy an Object Using the AWS SDK for Java \(p. 201\)](#)
- [Copy an Object Using the AWS SDK for .NET \(p. 202\)](#)
- [Copy an Object Using the AWS SDK for PHP \(p. 205\)](#)
- [Copy an Object Using the AWS SDK for Ruby \(p. 208\)](#)
- [Copy an Object Using the REST API \(p. 208\)](#)

The examples in this section show how to copy objects up to 5 GB in a single operation. For copying objects greater than 5 GB, you must use multipart upload API. For more information, see [Copying Objects Using the Multipart Upload API \(p. 210\)](#).

Copy an Object Using the AWS SDK for Java

The following tasks guide you through using the Java classes to copy an object in Amazon S3.

Copying Objects

1	Create an instance of the <code>AmazonS3Client</code> class.
2	Execute one of the <code>AmazonS3Client.copyObject</code> methods. You need to provide the request information, such as source bucket name, source key name, destination bucket name, and destination key. You provide this information by creating an instance of the <code>CopyObjectRequest</code> class or optionally providing this information directly with the <code>AmazonS3Client.copyObject</code> method.

The following Java code sample demonstrates the preceding tasks.

```
AmazonS3 s3client = new AmazonS3Client(new ProfileCredentialsProvider());
s3client.copyObject(sourceBucketName, sourceKey,
                    destinationBucketName, destinationKey);
```

Example

The following Java code example makes a copy of an object. The copied object with a different key is saved in the same source bucket. For instructions on how to create and test a working sample, see [Testing the Java Code Examples \(p. 545\)](#).

```
import java.io.IOException;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.CopyObjectRequest;

public class CopyObjectSingleOperation {
    private static String bucketName      = "**** Provide bucket name ****";
    private static String key             = "**** Provide key *** ";
    private static String destinationKey = "**** Provide dest. key ***";

    public static void main(String[] args) throws IOException {
        AmazonS3 s3client = new AmazonS3Client(new ProfileCredentialsProvider());

        try {
            // Copying object
            CopyObjectRequest copyObjRequest = new CopyObjectRequest(
                bucketName, key, bucketName, destinationKey);
            System.out.println("Copying object.");
            s3client.copyObject(copyObjRequest);

        } catch (AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException, " +
                "which means your request made it " +
                "to Amazon S3, 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 " +
                "an internal error while trying to " +
                "communicate with S3, " +
                "such as not being able to access the network.");
            System.out.println("Error Message: " + ace.getMessage());
        }
    }
}
```

Copy an Object Using the AWS SDK for .NET

The following tasks guide you through using the high-level .NET classes to upload a file. The API provides several variations, *overloads*, of the `Upload` method to easily upload your data.

Copying Objects

1	Create an instance of the <code>AmazonS3</code> class.
2	Execute one of the <code>AmazonS3.CopyObject</code> . You need to provide information such as source bucket, source key name, target bucket, and target key name. You provide this information by creating an instance of the <code>CopyObjectRequest</code> class.

The following C# code sample demonstrates the preceding tasks.

```
static IAmazonS3 client;
client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1);

CopyObjectRequest request = new CopyObjectRequest()
{
    SourceBucket      = bucketName,
    SourceKey        = objectKey,
    DestinationBucket = bucketName,
    DestinationKey   = destObjectKey
};
CopyObjectResponse response = client.CopyObject(request);
```

Example

The following C# code example makes a copy of an object. You will need to update code and provide your bucket names, and object keys. For instructions on how to create and test a working sample, see [Testing the .NET Code Examples \(p. 547\)](#).

```
using System;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazonaws.com.docsamples
{
    class CopyObject
    {
        static string sourceBucket      = "**** Bucket on which to enable logging ****";
        static string destinationBucket = "**** Bucket where you want logs stored ****";
        static string objectKey         = "**** Provide key name ****";
        static string destObjectKey     = "**** Provide destination key name ****";
        static IAmazonS3 client;

        public static void Main(string[] args)
        {
            using (client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1))

            {
                Console.WriteLine("Copying an object");
                CopyingObject();
            }
            Console.WriteLine("Press any key to continue... ");
            Console.ReadKey();
        }

        static void CopyingObject()
        {
            try
            {
                CopyObjectRequest request = new CopyObjectRequest
                {
                    SourceBucket      = sourceBucket,
                    SourceKey         = objectKey,
                    DestinationBucket = destinationBucket,
                    DestinationKey    = destObjectKey
                };
                CopyObjectResponse response = client.CopyObject(request);
            }
            catch (AmazonS3Exception s3Exception)
            {
                Console.WriteLine(s3Exception.Message,
                                  s3Exception.InnerException);
            }
        }
    }
}
```

Copy an Object Using the AWS SDK for PHP

This topic guides you through using classes from the AWS SDK for PHP to copy a single object and multiple objects within Amazon S3, from one bucket to another or within the same bucket.

Note

This topic assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 547\)](#) and have the AWS SDK for PHP properly installed.

The following tasks guide you through using PHP SDK classes to copy an object that is already stored in Amazon S3.

Copying an Object

1	Create an instance of an Amazon S3 client by using the Aws\S3\S3Client class factory() method.
2	To copy an object, execute the Aws\S3\S3Client::copyObject() method. You need to provide information such as source bucket, source key name, target bucket, and target key name.

The following PHP code sample demonstrates using the `copyObject()` method to copy an object that is already stored in Amazon S3.

```
use Aws\S3\S3Client;

$sourceBucket = '*** Your Source Bucket Name ***';
$sourceKeyname = '*** Your Source Object Key ***';
$targetBucket = '*** Your Target Bucket Name ***';
$targetKeyname = '*** Your Target Key Name ***';

// Instantiate the client.
$s3 = S3Client::factory();

// Copy an object.
$s3->copyObject(array(
    'Bucket'      => $targetBucket,
    'Key'         => $targetKeyname,
    'CopySource'  => "{$sourceBucket}/{$sourceKeyname}",
));

```

The following tasks guide you through using PHP classes to make multiple copies of an object within Amazon S3.

Copying Objects

1	Create an instance of an Amazon S3 client by using the Aws\S3\S3Client class factory() method.
2	To make multiple copies of an object, you execute a batch of calls to the Amazon S3 client getCommand() method, which is inherited from the GuzzleService\Client class. You provide the <code>CopyObject</code> command as the first argument and an array containing the source bucket, source key name, target bucket, and target key name as the second argument.

The following PHP code sample demonstrates making multiple copies of an object that is stored in Amazon S3.

```
use Aws\S3\S3Client;

$sourceBucket = '*** Your Source Bucket Name ***';
$sourceKeyname = '*** Your Source Object Key ***';
$targetBucket = '*** Your Target Bucket Name ***';
$targetKeyname = '*** Your Target Key Name ***';

// Instantiate the client.
$s3 = S3Client::factory();

// Perform a batch of CopyObject operations.
$batch = array();
for ($i = 1; $i <= 3; $i++) {
    $batch[] = $s3->getCommand('CopyObject', array(
        'Bucket'      => $targetBucket,
        'Key'         => "{$targetKeyname}-{$i}",
        'CopySource'  => "{$sourceBucket}/{$sourceKeyname}",
    ));
}
try {
    $successful = $s3->execute($batch);
    $failed = array();
} catch (\Guzzle\Service\Exception\CommandTransferException $e) {
    $successful = $e->getSuccessfulCommands();
    $failed = $e->getFailedCommands();
}
```

Example of Copying Objects within Amazon S3

The following PHP example illustrates the use of the `copyObject()` method to copy a single object within Amazon S3 and using a batch of calls to `CopyObject` using the `getCommand()` method to make multiple copies of an object.

```
<?php

// Include the AWS SDK using the Composer autoloader.
require 'vendor/autoload.php';

use Aws\S3\S3Client;

$sourceBucket = '*** Your Source Bucket Name ***';
$sourceKeyname = '*** Your Source Object Key ***';
$targetBucket = '*** Your Target Bucket Name ***';

// Instantiate the client.
$s3 = S3Client::factory();

// Copy an object.
$s3->copyObject(array(
    'Bucket'      => $targetBucket,
    'Key'         => "{$sourceKeyname}-copy",
    'CopySource'  => "{$sourceBucket}/{$sourceKeyname}",
));

// Perform a batch of CopyObject operations.
$batch = array();
for ($i = 1; $i <= 3; $i++) {
    $batch[] = $s3->getCommand('CopyObject', array(
        'Bucket'      => $targetBucket,
        'Key'         => "{$sourceKeyname}-copy-{$i}",
        'CopySource'  => "{$sourceBucket}/{$sourceKeyname}",
    ));
}
try {
    $successful = $s3->execute($batch);
    $failed = array();
} catch (\Guzzle\Service\Exception\CommandTransferException $e) {
    $successful = $e->getSuccessfulCommands();
    $failed = $e->getFailedCommands();
}
```

Related Resources

- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::copyObject\(\) Method](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::factory\(\) Method](#)
- [AWS SDK for PHP for Amazon S3 Guzzle\Service\Client Class](#)
- [AWS SDK for PHP for Amazon S3 Guzzle\Service\Client::getCommand\(\) Method](#)
- [AWS SDK for PHP for Amazon S3](#)
- [AWS SDK for PHP Documentation](#)

Copy an Object Using the AWS SDK for Ruby

The following tasks guide you through using the Ruby classes to copy an object in Amazon S3, from one bucket to another or to copy an object within the same bucket.

Copying Objects

1	Create an instance of the <code>AWS::S3</code> class by providing your AWS credentials.
2	Execute either the <code>AWS::S3::S3Object#copy_to</code> or <code>AWS::S3::S3Object#copy_from</code> method. You need to provide the request information, such as source bucket name, source key name, destination bucket name, and destination key.

The following Ruby code sample demonstrates the preceding tasks using the `#copy_to` method to copy an object from one bucket to another.

```
s3 = AWS::S3.new

# Upload a file and set server-side encryption.
bucket1 = s3.buckets[source_bucket]
bucket2 = s3.buckets[target_bucket]
obj1 = bucket1.objects[source_key]
obj2 = bucket2.objects[target_key]

obj1.copy_to(obj2)
```

Example

The following Ruby script example makes a copy of an object using the `#copy_from` method. The copied object with a different key is saved in the same source bucket. For instructions about how to create and test a working sample, see [Using the AWS SDK for Ruby \(p. 549\)](#).

```
#!/usr/bin/env ruby

require 'rubygems'
require 'aws-sdk'

bucket_name = '*** Provide bucket name ***'
source_key = '*** Provide source key ***'
target_key = '*** Provide target key ***'

# Get an instance of the S3 interface.
s3 = AWS::S3.new

# Copy the object.
s3.buckets[bucket_name].objects[target_key].copy_from(source_key)

puts "Copying file #{source_key} to #{target_key}."
```

Copy an Object Using the REST API

This example describes how to copy an object using REST. For more information about the REST API, go to [PUT Object \(Copy\)](#).

This example copies the `flotsam` object from the `pacific` bucket to the `jetsam` object of the `atlantic` bucket, preserving its metadata.

```
PUT /jetsam HTTP/1.1
Host: atlantic.s3.amazonaws.com
x-amz-copy-source: /pacific/flotsam
Authorization: AWS AKIAIOSFODNN7EXAMPLE:ENoSbxYByFA0UGLZUqJN5EUnLDg=
Date: Wed, 20 Feb 2008 22:12:21 +0000
```

The signature was generated from the following information.

```
PUT\r\n
\r\n
\r\n
Wed, 20 Feb 2008 22:12:21 +0000\r\n

x-amz-copy-source:/pacific/flotsam\r\n
/atlantic/jetsam
```

Amazon S3 returns the following response that specifies the ETag of the object and when it was last modified.

```
HTTP/1.1 200 OK
x-amz-id-2: Vyaxt7qEbzbv34BnSu5hctyyNSlHTYZFMWK4FtzO+iX8JQNyaLdTshL0KxatbaOZt
x-amz-request-id: 6B13C3C5B34AF333
Date: Date: Wed, 20 Feb 2008 22:13:01 +0000

Content-Type: application/xml
Transfer-Encoding: chunked
Connection: close
Server: AmazonS3
<?xml version="1.0" encoding="UTF-8"?>

<CopyObjectResult>
    <LastModified>2008-02-20T22:13:01</LastModified>
    <ETag>"7e9c608af58950deeb370c98608ed097"</ETag>
</CopyObjectResult>
```

Copying Objects Using the Multipart Upload API

Topics

- [Copy an Object Using the AWS SDK for Java Multipart Upload API \(p. 210\)](#)
- [Copy an Object Using the AWS SDK for .NET Multipart Upload API \(p. 213\)](#)
- [Copy Object Using the REST Multipart Upload API \(p. 218\)](#)

The examples in this section show you how to copy objects greater than 5 GB using the multipart upload API. You can copy objects less than 5 GB in a single operation. For more information, see [Copying Objects in a Single Operation \(p. 200\)](#).

Copy an Object Using the AWS SDK for Java Multipart Upload API

The following task guides you through using the Java SDK to copy an Amazon S3 object from one source location to another, such as from one bucket to another. You can use the code demonstrated here to copy objects greater than 5 GB. For objects less than 5 GB, use the single operation copy described in [Copy an Object Using the AWS SDK for Java \(p. 201\)](#).

Copying Objects

1	Create an instance of the <code>AmazonS3Client</code> class by providing your AWS credentials.
2	Initiate a multipart copy by executing the <code>AmazonS3Client.initiateMultipartUpload</code> method. Create an instance of <code>InitiateMultipartUploadRequest</code> . You will need to provide a bucket name and a key name.
3	Save the upload ID from the response object that the <code>AmazonS3Client.initiateMultipartUpload</code> method returns. You will need to provide this upload ID for each subsequent multipart upload operation.
4	Copy all the parts. For each part copy, create a new instance of the <code>CopyPartRequest</code> class and provide part information including source bucket, destination bucket, object key, uploadID, first byte of the part, last byte of the part, and the part number.
5	Save the response of the <code>CopyPartRequest</code> method in a list. The response includes the ETag value and the part number. You will need the part number to complete the multipart upload.
6	Repeat tasks 4 and 5 for each part.
7	Execute the <code>AmazonS3Client.completeMultipartUpload</code> method to complete the copy.

The following Java code sample demonstrates the preceding tasks.

```
// Step 1: Create instance and provide credentials.  
AmazonS3Client s3Client = new AmazonS3Client(new  
    PropertiesCredentials(  
        LowLevel_LargeObjectCopy.class.getResourceAsStream(  
            "AwsCredentials.properties")));
```

```
// Create lists to hold copy responses
List<CopyPartResult> copyResponses =
    new ArrayList<CopyPartResult>();

// Step 2: Initialize
InitiateMultipartUploadRequest initiateRequest =
    new InitiateMultipartUploadRequest(targetBucketName, targetObjectKey);

InitiateMultipartUploadResult initResult =
    s3Client.initiateMultipartUpload(initiateRequest);

// Step 3: Save upload Id.
String uploadId = initResult.getUploadId();

try {

    // Get object size.
    GetObjectMetadataRequest metadataRequest =
        new GetObjectMetadataRequest(sourceBucketName, sourceObjectKey);

    ObjectMetadata metadataResult = s3Client.getObjectMetadata(metadataRequest);

    long objectSize = metadataResult.getContentLength(); // in bytes

    // Step 4. Copy parts.
    long partSize = 5 * (long) Math.pow(2.0, 20.0); // 5 MB
    long bytePosition = 0;
    for (int i = 1; bytePosition < objectSize; i++) {
        // Step 5. Save copy response.
        CopyPartRequest copyRequest = new CopyPartRequest()
            .withDestinationBucketName(targetBucketName)
            .withDestinationKey(targetObjectKey)
            .withSourceBucketName(sourceBucketName)
            .withSourceKey(sourceObjectKey)
            .withUploadId(initResult.getUploadId())
            .withFirstByte(bytePosition)
            .withLastByte(bytePosition + partSize - 1 >= objectSize ? objectSize
- 1 : bytePosition + partSize - 1)
            .withPartNumber(i);

        copyResponses.add(s3Client.copyPart(copyRequest));
        bytePosition += partSize;
    }
    // Step 7. Complete copy operation.
    CompleteMultipartUploadResult completeUploadResponse =
        s3Client.completeMultipartUpload(completeRequest);
} catch (Exception e) {
    System.out.println(e.getMessage());
}
```

Example

The following Java code example copies an object from one Amazon S3 bucket to another. For instructions on how to create and test a working sample, see [Testing the Java Code Examples \(p. 545\)](#).

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.s3.*;
import com.amazonaws.services.s3.model.*;

public class LowLevel_LargeObjectCopy {

    public static void main(String[] args) throws IOException {
        String sourceBucketName = "**** Source-Bucket-Name ****";
        String targetBucketName = "**** Target-Bucket-Name ****";
        String sourceObjectKey = "**** Source-Object-Key ****";
        String targetObjectKey = "**** Target-Object-Key ****";
        AmazonS3Client s3Client = new AmazonS3Client(new
            PropertiesCredentials(
                LowLevel_LargeObjectCopy.class.getResourceAsStream(
                    "AwsCredentials.properties")));
        // List to store copy part responses.

        List<CopyPartResult> copyResponses =
            new ArrayList<CopyPartResult>();

        InitiateMultipartUploadRequest initiateRequest =
            new InitiateMultipartUploadRequest(targetBucketName, targetObjectKey);

        InitiateMultipartUploadResult initResult =
            s3Client.initiateMultipartUpload(initiateRequest);

        try {
            // Get object size.
           .GetObjectMetadataRequest metadataRequest =
                new GetObjectMetadataRequest(sourceBucketName, sourceObjectKey);

            ObjectMetadata metadataResult = s3Client.getObjectMetadata(metadataRe
quest);
            long objectSize = metadataResult.getContentLength(); // in bytes

            // Copy parts.
            long partSize = 5 * (long) Math.pow(2.0, 20.0); // 5 MB

            long bytePosition = 0;
            for (int i = 1; bytePosition < objectSize; i++) {
                CopyPartRequest copyRequest = new CopyPartRequest()
                    .withDestinationBucketName(targetBucketName)
                    .withDestinationKey(targetObjectKey)
                    .withSourceBucketName(sourceBucketName)
                    .withSourceKey(sourceObjectKey)
                    .withUploadId(initResult.getUploadId())
            }
        }
    }
}
```

```

        .withFirstByte(bytePosition)
        .withLastByte(bytePosition + partSize - 1 >= objectSize ? objectSize - 1 : bytePosition + partSize - 1)
        .withPartNumber(i);

    copyResponses.add(s3Client.copyPart(copyRequest));
    bytePosition += partSize;

}
CompleteMultipartUploadRequest completeRequest = new
    CompleteMultipartUploadRequest(
        targetBucketName,
        targetObjectKey,
        initResult.getUploadId(),
        GetETags(copyResponses));

    CompleteMultipartUploadResult completeUploadResponse =
        s3Client.completeMultipartUpload(completeRequest);
} catch (Exception e) {
    System.out.println(e.getMessage());
}
}

// Helper function that constructs ETags.
static List<PartETag> GetETags(List<CopyPartResult> responses)
{
    List<PartETag> etags = new ArrayList<PartETag>();
    for (CopyPartResult response : responses)
    {
        etags.add(new PartETag(response.getPartNumber(), re
sponse.getETag()));
    }
    return etags;
}
}

```

Copy an Object Using the AWS SDK for .NET Multipart Upload API

The following task guides you through using the .NET SDK to copy an Amazon S3 object from one source location to another, such as from one bucket to another. You can use the code demonstrated here to copy objects that are greater than 5 GB. For objects less than 5 GB, use the single operation copy described in [Copy an Object Using the AWS SDK for .NET \(p. 202\)](#).

Copying Objects

1	Create an instance of the <code>AmazonS3Client</code> class by providing your AWS credentials.
2	Initiate a multipart copy by executing the <code>AmazonS3Client.InitiateMultipartUpload</code> method. Create an instance of the <code>InitiateMultipartUploadRequest</code> . You will need to provide a bucket name and key name.
3	Save the upload ID from the response object that the <code>AmazonS3Client.InitiateMultipartUpload</code> method returns. You will need to provide this upload ID for each subsequent multipart upload operation.

4	Copy all the parts. For each part copy, create a new instance of the <code>CopyPartRequest</code> class and provide part information including source bucket, destination bucket, object key, uploadID, first byte of the part, last byte of the part, and the part number.
5	Save the response of the <code>CopyPartRequest</code> method in a list. The response includes the ETag value and the part number you will need to complete the multipart upload.
6	Repeat tasks 4 and 5 for each part.
7	Execute the <code>AmazonS3Client.CompleteMultipartUpload</code> method to complete the copy.

The following C# code sample demonstrates the preceding tasks.

```
// Step 1. Create instance and provide credentials.
IAmazonS3 s3Client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1);

// List to store upload part responses.
List<UploadPartResponse> uploadResponses = new List<UploadPartResponse>();
List<CopyPartResponse> copyResponses = new List<CopyPartResponse>();
InitiateMultipartUploadRequest initiateRequest =
    new InitiateMultipartUploadRequest
    {
        BucketName = targetBucket,
        Key = targetObjectKey
    };

// Step 2. Initialize.
InitiateMultipartUploadResponse initResponse = s3Client.InitiateMultipartUpload(initiateRequest);

// Step 3. Save Upload Id.
String uploadId = initResponse.UploadId;

try
{
    // Get object size.
    GetObjectMetadataRequest metadataRequest = new GetObjectMetadataRequest
    {
        BucketName = sourceBucket,
        Key        = sourceObjectKey
    };

    GetObjectMetadataResponse metadataResponse =
        s3Client.GetObjectMetadata(metadataRequest);
    long objectSize = metadataResponse.ContentLength; // in bytes

    // Copy parts.
    long partSize = 5 * (long)Math.Pow(2, 20); // 5 MB

    long bytePosition = 0;
    for (int i = 1; bytePosition < objectSize; i++)
    {

        CopyPartRequest copyRequest = new CopyPartRequest
```

```
        {
            DestinationBucket = targetBucket,
            DestinationKey = targetObjectKey,
            SourceBucket = sourceBucket,
            SourceKey = sourceObjectKey,
            UploadId = uploadId,
            FirstByte = bytePosition,
            LastByte = bytePosition + partSize - 1 >= objectSize ? objectSize
            - 1 : bytePosition + partSize - 1,
            PartNumber = i
        };

        copyResponses.Add(s3Client.CopyPart(copyRequest));

        bytePosition += partSize;
    }

    CompleteMultipartUploadRequest completeRequest =
    new CompleteMultipartUploadRequest
    {
        BucketName = targetBucket,
        Key = targetObjectKey,
        UploadId = initResponse.UploadId
    };

    completeRequest.AddPartETags(copyResponses);
    CompleteMultipartUploadResponse completeUploadResponse = s3Client.CompleteMultipartUpload(completeRequest);

}
catch (Exception e) {
    Console.WriteLine(e.Message);
}
```

Example

The following C# code example copies an object from one Amazon S3 bucket to another. For instructions on how to create and test a working sample, see [Testing the .NET Code Examples \(p. 547\)](#).

```
using System;
using System.Collections.Generic;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples
{
    class CopyObjectUsingMPUapi
    {

        static string sourceBucket      = "**** Source bucket name ****";
        static string targetBucket     = "**** Target bucket name ****";
        static string sourceObjectKey  = "**** Source object key ****";
        static string targetObjectKey = "**** Target object key ****";

        static void Main(string[] args)
        {
            IAmazonS3 s3Client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1);

            // List to store upload part responses.
            List<UploadPartResponse> uploadResponses = new List<UploadPartResponse>();

            List<CopyPartResponse> copyResponses = new List<CopyPartResponse>();

            InitiateMultipartUploadRequest initiateRequest =
                new InitiateMultipartUploadRequest
                {
                    BucketName = targetBucket,
                    Key = targetObjectKey
                };

            InitiateMultipartUploadResponse initResponse =
                s3Client.InitiateMultipartUpload(initiateRequest);
            String uploadId = initResponse.UploadId;

            try
            {
                // Get object size.
               GetObjectMetadataRequest metadataRequest = new GetObjectMetadataRequest
                {
                    BucketName = sourceBucket,
                    Key = sourceObjectKey
                };

                GetObjectMetadataResponse metadataResponse =
                    s3Client.GetObjectMetadata(metadataRequest);
                long objectSize = metadataResponse.ContentLength; // in bytes

                // Copy parts.
                long partSize = 5 * (long)Math.Pow(2, 20); // 5 MB
            }
        }
    }
}
```

```
        long bytePosition = 0;
        for (int i = 1; bytePosition < objectSize; i++)
        {

            CopyPartRequest copyRequest = new CopyPartRequest
            {
                DestinationBucket = targetBucket,
                DestinationKey = targetObjectKey,
                SourceBucket = sourceBucket,
                SourceKey = sourceObjectKey,
                UploadId = uploadId,
                FirstByte = bytePosition,
                LastByte = bytePosition + partSize - 1 >= objectSize
? objectSize - 1 : bytePosition + partSize - 1,
                PartNumber = i
            };

            copyResponses.Add(s3Client.CopyPart(copyRequest));

            bytePosition += partSize;
        }
        CompleteMultipartUploadRequest completeRequest =
            new CompleteMultipartUploadRequest
            {
                BucketName = targetBucket,
                Key = targetObjectKey,
                UploadId = initResponse.UploadId
            };

        completeRequest.AddPartETags(copyResponses);
        CompleteMultipartUploadResponse completeUploadResponse =
s3Client.CompleteMultipartUpload(completeRequest);

    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
}

// Helper function that constructs ETags.
static List<PartETag> GetETags(List<CopyPartResponse> responses)
{
    List<PartETag> etags = new List<PartETag>();
    foreach (CopyPartResponse response in responses)
    {
        etags.Add(new PartETag(response.PartNumber, response.ETag));
    }
    return etags;
}
}
```

Copy Object Using the REST Multipart Upload API

The following sections in the *Amazon Simple Storage Service API Reference* describe the REST API for multipart upload. For copying an existing object you use the Upload Part (Copy) API and specify the source object by adding the `x-amz-copy-source` request header in your request.

- [Initiate Multipart Upload](#)
- [Upload Part](#)
- [Upload Part \(Copy\)](#)
- [Complete Multipart Upload](#)
- [Abort Multipart Upload](#)
- [List Parts](#)
- [List Multipart Uploads](#)

You can use these APIs to make your own REST requests, or you can use one of the SDKs we provide. For more information about the SDKs, see [API Support for Multipart Upload \(p. 128\)](#).

Listing Object Keys

Keys can be listed by prefix. By choosing a common prefix for the names of related keys and marking these keys with a special character that delimits hierarchy, you can use the list operation to select and browse keys hierarchically. This is similar to how files are stored in directories within a file system.

Amazon S3 exposes a list operation that lets you enumerate the keys contained in a bucket. Keys are selected for listing by bucket and prefix. For example, consider a bucket named "dictionary" that contains a key for every English word. You might make a call to list all the keys in that bucket that start with the letter "q". List results are always returned in lexicographic (alphabetical) order.

Both the SOAP and REST list operations return an XML document that contains the names of matching keys and information about the object identified by each key.

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

Groups of keys that share a prefix terminated by a special delimiter can be rolled up by that common prefix for the purposes of listing. This enables applications to organize and browse their keys hierarchically, much like how you would organize your files into directories in a file system. For example, to extend the dictionary bucket to contain more than just English words, you might form keys by prefixing each word with its language and a delimiter, such as "French/logical". Using this naming scheme and the hierarchical listing feature, you could retrieve a list of only French words. You could also browse the top-level list of available languages without having to iterate through all the lexicographically intervening keys.

For more information on this aspect of listing, see [Listing Keys Hierarchically Using a Prefix and Delimiter \(p. 219\)](#).

List Implementation Efficiency

List performance is not substantially affected by the total number of keys in your bucket, nor by the presence or absence of the prefix, marker, maxkeys, or delimiter arguments.

Iterating Through Multi-Page Results

As buckets can contain a virtually unlimited number of keys, the complete results of a list query can be extremely large. To manage large result sets, Amazon S3 API support pagination to split them into multiple responses. Each list keys response returns a page of up to 1,000 keys with an indicator indicating if the response is truncated. You send a series of list keys requests until you have received all the keys. AWS SDK wrapper libraries provide the same pagination.

The following Java and .NET SDK examples show how to use pagination when listing keys in a bucket:

- [Listing Keys Using the AWS SDK for Java \(p. 220\)](#)
- [Listing Keys Using the AWS SDK for .NET \(p. 223\)](#)

Listing Keys Hierarchically Using a Prefix and Delimiter

The prefix and delimiter parameters limit the kind of results returned by a list operation. Prefix limits results to only those keys that begin with the specified prefix, and delimiter causes list to roll up all keys that share a common prefix into a single summary list result.

The purpose of the prefix and delimiter parameters is to help you organize and then browse your keys hierarchically. To do this, first pick a delimiter for your bucket, such as slash (/), that doesn't occur in any of your anticipated key names. Next, construct your key names by concatenating all containing levels of the hierarchy, separating each level with the delimiter.

For example, if you were storing information about cities, you might naturally organize them by continent, then by country, then by province or state. Because these names don't usually contain punctuation, you might select slash (/) as the delimiter. The following examples use a slash (/) delimiter.

- Europe/France/Aquitaine/Bordeaux
- North America/Canada/Quebec/Montreal
- North America/USA/Washington/Bellevue
- North America/USA/Washington/Seattle

If you stored data for every city in the world in this manner, it would become awkward to manage a flat key namespace. By using *Prefix* and *Delimiter* with the list operation, you can use the hierarchy you've created to list your data. For example, to list all the states in USA, set *Delimiter*='/' and *Prefix*='North America/USA/'. To list all the provinces in Canada for which you have data, set *Delimiter*='/' and *Prefix*='North America/Canada/'.

A list request with a delimiter lets you browse your hierarchy at just one level, skipping over and summarizing the (possibly millions of) keys nested at deeper levels. For example, assume you have a bucket (ExampleBucket) the following keys.

```
sample.jpg
photos/2006/January/sample.jpg
photos/2006/February/sample2.jpg
photos/2006/February/sample3.jpg
photos/2006/February/sample4.jpg
```

The sample bucket has only the `sample.jpg` object at the root level. To list only the root level objects in the bucket you send a GET request on the bucket with "/" delimiter character. In response, Amazon S3 returns the `sample.jpg` object key because it does not contain the "/" delimiter character. All other keys contain the delimiter character. Amazon S3 groups these keys and return a single `CommonPrefixes` element with prefix value `photos/` that is a substring from the beginning of these keys to the first occurrence of the specified delimiter.

```
<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Name>ExampleBucket</Name>
  <Prefix></Prefix>
  <Marker></Marker>
  <MaxKeys>1000</MaxKeys>
  <Delimiter>/</Delimiter>
  <IsTruncated>false</IsTruncated>
  <Contents>
    <Key>sample.jpg</Key>
    <LastModified>2011-07-24T19:39:30.000Z</LastModified>
    <ETag>"d1a7fb5eab1c16cb4f7cf341cf188c3d"</ETag>
    <Size>6</Size>
    <Owner>
      <ID>75cc57f09aa0c8caeab4f8c24e99d10f8e7faebf76c078efc7c6caea54ba06a</ID>
      <DisplayName>displayname</DisplayName>
    </Owner>
    <StorageClass>STANDARD</StorageClass>
  </Contents>
  <CommonPrefixes>
    <Prefix>photos/</Prefix>
  </CommonPrefixes>
</ListBucketResult>
```

Listing Keys Using the AWS SDK for Java

The following tasks guide you through using the Java classes to list object keys in a bucket. You have many options for listing the objects in your bucket. Note that for buckets with large number of objects you might get truncated results when listing the objects. You should check to see if the returned object listing is truncated, and use the `AmazonS3.listNextBatchOfObjects` operation to retrieve additional results.

Listing Object Keys

1	Create an instance of the <code>AmazonS3Client</code> class by providing your AWS credentials.
2	Execute one of the <code>AmazonS3Client.listObjects</code> methods. You need to provide the request information, such as the bucket name, and the optional key prefix. You provide this information by creating an instance of the <code>ListObjectsRequest</code> class. For listing objects, Amazon S3 returns up to 1,000 keys in the response. If you have more than 1,000 keys in your bucket, the response will be truncated. You should always check for if the response is truncated.

The following Java code sample demonstrates the preceding tasks.

```
AmazonS3 s3client = new AmazonS3Client(new ProfileCredentialsProvider());
```

```
ListObjectsRequest listObjectsRequest = new ListObjectsRequest()  
.withBucketName(bucketName)  
.withPrefix("m");  
ObjectListing objectListing;  
  
do {  
    objectListing = s3client.listObjects(listObjectsRequest);  
    for (S3ObjectSummary objectSummary :  
        objectListing.getObjectSummaries()) {  
        System.out.println( " - " + objectSummary.getKey() + " " +  
            "(size = " + objectSummary.getSize() +  
            " )");  
    }  
    listObjectsRequest.setMarker(objectListing.getNextMarker());  
} while (objectListing.isTruncated());
```

Example

The following Java code example list object keys in an Amazon S3 bucket. For instructions on how to create and test a working sample, see [Testing the Java Code Examples \(p. 545\)](#).

```
import java.io.IOException;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.ListObjectsRequest;
import com.amazonaws.services.s3.model.ObjectListing;
import com.amazonaws.services.s3.model.S3ObjectSummary;

public class ListKeys {
    private static String bucketName = "**** bucket name ****";

    public static void main(String[] args) throws IOException {
        AmazonS3 s3client = new AmazonS3Client(new ProfileCredentialsProvider());

        try {
            System.out.println("Listing objects");

            ListObjectsRequest listObjectsRequest = new ListObjectsRequest()
                .withBucketName(bucketName)
                .withPrefix("m");
            ObjectListing objectListing;
            do {
                objectListing = s3client.listObjects(listObjectsRequest);
                for (S3ObjectSummary objectSummary :
                    objectListing.getObjectSummaries()) {
                    System.out.println(" - " + objectSummary.getKey() + " " +
                        "(size = " + objectSummary.getSize() +
                        ")");
                }
                listObjectsRequest.setMarker(objectListing.getNextMarker());
            } while (objectListing.isTruncated());
        } catch (AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException, " +
                "which means your request made it " +
                "to Amazon S3, 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 " +
                "an internal error while trying to communicate" +
                " with S3, " +
                "such as not being able to access the network.");
            System.out.println("Error Message: " + ace.getMessage());
        }
    }
}
```

```
    }  
}
```

Listing Keys Using the AWS SDK for .NET

The following tasks guide you through using the .NET classes to list object keys in a bucket.

Listing Object Keys

1	Create an instance of the <code>AmazonS3</code> class.
2	Execute one of the <code>AmazonS3.ListObjects</code> . You need to provide the bucket name to list the keys. You can also specify optional information, such as retrieving keys starting with a specific prefix, and limiting result set to a specific number of keys. By default, the result set returns up to 1,000 keys. You provide this information by creating an instance of the <code>ListObjectsRequest</code> class.
3	Process the <code>ListObjectResponse</code> by iterating over the <code>ListObjectResponse.S3Objects</code> collection. You should check to see if the result was truncated. If yes, you send a request for the next set of keys by setting the <code>ListObjectRequest.Marker</code> value to the <code>NextMarker</code> value received in the previous response.

The following C# code sample demonstrates the preceding tasks.

```
static IAmazonS3 client;  
client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1);  
  
ListObjectsRequest request = new ListObjectsRequest  
{  
    BucketName = bucketName,  
    Prefix = "j",  
    MaxKeys = 2  
};  
  
do  
{  
    ListObjectsResponse response = client.ListObjects(request);  
  
    // Process response.  
    ...  
  
    // If response is truncated, set the marker to get the next  
    // set of keys.  
    if (response.IsTruncated)  
    {  
        request.Marker = response.NextMarker;  
    }  
    else  
    {  
        request = null;  
    }  
} while (request != null);
```

Example

The following C# code example lists keys in the specified bucket. The example illustrates the use of `AmazonS3.ListObjects` method. It also illustrates how you can specify options to list keys, such as listing keys with a specific prefix, listing keys that start after a specific marker, and listing only a specific number of keys. For instructions on how to create and test a working sample, see [Testing the .NET Code Examples \(p. 547\)](#).

```
using System;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples
{
    class ListObjects
    {
        static string bucketName = "**** Provide bucket name ****";
        static IAmazonS3 client;

        public static void Main(string[] args)
        {
            using (client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1))

            {
                Console.WriteLine("Listing objects stored in a bucket");
                ListingObjects();
            }

            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }

        static void ListingObjects()
        {
            try
            {
                ListObjectsRequest request = new ListObjectsRequest
                {
                    BucketName = bucketName,
                    Prefix = "j",
                    MaxKeys = 2
                };

                do
                {
                    ListObjectsResponse response = client.ListObjects(request);

                    // Process response.
                    foreach (S3Object entry in response.S3Objects)
                    {
                        Console.WriteLine("key = {0} size = {1}",
                            entry.Key, entry.Size);
                    }

                    // If response is truncated, set the marker to get the next
                    // set of keys.
                }
            }
        }
    }
}
```

```
        if (response.IsTruncated)
    {
        request.Marker = response.NextMarker;
    }
    else
    {
        request = null;
    }
} while (request != null);
}
catch (AmazonS3Exception amazonS3Exception)
{
    if (amazonS3Exception.ErrorCode != null &&
        (amazonS3Exception.ErrorCode.Equals("InvalidAccessKeyId")
        ||
        amazonS3Exception.ErrorCode.Equals("InvalidSecurity")))
    {
        Console.WriteLine("Check the provided AWS Credentials.");
        Console.WriteLine(
            "To sign up for service, go to http://aws.amazon.com/s3");
    }
    else
    {
        Console.WriteLine(
            "Error occurred. Message:'{0}' when listing objects",
            amazonS3Exception.Message);
    }
}
}
```

Listing Keys Using the AWS SDK for PHP

This topic guides you through using classes from the AWS SDK for PHP to list the object keys contained in an Amazon S3 bucket.

Note

This topic assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 547\)](#) and have the AWS SDK for PHP properly installed.

To list the object keys contained in a bucket using the AWS SDK for PHP you first must list the objects contained in the bucket and then extract the key from each of the listed objects. When listing objects in a bucket you have the option of using the low-level [Aws\S3\S3Client::listObjects\(\)](#) method or the high-level [Aws\S3\Iterator\ListObjects](#) iterator.

The low-level `listObjects()` method maps to the underlying Amazon S3 REST API. Each `listObjects()` request returns a page of up to 1,000 objects. If you have more than 1,000 objects in the bucket, your response will be truncated and you will need to send another `listObjects()` request to retrieve the next set of 1,000 objects.

You can use the high-level `ListObjects` iterator to make your task of listing the objects contained in a bucket a bit easier. To use the `ListObjects` iterator to create a list of objects you execute the Amazon S3 client `getIterator()` method that is inherited from `GuzzleService\Client` class with the `ListObjects` command as the first argument and an array to contain the returned objects from the specified bucket as the second argument. When used as a `ListObjects` iterator the `getIterator()` method returns all

the objects contained in the specified bucket. There is no 1,000 object limit, so you don't need to worry if the response is truncated or not.

The following tasks guide you through using the PHP Amazon S3 client methods to list the objects contained in a bucket from which you can list the object keys.

Listing Object Keys

1	Create an instance of an Amazon S3 client by using the Aws\S3\S3Client class factory method.
2	Execute the high-level Amazon S3 client <code>getIterator()</code> method with the <code>ListObjects</code> command as the first argument and an array to contain the returned objects from the specified bucket as the second argument. Or you can execute the low-level Amazon S3 client <code>listObjects()</code> method with an array to contain the returned objects from the specified bucket as the argument.
3	Extract the object key from each object in the list of returned objects.

The following PHP code sample demonstrates how to list the objects contained in a bucket from which you can list the object keys.

```
use Aws\S3\S3Client;

// Instantiate the client.
$s3 = S3Client::factory();

$bucket = '*** Bucket Name ***';

// Use the high-level iterators (returns ALL of your objects).
$objects = $s3->getIterator('ListObjects', array('Bucket' => $bucket));

echo "Keys retrieved!\n";
foreach ($objects as $object) {
    echo $object['Key'] . "\n";
}

// Use the plain API (returns ONLY up to 1000 of your objects).
$result = $s3->listObjects(array('Bucket' => $bucket));

echo "Keys retrieved!\n";
foreach ($result['Contents'] as $object) {
    echo $object['Key'] . "\n"
}
```

Example of Listing Object Keys

The following PHP example demonstrates how to list the keys from a specified bucket. It shows how to use the high-level `getIterator()` method to list the objects in a bucket and then how to extract the key from each of the objects in the list. It also shows how to use the low-level `listObjects()` method to list the objects in a bucket and then how to extract the key from each of the objects in the list returned. For information about running the PHP examples in this guide, go to [Running PHP Examples \(p. 548\)](#).

```
<?php

// Include the AWS SDK using the Composer autoloader.
require 'vendor/autoload.php';

use Aws\S3\S3Client;
use Aws\S3\Exception\S3Exception;

$bucket = '*** Your Bucket Name ***';

// Instantiate the client.
$s3 = S3Client::factory();

// Use the high-level iterators (returns ALL of your objects).
try {
    $objects = $s3->getIterator('ListObjects', array(
        'Bucket' => $bucket
    ));

    echo "Keys retrieved!\n";
    foreach ($objects as $object) {
        echo $object['Key'] . "\n";
    }
} catch (S3Exception $e) {
    echo $e->getMessage() . "\n";
}

// Use the plain API (returns ONLY up to 1000 of your objects).
try {
    $result = $s3->listObjects(array('Bucket' => $bucket));

    echo "Keys retrieved!\n";
    foreach ($result['Contents'] as $object) {
        echo $object['Key'] . "\n";
    }
} catch (S3Exception $e) {
    echo $e->getMessage() . "\n";
}
```

Related Resources

- [AWS SDK for PHP for Amazon S3 Aws\S3\Iterator\ListObjects](#)
- [AWS SDK for PHP for Amazon S3 Guzzle\Service\Client Class](#)
- [AWS SDK for PHP for Amazon S3 Guzzle\Service\Client::getIterator\(\) Method](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::factory\(\) Method](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::listObjects\(\) Method](#)
- [AWS SDK for PHP for Amazon S3](#)

- AWS SDK for PHP Documentation

Listing Keys Using the REST API

You can use the AWS SDK to retrieve an object. However, if your application requires it, you can send REST requests directly. You can send a GET request to retrieve some or all of the objects in a bucket or you can use selection criteria to return a subset of the objects in a bucket. For more information, go to [GET Bucket \(List Objects\)](#).

Related Resources

- [Using the AWS SDKs and Explorers \(p. 542\)](#)

Deleting Objects

Topics

- [Deleting Objects from a Version-Enabled Bucket \(p. 228\)](#)
- [Deleting Objects from an MFA-Enabled Bucket \(p. 229\)](#)
- [Related Resources \(p. 229\)](#)
- [Deleting One Object Per Request \(p. 229\)](#)
- [Deleting Multiple Objects Per Request \(p. 239\)](#)

You can delete one or more objects directly from Amazon S3. You have the following options when deleting an object:

- **Delete a single object**—Amazon S3 provides the DELETE API that you can use to delete one object in a single HTTP request.
- **Delete multiple objects**—Amazon S3 also provides the Multi-Object Delete API that you can use to delete up to 1000 objects in a single HTTP request.

When deleting objects from a bucket that is not version-enabled, you provide only the object key name, however, when deleting objects from a version-enabled bucket, you can optionally provide version ID of the object to delete specific a version of the object.

Deleting Objects from a Version-Enabled Bucket

If your bucket is version-enabled, then multiple versions of the same object can exist in the bucket. When working with version-enabled buckets, the delete API enables the following options:

- **Specify a non-versioned delete request**—That is, you specify only the object's key, and not the version ID. In this case, Amazon S3 creates a delete marker and returns its version ID in the response. This makes your object disappear from the bucket. For information about object versioning and the delete marker concept, see [Object Versioning \(p. 84\)](#).
- **Specify a versioned delete request**—That is, you specify both the key and also a version ID. In this case the following two outcomes are possible:
 - If the version ID maps to a specific object version, then Amazon S3 deletes the specific version of the object.
 - If the version ID maps to the delete marker of that object, Amazon S3 deletes the delete marker. This makes the object reappear in your bucket.

Deleting Objects from an MFA-Enabled Bucket

When deleting objects from an Multi Factor Authentication (MFA) enabled bucket, note the following:

- If you provide an invalid MFA token, the request always fails.
- If you have MFA-enabled bucket, and you make a versioned delete request (you provide an object key and version ID), the request will fail if you don't provide a valid MFA token. In addition, when using the Multi-Object Delete API on an MFA-enabled bucket, if any of the deletes is a versioned delete request (that is, you specify object key and version ID), the entire request will fail if you don't provide MFA token.

On the other hand, in the following cases the request succeeds:

- If you have an MFA enabled bucket, and you make a non-versioned delete request (you are not deleting a versioned object), and you don't provide MFA token, the delete succeeds.
- If you have a Multi-Object Delete request specifying only non-versioned objects to delete from an MFA-enabled bucket, and you don't provide an MFA token, the deletions succeed.

For information on MFA delete, see [MFA Delete \(p. 423\)](#).

Related Resources

- [Using the AWS SDKs and Explorers \(p. 542\)](#)

Deleting One Object Per Request

Topics

- [Deleting an Object Using the AWS SDK for Java \(p. 229\)](#)
- [Deleting an Object Using the AWS SDK for .NET \(p. 233\)](#)
- [Deleting an Object Using the AWS SDK for PHP \(p. 237\)](#)
- [Deleting an Object Using the REST API \(p. 239\)](#)

Amazon S3 provides the DELETE API (see [DELETE Object](#)) for you to delete one object per request. To learn more about object deletion, see [Deleting Objects \(p. 228\)](#).

You can use the REST API directly or use the wrapper libraries provided by the AWS SDKs that can simplify your application development.

Deleting an Object Using the AWS SDK for Java

The following tasks guide you through using the AWS SDK for Java classes to delete an object.

Deleting an Object (Non-Versioned Bucket)

1	Create an instance of the <code>AmazonS3Client</code> class.
2	Execute one of the <code>AmazonS3Client.deleteObject</code> methods. You can provide a bucket name and an object name as parameters or provide the same information in a <code>DeleteObjectRequest</code> object and pass the object as a parameter. If you have not enabled versioning on the bucket, the operation deletes the object. If you have enabled versioning, the operation adds a delete marker. For more information, see Deleting One Object Per Request (p. 229) .

The following Java sample demonstrates the preceding steps. The sample uses the `DeleteObjectRequest` class to provide a bucket name and an object key.

```
AmazonS3 s3client = new AmazonS3Client(new ProfileCredentialsProvider());  
s3client.deleteObject(new DeleteObjectRequest(bucketName, keyName));
```

Deleting a Specific Version of an Object (Version-Enabled Bucket)

1	Create an instance of the <code>AmazonS3Client</code> class.
2	Execute one of the <code>AmazonS3Client.deleteVersion</code> methods. You can provide a bucket name and an object key directly as parameters or use the <code>DeleteVersionRequest</code> to provide the same information.

The following Java sample demonstrates the preceding steps. The sample uses the `DeleteVersionRequest` class to provide a bucket name, an object key, and a version Id.

```
AmazonS3 s3client = new AmazonS3Client(new ProfileCredentialsProvider());  
s3client.deleteObject(new DeleteVersionRequest(bucketName, keyName, versionId));
```

Example 1: Deleting an Object (Non-Versioned Bucket)

The following Java example deletes an object from a bucket. If you have not enabled versioning on the bucket, Amazon S3 deletes the object. If you have enabled versioning, Amazon S3 adds a delete marker and the object is not deleted. For information about how to create and test a working sample, see [Testing the Java Code Examples \(p. 545\)](#).

```
import java.io.IOException;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.DeleteObjectRequest;

public class DeleteAnObjectNonVersionedBucket {

    private static String bucketName = "**** Provide a Bucket Name ****";
    private static String keyName     = "**** Provide a Key Name *****";

    public static void main(String[] args) throws IOException {
        AmazonS3 s3Client = new AmazonS3Client(new ProfileCredentialsProvider());

        try {
            s3Client.deleteObject(new DeleteObjectRequest(bucketName, keyName));

        } catch (AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException.");
            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.");
            System.out.println("Error Message: " + ace.getMessage());
        }
    }
}
```

Example 2: Deleting an Object (Versioned Bucket)

The following Java example deletes a specific version of an object from a versioned bucket. The `deleteObject` request removes the specific object version from the bucket.

To test the sample, you must provide a bucket name. The code sample performs the following tasks:

1. Enable versioning on the bucket.
2. Add a sample object to the bucket. In response, Amazon S3 returns the version ID of the newly added object.
3. Delete the sample object using the `deleteVersion` method. The `DeleteVersionRequest` class specifies both an object key name and a version ID.

For information about how to create and test a working sample, see [Testing the Java Code Examples \(p. 545\)](#).

```
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.util.Random;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.BucketVersioningConfiguration;
import com.amazonaws.services.s3.model.CannedAccessControlList;
import com.amazonaws.services.s3.model.DeleteVersionRequest;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.PutObjectRequest;
import com.amazonaws.services.s3.model.PutObjectResult;
import com.amazonaws.services.s3.model.SetBucketVersioningConfigurationRequest;

public class DeleteAnObjectVersionEnabledBucket {

    static String bucketName = "*** Provide a Bucket Name ***";
    static String keyName     = "*** Provide a Key Name ****";
    static AmazonS3Client s3Client;

    public static void main(String[] args) throws IOException {
        s3Client = new AmazonS3Client(new ProfileCredentialsProvider());
        try {
            // Make the bucket version-enabled.
            enableVersioningOnBucket(s3Client, bucketName);

            // Add a sample object.
            String versionId = putAnObject(keyName);

            s3Client.deleteVersion(
                new DeleteVersionRequest(
                    bucketName,
                    keyName,
                    versionId));

        } catch (AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException.");
            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.");
        System.out.println("Error Message: " + ace.getMessage());
    }
}

static void enableVersioningOnBucket(AmazonS3Client s3Client,
    String bucketName) {
    BucketVersioningConfiguration config = new BucketVersioningConfiguration()
        .withStatus(BucketVersioningConfiguration.ENABLED);
    SetBucketVersioningConfigurationRequest setBucketVersioningConfigurationRequest =
        new SetBucketVersioningConfigurationRequest(
            bucketName, config);
    s3Client.setBucketVersioningConfiguration(setBucketVersioningConfigurationRequest);
}

static String putAnObject(String keyName) {
    String content = "This is the content body!";
    String key = "ObjectToDelete-" + new Random().nextInt();
    ObjectMetadata metadata = new ObjectMetadata();
    metadata.setHeader("Subject", "Content-As-Object");
    metadata.setHeader("Content-Length", content.length());
    PutObjectRequest request = new PutObjectRequest(bucketName, key,
        new ByteArrayInputStream(content.getBytes()), metadata)
        .withCannedAcl(CannedAccessControlList.AuthenticatedRead);
    PutObjectResult response = s3Client.putObject(request);
    return response.getVersionId();
}
```

Deleting an Object Using the AWS SDK for .NET

You can delete an object from a bucket. If you have versioning enabled on the bucket, you can also delete a specific version of an object.

The following tasks guide you through using the .NET classes to delete an object.

Deleting an Object (Non-Versioned Bucket)

1	Create an instance of the <code>AmazonS3Client</code> class by providing your AWS credentials.
2	Execute the <code>AmazonS3.DeleteObject</code> method by providing a bucket name and an object key in an instance of <code>DeleteObjectRequest</code> . If you have not enabled versioning on the bucket, the operation deletes the object. If you have enabled versioning, the operation adds a delete marker. For more information, see Deleting One Object Per Request (p. 229) .

The following C# code sample demonstrates the preceding steps.

```
static IAmazonS3 client;
client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1);

DeleteObjectRequest deleteObjectRequest =
    new DeleteObjectRequest
    {
        BucketName = bucketName,
        Key = keyName
    };

using (client = Amazon.AWSClientFactory.CreateAmazonS3Client(
    accessKeyID, secretAccessKeyID))
{
    client.DeleteObject(deleteObjectRequest);
}
```

Deleting a Specific Version of an Object (Version-Enabled Bucket)

1	Create an instance of the <code>AmazonS3Client</code> class by providing your AWS credentials.
2	Execute the <code>AmazonS3.DeleteObject</code> method by providing a bucket name, an object key name, and object version Id in an instance of <code>DeleteObjectRequest</code> . The <code>DeleteObject</code> method deletes the specific version of the object.

The following C# code sample demonstrates the preceding steps.

```
IAmazonS3 client
client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1)

DeleteObjectRequest request = new DeleteObjectRequest
{
    BucketName = bucketName,
    Key = keyName,
    VersionId = versionID
};

using (client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1))
{
    client.DeleteObject(deleteObjectRequest);
    Console.WriteLine("Deleting an object");
}
```

Example 1: Deleting an Object (Non-Versioned Bucket)

The following C# code example deletes an object from a bucket. It does not provide a version Id in the delete request. If you have not enabled versioning on the bucket, Amazon S3 deletes the object. If you have enabled versioning, Amazon S3 adds a delete marker and the object is not deleted. For information about how to create and test a working sample, see [Testing the .NET Code Examples \(p. 547\)](#).

```
using System;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples
{
    class DeleteObjectNonVersionedBucket
    {
        static string bucketName = "**** Provide a bucket name ****";
        static string keyName    = "**** Provide a key name ****";
        static IAmazonS3 client;

        public static void Main(string[] args)
        {
            using (client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1))

            {
                DeleteObjectRequest deleteObjectRequest = new DeleteObjectRequest

                {
                    BucketName = bucketName,
                    Key = keyName
                };
                try
                {
                    client.DeleteObject(deleteObjectRequest);
                    Console.WriteLine("Deleting an object");
                }
                catch (AmazonS3Exception s3Exception)
                {
                    Console.WriteLine(s3Exception.Message,
                                      s3Exception.InnerException);
                }
            }
            Console.WriteLine("Press any key to continue... ");
            Console.ReadKey();
        }
    }
}
```

Example 2: Deleting an Object (Versioned Bucket)

The following C# code example deletes an object from a versioned bucket. The `DeleteObjectRequest` instance specifies an object key name and a version ID. The `DeleteObject` method removes the specific object version from the bucket.

To test the sample, you must provide a bucket name. The code sample performs the following tasks:

1. Enable versioning on the bucket.
2. Add a sample object to the bucket. In response, Amazon S3 returns the version ID of the newly added object. You can also obtain version IDs of an object by sending a `ListVersions` request.

```
var listResponse = client.ListVersions(new ListVersionsRequest { BucketName = bucketName, Prefix = keyName });
```

3. Delete the sample object using the `DeleteObject` method. The `DeleteObjectRequest` class specifies both an object key name and a version ID.

For information about how to create and test a working sample, see [Testing the .NET Code Examples \(p. 547\)](#).

```
using System;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazonaws.com.docsamples
{
    class DeleteObjectVersion
    {
        static string bucketName = "**** Provide a Bucket Name ****";
        static string keyName     = "**** Provide a Key Name ****";
        static IAmazonS3 client;

        public static void Main(string[] args)
        {
            using (client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1))
            {
                try
                {
                    // Make the bucket version-enabled.
                    EnableVersioningOnBucket(bucketName);

                    // Add a sample object.
                    string versionID = PutAnObject(keyName);

                    // Delete the object by specifying an object key and a
version ID.
                    DeleteObjectRequest request = new DeleteObjectRequest
                    {
                        BucketName = bucketName,
                        Key = keyName,
                        VersionId = versionID
                    };
                    Console.WriteLine("Deleting an object");
                    client.DeleteObject(request);
                }
            }
        }
    }
}
```

```
        }
        catch (AmazonS3Exception s3Exception)
        {
            Console.WriteLine(s3Exception.Message,
                s3Exception.InnerException);
        }
    }
    Console.WriteLine("Press any key to continue... ");
    Console.ReadKey();
}

static void EnableVersioningOnBucket(string bucketName)
{
    PutBucketVersioningRequest setBucketVersioningRequest = new PutBucketVersioningRequest
    {
        BucketName = bucketName,
        VersioningConfig = new S3BucketVersioningConfig { Status = VersionStatus.Enabled };
    };
    client.PutBucketVersioning(setBucketVersioningRequest);
}

static string PutAnObject(string objectKey)
{
    PutObjectRequest request = new PutObjectRequest
    {
        BucketName = bucketName,
        Key = objectKey,
        ContentBody = "This is the content body!"
    };

    PutObjectResponse response = client.PutObject(request);
    return response.VersionId;
}
}
```

Deleting an Object Using the AWS SDK for PHP

This topic guides you through using classes from the AWS SDK for PHP to delete an object from a non-versioned bucket. For information on deleting an object from a versioned bucket, see [Deleting an Object Using the REST API \(p. 239\)](#).

Note

This topic assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 547\)](#) and have the AWS SDK for PHP properly installed.

Deleting One Object (Non-Versioned Bucket)

1	Create an instance of an Amazon S3 client by using the Aws\S3\S3Client class factory() method.
---	--

2	Execute the Aws\S3\S3Client::deleteObject() method. You must provide a bucket name and a key name in the array parameter's required keys, Bucket and Key. If you have not enabled versioning on the bucket, the operation deletes the object. If you have enabled versioning, the operation adds a delete marker. For more information, see Deleting Objects (p. 228) .
---	--

The following PHP code sample demonstrates how to delete an object from an Amazon S3 bucket using the `deleteObject()` method.

```
use Aws\S3\S3Client;

$s3 = S3Client::factory();

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

$result = $s3->deleteObject(array(
    'Bucket' => $bucket,
    'Key'     => $keyname
));
```

Example Deleting an Object from a Non-Versioned Bucket

The following PHP code example deletes an object from a bucket. It does not provide a version Id in the delete request. If you have not enabled versioning on the bucket, Amazon S3 deletes the object. If you have enabled versioning, Amazon S3 adds a delete marker and the object is not deleted. For information about running the PHP examples in this guide, go to [Running PHP Examples \(p. 548\)](#). For information on deleting an object from a versioned bucket, see [Deleting an Object Using the REST API \(p. 239\)](#).

```
<?php

// Include the AWS SDK using the Composer autoloader.
require 'vendor/autoload.php';

use Aws\S3\S3Client;

$s3 = S3Client::factory();

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

$result = $s3->deleteObject(array(
    'Bucket' => $bucket,
    'Key'     => $keyname
));
```

Related Resources

- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::factory\(\) Method](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::deleteObject\(\) Method](#)
- [AWS SDK for PHP for Amazon S3](#)
- [AWS SDK for PHP Documentation](#)

Deleting an Object Using the REST API

You can use the AWS SDKs to delete an object. However, if your application requires it, you can send REST requests directly. For more information, go to [DELETE Object](#) in the *Amazon Simple Storage Service API Reference*.

Deleting Multiple Objects Per Request

Topics

- [Deleting Multiple Objects Using the AWS SDK for Java \(p. 239\)](#)
- [Deleting Multiple Objects Using the AWS SDK for .NET \(p. 248\)](#)
- [Deleting Multiple Objects Using the AWS SDK for PHP \(p. 257\)](#)
- [Deleting Multiple Objects Using the REST API \(p. 261\)](#)

Amazon S3 provides the Multi-Object Delete API (see [Delete - Multi-Object Delete](#)) that enables you to delete multiple objects in a single request. The API supports two modes for the response; verbose and quiet. By default, the operation uses verbose mode in which the response includes the result each keys deletion that was encountered in your request. In quiet mode, the response includes only keys where the delete operation encountered an error.

If all keys were successfully deleted when using the quiet mode, Amazon S3 returns empty response.

To learn more about object deletion, see [Deleting Objects \(p. 228\)](#).

You can use the REST API directly or use the AWS SDKs.

Deleting Multiple Objects Using the AWS SDK for Java

The following tasks guide you through using the AWS SDK for Java classes to delete multiple objects in a single HTTP request.

Deleting Multiple Objects (Non-Versioned Bucket)

1	Create an instance of the <code>AmazonS3Client</code> class.
2	Create an instance of the <code>DeleteObjectsRequest</code> class and provide a list of objects keys you want to delete.
3	Execute the <code>AmazonS3Client.deleteObjects</code> method.

The following Java code sample demonstrates the preceding steps.

```
DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest(bucketName);

List<KeyVersion> keys = new ArrayList<KeyVersion>();
keys.add(new KeyVersion(keyName1));
keys.add(new KeyVersion(keyName2));
keys.add(new KeyVersion(keyName3));

multiObjectDeleteRequest.setKeys(keys);

try {
```

```
    DeleteObjectsResult delObjRes = s3Client.deleteObjects(multiObjectDelete
Request);
    System.out.format("Successfully deleted all the %s items.\n", delObjRes.get
DeletedObjects().size());
} catch (MultiObjectDeleteException e) {
    // Process exception.
}
```

In the event of an exception, you can review the `MultiObjectDeleteException` to determine which objects failed to delete and why as shown in the following Java example.

```
System.out.format("%s \n", e.getMessage());
System.out.format("No. of objects successfully deleted = %s\n", e.getDeletedOb
jects().size());
System.out.format("No. of objects failed to delete = %s\n", e.ge
tErrors().size());
System.out.format("Printing error data...\n");
for (DeleteError deleteError : e.getErrors()){
    System.out.format("Object Key: %s\t%s\t%s\n",
        deleteError.getKey(), deleteError.getCode(), deleteError.getMes
sage());
}
```

The following tasks guide you through deleting objects from a version-enabled bucket.

Deleting Multiple Objects (Version-Enabled Bucket)

1	Create an instance of the <code>AmazonS3Client</code> class.
2	Create an instance of the <code>DeleteObjectsRequest</code> class and provide a list of objects keys and optionally the version IDs of the objects that you want to delete. If you specify the version ID of the object that you want to delete, Amazon S3 deletes the specific object version. If you don't specify the version ID of the object that you want to delete, Amazon S3 adds a delete marker. For more information, see Deleting One Object Per Re quest (p. 229) .
3	Execute the <code>AmazonS3Client.deleteObjects</code> method.

The following Java code sample demonstrates the preceding steps.

```
List<KeyVersion> keys = new ArrayList<KeyVersion>();
// Provide a list of object keys and versions.

DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest(buck
etName)
.withKeys(keys);

try {
    DeleteObjectsResult delObjRes = s3Client.deleteObjects(multiObjectDelete
Request);
    System.out.format("Successfully deleted all the %s items.\n", delObjRes.get
DeletedObjects().size());
```

```
} catch (MultiObjectDeleteException e) {  
    // Process exception.  
}
```

Example 1: Multi-Object Delete (Non-Versioned Bucket)

The following Java code example uses the Multi-Object Delete API to delete objects from a non-versioned bucket. The example first uploads the sample objects to the bucket and then uses the `deleteObjects` method to delete the objects in a single request.

For information about how to create and test a working sample, see [Testing the Java Code Examples \(p. 545\)](#).

```
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.CannedAccessControlList;
import com.amazonaws.services.s3.model.DeleteObjectsRequest;
import com.amazonaws.services.s3.model.DeleteObjectsRequest.KeyVersion;
import com.amazonaws.services.s3.model.DeleteObjectsResult;
import com.amazonaws.services.s3.model.MultiObjectDeleteException;
import com.amazonaws.services.s3.model.MultiObjectDeleteException.DeleteError;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.PutObjectRequest;
import com.amazonaws.services.s3.model.PutObjectResult;

public class DeleteMultipleObjectsNonVersionedBucket {

    static String bucketName = "**** Provide a bucket name ****";
    static AmazonS3Client s3Client;

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

        try {
            s3Client = new AmazonS3Client(new ProfileCredentialsProvider());
            // Upload sample objects. Because the bucket is not version-enabled,
            // the KeyVersions list returned will have null values for version
            IDs.
            List<KeyVersion> keysAndVersions1 = putObjects(3);

            // Delete specific object versions.
            multiObjectNonVersionedDelete(keysAndVersions1);

        } catch (AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException.");
            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.");
            System.out.println("Error Message: " + ace.getMessage());
        }
    }
}
```

```
}

static List<KeyVersion> putObjects(int number) {
    List<KeyVersion> keys = new ArrayList<KeyVersion>();
    String content = "This is the content body!";
    for (int i = 0; i < number; i++) {
        String key = "ObjectToDelete-" + new Random().nextInt();
        ObjectMetadata metadata = new ObjectMetadata();
        metadata.setHeader("Subject", "Content-As-Object");
        metadata.setHeader("Content-Length", (long)content.length());
        PutObjectRequest request = new PutObjectRequest(bucketName, key,
            new ByteArrayInputStream(content.getBytes()), metadata)
                .withCannedAcl(CannedAccessControlList.AuthenticatedRead);

        PutObjectResult response = s3Client.putObject(request);
        KeyVersion keyVersion = new KeyVersion(key, response.getVersionId());

        keys.add(keyVersion);
    }
    return keys;
}

static void multiObjectNonVersionedDelete(List<KeyVersion> keys) {

    // Multi-object delete by specifying only keys (no version ID).
    DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest(
        bucketName).withQuiet(false);

    // Create request that include only object key names.
    List<KeyVersion> justKeys = new ArrayList<KeyVersion>();
    for (KeyVersion key : keys) {
        justKeys.add(new KeyVersion(key.getKey()));
    }
    multiObjectDeleteRequest.setKeys(justKeys);
    // Execute DeleteObjects - Amazon S3 add delete marker for each object

    // deletion. The objects no disappear from your bucket (verify).
    DeleteObjectsResult delObjRes = null;
    try {
        delObjRes = s3Client.deleteObjects(multiObjectDeleteRequest);
        System.out.format("Successfully deleted all the %s items.\n", de
1ObjRes.getDeletedObjects().size());
    } catch (MultiObjectDeleteException mode) {
        printDeleteResults(mode);
    }
}
static void printDeleteResults(MultiObjectDeleteException mode) {
    System.out.format("%s \n", mode.getMessage());
    System.out.format("No. of objects successfully deleted = %s\n",
mode.getDeletedObjects().size());
    System.out.format("No. of objects failed to delete = %s\n", mode.ge
tErrors().size());
    System.out.format("Printing error data...\n");
    for (DeleteError deleteError : mode.getErrors()){
        System.out.format("Object Key: %s\t%s\t%s\n",
deleteError.getKey(), deleteError.getCode(), deleteError.get
Message());
    }
}
```

```
    }  
}
```

Example 2: Multi-Object Delete (Version-Enabled Bucket)

The following Java code example uses the Multi-Object Delete API to delete objects from a version-enabled bucket.

Before you can test the sample, you must create a sample bucket and provide the bucket name in the example. You can use the AWS Management Console to create a bucket.

The example performs the following actions:

1. Enable versioning on the bucket.
2. Perform a versioned-delete.

The example first uploads the sample objects. In response, Amazon S3 returns the version IDs for each sample object that you uploaded. The example then deletes these objects using the Multi-Object Delete API. In the request, it specifies both the object keys and the version IDs (that is, versioned delete).

3. Perform a non-versioned delete.

The example uploads the new sample objects. Then, it deletes the objects using the Multi-Object API. However, in the request, it specifies only the object keys. In this case, Amazon S3 adds the delete markers and the objects disappear from your bucket.

4. Delete the delete markers.

To illustrate how the delete markers work, the sample deletes the delete markers. In the Multi-Object Delete request, it specifies the object keys and the version IDs of the delete markers it received in the response in the preceding step. This action makes the objects reappear in your bucket.

For information about how to create and test a working sample, see [Testing the Java Code Examples \(p. 545\)](#).

```
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.BucketVersioningConfiguration;
import com.amazonaws.services.s3.model.CannedAccessControlList;
import com.amazonaws.services.s3.model.DeleteObjectsRequest;
import com.amazonaws.services.s3.model.DeleteObjectsRequest.KeyVersion;
import com.amazonaws.services.s3.model.DeleteObjectsResult;
import com.amazonaws.services.s3.model.DeleteObjectsResult.DeletedObject;
import com.amazonaws.services.s3.model.MultiObjectDeleteException;
import com.amazonaws.services.s3.model.MultiObjectDeleteException.DeleteError;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.PutObjectRequest;
import com.amazonaws.services.s3.model.PutObjectResult;
import com.amazonaws.services.s3.model.SetBucketVersioningConfigurationRequest;

public class DeleteMultipleObjectsVersionEnabledBucket {

    static String bucketName = "**** Provide a bucket name ****";
}
```

```
static AmazonS3Client s3Client;

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

    try {
        s3Client = new AmazonS3Client(new ProfileCredentialsProvider());

        // 1. Enable versioning on the bucket.
        enableVersioningOnBucket(s3Client, bucketName);

        // 2a. Upload sample objects.
        List<KeyVersion> keysAndVersions1 = putObjects(3);
        // 2b. Delete specific object versions.
        multiObjectVersionedDelete(keysAndVersions1);

        // 3a. Upload samples objects.
        List<KeyVersion> keysAndVersions2 = putObjects(3);
        // 3b. Delete objects using only keys. Amazon S3 creates a delete
marker and
        // returns its version Id in the response.
        DeleteObjectsResult response = multiObjectNonVersionedDelete(key
sAndVersions2);
        // 3c. Additional exercise - using multi-object versioned delete,
remove the
        // delete markers received in the preceding response. This results
in your objects
        // reappear in your bucket
        multiObjectVersionedDeleteRemoveDeleteMarkers(response);

    } catch (AmazonServiceException ase) {
        System.out.println("Caught an AmazonServiceException.");
        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.");
        System.out.println("Error Message: " + ace.getMessage());
    }
}

static void enableVersioningOnBucket(AmazonS3Client s3Client,
    String bucketName) {
    BucketVersioningConfiguration config = new BucketVersioningConfigura
tion()
        .withStatus(BucketVersioningConfiguration.ENABLED);
    SetBucketVersioningConfigurationRequest setBucketVersioningConfigura
tionRequest = new SetBucketVersioningConfigurationRequest(
        bucketName, config);
    s3Client.setBucketVersioningConfiguration(setBucketVersioningConfigura
tionRequest);
}

static List<KeyVersion> putObjects(int number) {
    List<KeyVersion> keys = new ArrayList<KeyVersion>();
    String content = "This is the content body!";
    for (int i = 0; i < number; i++) {
```

```
String key = "ObjectToDelete-" + new Random().nextInt();
ObjectMetadata metadata = new ObjectMetadata();
metadata.setHeader("Subject", "Content-As-Object");
metadata.setHeader("Content-Length", (long)content.length());
PutObjectRequest request = new PutObjectRequest(bucketName, key,
    new ByteArrayInputStream(content.getBytes()), metadata)
    .withCannedAcl(CannedAccessControlList.AuthenticatedRead);

PutObjectResult response = s3Client.putObject(request);
KeyVersion keyVersion = new KeyVersion(key, response.getVersionId());

keys.add(keyVersion);
}
return keys;
}

static void multiObjectVersionedDelete(List<KeyVersion> keys) {
    DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest(
        bucketName).withKeys(keys);

    DeleteObjectsResult delObjRes = null;
    try {
        delObjRes = s3Client.deleteObjects(multiObjectDeleteRequest);
        System.out.format("Successfully deleted all the %s items.\n", de
lObjRes.getDeletedObjects().size());
    } catch(MultiObjectDeleteException mode) {
        printDeleteResults(mode);
    }
}

static DeleteObjectsResult multiObjectNonVersionedDelete(List<KeyVersion>
keys) {

    // Multi-object delete by specifying only keys (no version ID).
    DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest(
        bucketName);

    // Create request that include only object key names.
    List<KeyVersion> justKeys = new ArrayList<KeyVersion>();
    for (KeyVersion key : keys) {
        justKeys.add(new KeyVersion(key.getKey()));
    }

    multiObjectDeleteRequest.setKeys(justKeys);
    // Execute DeleteObjects - Amazon S3 add delete marker for each object

    // deletion. The objects no disappear from your bucket (verify).
    DeleteObjectsResult delObjRes = null;
    try {
        delObjRes = s3Client.deleteObjects(multiObjectDeleteRequest);
        System.out.format("Successfully deleted all the %s items.\n", de
lObjRes.getDeletedObjects().size());
    } catch (MultiObjectDeleteException mode) {
        printDeleteResults(mode);
    }
    return delObjRes;
}
```

```

        }
    static void multiObjectVersionedDeleteRemoveDeleteMarkers(
        DeleteObjectsResult response) {

        List<KeyVersion> keyVersionList = new ArrayList<KeyVersion>();
        for (DeletedObject deletedObject : response.getDeletedObjects()) {
            keyVersionList.add(new KeyVersion(deletedObject.getKey(),
                deletedObject.getDeleteMarkerVersionId()));
        }
        // Create a request to delete the delete markers.
        DeleteObjectsRequest multiObjectDeleteRequest2 = new DeleteObjects
Request(
            bucketName).withKeys(keyVersionList);

        // Now delete the delete marker bringing your objects back to the
        bucket.
        DeleteObjectsResult delObjRes = null;
        try {
            delObjRes = s3Client.deleteObjects(multiObjectDeleteRequest2);
            System.out.format("Successfully deleted all the %s items.\n", de
lObjRes.getDeletedObjects().size());
            } catch (MultiObjectDeleteException mode) {
                printDeleteResults(mode);
            }
        }
        static void printDeleteResults(MultiObjectDeleteException mode) {
            System.out.format("%s \n", mode.getMessage());
            System.out.format("No. of objects successfully deleted = %s\n",
mode.getDeletedObjects().size());
            System.out.format("No. of objects failed to delete = %s\n", mode.ge
tErrors().size());
            System.out.format("Printing error data...\n");
            for (DeleteError deleteError : mode.getErrors()){
                System.out.format("Object Key: %s\t%s\t%s\n",
                    deleteError.getKey(), deleteError.getCode(), deleteError.get
Message());
            }
        }
    }
}

```

Deleting Multiple Objects Using the AWS SDK for .NET

The following tasks guide you through using the AWS SDK for .NET classes to delete multiple objects in a single HTTP request.

Deleting Multiple Objects (Non-Versioned Bucket)

1	Create an instance of the <code>AmazonS3Client</code> class.
2	Create an instance of the <code>DeleteObjectsRequest</code> class and provide list of the object keys you want to delete.
3	Execute the <code>AmazonS3Client.DeleteObjects</code> method. If one or more objects fail to delete, Amazon S3 throws a <code>DeleteObjectsException</code> .

The following C# code sample demonstrates the preceding steps.

```
DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest();
multiObjectDeleteRequest.BucketName = bucketName;

multiObjectDeleteRequest.AddKey("<object Key>", null); // version ID is null.
multiObjectDeleteRequest.AddKey("<object Key>", null);
multiObjectDeleteRequest.AddKey("<object Key>", null);

try
{
    DeleteObjectsResponse response = client.DeleteObjects(multiObjectDelete
Request);
    Console.WriteLine("Successfully deleted all the {0} items", response.Delete
dObjects.Count);
}
catch (DeleteObjectsException e)
{
    // Process exception.
}
```

The `DeleteObjectsRequest` can also take the list of `KeyVersion` objects as parameter. For bucket without versioning, version ID is null.

```
List<KeyVersion> keys = new List<KeyVersion>();
KeyVersion keyVersion = new KeyVersion
{
    Key = key,
    VersionId = null // For buckets without versioning.
};

keys.Add(keyVersion);
List<KeyVersion> keys = new List<KeyVersion>();
...
DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest
{
    BucketName = bucketName,
    Objects = keys // This includes the object keys and null version IDs.
};
```

In the event of an exception, you can review the `DeleteObjectsException` to determine which objects failed to delete and why as shown in the following C# code example.

```
DeleteObjectsResponse errorResponse = e.Response;
Console.WriteLine("No. of objects successfully deleted = {0}", errorResponse.De
letedObjects.Count);
Console.WriteLine("No. of objects failed to delete = {0}", errorResponse.Delet
eErrors.Count);
Console.WriteLine("Printing error data...");
foreach (DeleteError deleteError in errorResponse.DeleteErrors)
{
    Console.WriteLine("Object Key: {0}\t{1}\t{2}", deleteError.Key, deleteEr
ror.Code, deleteError.Message);
}
```

The following tasks guide you through deleting objects from a version-enabled bucket.

Deleting Multiple Objects (Version-Enabled Bucket)

1	Create an instance of the <code>AmazonS3Client</code> class.
2	Create an instance of the <code>DeleteObjectsRequest</code> class and provide a list of object keys and optionally the version IDs of the objects that you want to delete. If you specify the version ID of the object you want to delete, Amazon S3 deletes the specific object version. If you don't specify the version ID of the object that you want to delete, Amazon S3 adds a delete marker. For more information, see Deleting One Object Per Request (p. 229) .
3	Execute the <code>AmazonS3Client.DeleteObjects</code> method.

The following C# code sample demonstrates the preceding steps.

```
List<KeyVersion> keysAndVersions = new List<KeyVersion>();
// provide a list of object keys and versions.

DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest
{
    BucketName = bucketName,
    Objects = keysAndVersions
};

try
{
    DeleteObjectsResponse response = client.DeleteObjects(multiObjectDelete
Request);
    Console.WriteLine("Successfully deleted all the {0} items", response.Delete
dObjects.Count);
}
catch (DeleteObjectsException e)
{
    // Process exception.
}
```

Example 1: Multi-Object Delete (Non-Versioned Bucket)

The following C# code example uses the Multi-Object API to delete objects from a bucket that is not version-enabled. The example first uploads the sample objects to the bucket and then uses the `DeleteObjects` method to delete the objects in a single request. In the `DeleteObjectsRequest`, the example specifies only the object key names because the version IDs are null.

For information about how to create and test a working sample, see [Testing the .NET Code Examples \(p. 547\)](#).

```
using System;
using System.Collections.Generic;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazonaws.com.docsamples
{
    class DeleteMultipleObjects
    {
        static string bucketName = "**** Provide a bucket name ****";
        static IAmazonS3 client;

        public static void Main(string[] args)
        {
            using (client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1))
            {
                var keysAndVersions = PutObjects(3);
                // Delete the objects.
                MultiObjectDelete(keysAndVersions);
            }

            Console.WriteLine("Click ENTER to continue.....");
            Console.ReadLine();
        }

        static void MultiObjectDelete(List<KeyVersion> keys)
        {
            // a. multi-object delete by specifying the key names and version
            // IDs.
            DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest
            {
                BucketName = bucketName,
                Objects = keys // This includes the object keys and null version
                // IDs.
            };
            multiObjectDeleteRequest.AddKey("AWS_SDK_copy2.dll", null);
            try
            {
                DeleteObjectsResponse response = client.DeleteObjects(multiObjectDeleteRequest);
                Console.WriteLine("Successfully deleted all the {0} items",
                    response.DeletedObjects.Count);
            }
            catch (DeleteObjectsException e)
            {
                PrintDeletionReport(e);
            }
        }
    }
}
```

```
        }

    private static void PrintDeletionReport(DeleteObjectsException e)
    {
        // var errorResponse = e.ErrorResponse;
        DeleteObjectsResponse errorResponse = e.Response;
        Console.WriteLine("x {0}", errorResponse.DeletedObjects.Count);

        Console.WriteLine("No. of objects successfully deleted = {0}", errorResponse.DeletedObjects.Count);
        Console.WriteLine("No. of objects failed to delete = {0}", errorResponse.DeleteErrors.Count);

        Console.WriteLine("Printing error data...");
        foreach (DeleteError deleteError in errorResponse.DeleteErrors)
        {
            Console.WriteLine("Object Key: {0}\t{1}\t{2}", deleteError.Key,
deleteError.Code, deleteError.Message);
        }
    }

    static List<KeyVersion> PutObjects(int number)
    {
        List<KeyVersion> keys = new List<KeyVersion>();
        for (int i = 0; i < number; i++)
        {
            string key = "ExampleObject-" + new System.Random().Next();
            PutObjectRequest request = new PutObjectRequest
            {
                BucketName = bucketName,
                Key = key,
                ContentBody = "This is the content body!",
            };

            PutObjectResponse response = client.PutObject(request);
            KeyVersion keyVersion = new KeyVersion
            {
                Key = key,
                // For non-versioned bucket operations, we only need object
key.
                // VersionId = response.VersionId
            };
            keys.Add(keyVersion);
        }
        return keys;
    }
}
```

Example 2: Multi-Object Delete (Version-Enabled Bucket)

The following C# code example uses the Multi-Object API to delete objects from a version-enabled bucket. In addition to showing the DeleteObjects Multi-Object Delete API usage, it also illustrates how versioning works in a version-enabled bucket.

Before you can test the sample, you must create a sample bucket and provide the bucket name in the example. You can use the AWS Management Console to create a bucket.

The example performs the following actions:

1. Enable versioning on the bucket.
2. Perform a versioned-delete.

The example first uploads the sample objects. In response, Amazon S3 returns the version IDs for each sample object that you uploaded. The example then deletes these objects using the Multi-Object Delete API. In the request, it specifies both the object keys and the version IDs (that is, versioned delete).

3. Perform a non-versioned delete.

The example uploads the new sample objects. Then, it deletes the objects using the Multi-Object API. However, in the request, it specifies only the object keys. In this case, Amazon S3 adds the delete markers and the objects disappear from your bucket.

4. Delete the delete markers.

To illustrate how the delete markers work, the sample deletes the delete markers. In the Multi-Object Delete request, it specifies the object keys and the version IDs of the delete markers it received in the response in the preceding step. This action makes the objects reappear in your bucket.

For information about how to create and test a working sample, see [Testing the .NET Code Examples \(p. 547\)](#).

```
using System;
using System.Collections.Generic;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples
{
    class DeleteMultipleObjectsVersionedBucket
    {
        static string bucketName = "**** Provide a bucket name ***";
        static IAmazonS3 client;

        public static void Main(string[] args)
        {
            using (client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1))

            {

                // 1. Enable versioning on the bucket.
                EnableVersioningOnBucket(bucketName);

                // 2a. Upload the sample objects.
                var keysAndVersions1 = PutObjects(3);
                // 2b. Delete the specific object versions.
                VersionedDelete(keysAndVersions1);
            }
        }
}
```

```
// 3a. Upload the sample objects.  
var keysAndVersions2 = PutObjects(3);  
  
// 3b. Delete objects using only keys. Amazon S3 creates a delete  
marker and  
// returns its version Id in the response.  
List<DeletedObject> deletedObjects = NonVersionedDelete(keysAnd  
Versions2);  
  
// 3c. Additional exercise - using a multi-object versioned  
delete, remove the  
// delete markers received in the preceding response. This  
results in your objects  
// reappearing in your bucket.  
RemoveMarkers(deletedObjects);  
}  
  
Console.WriteLine("Click ENTER to continue.....");  
Console.ReadLine();  
}  
  
private static void PrintDeletionReport(DeleteObjectsException e)  
{  
    var errorResponse = e.Response;  
    Console.WriteLine("No. of objects successfully deleted = {0}", er  
rorResponse.DeletedObjects.Count);  
    Console.WriteLine("No. of objects failed to delete = {0}", errorRe  
sponse.DeleteErrors.Count);  
    Console.WriteLine("Printing error data...");  
    foreach (DeleteError deleteError in errorResponse.DeleteErrors)  
    {  
        Console.WriteLine("Object Key: {0}\t{1}\t{2}", deleteError.Key,  
deleteError.Code, deleteError.Message);  
    }  
}  
  
static void EnableVersioningOnBucket(string bucketName)  
{  
    PutBucketVersioningRequest setBucketVersioningRequest = new PutBuck  
etVersioningRequest  
    {  
        BucketName = bucketName,  
        VersioningConfig = new S3BucketVersioningConfig { Status =  
VersionStatus.Enabled }  
    };  
    client.PutBucketVersioning(setBucketVersioningRequest);  
}  
  
static void VersionedDelete(List<KeyVersion> keys)  
{  
    // a. Perform a multi-object delete by specifying the key names and  
version IDs.  
    DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjects  
Request  
    {  
        BucketName = bucketName,  
        Objects = keys // This includes the object keys and specific
```

```

version IDs.
    };
    try
    {
        Console.WriteLine("Executing VersionedDelete...");
        DeleteObjectsResponse response = client.DeleteObjects(multiObjectDeleteRequest);
        Console.WriteLine("Successfully deleted all the {0} items",
response.DeletedObjects.Count);
    }
    catch (DeleteObjectsException e)
    {
        PrintDeletionReport(e);
    }
}

static List<DeletedObject> NonVersionedDelete(List<KeyVersion> keys)
{
    // Create a request that includes only the object key names.
    DeleteObjectsRequest multiObjectDeleteRequest = new DeleteObjectsRequest();
    multiObjectDeleteRequest.BucketName = bucketName;

    foreach (var key in keys)
    {
        multiObjectDeleteRequest.AddKey(key.Key);
    }
    // Execute DeleteObjects - Amazon S3 add delete marker for each
object
    // deletion. The objects disappear from your bucket.
    // You can verify that using the Amazon S3 console.
    DeleteObjectsResponse response;
    try
    {
        Console.WriteLine("Executing NonVersionedDelete...");
        response = client.DeleteObjects(multiObjectDeleteRequest);
        Console.WriteLine("Successfully deleted all the {0} items",
response.DeletedObjects.Count);
    }
    catch (DeleteObjectsException e)
    {
        PrintDeletionReport(e);
        throw; // Some deletes failed. Investigate before continuing.
    }
    // This response contains the DeletedObjects list which we use to
delete the delete markers.
    return response.DeletedObjects;
}

private static void RemoveMarkers(List<DeletedObject> deletedObjects)
{
    List<KeyVersion> keyVersionList = new List<KeyVersion>();

    foreach (var deletedObject in deletedObjects)
    {
        KeyVersion keyVersion = new KeyVersion
        {
            Key = deletedObject.Key,

```

```
        VersionId = deletedObject.DeleteMarkerVersionId
    };
    keyVersionList.Add(keyVersion);
}
// Create another request to delete the delete markers.
var multiObjectDeleteRequest = new DeleteObjectsRequest
{
    BucketName = bucketName,
    Objects = keyVersionList
};

// Now, delete the delete marker to bring your objects back to the
bucket.
try
{
    Console.WriteLine("Removing the delete markers .....");
    var deleteObjectResponse = client.DeleteObjects(multiObjectDe
leteRequest);
    Console.WriteLine("Successfully deleted all the {0} delete
markers",
                      deleteObjectResponse.DeletedOb
jects.Count);
}
catch (DeleteObjectsException e)
{
    PrintDeletionReport(e);
}
}

static List<KeyVersion> PutObjects(int number)
{
    List<KeyVersion> keys = new List<KeyVersion>();

    for (int i = 0; i < number; i++)
    {
        string key = "ObjectToDelete-" + new System.Random().Next();
        PutObjectRequest request = new PutObjectRequest
        {
            BucketName = bucketName,
            Key = key,
            ContentBody = "This is the content body!",
        };

        PutObjectResponse response = client.PutObject(request);
        KeyVersion keyVersion = new KeyVersion
        {
            Key = key,
            VersionId = response.VersionId
        };

        keys.Add(keyVersion);
    }
    return keys;
}
}
```

Deleting Multiple Objects Using the AWS SDK for PHP

This topic guides you through using classes from the AWS SDK for PHP to delete multiple objects from versioned and non-versioned Amazon S3 buckets. For more information about versioning, see [Using Versioning \(p. 422\)](#).

Note

This topic assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 547\)](#) and have the AWS SDK for PHP properly installed.

The following tasks guide you through using the PHP SDK classes to delete multiple objects from a non-versioned bucket.

Deleting Multiple Objects (Non-Versioned Bucket)

1	Create an instance of an Amazon S3 client by using the Aws\S3\S3Client class factory() method.
2	Execute the Aws\S3\S3Client::deleteObjects() method. You need to provide a bucket name and an array of object keys as parameters. You can specify up to 1000 keys.

The following PHP code sample demonstrates deleting multiple objects from an Amazon S3 non-versioned bucket.

```
use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname1 = '*** Your Object Key1 ***';
$keyname2 = '*** Your Object Key2 ***';
$keyname3 = '*** Your Object Key3 ***';

$s3 = S3Client::factory();

// Delete objects from a bucket
$result = $s3->deleteObjects(array(
    'Bucket' => $bucket,
    'Objects' => array(
        array('Key' => $keyname1),
        array('Key' => $keyname2),
        array('Key' => $keyname3),
    )
));
```

The following tasks guide you through deleting multiple objects from an Amazon S3 version-enabled bucket.

Deleting Multiple Objects (Version-Enabled Bucket)

1	Create an instance of an Amazon S3 client by using the Aws\S3\S3Client class factory() method.
---	--

2 Execute the `Aws\S3\S3Client::deleteObjects()` method and provide a list of objects keys and optionally the version IDs of the objects that you want to delete.

If you specify version ID of the object that you want to delete, Amazon S3 deletes the specific object version. If you don't specify the version ID of the object that you want to delete, Amazon S3 adds a delete marker. For more information, see [Deleting One Object Per Request \(p. 229\)](#).

The following PHP code sample demonstrates deleting multiple objects from an Amazon S3 version-enabled bucket.

```
use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';
$versionId1 = '*** Your Object Key Version ID1 ***';
$versionId2 = '*** Your Object Key Version ID2 ***';
$versionId3 = '*** Your Object Key Version ID3 ***';

$s3 = S3Client::factory();

// Delete object versions from a versioning-enabled bucket.
$result = $s3->deleteObjects(array(
    'Bucket' => $bucket,
    'Objects' => array(
        array('Key' => $keyname, 'VersionId' => $versionId1),
        array('Key' => $keyname, 'VersionId' => $versionId2),
        array('Key' => $keyname, 'VersionId' => $versionId3),
    )
));
```

Amazon S3 returns a response that shows the objects that were deleted and objects it could not delete because of errors (for example, permission errors).

The following PHP code sample prints the object keys for objects that were deleted. It also prints the object keys that were not deleted and the related error messages.

```
echo "The following objects were deleted successfully:\n";
foreach ($result['Deleted'] as $object) {
    echo "Key: {$object['Key']}, VersionId: {$object['VersionId']}\n";
}

echo "\nThe following objects could not be deleted:\n";
foreach ($result['Errors'] as $object) {
    echo "Key: {$object['Key']}, VersionId: {$object['VersionId']}\n";
}
```

Example 1: Multi-Object Delete (Non-Versioned Bucket)

The following PHP code example uses the `deleteObjects()` method to delete multiple objects from a bucket that is not version-enabled.

The example performs the following actions:

1. Creates a few objects by using the `Aws\S3\S3Client::putObject()` method.
2. Lists the objects and gets the keys of the created objects using the `Aws\S3\S3Client::listObjects()` method.
3. Performs a non-versioned delete by using the `Aws\S3\S3Client::deleteObjects()` method.

For information about running the PHP examples in this guide, go to [Running PHP Examples \(p. 548\)](#).

```
<?php

// Include the AWS SDK using the Composer autoloader.
require 'vendor/autoload.php';

use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';

// Instantiate the client.
$s3 = S3Client::factory();

// 1. Create a few objects.
for ($i = 1; $i <= 3; $i++) {
    $s3->putObject(array(
        'Bucket' => $bucket,
        'Key'     => "key{$i}",
        'Body'    => "content {$i}",
    ));
}

// 2. List the objects and get the keys.
$keys = $s3->listObjects(array('Bucket' => $bucket))
    ->getPath('Contents/*/Key');

// 3. Delete the objects.
$result = $s3->deleteObjects(array(
    'Bucket'  => $bucket,
    'Objects' => array_map(function ($key) {
        return array('Key' => $key);
    }, $keys),
));
}
```

Example 2: Multi-Object Delete (Version-Enabled Bucket)

The following PHP code example uses the `deleteObjects()` method to delete multiple objects from a version-enabled bucket.

The example performs the following actions:

1. Enables versioning on the bucket by using the `Aws\S3\S3Client::putBucketVersioning()` method.
2. Creates a few versions of an object by using the `Aws\S3\S3Client::putObject()` method.
3. Lists the objects versions and gets the keys and version IDs for the created object versions using the `Aws\S3\S3Client::listObjectVersions()` method.
4. Performs a versioned-delete by using the `Aws\S3\S3Client::deleteObjects()` method with the retrieved keys and versions IDs.
5. Disables versioning on the bucket by using the `Aws\S3\S3Client::putBucketVersioning()` method.

For information about running the PHP examples in this guide, go to [Running PHP Examples \(p. 548\)](#).

```
<?php

// Include the AWS SDK using the Composer autoloader.
require 'vendor/autoload.php';

use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

// Instantiate the client.
$s3 = S3Client::factory();

// 1. Enable object versioning for the bucket.
$s3->putBucketVersioning(array(
    'Bucket' => $bucket,
    'Status' => 'Enabled',
));

// 2. Create a few versions of an object.
for ($i = 1; $i <= 3; $i++) {
    $s3->putObject(array(
        'Bucket' => $bucket,
        'Key'      => $keyname,
        'Body'     => "content {$i}",
    ));
}

// 3. List the objects versions and get the keys and version IDs.
$versions = $s3->listObjectVersions(array('Bucket' => $bucket))
    ->getPath('Versions');

// 4. Delete the object versions.
$result = $s3->deleteObjects(array(
    'Bucket'  => $bucket,
    'Objects' => array_map(function ($version) {
        return array(
            'Key'       => $version['Key'],
        );
    }),
));
```

```
        'VersionId' => $version['VersionId']
    );
}, $versions),
));

echo "The following objects were deleted successfully:\n";
foreach ($result['Deleted'] as $object) {
    echo "Key: {$object['Key']}, VersionId: {$object['VersionId']}\\n";
}

echo "\\nThe following objects could not be deleted:\\n";
foreach ($result['Errors'] as $object) {
    echo "Key: {$object['Key']}, VersionId: {$object['VersionId']}\\n";
}

// 5. Suspend object versioning for the bucket.
$s3->putBucketVersioning(array(
    'Bucket' => $bucket,
    'Status' => 'Suspended',
));

```

Related Resources

- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::factory\(\) Method](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::deleteObject\(\) Method](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::listObjects\(\) Method](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::listObjectVersions\(\) Method](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::putObject\(\) Method](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::putBucketVersioning\(\) Method](#)
- [AWS SDK for PHP for Amazon S3](#)
- [AWS SDK for PHP Documentation](#)

Deleting Multiple Objects Using the REST API

You can use the AWS SDKs to delete multiple objects using the Multi-Object Delete API. However, if your application requires it, you can send REST requests directly. For more information, go to [Delete Multiple Objects](#) in the *Amazon Simple Storage Service API Reference*.

Restoring Objects

Topics

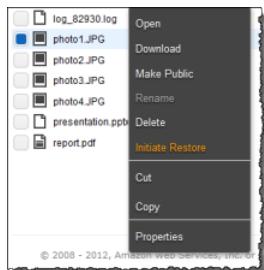
- [Restore an Object Using the Amazon S3 Console \(p. 262\)](#)
- [Restore an Object Using the AWS SDK for Java \(p. 263\)](#)
- [Restore an Object Using the AWS SDK for .NET \(p. 265\)](#)
- [Restore an Object Using REST API \(p. 268\)](#)

You must initiate a restore request before you can access an archived object. This section describes how to initiate a restore request programmatically. For more information about archiving objects, see [Object Lifecycle Management \(p. 86\)](#).

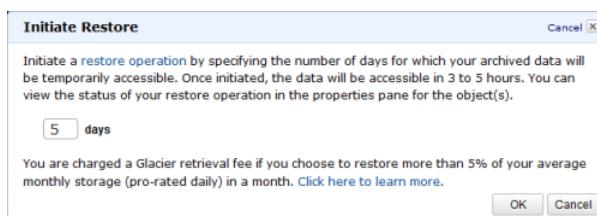
It takes about four hours for a restore operation to make a copy of the archive available for you to access. After initiating a restore, you can check status of the restoration in the console (see [Restoring GLACIER Objects by Using Amazon S3 Console \(p. 98\)](#)) or send HEAD request to programmatically retrieve object's metadata which includes the restoration status information.

Restore an Object Using the Amazon S3 Console

You can use Amazon S3 console to restore a copy of an archived object. In the console you right-mouse click on the object and select **Initiate Restore**.



You specify the number of days you want the object copy restored.



It takes about three to five hours for Amazon S3 to complete the restoration. The object properties in the console shows the object restoration status.



When object copy is restored, the object properties in the console shows the object is restored and when Amazon S3 will remove the restored copy. The console also gives you option to modify the restoration period.



Note that when you restore an archive you are paying for both the archive and a copy you restored temporarily. For information about pricing, go to the [Pricing](#) section of the *Amazon S3 product detail page*.

Amazon S3 restores a temporary copy of the object only for the specified duration. After that Amazon S3 deletes the restored object copy. You can modify the expiration period of a restored copy, by reissuing a restore, in which case Amazon S3 updates the expiration period, relative to the current time.

Amazon S3 calculates expiration time of the restored object copy by adding the number of days specified in the restoration request to the current time and rounding the resulting time to the next day midnight UTC. For example, if an object was created on 10/15/2012 10:30 am UTC and the restoration period was specified as 3 days, then the restored copy expires on 10/19/2012 00:00 UTC at which time Amazon S3 delete the object copy.

You can restore an object copy for any number of days. However you should restore objects only for the duration you need because of the storage costs associated with the object copy. For pricing information, go to the [Pricing](#) section of the Amazon S3 product detail page.

Restore an Object Using the AWS SDK for Java

The following tasks guide you through use the AWS SDK for Java to initiate a restoration of an archived object.

Downloading Objects

1	Create an instance of the <code>AmazonS3Client</code> class.
2	Create an instance of <code>RestoreObjectRequest</code> class by providing bucket name, object key to restore and the number of days for which you the object copy restored.
3	Execute one of the <code>AmazonS3.RestoreObject</code> methods to initiate the archive restoration.

The following Java code sample demonstrates the preceding tasks.

```
String bucketName = "examplebucket";
String objectkey = "examplekey";
AmazonS3Client s3Client = new AmazonS3Client();

RestoreObjectRequest request = new RestoreObjectRequest(bucketName, objectkey,
```

```
    2);
s3Client.restoreObject(request);
```

Amazon S3 maintains the restoration status in the object metadata. You can retrieve object metadata and check the value of the `RestoreInProgress` property as shown in the following Java code snippet.

```
String bucketName = "examplebucket";
String objectkey = "examplekey";
AmazonS3Client s3Client = new AmazonS3Client();

client = new AmazonS3Client();

GetObjectMetadataRequest request = new GetObjectMetadataRequest(bucketName,
objectKey);

ObjectMetadata response = s3Client.getObjectMetadata(request);

Boolean restoreFlag = response.getOngoingRestore();
System.out.format("Restoration status: %s.\n",
(restoreFlag == true) ? "in progress" : "finished");
```

Example

The following Java code example initiates a restoration request for the specified archived object. You must update the code and provide a bucket name and an archived object key name. For instructions on how to create and test a working sample, see [Testing the Java Code Examples \(p. 545\)](#).

```
import java.io.IOException;

import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.AmazonS3Exception;
import com.amazonaws.services.s3.model.GetObjectMetadataRequest;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.RestoreObjectRequest;

public class RestoreArchivedObject {

    public static String bucketName = "**** Provide bucket name ***";
    public static String objectKey = "**** Provide object key name ***";
    public static AmazonS3Client s3Client;

    public static void main(String[] args) throws IOException {
        AmazonS3Client s3Client = new AmazonS3Client(new ProfileCredentialsProvider());

        try {

            RestoreObjectRequest requestRestore = new RestoreObjectRequest(bucketName, objectKey, 2);
            s3Client.restoreObject(requestRestore);

            GetObjectMetadataRequest requestCheck = new GetObjectMetadataRequest(bucketName, objectKey);
            ObjectMetadata response = s3Client.getObjectMetadata(requestCheck);

            Boolean restoreFlag = response.getOngoingRestore();
            System.out.format("Restoration status: %s.\n",
                (restoreFlag == true) ? "in progress" : "finished");

        } catch (AmazonS3Exception amazonS3Exception) {
            System.out.format("An Amazon S3 error occurred. Exception: %s",
                amazonS3Exception.toString());
        } catch (Exception ex) {
            System.out.format("Exception: %s", ex.toString());
        }
    }
}
```

Restore an Object Using the AWS SDK for .NET

The following tasks guide you through using the AWS SDK for .NET to initiate a restoration of an archived object.

Downloading Objects

1	Create an instance of the <code>AmazonS3</code> class.
---	--

2	Create an instance of <code>RestoreObjectRequest</code> class by providing bucket name, object key to restore and the number of days for which you want the object copy restored.
3	Execute one of the <code>AmazonS3.RestoreObject</code> methods to initiate the archive restoration.

The following C# code sample demonstrates the preceding tasks.

```
IAmazonS3 client;
string bucketName = "examplebucket";
string objectKey = "examplekey";

client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1);

RestoreObjectRequest restoreRequest = new RestoreObjectRequest()
{
    BucketName = bucketName,
    Key = objectKey,
    Days = 2
};

client.RestoreObject(restoreRequest);
```

Amazon S3 maintains the restoration status in the object metadata. You can retrieve object metadata and check the value of the `RestoreInProgress` property as shown in the following C# code snippet.

```
IAmazonS3 client;
string bucketName = "examplebucket";
string objectKey = "examplekey";

client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1);

GetObjectMetadataRequest metadataRequest = new GetObjectMetadataRequest()
{
    BucketName = bucketName,
    Key = objectKey
};
GetObjectMetadataResponse response = client.GetObjectMetadata(metadataRequest);
Console.WriteLine("Restoration status: {0}", response.RestoreInProgress);
if (response.RestoreInProgress == false)
    Console.WriteLine("Restored object copy expires on: {0}", response.RestoreExpiration);
```

Example

The following C# code example initiates a restoration request for the specified archived object. You must update the code and provide a bucket name and an archived object key name. For instructions on how to create and test a working sample, see [Testing the .NET Code Examples \(p. 547\)](#).

```
using System;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples
{
    class RestoreArchivedObject
    {
        static string bucketName = "**** provide bucket name ****";
        static string objectKey = "**** archived object keyname ****";

        static IAmazonS3 client;

        public static void Main(string[] args)
        {
            try
            {
                using (client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1))
                {
                    RestoreObject(client, bucketName, objectKey);
                    CheckRestorationStatus(client, bucketName, objectKey);
                }

                Console.WriteLine("Example complete. To continue, click Enter... ");
                Console.ReadKey();
            }
            catch (AmazonS3Exception amazonS3Exception)
            {
                Console.WriteLine("S3 error occurred. Exception: " +
amazonS3Exception.ToString());
            }
            catch (Exception e)
            {
                Console.WriteLine("Exception: " + e.ToString());
            }
        }

        static void RestoreObject(IAmazonS3 client, string bucketName, string
objectKey)
        {
            RestoreObjectRequest restoreRequest = new RestoreObjectRequest
            {
                BucketName = bucketName,
                Key = objectKey,
                Days = 2
            };
            RestoreObjectResponse response = client.RestoreObject(restore
Request);
        }
    }
}
```

```
    static void CheckRestorationStatus(IAmazonS3 client, string bucketName,
string objectKey)
    {
        GetObjectMetadataRequest metadataRequest = new GetObjectMetadataRe
quest
        {
            BucketName = bucketName,
            Key = objectKey
        };
        GetObjectMetadataResponse response = client.GetObject
Metadata(metadataRequest);
        Console.WriteLine("Restoration status: {0}", response.RestoreInPro
gress);
        if (response.RestoreInProgress == false)
            Console.WriteLine("Restored object copy expires on: {0}", re
sponse.RestoreExpiration);
    }
}
```

Restore an Object Using REST API

Amazon S3 provides an API for you to initiate an archive restoration. For more information, go to [POST Object restore](#) in the *Amazon Simple Storage Service API Reference*.

Managing Access Permissions to Your Amazon S3 Resources

By default, all Amazon S3 resources—buckets, objects, and related subresources (for example, lifecycle configuration and website configuration)—are private: only the resource owner, an AWS account that created it, can access the resource. The resource owner can optionally grant access permissions to others by writing an access policy.

Amazon S3 offers access policy options broadly categorized as resource-based policies and user policies. Access policies you attach to your resources (buckets and objects) are referred to as resource-based policies. For example, bucket policies and access control lists (ACLs) are resource-based policies. You can also attach access policies to users in your account. These are called user policies. You may choose to use resource-based policies, user policies, or some combination of these to manage permissions to your Amazon S3 resources. The introductory topics provide general guidelines for managing permissions.

We recommend you first review the access control overview topics. For more information, see [Introduction to Managing Access Permissions to Your Amazon S3 Resources \(p. 269\)](#). Then for more information about specific access policy options, see the following topics:

- [Using Bucket Policies and User Policies \(p. 312\)](#)
- [Managing Access with ACLs \(p. 363\)](#)

Introduction to Managing Access Permissions to Your Amazon S3 Resources

Topics

- [Overview of Managing Access \(p. 270\)](#)
- [How Amazon S3 Authorizes a Request \(p. 275\)](#)
- [Guidelines for Using the Available Access Policy Options \(p. 280\)](#)
- [Example Walkthroughs: Managing Access to Your Amazon S3 Resources \(p. 283\)](#)

The topics in this section provide an overview of managing access permissions to your Amazon S3 resources and provides guidelines for when to use which access control method. The topic also provides introductory example walkthroughs. We recommend you review these topics in order.

Overview of Managing Access

Topics

- [Amazon S3 Resources \(p. 270\)](#)
- [Resource Operations \(p. 271\)](#)
- [Managing Access to Resources \(Access Policy Options\) \(p. 271\)](#)
- [So Which Access Control Method Should I Use? \(p. 273\)](#)
- [Related Topics \(p. 274\)](#)

When granting permissions, you decide who is getting them, which Amazon S3 resources they are getting permissions for, and specific actions you want to allow on those resources.

Amazon S3 Resources

Buckets and objects are primary Amazon S3 resources, and both have associated subresources. For example, bucket subresources include the following:

- `lifecycle` – Stores lifecycle configuration information (see [Object Lifecycle Management \(p. 86\)](#)).
- `website` – Stores website configuration information if you configure your bucket for website hosting (see [Hosting a Static Website on Amazon S3 \(p. 448\)](#)).
- `versioning` – Stores versioning configuration (see [PUT Bucket versioning](#)).
- `policy` and `acl` (Access Control List) – Store access permission information for the bucket.
- `cors` (Cross-Origin Resource Sharing) – Supports configuring your bucket to allow cross-origin requests (see [Enabling Cross-Origin Resource Sharing \(p. 110\)](#)).
- `logging` – Enables you to request Amazon S3 to save bucket access logs.

Object subresources include the following:

- `acl` – Stores a list of access permissions on the object. This topic discusses how to use this subresource to manage object permissions (see [Managing Access with ACLs \(p. 363\)](#)).
- `restore` – Supports temporarily restoring an archived object (see [POST Object restore](#)). An object in the Glacier storage class is an archived object. To access the object, you must first initiate a restore request, which restores a copy of the archived object. In the request, you specify the number of days that you want the restored copy to exist. For more information about archiving objects, see [Object Lifecycle Management \(p. 86\)](#).

About the Resource Owner

By default, all Amazon S3 resources are private. Only a resource owner can access the resource. The resource owner refers to the AWS account that creates the resource. For example:

- The AWS account that you use to create buckets and objects owns those resources.
- If you create an AWS Identity and Access Management (IAM) user in your AWS account, your AWS account is the parent owner. If the IAM user uploads an object, the parent account, to which the user belongs, owns the object.
- A bucket owner can grant cross-account permissions to another AWS account (or users in another account) to upload objects. In this case, the AWS account that uploads objects owns those objects. The bucket owner does not have permissions on the objects that other accounts own, with the following exceptions:
 - The bucket owner pays the bills. The bucket owner can deny access to any objects, or delete any objects in the bucket, regardless of who owns them.

- The bucket owner can archive any objects or restore archived objects regardless of who owns them. Archival refers to the storage class used to store the objects. For more information, see [Object Lifecycle Management \(p. 86\)](#).

Important

AWS recommends not using the root credentials of your AWS account to make requests. Instead, create an IAM user, and grant that user full access. We refer to these users as administrator users. You can use the administrator user credentials, instead of root credentials of your account, to interact with AWS and perform tasks, such as create a bucket, create users, and grant them permissions. For more information, go to [Root Account Credentials vs. IAM User Credentials](#) in the *AWS General Reference* and [IAM Best Practices](#) in the *Using IAM*.

The following diagram shows an AWS account owning resources, the IAM users, buckets, and objects.



Resource Operations

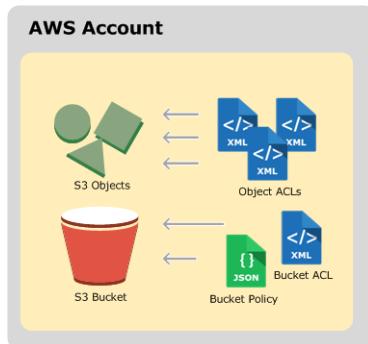
Amazon S3 provides a set of operations to work with the Amazon S3 resources. For a list of available operations, go to [Operations on Buckets](#) and [Operations on Objects](#) in the *Amazon Simple Storage Service API Reference*.

Managing Access to Resources (Access Policy Options)

Managing access refers to granting others (AWS accounts and users) permission to perform the resource operations by writing an access policy. For example, you can grant `PUT Object` permission to a user in an AWS account so the user can upload objects to your bucket. In addition to granting permissions to individual users and accounts, you can grant permissions to everyone (also referred as anonymous access) or to all authenticated users (users with AWS credentials). For example, if you configure your bucket as a website, you may want to make objects public by granting the `GET Object` permission to everyone.

Access policy describes who has access to what. You can associate an access policy with a resource (bucket and object) or a user. Accordingly, you can categorize the available Amazon S3 access policies as follows:

- **Resource-based policies** – Bucket policies and access control lists (ACLs) are resource-based because you attach them to your Amazon S3 resources.



- **ACL** – Each bucket and object has an ACL associated with it. An ACL is a list of grants identifying grantee and permission granted. You use ACLs to grant basic read/write permissions to other AWS accounts. ACLs use an Amazon S3-specific XML schema.

The following is an example bucket ACL. The grant in the ACL shows a bucket owner as having full control permission.

```
<?xml version="1.0" encoding="UTF-8"?>
<AccessControlPolicy xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Owner>
    <ID>*** Owner-Canonical-User-ID ***</ID>
    <DisplayName>owner-display-name</DisplayName>
  </Owner>
  <AccessControlList>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:type="Canonical User">
        <ID>*** Owner-Canonical-User-ID ***</ID>
        <DisplayName>display-name</DisplayName>
      </Grantee>
      <Permission>FULL_CONTROL</Permission>
    </Grant>
  </AccessControlList>
</AccessControlPolicy>
```

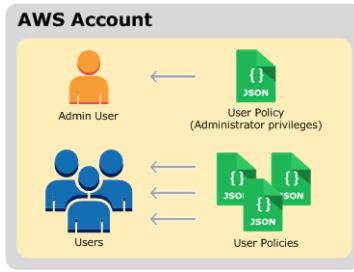
Both bucket and object ACLs use the same XML schema.

- **Bucket Policy** – For your bucket, you can add a bucket policy to grant other AWS accounts or IAM users permissions for the bucket and the objects in it. Any object permissions apply only to the objects that the bucket owner creates. Bucket policies supplement, and in many cases, replace ACL-based access policies.

The following is an example bucket policy. You express bucket policy (and user policy) using a JSON file. The policy grants anonymous read permission on all objects in a bucket. The bucket policy has one statement, which allows the `s3:GetObject` action (read permission) on objects in a bucket named `examplebucket`. By specifying the `principal` with a wild card (*), the policy grants anonymous access.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "*",
      "Action": [ "s3:GetObject" ],
      "Resource": [ "arn:aws:s3:::examplebucket/*" ]
    }
  ]
}
```

- **User policies** – You can use AWS Identity and Access Management (IAM) to manage access to your Amazon S3 resources. Using IAM, you can create IAM users, groups, and roles in your account and attach access policies to them granting them access to AWS resources including Amazon S3.



For more information about IAM, go to [AWS Identity and Access Management \(IAM\)](#) product detail page.

The following is an example of a user policy. You cannot grant anonymous permissions in an IAM user policy, because the policy is attached to a user. The example policy allows the associated user that it's attached to perform six different Amazon S3 actions on a bucket and the objects in it. You can attach this policy to a specific IAM user, group, or role.

```
{  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:PutObject",  
                "s3:GetObject",  
                "s3:DeleteObject",  
                "s3>ListAllMyBuckets",  
                "s3:GetBucketLocation",  
                "s3>ListBucket"  
            ],  
            "Resource": "arn:aws:s3:::examplebucket/*"  
        }  
    ]  
}
```

When Amazon S3 receives a request, it must evaluate all the access policies to determine whether to authorize or deny the request. For more information about how Amazon S3 evaluates these policies, see [How Amazon S3 Authorizes a Request \(p. 275\)](#).

So Which Access Control Method Should I Use?

With the options available to write an access policy, the following questions arise:

- When should I use which access control method? For example, to grant bucket permissions, should I use bucket policy or bucket ACL? I own a bucket and the objects in the bucket. Should I use a resource-based access policy or an IAM user policy? If I use a resource-based access policy, should I use a bucket policy or an object ACL to manage object permissions?
- I own a bucket, but I don't own all of the objects in it. How are access permissions managed for the objects that somebody else owns?
- If I grant access by using a combination of these access policy options, how does Amazon S3 determine if a user has permission to perform a requested operation?

The following sections explains these access control alternatives, how Amazon S3 evaluates access control mechanisms, when to use which access control method, and also provide example walkthroughs.

[How Amazon S3 Authorizes a Request \(p. 275\)](#)

[Guidelines for Using the Available Access Policy Options \(p. 280\)](#)

[Example Walkthroughs: Managing Access to Your Amazon S3 Resources \(p. 283\)](#)

Related Topics

We recommend that you first review the introductory topics that explain the options available for you to manage access to your Amazon S3 resources. For more information, see [Introduction to Managing Access Permissions to Your Amazon S3 Resources \(p. 269\)](#). You can then use the following topics for more information about specific access policy options.

- [Using Bucket Policies and User Policies \(p. 312\)](#)
- [Managing Access with ACLs \(p. 363\)](#)

How Amazon S3 Authorizes a Request

Topics

- [Related Topics \(p. 276\)](#)
- [How Amazon S3 Authorizes a Request for a Bucket Operation \(p. 276\)](#)
- [How Amazon S3 Authorizes a Request for an Object Operation \(p. 278\)](#)

When Amazon S3 receives a request—for example, a bucket or an object operation—it first verifies that the requester has the necessary permissions. Amazon S3 evaluates all the relevant access policies, user policies, and resource-based policies (bucket policy, bucket ACL, object ACL) in deciding whether to authorize the request. The following are some of the example scenarios:

- If the requester is an IAM user, Amazon S3 must determine if the parent AWS account to which the user belongs has granted the user necessary permission to perform the operation. In addition, if the request is for a bucket operation, such as a request to list the bucket content, Amazon S3 must verify that the bucket owner has granted permission for the requester to perform the operation.

Note

To perform a specific operation on a resource, an IAM user needs permission from both the parent AWS account to which it belongs and the AWS account that owns the resource.

- If the request is for an operation on an object that the bucket owner does not own, in addition to making sure the requester has permissions from the object owner, Amazon S3 must also check the bucket policy to ensure the bucket owner has not set explicit deny on the object.

Note

A bucket owner (who pays the bill) can explicitly deny access to objects in the bucket regardless of who owns it. The bucket owner can also delete any object in the bucket.

In order to determine whether the requester has permission to perform the specific operation, Amazon S3 does the following, in order, when it receives a request:

1. Converts all the relevant access policies (user policy, bucket policy, ACLs) at run time into a set of policies for evaluation.
2. Evaluates the resulting set of policies in the following steps. In each step, Amazon S3 evaluates a subset of policies in a specific context, based on the context authority.
 - a. **User context** – In the user context, the parent account to which the user belongs is the context authority.

Amazon S3 evaluates a subset of policies owned by the parent account. This subset includes the user policy that the parent attaches to the user. If the parent also owns the resource in the request (bucket, object), Amazon S3 also evaluates the corresponding resource policies (bucket policy, bucket ACL, and object ACL) at the same time.

A user must have permission from the parent account to perform the operation.

This step applies only if the request is made by a user in an AWS account. If the request is made using root credentials of an AWS account, Amazon S3 skips this step.

- b. **Bucket context** – In the bucket context, Amazon S3 evaluates policies owned by the AWS account that owns the bucket.

If the request is for a bucket operation, the requester must have permission from the bucket owner. If the request is for an object, Amazon S3 evaluates all the policies owned by the bucket owner to check if the bucket owner has not explicitly denied access to the object. If there is an explicit deny set, Amazon S3 does not authorize the request.

- c. **Object context** – If the request is for an object, Amazon S3 evaluates the subset of policies owned by the object owner.

The following sections describe in detail and provide examples:

- [How Amazon S3 Authorizes a Request for a Bucket Operation \(p. 276\)](#)
- [How Amazon S3 Authorizes a Request for an Object Operation \(p. 278\)](#)

Related Topics

We recommend you first review the introductory topics that explain the options for managing access to your Amazon S3 resources. For more information, see [Introduction to Managing Access Permissions to Your Amazon S3 Resources \(p. 269\)](#). You can then use the following topics for more information about specific access policy options.

- [Using Bucket Policies and User Policies \(p. 312\)](#)
- [Managing Access with ACLs \(p. 363\)](#)

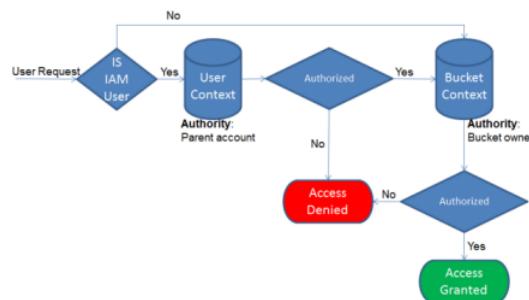
How Amazon S3 Authorizes a Request for a Bucket Operation

When Amazon S3 receives a request for a bucket operation, Amazon S3 converts all the relevant permissions—resource-based permissions (bucket policy, bucket access control list (ACL)) and IAM user policy if the request is from a user—into a set of policies to evaluate at run time. It then evaluates the resulting set of policies in a series of steps according to a specific context—user context or bucket context.

1. **User context** – If the requester is an IAM user, the user must have permission from the parent AWS account to which it belongs. In this step, Amazon S3 evaluates a subset of policies owned by the parent account (also referred to as the context authority). This subset of policies includes the user policy that the parent account attaches to the user. If the parent also owns the resource in the request (in this case, the bucket), Amazon S3 also evaluates the corresponding resource policies (bucket policy and bucket ACL) at the same time.
2. **Bucket context** – The requester must have permissions from the bucket owner to perform a specific bucket operation. In this step, Amazon S3 evaluates a subset of policies owned by the AWS account that owns the bucket.

The bucket owner can grant permission by using a bucket policy or bucket ACL. Note that, if the AWS account that owns the bucket is also the parent account of an IAM user, then it can configure bucket permissions in a user policy.

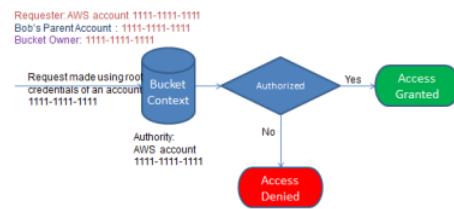
The following is a graphical illustration of the context-based evaluation for bucket operation.



The following examples illustrate the evaluation logic.

Example 1: Bucket Operation Requested by Bucket Owner

In this example, the bucket owner sends a request for a bucket operation using the root credentials of the AWS account.

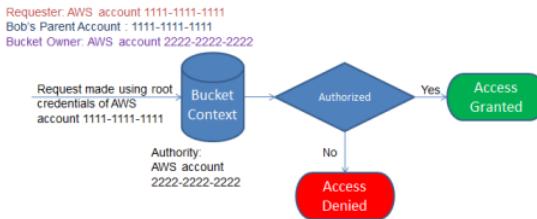


Amazon S3 performs the context evaluation as follows:

1. Because the request is made by using root credentials of an AWS account, the user context is not evaluated.
2. In the bucket context, Amazon S3 reviews the bucket policy to determine if the requester has permission to perform the operation. Amazon S3 authorizes the request.

Example 2: Bucket Operation Requested by an AWS Account That Is Not the Bucket Owner

In this example, a request is made using root credentials of AWS account 1111-1111-1111 for a bucket operation owned by AWS account 2222-2222-2222. No IAM users are involved in this request.

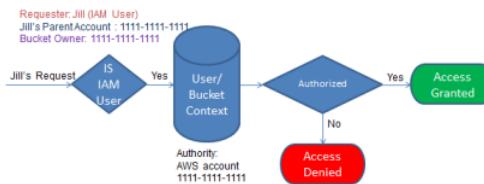


In this case, Amazon S3 evaluates the context as follows:

1. Because the request is made using root credentials of an AWS account, the user context is not evaluated.
2. In the bucket context, Amazon S3 examines the bucket policy. If the bucket owner (AWS account 2222-2222-2222) has not authorized AWS account 1111-1111-1111 to perform the requested operation, Amazon S3 denies the request. Otherwise, Amazon S3 grants the request and performs the operation.

Example 3: Bucket Operation Requested by an IAM User Whose Parent AWS Account Is Also the Bucket Owner

In the example, the request is sent by Jill, an IAM user in AWS account 1111-1111-1111, which also owns the bucket.



Amazon S3 performs the following context evaluation:

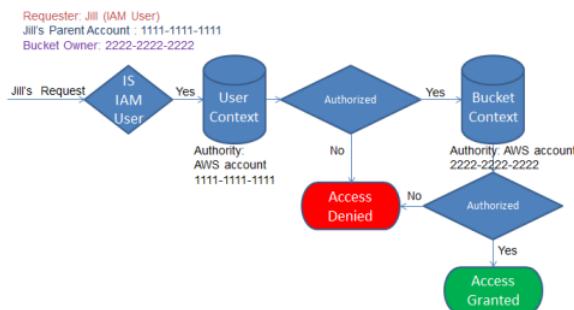
1. Because the request is from an IAM user, in the user context, Amazon S3 evaluates all policies that belong to the parent AWS account to determine if Jill has permission to perform the operation.

In this example, parent AWS account 1111-1111-1111, to which the user belongs, is also the bucket owner. As a result, in addition to the user policy, Amazon S3 also evaluates the bucket policy and bucket ACL in the same context, because they belong to the same account.

2. Because Amazon S3 evaluated the bucket policy and bucket ACL as part of the user context, it does not evaluate the bucket context.

Example 4: Bucket Operation Requested by an IAM User Whose Parent AWS Account Is Not the Bucket Owner

In this example, the request is sent by Jill, an IAM user whose parent AWS account is 1111-1111-1111, but the bucket is owned by another AWS account, 2222-2222-2222.



Jill will need permissions from both the parent AWS account and the bucket owner. Amazon S3 evaluates the context as follows:

1. Because the request is from an IAM user, Amazon S3 evaluates the user context by reviewing the policies authored by the account to verify that Jill has the necessary permissions. If Jill has permission, then Amazon S3 moves on to evaluate the bucket context; if not, it denies the request.
2. In the bucket context, Amazon S3 verifies that bucket owner 2222-2222-2222 has granted Jill (or her parent AWS account) permission to perform the requested operation. If she has that permission, Amazon S3 grants the request and performs the operation; otherwise, Amazon S3 denies the request.

How Amazon S3 Authorizes a Request for an Object Operation

When Amazon S3 receives a request for an object operation, it converts all the relevant permissions—resource-based permissions (object access control list (ACL), bucket policy, bucket ACL) and IAM user policies—into a set of policies to be evaluated at run time. It then evaluates the resulting

set of policies in a series of steps. In each step, it evaluates a subset of policies in three specific contexts—user context, bucket context, and object context.

1. **User context** – If the requester is an IAM user, the user must have permission from the parent AWS account to which it belongs. In this step, Amazon S3 evaluates a subset of policies owned by the parent account (also referred as the context authority). This subset of policies includes the user policy that the parent attaches to the user. If the parent also owns the resource in the request (bucket, object), Amazon S3 evaluates the corresponding resource policies (bucket policy, bucket ACL, and object ACL) at the same time.

Note

If the parent AWS account owns the resource (bucket or object), it can grant resource permissions to its IAM user by using either the user policy or the resource policy.

2. **Bucket context** – In this context, Amazon S3 evaluates policies owned by the AWS account that owns the bucket.

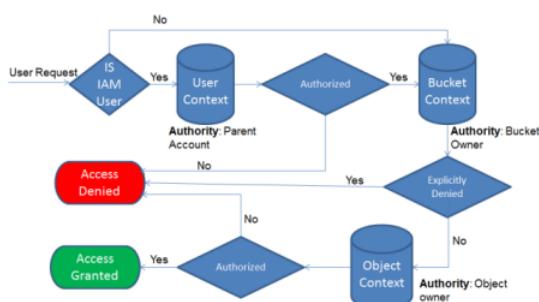
If the AWS account that owns the object in the request is not same as the bucket owner, in the bucket context Amazon S3 checks the policies if the bucket owner has explicitly denied access to the object. If there is an explicit deny set on the object, Amazon S3 does not authorize the request.

3. **Object context** – The requester must have permissions from the object owner to perform a specific object operation. In this step, Amazon S3 evaluates the object ACL.

Note

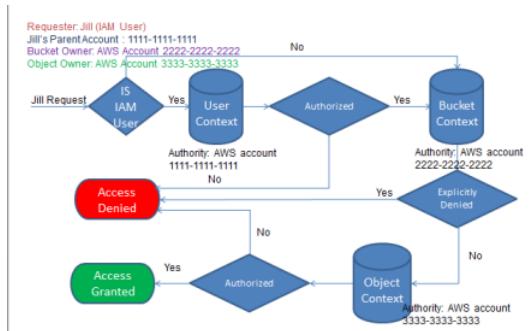
If bucket and object owners are the same, access to the object can be granted in the bucket policy, which is evaluated at the bucket context. If the owners are different, the object owners must use an object ACL to grant permissions. If the AWS account that owns the object is also the parent account to which the IAM user belongs, it can configure object permissions in a user policy, which is evaluated at the user context. For more information about using these access policy alternatives, see [Guidelines for Using the Available Access Policy Options \(p. 280\)](#).

The following is an illustration of the context-based evaluation for an object operation.



Example 1: Object Operation Request

In this example, IAM user Jill, whose parent AWS account is 1111-1111-1111, sends an object operation request (for example, Get object) for an object owned by AWS account 3333-3333-3333 in a bucket owned by AWS account 2222-2222-2222.



Jill will need permission from the parent AWS account, the bucket owner, and the object owner. Amazon S3 evaluates the context as follows:

- Because the request is from an IAM user, Amazon S3 evaluates the user context to verify that the parent AWS account 1111-1111-1111 has given Jill permission to perform the requested operation. If she has that permission, Amazon S3 evaluates the bucket context. Otherwise, Amazon S3 denies the request.
- In the bucket context, the bucket owner, AWS account 2222-2222-2222, is the context authority. Amazon S3 evaluates the bucket policy to determine if the bucket owner has explicitly denied Jill access to the object.
- In the object context, the context authority is AWS account 3333-3333-3333, the object owner. Amazon S3 evaluates the object ACL to determine if Jill has permission to access the object. If she does, Amazon S3 authorizes the request.

Guidelines for Using the Available Access Policy Options

Amazon S3 supports resource-based policies and user policies to manage access to your Amazon S3 resources (see [Managing Access to Resources \(Access Policy Options\) \(p. 271\)](#)). Resource-based policies include bucket policies, bucket ACLs, and object ACLs. This section describes specific scenarios for using resource-based access policies to manage access to your Amazon S3 resources.

When to Use an ACL-based Access Policy (Bucket and Object ACLs)

Both buckets and objects have associated ACLs that you can use to grant permissions. The following sections describe scenarios for using object ACLs and bucket ACLs.

When to Use an Object ACL

In addition to an object ACL, there are other ways an object owner can manage object permissions. For example:

- If the AWS account that owns the object also owns the bucket, then it can write a bucket policy to manage the object permissions.
- If the AWS account that owns the object wants to grant permission to a user in its account, it can use a user policy.

So when do you use object ACLs to manage object permissions? The following are the scenarios when you use object ACLs to manage object permissions.

- **An object ACL is the only way to manage access to objects not owned by the bucket owner –** An AWS account that owns the bucket can grant another AWS account permission to upload objects. The bucket owner does not own these objects. The AWS account that created the object must grant permissions using object ACLs.

Note

A bucket owner cannot grant permissions on objects it does not own. For example, a bucket policy granting object permissions applies only to objects owned by the bucket owner. However, the bucket owner, who pays the bills, can write a bucket policy to deny access to any objects in the bucket, regardless of who owns it. The bucket owner can also delete any objects in the bucket.

- **Permissions vary by object and you need to manage permissions at the object level –** You can write a single policy statement granting an AWS account read permission on millions of objects with a specific key name prefix. For example, grant read permission on objects starting with key name prefix "logs". However, if your access permissions vary by object, granting permissions to individual objects using a bucket policy may not be practical. Also the bucket policies are limited to 20 KB in size.

In this case, you may find using object ACLs a suitable alternative. Although, even an object ACL is also limited to a maximum of 100 grants (see [Access Control List \(ACL\) Overview \(p. 363\)](#)).

- **Object ACLs control only object-level permissions –** There is a single bucket policy for the entire bucket, but object ACLs are specified per object.

An AWS account that owns a bucket can grant another AWS account permission to manage access policy. It allows that account to change anything in the policy. To better manage permissions, you may choose not to give such a broad permission, and instead grant only the READ-ACP and WRITE-ACP permissions on a subset of objects. This limits the account to manage permissions only on specific objects by updating individual object ACLs.

When to Use a Bucket ACL

The only recommended use case for the bucket ACL is to grant write permission to the Amazon S3 Log Delivery group to write access log objects to your bucket (see [Server Access Logging \(p. 530\)](#)). If you want Amazon S3 to deliver access logs to your bucket, you will need to grant write permission on the bucket to the Log Delivery group. The only way you can grant necessary permissions to the Log Delivery group is via a bucket ACL, as shown in the following bucket ACL fragment.

```
<?xml version="1.0" encoding="UTF-8"?>
<AccessControlPolicy xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Owner>
    ...
  </Owner>
  <AccessControlList>
    <Grant>
      ...
    </Grant>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="Group">
        <URI>http://acs.amazonaws.com/groups/s3/LogDelivery</URI>
      </Grantee>
      <Permission>WRITE</Permission>
    </Grant>
  </AccessControlList>
</AccessControlPolicy>
```

When to Use a Bucket Policy

If an AWS account that owns a bucket wants to grant permission to users in its account, it can use either a bucket policy or a user policy. But in the following scenarios, you will need to use a bucket policy.

- **You want to manage cross-account permissions for all Amazon S3 permissions** – You can use ACLs to grant cross-account permissions to other accounts, but ACLs support only a finite set of permission ([What Permissions Can I Grant? \(p. 365\)](#)), these don't include all Amazon S3 permissions. For example, you cannot grant permissions on bucket subresources (see [Managing Access Permissions to Your Amazon S3 Resources \(p. 269\)](#)) using an ACL.

Although both bucket and user policies support granting permission for all Amazon S3 operations (see [Specifying Permissions in a Policy \(p. 315\)](#)), the user policies are for managing permissions for users in your account. For cross-account permissions to other AWS accounts or users in another account, you must use a bucket policy.

When to Use a User Policy

In general, you can use either a user policy or a bucket policy to manage permissions. You may choose to manage permissions by creating users and managing permissions individually by attaching policies to users (or user groups), or you may find that resource-based policies, such as a bucket policy, work better for your scenario.

Note that AWS Identity and Access Management (IAM) enables you to create multiple users within your AWS account and manage their permissions via user policies. An IAM user must have permissions from the parent account to which it belongs, and from the AWS account that owns the resource the user wants to access. The permissions can be granted as follows:

- **Permission from the parent account** – The parent account can grant permissions to its user by attaching a user policy.
- **Permission from the resource owner** – The resource owner can grant permission to either the IAM user (using a bucket policy) or the parent account (using a bucket policy, bucket ACL, or object ACL).

This is akin to a child who wants to play with a toy that belongs to someone else. In this case, the child must get permission from a parent to play with the toy and permission from the toy owner.

Permission Delegation

If an AWS account owns a resource, it can grant those permissions to another AWS account. That account can then delegate those permissions, or a subset of them, to users in the account. This is referred to as permission delegation. But an account that receives permissions from another account cannot delegate permission cross-account to another AWS account.

Related Topics

We recommend you first review all introductory topics that explain how you manage access to your Amazon S3 resources and related guidelines. For more information, see [Introduction to Managing Access Permissions to Your Amazon S3 Resources \(p. 269\)](#). You can then use the following topics for more information about specific access policy options.

- [Using Bucket Policies and User Policies \(p. 312\)](#)
- [Managing Access with ACLs \(p. 363\)](#)

Example Walkthroughs: Managing Access to Your Amazon S3 Resources

This topic provides the following introductory walkthrough examples for granting access to Amazon S3 resources. These examples use the AWS Management Console to create resources (buckets, objects, users) and grant them permissions. The examples then show you how to verify permissions using the command line tools, so you don't have to write any code. We provide commands using both the AWS Command Line Interface (CLI) and the AWS Tools for Windows PowerShell.

- [Example 1: Bucket Owner Granting Its Users Bucket Permissions \(p. 287\)](#)

The IAM users you create in your account have no permissions by default. In this exercise, you grant a user permission to perform bucket and object operations.

- [Example 2: Bucket Owner Granting Cross-Account Bucket Permissions \(p. 292\)](#)

In this exercise, a bucket owner, Account A, grants cross-account permissions to another AWS account, Account B. Account B then delegates those permissions to users in its account.

- **Managing object permissions when the object and bucket owners are not the same**

The example scenarios in this case are about a bucket owner granting object permissions to others, but not all objects in the bucket are owned by the bucket owner. What permissions does the bucket owner need, and how can it delegate those permissions?

The AWS account that creates a bucket is called the bucket owner. The owner can grant other AWS accounts permission to upload objects, and the AWS accounts that create objects own them. The bucket owner has no permissions on those objects created by other AWS accounts. If the bucket owner writes a bucket policy granting access to objects, the policy does not apply to objects that are owned by other accounts.

In this case, the object owner must first grant permissions to the bucket owner using an object ACL. The bucket owner can then delegate those object permissions to others, to users in its own account, or to another AWS account, as illustrated by the following examples.

- [Example 3: Bucket Owner Granting Its Users Permissions to Objects It Does Not Own \(p. 298\)](#)

In this exercise, the bucket owner first gets permissions from the object owner. The bucket owner then delegates those permissions to users in its own account.

- [Example 4: Bucket Owner Granting Cross-account Permission to Objects It Does Not Own \(p. 303\)](#)

After receiving permissions from the object owner, the bucket owner cannot delegate permission to other AWS accounts because cross-account delegation is not supported (see [Permission Delegation \(p. 282\)](#)). Instead, the bucket owner can create an IAM role with permissions to perform specific operations (such as get object) and allow another AWS account to assume that role. Anyone who assumes the role can then access objects. This example shows how a bucket owner can use an IAM role to enable this cross-account delegation.

Before You Try the Example Walkthroughs

These examples use the AWS Management Console to create resources and grant permissions. And to test permissions, the examples use the command line tools, AWS Command Line Interface (CLI) and AWS Tools for Windows PowerShell, so you don't need to write any code. To test permissions you will need to set up one of these tools. For more information, see [Setting Up the Tools for the Example Walkthroughs \(p. 284\)](#).

In addition, when creating resources these examples don't use root credentials of an AWS account. Instead, you create an administrator user in these accounts to perform these tasks.

About Using an Administrator User to Create Resources and Grant Permissions

AWS Identity and Access Management (IAM) recommends not using the root credentials of your AWS account to make requests. Instead, create an IAM user, grant that user full access, and then use that user's credentials to interact with AWS. We refer to this user as an administrator user. For more information, go to [Root Account Credentials vs. IAM User Credentials](#) in the *AWS General Reference* and [IAM Best Practices](#) in *Using IAM*.

All example walkthroughs in this section use the administrator user credentials. If you have not created an administrator user for your AWS account, the topics show you how.

Note that to sign in to the AWS Management Console using the user credentials, you will need to use the IAM User Sign-In URL. The IAM console provides this URL for your AWS account. The topics show you how to get the URL.

Setting Up the Tools for the Example Walkthroughs

The introductory examples (see [Example Walkthroughs: Managing Access to Your Amazon S3 Resources \(p. 283\)](#)) use the AWS Management Console to create resources and grant permissions. And to test permissions, the examples use the command line tools, AWS Command Line Interface (CLI) and AWS Tools for Windows PowerShell, so you don't need to write any code. To test permissions, you must set up one of these tools.

To set up the AWS CLI

1. Download and configure the AWS CLI. For instructions, see the following topics in the *AWS Command Line Interface User Guide*.

[Getting Set Up with the AWS Command Line Interface](#)

[Installing the AWS Command Line Interface](#)

[Configuring the AWS Command Line Interface](#)

2. Set the default profile.

You will store user credentials in the AWS CLI config file. Create a default profile in the config file using your AWS account credentials.

```
[default]
aws_access_key_id = access key ID
aws_secret_access_key = secret access key
region = us-west-2
```

3. Verify the setup by entering the following command at the command prompt. Both these commands don't provide credentials explicitly, so the credentials of the default profile are used.

- Try the help command

```
aws help
```

- Use aws s3 ls to get a list of buckets on the configured account.

```
aws s3 ls
```

As you go through the example walkthroughs, you will create users, and you will save user credentials in the config files by creating profiles, as the following example shows. Note that these profiles have names (AccountAadmin and AccountBadmin):

```
[profile AccountAadmin]
aws_access_key_id = User AccountAadmin access key ID
aws_secret_access_key = User AccountAadmin secret access key
region = us-west-2

[profile AccountBadmin]
aws_access_key_id = Account B access key ID
aws_secret_access_key = Account B secret access key
region = us-east-1
```

To execute a command using these user credentials, you add the `--profile` parameter specifying the profile name. The following AWS CLI command retrieves a listing of objects in `examplebucket` and specifies the `AccountBadmin` profile.

```
aws s3 ls s3://examplebucket --profile AccountBadmin
```

Alternatively, you can configure one set of user credentials as the default profile by changing the `AWS_DEFAULT_PROFILE` environment variable from the command prompt. Once you've done this, whenever you execute AWS CLI commands without the `--profile` parameter, the AWS CLI will use the profile you set in the environment variable as the default profile.

```
$ export AWS_DEFAULT_PROFILE=AccountAadmin
```

To set up AWS Tools for Windows PowerShell

1. Download and configure the AWS Tools for Windows PowerShell. For instructions, go to [Download and Install the AWS Tools for Windows PowerShell](#) in the *AWS Tools for Windows PowerShell User Guide*.

Note

In order to load the AWS Tools for Windows PowerShell module, you need to enable PowerShell script execution. For more information, go to [Enable Script Execution](#) in the *AWS Tools for Windows PowerShell User Guide*.

2. For these exercises, you will specify AWS credentials per session using the `Set-AWSCredentials` command. The command saves the credentials to a persistent store (`-StoreAs` parameter).

```
Set-AWSCredentials -AccessKey AccessKeyID -SecretKey SecretAccessKey -storeas
string
```

3. Verify the setup.
 - Execute the `Get-Command` to retrieve a list of available commands you can use for Amazon S3 operations.

```
Get-Command -module awspowershell -noun s3* -StoredCredentials string
```

- Execute the `Get-S3Object` command to retrieve a list of objects in a bucket.

```
Get-S3Object -BucketName bucketname -StoredCredentials string
```

For a list of commands, go to [Amazon Simple Storage Service Cmdlets](#).

Now you are ready to try the exercises. Follow the links provided at the beginning of the section.

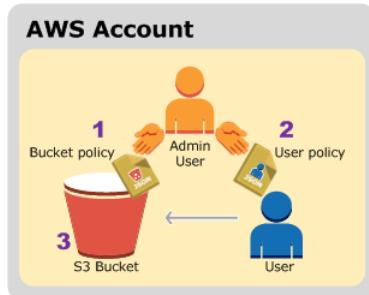
Example 1: Bucket Owner Granting Its Users Bucket Permissions

Topics

- Step 0: Preparing for the Walkthrough (p. 287)
- Step 1: Create Resources (a Bucket and an IAM User) in Account A and Grant Permissions (p. 288)
- Step 2: Test Permissions (p. 290)

In this exercise, an AWS account owns a bucket, and it has an IAM user in the account. The user by default has no permissions. The parent account must grant permissions to the user to perform any tasks. Both the bucket owner and the parent account to which the user belongs are the same. Therefore, the AWS account can use a bucket policy, a user policy, or both to grant its user permissions on the bucket. You will grant some permissions using a bucket policy and grant other permissions using a user policy.

The following steps summarize the walkthrough:



1. Account administrator creates a bucket policy granting a set of permissions to the user.
2. Account administrator attaches a user policy to the user granting additional permissions.
3. User then tries permissions granted via both the bucket policy and the user policy.

For this example, you will need an AWS account. Instead of using the root credentials of the account, you will create an administrator user (see [About Using an Administrator User to Create Resources and Grant Permissions \(p. 284\)](#)). We refer to the AWS account and the administrator user as follows:

Account ID	Account Referred To As	Administrator User in the Account
1111-1111-1111	Account A	AccountAdmin

All the tasks of creating users and granting permissions are done in the AWS Management Console. To verify permissions, the walkthrough uses the command line tools, AWS Command Line Interface (CLI) and AWS Tools for Windows PowerShell, to verify the permissions, so you don't need to write any code.

Step 0: Preparing for the Walkthrough

1. Make sure you have an AWS account and that it has a user with administrator privileges.
 - a. Sign up for an account, if needed. We refer to this account as Account A.
 - i. Go to <http://aws.amazon.com/s3> and click **Sign Up**.
 - ii. Follow the on-screen instructions.

AWS will notify you by email when your account is active and available for you to use.

- b. In Account A, create an administrator user AccountAadmin. Using Account A credentials, sign in to the [IAM console](#) and do the following:
 - i. Create user AccountAadmin and note down the user security credentials.
For instructions, go to [Creating an IAM User in Your AWS Account](#) in *Using IAM*.
 - ii. Grant AccountAadmin administrator privileges by attaching a user policy giving full access.
For instructions, go to [Managing Policies \(AWS Management Console\)](#) in *Using IAM*.
 - iii. Note down the **IAM User Sign-In URL** for AccountAadmin. You will need to use this URL when signing in to the AWS Management Console. For more information about where to find it, go to [How IAM Users Sign in to Your AWS Account](#) in *Using IAM*. Note down the URL for each of the accounts.
2. Set up either the AWS Command Line Interface (CLI) or the AWS Tools for Windows PowerShell. Make sure you save administrator user credentials as follows:
 - If using the AWS CLI, create two profiles, AccountAadmin and AccountBadmin, in the config file.
 - If using the AWS Tools for Windows PowerShell, make sure you store credentials for the session as AccountAadmin and AccountBadmin.

For instructions, see [Setting Up the Tools for the Example Walkthroughs \(p. 284\)](#).

Step 1: Create Resources (a Bucket and an IAM User) in Account A and Grant Permissions

Using the credentials of user AccountAadmin in Account A, and the special IAM user sign-in URL, sign in to the AWS Management Console and do the following:

1. Create Resources (a bucket and an IAM user)
 - a. In the Amazon S3 console create a bucket. Note down the AWS region in which you created it. For instructions, go to [Creating a Bucket](#) in the *Amazon Simple Storage Service Console User Guide*.
 - b. In the IAM console, do the following:
 - i. Create a user, Dave.
For instructions, go to [Adding an IAM User \(AWS Management Console\)](#) in *Using IAM*.
 - ii. Note down the UserDave credentials.
 - iii. Note down the Amazon Resource Name (ARN) for user Dave. In the IAM console, select the user, and the **Summary** tab provides the user ARN.
2. Grant Permissions.
Because the bucket owner and the parent account to which the user belongs are the same, the AWS account can grant user permissions using a bucket policy, a user policy, or both. In this example, you do both. If the object is also owned by the same account, the bucket owner can grant object permissions in the bucket policy (or an IAM policy).
 - a. In the Amazon S3 console, attach the following bucket policy to `examplebucket`.

The policy has two statements.

- The first statement grants Dave the bucket operation permissions `s3:GetBucketLocation` and `s3>ListBucket`.
- The second statement grants the `s3:GetObject` permission. Because Account A also owns the object, the account administrator is able to grant the `s3:GetObject` permission.

In the `Principal` statement, Dave is identified by his user ARN. For more information about policy elements, see [Access Policy Language Overview \(p. 312\)](#).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "statement1",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::AccountA-ID:user/Dave"
            },
            "Action": [
                "s3:GetBucketLocation",
                "s3>ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::examplebucket"
            ]
        },
        {
            "Sid": "statement2",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::AccountA-ID:user/Dave"
            },
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::examplebucket/*"
            ]
        }
    ]
}
```

- Create an inline policy for the user Dave by using the following policy. The policy grants Dave the `s3:PutObject` permission. You need to update the policy by providing your bucket name.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PermissionForObjectOperations",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": [

```

```
        "arn:aws:s3:::examplebucket/*"
    ]
}
}
```

For instructions, go to [Managing Inline Policies \(AWS Management Console\)](#) in *Using IAM*. Note you need to sign in to the console using Account B credentials.

Step 2: Test Permissions

Using Dave's credentials, verify that the permissions work. You can use either of the following two procedures.

Test Using the AWS CLI

1. Update the AWS CLI config file by adding the following UserDaveAccountA profile. For more information, see [Setting Up the Tools for the Example Walkthroughs \(p. 284\)](#).

```
[profile UserDaveAccountA]
aws_access_key_id = access-key
aws_secret_access_key = secret-access-key
region = us-east-1
```

2. Verify that Dave can perform the operations as granted in the user policy. Upload a sample object using the following AWS CLI put-object command.

The --body parameter in the command identifies the source file to upload. For example, if the file is in the root of the C: drive on a Windows machine, you specify c:\HappyFace.jpg. The --key parameter provides the key name for the object.

```
aws s3api put-object --bucket examplebucket --key HappyFace.jpg --body
HappyFace.jpg --profile UserDaveAccountA
```

Execute the following AWS CLI command to get the object.

```
aws s3api get-object --bucket examplebucket --key HappyFace.jpg OutputFile.jpg
--profile UserDaveAccountA
```

Test Using the AWS Tools for Windows PowerShell

1. Store Dave's credentials as AccountADave. You then use these credentials to PUT and GET an object.

```
set-awscredentials -AccessKey AccessKeyID -SecretKey SecretAccessKey -storeas
AccountADave
```

2. Upload a sample object using the AWS Tools for Windows PowerShell Write-S3Object command using user Dave's stored credentials.

```
Write-S3Object -bucketname examplebucket -key HappyFace.jpg -file HappyFace.jpg -StoredCredentials AccountADave
```

Download the previously uploaded object.

```
Read-S3Object -bucketname examplebucket -key HappyFace.jpg -file Output.jpg -StoredCredentials AccountADave
```

Example 2: Bucket Owner Granting Cross-Account Bucket Permissions

Topics

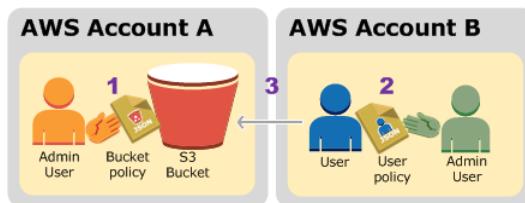
- Step 0: Preparing for the Walkthrough (p. 293)
- Step 1: Do the Account A Tasks (p. 294)
- Step 2: Do the Account B Tasks (p. 295)
- Step 3: Extra Credit: Try Explicit Deny (p. 297)
- Step 4: Clean Up (p. 298)

An AWS account—for example, Account A—can grant another AWS account, Account B, permission to access its resources such as buckets and objects. Account B can then delegate those permissions to users in its account. In this example scenario, a bucket owner grants cross-account permission to another account to perform specific bucket operations.

Note

Account A can also directly grant a user in Account B permissions using a bucket policy. But the user will still need permission from the parent account, Account B, to which the user belongs, even if Account B does not have permissions from Account A. As long as the user has permission from both the resource owner and the parent account, the user will be able to access the resource.

The following is a summary of the walkthrough steps:



1. Account A administrator user attaches a bucket policy granting cross-account permissions to Account B to perform specific bucket operations.

Note that administrator user in Account B will automatically inherit the permissions.
2. Account B administrator user attaches user policy to the user delegating the permissions it received from Account A.
3. User in Account B then verifies permissions by accessing a object in the bucket owned by Account A.

For this example, you need two accounts. The following table shows how we refer to these accounts and the administrator users in them. Per IAM guidelines (see [About Using an Administrator User to Create Resources and Grant Permissions \(p. 284\)](#)) we do not use the account root credentials in this walkthrough. Instead, you create an administrator user in each account and use those credentials in creating resources and granting them permissions.

AWS Account ID	Account Referred To As	Administrator User in the Account
1111-1111-1111	Account A	AccountAadmin
2222-2222-2222	Account B	AccountBadmin

All the tasks of creating users and granting permissions are done in the AWS Management Console. To verify permissions, the walkthrough uses the command line tools, AWS Command Line Interface (CLI) and AWS Tools for Windows PowerShell, so you don't need to write any code.

Step 0: Preparing for the Walkthrough

1. Make sure you have two AWS accounts and that each account has one administrator user as shown in the table in the preceding section.
 - a. Sign up for an AWS account, if needed.
 - i. Go to <http://aws.amazon.com/s3> and click **Sign Up**.
 - ii. Follow the on-screen instructions.AWS will notify you by email when your account is active and available for you to use.
 - b. Using Account A credentials, sign in to the [IAM console](#) to create the administrator user:
 - i. Create user AccountAadmin and note down the security credentials. For instructions, go to [Creating an IAM User in Your AWS Account](#) in *Using IAM*.
 - ii. Grant AccountAadmin administrator privileges by attaching a user policy giving full access. For instructions, go to [Managing Policies \(AWS Management Console\)](#) in *Using IAM*.
 - c. While you are in the IAM console, note down the **IAM User Sign-In URL** on the **Dashboard**. All users in the account must use this URL when signing in to the AWS Management Console. For more information, go to [How IAM Users Sign In to Your AWS Account](#) in *Using IAM*.
 - d. Repeat the preceding step using Account B credentials and create administrator user AccountBadmin.
 2. Set up either the AWS Command Line Interface (CLI) or the AWS Tools for Windows PowerShell. Make sure you save administrator user credentials as follows:
 - If using the AWS CLI, create two profiles, AccountAadmin and AccountBadmin, in the config file.
 - If using the AWS Tools for Windows PowerShell, make sure you store credentials for the session as AccountAadmin and AccountBadmin.For instructions, see [Setting Up the Tools for the Example Walkthroughs \(p. 284\)](#).
 3. Save the administrator user credentials, also referred to as profiles. You can use the profile name instead of specifying credentials for each command you enter. For more information, see [Setting Up the Tools for the Example Walkthroughs \(p. 284\)](#).
 - a. Add profiles in the AWS CLI config file for each of the administrator users in the two accounts.

```
[profile AccountAadmin]
aws_access_key_id = access-key-ID
aws_secret_access_key = secret-access-key
region = us-east-1

[profile AccountBadmin]
aws_access_key_id = access-key-ID
aws_secret_access_key = secret-access-key
region = us-east-1
```

- b. If you are using the AWS Tools for Windows PowerShell

```
set-awscredentials -AccessKey AcctA-access-key-ID -SecretKey AcctA-secret-access-key -storeas AccountAadmin  
set-awscredentials -AccessKey AcctB-access-key-ID -SecretKey AcctB-secret-access-key -storeas AccountBadmin
```

Step 1: Do the Account A Tasks

Step 1.1: Sign In to the AWS Management Console

Using the IAM user sign-in URL for Account A first sign in to the AWS Management Console as AccountAadmin user. This user will create a bucket and attach a policy to it.

Step 1.2: Create a Bucket

1. In the Amazon S3 console, create a bucket. This exercise assumes the bucket is created in the US Standard region and is named examplebucket.

For instructions, go to [Creating a Bucket](#) in the *Amazon Simple Storage Service Console User Guide*.

2. Upload a sample object to the bucket.

For instructions, go to [Add an Object to a Bucket](#) in the *Amazon Simple Storage Service Getting Started Guide*.

Step 1.3: Attach a Bucket Policy to Grant Cross-Account Permissions to Account B

The bucket policy grants the s3:GetBucketLocation and s3>ListBucket permissions to Account B. It is assumed you are still signed into the console using AccountAadmin user credentials.

1. Attach the following bucket policy to examplebucket. The policy grants Account B permission for the s3:GetBucketLocation and s3>ListBucket actions.

For instructions on editing bucket permissions, go to [Editing Bucket Permissions](#) in the *Amazon Simple Storage Service Console User Guide*. Follow these steps to add a bucket policy.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "Example permissions",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "arn:aws:iam::AccountB-ID:root"  
            },  
            "Action": [  
                "s3:GetBucketLocation",  
                "s3>ListBucket"  
            ],  
            "Resource": [  
                "arn:aws:s3:::examplebucket"  
            ]  
        }  
    ]  
}
```

```
    ]  
}
```

2. Verify Account B (and thus its administrator user) can perform the operations.

- Using the AWS CLI

```
aws s3 ls s3://examplebucket --profile AccountBadmin  
aws s3api get-bucket-location --bucket examplebucket --profile AccountBadmin
```

- Using the AWS Tools for Windows PowerShell

```
get-s3object -BucketName example2bucket -StoredCredentials AccountBadmin  
get-s3bucketlocation -BucketName example2bucket -StoredCredentials Account  
Badmin
```

Step 2: Do the Account B Tasks

Now the Account B administrator creates a user, Dave, and delegates the Dave permissions received from Account A.

Step 2.1: Sign In to the AWS Management Console

Using the IAM user sign-in URL for Account B, first sign in to the AWS Management Console as AccountBadmin user.

Step 2.2: Create User Dave in Account B

1. In the IAM console, create a user, Dave.

For instructions, go to [Adding an IAM User \(AWS Management Console\)](#) in *Using IAM*.

2. Note down the UserDave credentials.

Step 2.3: Delegate Permissions to User Dave

- Create an inline policy for the user Dave by using the following policy. You will need to update the policy by providing your bucket name.

It is assumed you are signed in to the console using AccountBadmin user credentials.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "Example",  
            "Effect": "Allow",  
            "Action": [  
                "s3>ListBucket"  
            ],  
            "Resource": [  
                "arn:aws:s3:::examplebucket"  
            ]  
        }  
    ]  
}
```

```
        ]  
    }  
}  
}
```

For instructions, go to [Managing Inline Policies \(AWS Management Console\)](#) in *Using IAM*.

Step 2.4: Test Permissions

Now Dave in Account B can list the contents of `examplebucket` owned by Account A. You can verify the permissions using either of the following procedures.

Test Using the AWS CLI

1. Add the UserDave profile to the AWS CLI config file. For more information about the config file, see [Setting Up the Tools for the Example Walkthroughs \(p. 284\)](#).

```
[profile UserDave]  
aws_access_key_id = access-key  
aws_secret_access_key = secret-access-key  
region = us-east-1
```

2. At the command prompt, enter the following AWS CLI command to verify Dave can now get an object list from the `examplebucket` owned by Account A. Note the command specifies the UserDave profile.

```
aws s3 ls s3://examplebucket --profile UserDave
```

Dave does not have any other permissions. So if he tries any other operation—for example, the following get bucket location—Amazon S3 returns permission denied.

```
aws s3api get-bucket-location --bucket examplebucket --profile UserDave
```

Test Using AWS Tools for Windows PowerShell

1. Store Dave's credentials as AccountBDave.

```
set-awscredentials -AccessKey AccessKeyID -SecretKey SecretAccessKey -storeas AccountBDave
```

2. Try the List Bucket command.

```
get-s3object -BucketName example2bucket -StoredCredentials AccountBDave
```

Dave does not have any other permissions. So if he tries any other operation—for example, the following get bucket location—Amazon S3 returns permission denied.

```
get-s3bucketlocation -BucketName example2bucket -StoredCredentials Account BDave
```

Step 3: Extra Credit: Try Explicit Deny

You can have permissions granted via an ACL, a bucket policy, and a user policy. But if there is an explicit deny set via either a bucket policy or a user policy, the explicit deny takes precedence over any other permissions. For testing, let's update the bucket policy and explicitly deny Account B the s3:ListBucket permission. The policy also grants s3>ListBucket permission, but explicit deny takes precedence, and Account B or users in Account B will not be able to list objects in examplebucket.

1. Using credentials of user AccountAadmin in Account A, replace the bucket policy by the following.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "Example permissions",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "arn:aws:iam::AccountB-ID:root"  
            },  
            "Action": [  
                "s3:GetBucketLocation",  
                "s3>ListBucket"  
            ],  
            "Resource": [  
                "arn:aws:s3:::examplebucket"  
            ]  
        },  
        {  
            "Sid": "Deny permission",  
            "Effect": "Deny",  
            "Principal": {  
                "AWS": "arn:aws:iam::AccountB-ID:root"  
            },  
            "Action": [  
                "s3>ListBucket"  
            ],  
            "Resource": [  
                "arn:aws:s3:::examplebucket"  
            ]  
        }  
    ]  
}
```

2. Now if you try to get a bucket list using AccountBadmin credentials, you will get access denied.

- Using the AWS CLI:

```
aws s3 ls s3://examplebucket --profile AccountBadmin
```

- Using the AWS Tools for Windows PowerShell:

```
get-s3object -BucketName example2bucket -StoredCredentials AccountBDave
```

Step 4: Clean Up

1. After you are done testing, you can do the following to clean up.
 - Sign in to the AWS Management Console ([AWS Management Console](#)) using Account A credentials, and do the following:
 - In the Amazon S3 console, remove the bucket policy attached to `examplebucket`. In the bucket **Properties**, delete the policy in the **Permissions** section.
 - If the bucket is created for this exercise, in the Amazon S3 console, delete the objects and then delete the bucket.
 - In the IAM console, remove the AccountAdmin user.
2. Sign in to the AWS Management Console ([AWS Management Console](#)) using Account B credentials. In the IAM console, delete user AccountBadmin.

Example 3: Bucket Owner Granting Its Users Permissions to Objects It Does Not Own

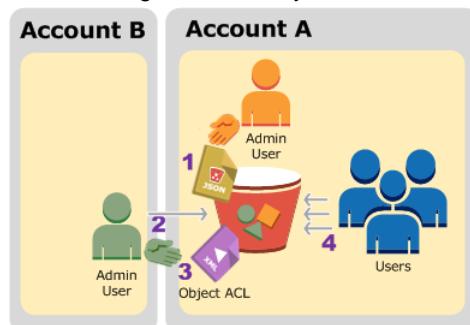
Topics

- [Step 0: Preparing for the Walkthrough \(p. 299\)](#)
- [Step 1: Do the Account A Tasks \(p. 300\)](#)
- [Step 2: Do the Account B Tasks \(p. 301\)](#)
- [Step 3: Test Permissions \(p. 301\)](#)
- [Step 4: Clean Up \(p. 302\)](#)

The scenario for this example is that a bucket owner wants to grant permission to access objects, but not all objects in the bucket are owned by the bucket owner. How can a bucket owner grant permission on objects it does not own? For this example, the bucket owner is trying to grant permission to users in its own account.

A bucket owner can enable other AWS accounts to upload objects. These objects are owned by the accounts that created them. The bucket owner does not own objects that were not created by the bucket owner. Therefore, for the bucket owner to grant access to these objects, the object owner must first grant permission to the bucket owner using an object ACL. The bucket owner can then delegate those permissions via a bucket policy. In this example, the bucket owner delegates permission to users in its own account.

The following is a summary of the walkthrough steps:



1. Account A administrator user attaches a bucket policy with two statements.
 - Allow cross-account permission to Account B to upload objects.
 - Allow a user in its own account to access objects in the bucket.
2. Account B administrator user uploads objects to the bucket owned by Account A.
3. Account B administrator updates the object ACL adding grant that gives the bucket owner full-control permission on the object.
4. User in Account A verifies by accessing objects in the bucket, regardless of who owns them.

For this example, you need two accounts. The following table shows how we refer to these accounts and the administrator users in these accounts. Per IAM guidelines (see [About Using an Administrator User to Create Resources and Grant Permissions \(p. 284\)](#)) we do not use the account root credentials in this walkthrough. Instead, you create an administrator user in each account and use those credentials in creating resources and granting them permissions.

AWS Account ID	Account Referred To As	Administrator User in the Account
1111-1111-1111	Account A	AccountAadmin
2222-2222-2222	Account B	AccountBadmin

All the tasks of creating users and granting permissions are done in the AWS Management Console. To verify permissions, the walkthrough uses the command line tools, AWS Command Line Interface (CLI) and AWS Tools for Windows PowerShell, so you don't need to write any code.

Step 0: Preparing for the Walkthrough

1. Make sure you have two AWS accounts and each account has one administrator user as shown in the table in the preceding section.
 - a. Sign up for an AWS account, if needed.
 - i. Go to <http://aws.amazon.com/s3> and click **Sign Up**.
 - ii. Follow the on-screen instructions. AWS will notify you by email when your account is active and available for you to use.
 - b. Using Account A credentials, sign in to the [IAM console](#) and do the following to create an administrator user:
 - Create user AccountAadmin and note down security credentials. For more information about adding users, go to [Creating an IAM User in Your AWS Account](#) in *Using IAM*.
 - Grant AccountAadmin administrator privileges by attaching a user policy giving full access. For instructions, go to [Managing Policies \(AWS Management Console\)](#) in *Using IAM*.
 - In the IAM console **Dashboard**, note down the **IAM User Sign-In URL**. Users in this account must use this URL when signing in to the AWS Management Console. For more information, go to [How IAM Users Sign in to Your AWS Account](#) in *Using IAM*.
 - c. Repeat the preceding step using Account B credentials and create administrator user AccountBadmin.

2. Set up either the AWS Command Line Interface (CLI) or the AWS Tools for Windows PowerShell. Make sure you save administrator user credentials as follows:
 - If using the AWS CLI, create two profiles, AccountAdmin and AccountBadmin, in the config file.
 - If using the AWS Tools for Windows PowerShell, make sure you store credentials for the session as AccountAdmin and AccountBadmin.

For instructions, see [Setting Up the Tools for the Example Walkthroughs \(p. 284\)](#).

Step 1: Do the Account A Tasks

Step 1.1: Sign In to the AWS Management Console

Using the IAM user sign-in URL for Account A first sign in to the AWS Management Console as AccountAdmin user. This user will create a bucket and attach a policy to it.

Step 1.2: Create a Bucket, a User, and Add a Bucket Policy Granting User Permissions

1. In the Amazon S3 console, create a bucket. This exercise assumes the bucket is created in the US Standard region and the name is examplebucket.

For instructions, go to [Creating a Bucket](#) in the *Amazon Simple Storage Service Console User Guide*.

2. In the IAM console, create a user Dave.

For instructions, go to [Adding an IAM User \(AWS Management Console\)](#) in *Using IAM*.

3. Note down the Dave credentials.

4. In the Amazon S3 console, attach the following bucket policy to examplebucket bucket. For instructions, go to [Editing Bucket Permissions](#) in the *Amazon Simple Storage Service Console User Guide*. Follow steps to add a bucket policy.

The policy grants Account B the s3:PutObject and s3>ListBucket permissions. The policy also grants user Dave the s3:GetObject permission.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "Statement1",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "arn:aws:iam::AccountB-ID:root"  
            },  
            "Action": [  
                "s3:PutObject"  
            ],  
            "Resource": [  
                "arn:aws:s3:::examplebucket/*"  
            ]  
        },  
        {  
            "Sid": "Statement3",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "arn:aws:iam::AccountA-ID:user/Dave"  
            },  
            "Action": [  
                "s3:GetObject"  
            ]  
        }  
    ]  
}
```

```
        "s3:GetObject"
    ],
    "Resource": [
        "arn:aws:s3:::examplebucket/*"
    ]
}
}
```

Step 2: Do the Account B Tasks

Now that Account B has permissions to perform operations on Account A's bucket, the Account B administrator will do the following:

- Upload an object to Account A's bucket.
- Add a grant in the object ACL to allow Account A, bucket owner, full control.

Using the AWS CLI

1. Using the `put-object` AWS CLI command, upload an object. The `--body` parameter in the command identifies the source file to upload. For example, if the file is on C: drive of a Windows machine, you would specify `c:\HappyFace.jpg`. The `--key` parameter provides the key name for the object.

```
aws s3api put-object --bucket examplebucket --key HappyFace.jpg --body HappyFace.jpg --profile AccountBadmin
```

2. Add a grant to the object ACL to allow the bucket owner full control of the object.

```
aws s3api put-object-acl --bucket examplebucket --key HappyFace.jpg --grant-full-control id="AccountA-CanonicalUserID" --profile AccountBadmin
```

Using the AWS Tools for Windows PowerShell

1. Using the `Write-S3Object` AWS Tools for Windows PowerShell command, upload an object.

```
Write-S3Object -BucketName examplebucket -key HappyFace.jpg -file HappyFace.jpg -StoredCredentials AccountBadmin
```

2. Add a grant to the object ACL to allow the bucket owner full control of the object.

```
Set-S3ACL -BucketName examplebucket -Key HappyFace.jpg -CannedACLName "bucket-owner-full-control" -StoredCreden
```

Step 3: Test Permissions

Now verify user Dave in Account A can access the object owned by Account B.

Using the AWS CLI

1. Add user Dave credentials to the AWS CLI config file and create a new profile, UserDaveAccountA. For more information, see [Setting Up the Tools for the Example Walkthroughs \(p. 284\)](#).

```
[profile UserDaveAccountA]
aws_access_key_id = access-key
aws_secret_access_key = secret-access-key
region = us-east-1
```

2. Execute the get-object AWS CLI command to download HappyFace.jpg and save it locally. You provide user Dave credentials by adding the --profile parameter.

```
aws s3api get-object --bucket examplebucket --key HappyFace.jpg Outputfile.jpg
--profile UserDaveAccountA
```

Using the AWS Tools for Windows PowerShell

1. Store user Dave AWS credentials, as UserDaveAccountA, to persistent store.

```
Set-AWSCredentials -AccessKey UserDave-AccessKey -SecretKey UserDave-SecretAccessKey -StoreAs UserDaveAccountA
```

2. Execute the Read-S3Object command to download the HappyFace.jpg object and save it locally. You provide user Dave credentials by adding the -StoredCredentials parameter.

```
Read-S3Object -BucketName examplebucket -Key HappyFace.jpg -File HappyFace.jpg
-StoredCredentials UserDaveAccountA
```

Step 4: Clean Up

1. After you are done testing, you can do the following to clean up.
 - Sign in to the AWS Management Console ([AWS Management Console](#)) using Account A credentials, and do the following:
 - In the Amazon S3 console, remove the bucket policy attached to *examplebucket*. In the bucket **Properties**, delete the policy in the **Permissions** section.
 - If the bucket is created for this exercise, in the Amazon S3 console, delete the objects and then delete the bucket.
 - In the IAM console, remove the AccountAdmin user.
2. Sign in to the AWS Management Console ([AWS Management Console](#)) using Account B credentials. In the IAM console, delete user AccountBadmin.

Example 4: Bucket Owner Granting Cross-account Permission to Objects It Does Not Own

Topics

- [Background: Cross-Account Permissions and Using IAM Roles \(p. 303\)](#)
- [Step 0: Preparing for the Walkthrough \(p. 304\)](#)
- [Step 1: Do the Account A Tasks \(p. 306\)](#)
- [Step 2: Do the Account B Tasks \(p. 308\)](#)
- [Step 3: Do the Account C Tasks \(p. 308\)](#)
- [Step 4: Clean Up \(p. 310\)](#)
- [Related Resources \(p. 311\)](#)

In this example scenario, you own a bucket and you have enabled other AWS accounts to upload objects. That is, your bucket can have objects that other AWS accounts own.

Now, suppose as a bucket owner, you need to grant cross-account permission on objects, regardless of who the owner is, to a user in another account. For example, that user could be a billing application that needs to access object metadata. There are two core issues:

- The bucket owner has no permissions on those objects created by other AWS accounts. So for the bucket owner to grant permissions on objects it does not own, the object owner, the AWS account that created the objects, must first grant permission to the bucket owner. The bucket owner can then delegate those permissions.
- Bucket owner account can delegate permissions to users in its own account (see [Example 3: Bucket Owner Granting Its Users Permissions to Objects It Does Not Own \(p. 298\)](#)), but it cannot delegate permissions to other AWS accounts, because cross-account delegation is not supported.

In this scenario, the bucket owner can create an AWS Identity and Access Management (IAM) role with permission to access objects, and grant another AWS account permission to assume the role temporarily enabling it to access objects in the bucket.

Background: Cross-Account Permissions and Using IAM Roles

IAM roles enable several scenarios to delegate access to your resources, and cross-account access is one of the key scenarios. In this example, the bucket owner, Account A, uses an IAM role to temporarily delegate object access cross-account to users in another AWS account, Account C. Each IAM role you create has two policies attached to it:

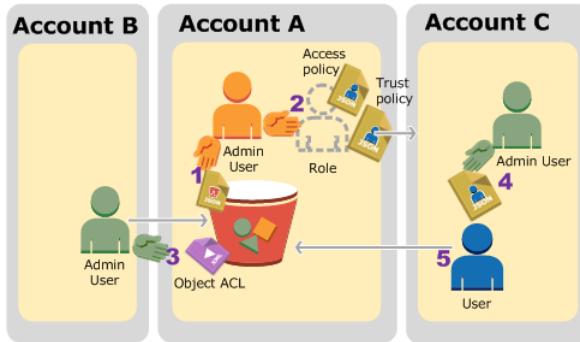
- A trust policy identifying another AWS account that can assume the role.
- An access policy defining what permissions—for example, `s3:GetObject`—are allowed when someone assumes the role. For a list of permissions you can specify in a policy, see [Specifying Permissions in a Policy \(p. 315\)](#).

The AWS account identified in the trust policy then grants its user permission to assume the role. The user can then do the following to access objects:

- Assume the role and, in response, get temporary security credentials.
- Using the temporary security credentials, access the objects in the bucket.

For more information about IAM roles, go to [Roles \(Delegation and Federation\)](#) in *Using IAM*.

The following is a summary of the walkthrough steps:



1. Account A administrator user attaches a bucket policy granting Account B conditional permission to upload objects.
2. Account A administrator creates an IAM role, establishing trust with Account C, so users in that account can access Account A. The access policy attached to the role limits what user in Account C can do when the user accesses Account A.
3. Account B administrator uploads an object to the bucket owned by Account A, granting full-control permission to the bucket owner.
4. Account C administrator creates a user and attaches a user policy that allows the user to assume the role.
5. User in Account C first assumes the role, which returns the user temporary security credentials. Using those temporary credentials, the user then accesses objects in the bucket.

For this example, you need three accounts. The following table shows how we refer to these accounts and the administrator users in these accounts. Per IAM guidelines (see [About Using an Administrator User to Create Resources and Grant Permissions \(p. 284\)](#)) we do not use the account root credentials in this walkthrough. Instead, you create an administrator user in each account and use those credentials in creating resources and granting them permissions

AWS Account ID	Account Referred To As	Administrator User in the Account
1111-1111-1111	Account A	AccountAadmin
2222-2222-2222	Account B	AccountBadmin
3333-3333-3333	Account C	AccountCadmin

Step 0: Preparing for the Walkthrough

Note

You may want to open a text editor and write down some of the information as you walk through the steps. In particular, you will need account IDs, canonical user IDs, IAM User Sign-in URLs for each account to connect to the console, and Amazon Resource Names (ARNs) of the IAM users, and roles.

1. Make sure you have three AWS accounts and each account has one administrator user as shown in the table in the preceding section.
 - a. Sign up for AWS accounts, as needed. We refer to these accounts as Account A, Account B, and Account C.
 - i. Go to <http://aws.amazon.com/s3> and click **Sign Up**.

- ii. Follow the on-screen instructions.

AWS will notify you by email when your account is active and available for you to use.

- b. Using Account A credentials, sign in to the [IAM console](#) and do the following to create an administrator user:
 - Create user AccountAadmin and note down security credentials. For more information about adding users, go to [Creating an IAM User in Your AWS Account](#) in *Using IAM*.
 - Grant AccountAadmin administrator privileges by attaching a user policy giving full access. For instructions, go to [Managing Policies \(AWS Management Console\)](#) in *Using IAM*.
 - In the IAM Console **Dashboard**, note down the **IAM User Sign-In URL**. Users in this account must use this URL when signing in to the AWS Management Console. For more information, go to [How IAM Users Sign In to Your AWS Account](#) in *Using IAM*.
- c. Repeat the preceding step to create administrator users in Account B and Account C.

2. For Account C, note down the account ID.

When you create an IAM role in Account A, the trust policy grants Account C permission to assume the role by specifying the account ID. You can find account information as follows:

- a. Go to <http://aws.amazon.com/> and from the **My Account/Console** drop-down menu, select **Security Credentials**.
 - b. Sign in using appropriate account credentials.
 - c. Click **Account Identifiers** and note down the **AWS Account ID** and the **Canonical User ID**.
3. When creating a bucket policy, you will need the following information. Note down these values:
 - **Canonical user ID of Account A** – When the Account A administrator grants conditional upload object permission to the Account B administrator, the condition specifies the canonical user ID of the Account A user that must get full-control of the objects.

Note

The canonical user ID is the Amazon S3–only concept. It is a 64-character obfuscated version of the account ID.

- **User ARN for Account B administrator** – You can find the user ARN in the IAM console. You will need to select the user and find the user's ARN in the **Summary** tab.

In the bucket policy, you grant AccountBadmin permission to upload objects and you specify the user using the ARN. Here's an example ARN value:

```
arn:aws:iam::AccountB-ID:user/AccountBadmin
```

4. Set up either the AWS Command Line Interface (CLI) or the AWS Tools for Windows PowerShell. Make sure you save administrator user credentials as follows:
 - If using the AWS CLI, create profiles, AccountAadmin and AccountBadmin, in the config file.
 - If using the AWS Tools for Windows PowerShell, make sure you store credentials for the session as AccountAadmin and AccountBadmin.

For instructions, see [Setting Up the Tools for the Example Walkthroughs \(p. 284\)](#).

Step 1: Do the Account A Tasks

In this example, Account A is the bucket owner. So user AccountAadmin in Account A will create a bucket, attach a bucket policy granting the Account B administrator permission to upload objects, create an IAM role granting Account C permission to assume the role so it can access objects in the bucket.

Step 1.1: Sign In to the AWS Management Console

Using the IAM User Sign-in URL for Account A, first sign in to the AWS Management Console as AccountAadmin user. This user will create a bucket and attach a policy to it.

Step 1.2: Create a Bucket and Attach a Bucket Policy

In the Amazon S3 console, do the following:

1. Create a bucket. This exercise assumes the bucket name is `examplebucket`.

For instructions, go to [Creating a Bucket](#) in the *Amazon Simple Storage Service Console User Guide*.

2. Attach the following bucket policy granting conditional permission to the Account B administrator permission to upload objects.

You need to update the policy by providing your own values for `examplebucket`, `AccountB-ID`, and the `CanonicalUserId-of-AWSaccountA-BucketOwner`.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "111",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "arn:aws:iam::AccountB-ID:user/AccountBadmin"  
            },  
            "Action": "s3:PutObject",  
            "Resource": "arn:aws:s3:::examplebucket/*"  
        },  
        {  
            "Sid": "112",  
            "Effect": "Deny",  
            "Principal": {  
                "AWS": "arn:aws:iam::AccountB-ID:user/AccountBadmin"  
            },  
            "Action": "s3:PutObject",  
            "Resource": "arn:aws:s3:::examplebucket/*",  
            "Condition": {  
                "StringNotEquals": {  
                    "s3:x-amz-grant-full-control": "id=CanonicalUserId-of-  
                    AWSaccountA-BucketOwner"  
                }  
            }  
        }  
    ]  
}
```

Step 1.3: Create an IAM Role to Allow Account C Cross-Account Access in Account A

In the IAM console, create an IAM role ("examplerole") that grants Account C permission to assume the role. Make sure you are still signed in as the Account A administrator because the role must be created in Account A.

1. Before creating the role, prepare the managed policy that defines the permissions that the role requires. You attach this policy to the role in a later step.
 - a. In the navigation pane on the left, click **Policies** and then click **Create Policy**.
 - b. Next to **Create Your Own Policy**, click **Select**.
 - c. Enter access-accountA-bucket in the **Policy Name** field.
 - d. Copy the following access policy and paste it into the **Policy Document** field. The access policy grants the role s3:GetObject permission so when Account C user assumes the role, it can only perform the s3:GetObject operation.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "s3:GetObject",  
            "Resource": "arn:aws:s3:::examplebucket/*"  
        }  
    ]  
}
```

- e. Click **Create Policy**.

The new policy appears in the list of managed policies.

2. In the navigation pane on the left, click **Roles** and then click **Create New Role**.
3. Enter exempleroles for the role name, and then click **Next Step**.
4. Under **Select Role Type**, select **Role for Cross-Account Access**, and then click the **Select** button next to **Provide access between AWS accounts you own**.
5. Enter the Account C account ID.

For this walkthrough you do not need to require users to have multi-factor authentication (MFA) to assume the role, so leave that option unselected.

6. Click **Next Step** to set the permissions that will be associated with the role.
7. Select the box next to the access-accountA-bucket policy that you created and then click **Next Step**.

The Review page appears so you can confirm the settings for the role before it's created. One very important item to note on this page is the link that you can send to your users who need to use this role. Users who click the link go straight to the Switch Role page with the Account ID and Role Name fields already filled in. You can also see this link later on the Role Summary page for any cross-account role.

8. After reviewing the role, click **Create Role**.

The exempleroles role is displayed in the list of roles.

9. Click the role name exempleroles.
10. Scroll down to the **Trust Relationships** section and expand the section.
11. Click **Show policy document** and verify the trust policy shown matches the following policy.

The following trust policy establishes trust with Account C, by allowing it the `sts:AssumeRole` action. For more information, go to [AssumeRole](#) in the *AWS Security Token Service API Reference*.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "arn:aws:iam::AccountC-ID:root"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

12. Note down the Amazon Resource Name (ARN) of the `examplerole` role you created.

Later in the following steps, you attach a user policy to allow an IAM user to assume this role, and you identify the role by the ARN value.

Step 2: Do the Account B Tasks

The `examplebucket` owned by Account A needs objects owned by other accounts. In this step, the Account B administrator uploads an object using the command line tools.

- Using the `put-object` AWS CLI command, upload an object to the `examplebucket`.

```
aws s3api put-object --bucket examplebucket --key HappyFace.jpg --body  
HappyFace.jpg --grant-full-control id="canonicalUserId-ofTheBucketOwner" -  
-profile AccountBadmin
```

Note the following:

- The `--Profile` parameter specifies `AccountBadmin` profile, so the object is owned by Account B.
- The parameter `grant-full-control` grants the bucket owner full-control permission on the object as required by the bucket policy.
- The `--body` parameter identifies the source file to upload. For example, if the file is on the C: drive of a Windows computer, you specify `c:\HappyFace.jpg`.

Step 3: Do the Account C Tasks

In the preceding steps, Account A has already created a role, `examplerole`, establishing trust with Account C. This allows users in Account C to access Account A. In this step, Account C administrator creates a user (Dave) and delegates him the `sts:AssumeRole` permission it received from Account A. This will allow Dave to assume the `examplerole` and temporarily gain access to Account A. The access policy that Account A attached to the role will limit what Dave can do when he accesses Account A—specifically, get objects in `examplebucket`.

Step 3.1: Create a User in Account C and Delegate Permission to Assume `examplerole`

1. Using the IAM user sign-in URL for Account C, first sign in to the AWS Management Console as AccountCadmin user.
2. In the IAM console, create a user Dave.

For instructions, go to [Adding an IAM User \(AWS Management Console\)](#) in *Using IAM*.
3. Note down the Dave credentials. Dave will need these credentials to assume the `examplerole` role.
4. Create an inline policy for the Dave IAM user to delegate the `sts:AssumeRole` permission to Dave on the `examplerole` role in account A.
 - a. In the navigation pane on the left, click **Users**.
 - b. Click the user name Dave.
 - c. Scroll down to the **Permissions** section, and then expand the **Inline Policies** section.
 - d. Click [click here \(or Create User Policy\)](#).
 - e. Click **Custom Policy**, and then click **Select**.
 - f. Enter a name for the policy in the **Policy Name** field.
 - g. Copy the following policy into the **Policy Document** field.

You will need to update the policy by providing the Account A ID.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": ["sts:AssumeRole"],  
            "Resource": "arn:aws:iam::AccountA-ID:role/examplerole"  
        }  
    ]  
}
```

- h. Click **Apply Policy**

5. Save Dave's credentials to the config file of the AWS CLI by adding another profile, AccountCDave.

```
[profile AccountCDave]  
aws_access_key_id = UserDaveAccessKeyId  
aws_secret_access_key = UserDaveSecretAccessKey  
region = us-west-2
```

Step 3.2: Assume Role (`examplerole`) and Access Objects

Now Dave can access objects in the bucket owned by Account A as follows:

- Dave first assumes the `examplerole` using his own credentials. This will return temporary credentials.
- Using the temporary credentials, Dave will then access objects in Account A's bucket.

1. At the command prompt, execute the following AWS CLI `assume-role` command using the AccountCDave profile.

You will need to update the ARN value in the command by providing the Account A ID where `examplerole` is defined.

```
aws sts assume-role --role-arn arn:aws:iam::accountA-ID:role/examplerole -  
-profile AccountCDave --role-session-name test
```

In response, AWS Security Token Service (STS) returns temporary security credentials (access key ID, secret access key, and a security token).

2. Save the temporary security credentials in the AWS CLI config file under the `TempCred` profile.

```
[profile TempCred]  
aws_access_key_id = temp-access-key-ID  
aws_secret_access_key = temp-secret-access-key  
aws_security_token = security-token  
region = us-west-2
```

3. At the command prompt, execute the following AWS CLI command to access objects using the temporary credentials. For example, the command specifies the head-object API to retrieve object metadata for the `HappyFace.jpg` object.

```
aws s3api get-object --bucket examplebucket --key HappyFace.jpg SaveFileAs.jpg  
--profile TempCred
```

Because the access policy attached to `examplerole` allows the actions, Amazon S3 processes the request. You can try any other action on any other object in the bucket.

If you try any other action—for example, `get-object-acl`—you will get permission denied because the role is not allowed that action.

```
aws s3api get-object-acl --bucket examplebucket --key HappyFace.jpg --profile  
TempCred
```

We used user Dave to assume the role and access the object using temporary credentials. It could also be an application in Account C that accesses objects in `examplebucket`. The application can obtain temporary security credentials, and Account C can delegate the application permission to assume `examplerole`.

Step 4: Clean Up

1. After you are done testing, you can do the following to clean up.
 - Sign in to the AWS Management Console ([AWS Management Console](#)) using account A credentials, and do the following:
 - In the Amazon S3 console, remove the bucket policy attached to `examplebucket`. In the bucket **Properties**, delete the policy in the **Permissions** section.
 - If the bucket is created for this exercise, in the Amazon S3 console, delete the objects and then delete the bucket.
 - In the IAM console, remove the `examplerole` you created in Account A.
 - In the IAM console, remove the `AccountAdmin` user.

2. Sign in to the AWS Management Console ([AWS Management Console](#)) using Account B credentials. In the IAM console, delete user AccountBadmin.
3. Sign in to the AWS Management Console ([AWS Management Console](#)) using Account C credentials. In the IAM console, delete user AccountCadmin and user Dave.

Related Resources

- [Creating a Role for Cross-Account Access \(AWS Management Console\)](#) in *Using IAM*.
- [Walkthrough: Cross-Account Access Using Roles](#) in *Using IAM*.
- [Managing Policies \(AWS Management Console\)](#) in *Using IAM*.

Using Bucket Policies and User Policies

Topics

- [Access Policy Language Overview \(p. 312\)](#)
- [Bucket Policy Examples \(p. 336\)](#)
- [User Policy Examples \(p. 342\)](#)

Bucket policy and user policy are two of the access policy options available for you to grant permission to your Amazon S3 resources. Both use JSON-based access policy language. The topics in this section describe the key policy language elements, with emphasis on Amazon S3–specific details, and provide example bucket and user policies.

Important

We recommend you first review the introductory topics that explain the basic concepts and options available for you to manage access to your Amazon S3 resources. For more information, see [Introduction to Managing Access Permissions to Your Amazon S3 Resources \(p. 269\)](#).

Access Policy Language Overview

The topics in this section describe the basic elements used in bucket and user policies as used in Amazon S3. For complete policy language information, see the [Overview of AWS IAM Policies](#) and the [AWS IAM Policy Reference](#) topics in the *Using IAM*.

Note

Bucket policies are limited to 20 KB in size.

Common Elements in an Access Policy

In its most basic sense, a policy contains the following elements:

- **Resources** – Buckets and objects are the Amazon S3 resources for which you can allow or deny permissions. In a policy, you use the Amazon Resource Name (ARN) to identify the resource.
- **Actions** – For each resource, Amazon S3 supports a set of operations. You identify resource operations you will allow (or deny) by using action keywords (see [Specifying Permissions in a Policy \(p. 315\)](#)).

For example, the `s3>ListBucket` permission will allow the user permission to the Amazon S3 [GET Bucket \(List Objects\)](#) operation.

- **Effect** – What the effect will be when the user requests the specific action—this can be either allow or deny.

If you do not explicitly grant access to (allow) a resource, access is implicitly denied. You can also explicitly deny access to a resource, which you might do in order to make sure that a user cannot access it, even if a different policy grants access.

- **Principal** – The account or user who is allowed access to the actions and resources in the statement. You specify a principal only in a bucket policy. It is the user, account, service, or other entity who is the recipient of this permission. In a user policy, the user to which the policy is attached is the implicit principal.

The following example bucket policy shows the preceding common policy elements. The policy allows Dave, a user in account `Account-ID`, `s3:GetBucketLocation`, `s3>ListBucket` and `s3:GetObject` Amazon S3 permissions on the `examplebucket` bucket.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "ExampleStatement1",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "arn:aws:iam::Account-ID:user/Dave"  
            },  
            "Action": [  
                "s3:GetBucketLocation",  
                "s3>ListBucket",  
                "s3GetObject"  
            ],  
            "Resource": [  
                "arn:aws:s3:::examplebucket"  
            ]  
        }  
    ]  
}
```

Because this is a bucket policy, it includes the `Principal` element, which specifies who gets the permission.

For more information about the access policy elements, see the following topics:

- [Specifying Resources in a Policy \(p. 313\)](#)
- [Specifying a Principal in a Policy \(p. 314\)](#)
- [Specifying Permissions in a Policy \(p. 315\)](#)
- [Specifying Conditions in a Policy \(p. 319\)](#)

The following topics provide additional policy examples:

- [Bucket Policy Examples \(p. 336\)](#)
- [User Policy Examples \(p. 342\)](#)

Specifying Resources in a Policy

The following is the common Amazon Resource Name (ARN) format to identify any resources in AWS.

```
arn:partition:service:region:namespace:relative-id
```

For your Amazon S3 resources,

- "aws" is a common partition name. If your resources are in China (Beijing) region, "aws-cn" is the partition name.
- "s3" is the service.
- you don't specify region and namespace.
- For Amazon S3, it can be a bucket-name or a bucket-name/object-key. You can use wild card.

Then the ARN format for Amazon S3 resources reduces to:

```
arn:aws:s3:::bucket_name
arn:aws:s3:::bucket_name/key_name
```

The following are examples of Amazon S3 resource ARNs.

- This ARN identifies the `/developers/design_info.doc` object in the `examplebucket` bucket.

```
arn:aws:s3:::examplebucket/developers/design_info.doc
```

- You can use a wildcard '*' character in the relative-ID part of the ARN. The following ARN uses '*' to identify all objects in the `examplebucket` bucket.

```
arn:aws:s3:::examplebucket/*
```

The following ARN uses '*' to indicate all Amazon S3 resource (all bucket and objects in your account).

```
arn:aws:s3:::*
```

- You can use policy variables in Amazon S3 ARNs. At policy evaluation time, these predefined variables are replaced by their corresponding values. Suppose you organize your bucket as a collection of folders, one folder for each of your users. The folder name is the same as the user name. To grant users permission to their folders, you can specify a policy variable in the resource ARN:

```
arn:aws:s3:::bucket_name/developers/${aws:username}/
```

At run time, when the policy is evaluated, the variable `${aws:username}` in the resource ARN is substituted with the user name making the request.

For more information, see the following resources:

- [Resource in Using IAM](#)
- [Policy Variables in Using IAM](#).
- [ARNs in the AWS General Reference](#)

For more information about other access policy language elements, see [Access Policy Language Overview \(p. 312\)](#).

Specifying a Principal in a Policy

The `Principal` element specifies the user, account, service, or other entity that is allowed or denied access to a resource. The `Principal` element is relevant only in a bucket policy; you don't specify it in a user policy because you attach user policy directly to a specific user. The following are examples of specifying `Principal`. For more information, go to [Principal in Using IAM](#).

- To grant permissions to an AWS account, identify the account using the following format.

```
"AWS" : "account-ARN"
```

For example:

```
"Principal": { "AWS": "arn:aws:iam::AccountNumber-WithoutHyphens:root" }
```

Amazon S3 also supports canonical user ID, an obfuscated form of the AWS account ID. You can specify this ID using the following format

```
"CanonicalUser": "64-digit-alphanumeric-value"
```

For example:

```
"Principal": { "CanonicalUser": "64-digit-alphanumeric-value" }
```

To find the canonical user ID associated with your AWS account

1. Go to <http://aws.amazon.com> and from the **My Account/Console** drop-down menu, select **Security Credentials**.
 2. Sign in using appropriate account credentials.
 3. Click **Account Identifiers**.
- To grant permission to an IAM user within your account, you must provide a "AWS" : "user-ARN" name-value pair.

```
"Principal": { "AWS": "arn:aws:iam::account-number-without-hyphens:user/username" }
```

- To grant permission to everyone, also referred as anonymous access, you set the wildcard, "*", as the Principal value. For example, if you configure your bucket as a website, you want all the objects in the bucket to be publicly accessible.

```
"Principal" : "*"
```

- You can also grant a CloudFront Origin Access Identity using the Canonical User ID associated with that identity. To learn more about CloudFront's support for serving private content, go to [Serving Private Content](#) topic in the *Amazon CloudFront Developer Guide*. You must specify the Canonical User ID for your CloudFront distribution's origin identity, not your AWS account.

For more information about other access policy language elements, see [Access Policy Language Overview \(p. 312\)](#).

Specifying Permissions in a Policy

Amazon S3 defines a set of permissions that you can specify in a policy. These are keywords, each of which maps to specific Amazon S3 operations (see [Operations on Buckets](#), and [Operations on Objects](#) in the *Amazon Simple Storage Service API Reference*).

Amazon S3 Permissions for Object Operations

Permissions	Amazon S3 Operations
s3:GetObject	GET Object , HEAD Object , GET Object Torrent When you grant this permission on a version-enabled bucket, you always get the latest version data.

Permissions	Amazon S3 Operations
s3:GetObjectVersion	GET Object , HEAD Object , GET Object Torrent To grant permission for version specific object data, you must grant this permission. That is, when you specify version number when making any of these requests, you need this Amazon S3 permission.
s3:PutObject	PUT Object , POST Object , Initiate Multipart Upload , Upload Part , Complete Multipart Upload PUT Object - Copy
s3:GetObjectAcl	GET Object ACL
s3:GetObjectVersionAcl	GET ACL (for a Specific Version of the Object)
s3:PutObjectAcl	PUT Object ACL
s3:PutObjectVersionAcl	PUT Object (for a Specific Version of the Object)
s3:DeleteObject	DELETE Object
s3:DeleteObjectVersion	DELETE Object (a Specific Version of the Object)
s3>ListMulti-partUploadParts	List Parts
s3:AbortMulti-partUpload	Abort Multipart Upload
s3:GetObjectTorrent	GET Object Torrent
s3:GetObjectVersionTorrent	GET Object Torrent versioning
s3:RestoreObject	POST Object restore

The following example bucket policy grants the `s3:PutObject` and the `s3:PutObjectAcl` permissions to a user (Dave). If you remove the `Principal` element, you can attach the policy to a user. These are object operations, and accordingly the relative-id portion of the Resource ARN identifies objects (`examplebucket/*`). For more information, see [Specifying Resources in a Policy \(p. 313\)](#).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "statement1",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:User/Dave"
      },
      "Action": [
        "s3:PutObject",
        "s3:PutObjectAcl"
      ],
      "Resource": "arn:aws:s3:::examplebucket/*"
    }
  ]
}
```

```

        "AWS": "arn:aws:iam::AccountB-ID:user/Dave"
    },
    "Action": [ "s3:PutObject", "s3:PutObjectAcl" ],
    "Resource": "arn:aws:s3:::examplebucket/*"
}
]
}

```

You can use a wildcard to grant permission for all Amazon S3 actions.

```
"Action": "/*"
```

Amazon S3 Permissions Related to Bucket Operations

Permission Keywords	Amazon S3 Operation(s) Covered
s3:CreateBucket	PUT Bucket
s3:DeleteBucket	DELETE Bucket
s3>ListBucket	GET Bucket (List Objects), HEAD Bucket
s3>ListBucketVersions	GET Bucket Object versions
s3>ListAllMyBuckets	GET Service
s3>ListBucketMultipartUploads	List Multipart Uploads

The following example user policy grants the `s3:CreateBucket`, `s3>ListAllMyBuckets`, and the `s3:GetBucketLocation` permissions to a user. Note that for all these permissions, you set the relative-id part of the Resource ARN to `"*"`. For all other bucket actions, you must specify a bucket name. For more information, see [Specifying Resources in a Policy \(p. 313\)](#).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "statement1",
            "Effect": "Allow",
            "Action": [
                "s3:CreateBucket", "s3>ListAllMyBuckets", "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::/*"
            ]
        }
    ]
}
```

```
        ]  
    }
```

Note that, if your user is going to use the console to view buckets, and see content of any of these buckets, the console will need the user to have the `s3:ListAllMyBuckets` and `s3:GetBucketLocation` permissions. For an example walkthrough, see [An Example Walkthrough: Using user policies to control access to your bucket \(p. 347\)](#).

Amazon S3 Permissions Related to Bucket Subresource Operations

Permissions	Amazon S3 Operation(s) Covered
<code>s3:GetBucketAcl</code>	GET Bucket acl
<code>s3:PutBucketAcl</code>	PUT Bucket acl
<code>s3:GetBucketCORS</code>	GET Bucket cors
<code>s3:PutBucketCORS</code>	PUT Bucket cors
<code>s3:GetBucketVersioning</code>	GET Bucket versioning
<code>s3:PutBucketVersioning</code>	PUT Bucket versioning
<code>s3:GetBucketRequestPayment</code>	GET Bucket requestPayment
<code>s3:PutBucketRequestPayment</code>	PUT Bucket requestPayment
<code>s3:GetBucketLocation</code>	GET Bucket location
<code>s3:GetBucketPolicy</code>	GET Bucket policy
<code>s3:DeleteBucketPolicy</code>	DELETE Bucket policy
<code>s3:PutBucketPolicy</code>	PUT Bucket policy
<code>s3:GetBucketNotification</code>	GET Bucket notification
<code>s3:PutBucketNotification</code>	PUT Bucket notification
<code>s3:GetBucketLogging</code>	GET Bucket logging
<code>s3:PutBucketLogging</code>	PUT Bucket logging
<code>s3:GetBucketTagging</code>	GET Bucket tagging
<code>s3:PutBucketTagging</code>	PUT Bucket tagging
<code>s3:GetBucketWebsite</code>	GET Bucket website
<code>s3:PutBucketWebsite</code>	PUT Bucket website
<code>s3:DeleteBucketWebsite</code>	DELETE Bucket website
<code>s3:GetLifecycleConfiguration</code>	GET Bucket lifecycle
<code>s3:PutLifecycleConfiguration</code>	PUT Bucket lifecycle
<code>s3:PutReplicationConfiguration</code>	PUT Bucket replication
<code>s3:GetReplicationConfiguration</code>	GET Bucket replication

Permissions	Amazon S3 Operation(s) Covered
s3:DeleteReplicationConfiguration	DELETE Bucket replication

The following user policy grants the `s3:GetBucketAcl` permission on the `examplebucket` bucket to user Dave.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "statement1",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::Account-ID:user/Dave"
            },
            "Action": [ "s3:GetObjectVersion", "s3:GetBucketAcl" ],
            "Resource": "arn:aws:s3:::examplebucket"
        }
    ]
}
```

You can delete objects either by explicitly calling the `DELETE` Object API or by configuring its lifecycle (see [Object Lifecycle Management \(p. 86\)](#)) so that Amazon S3 can remove the objects when their lifetime expires. To explicitly block users or accounts from deleting objects, you must explicitly deny them `s3:DeleteObject`, `s3:DeleteObjectVersion`, and `s3:PutLifecycleConfiguration` permissions. Note that, by default, users have no permissions. But as you create users, add users to groups, and grant them permissions, it is possible for users to get certain permissions that you did not intend to give. That is where you can use explicit deny, which supersedes all other permissions a user might have and denies the user permissions for specific actions.

Specifying Conditions in a Policy

The access policy language allows you to specify conditions when granting permissions. The `Condition` element (or `Condition` block) lets you specify conditions for when a policy is in effect. In the `Condition` element, which is optional, you build expressions in which you use Boolean operators (equal, less than, etc.) to match your condition against values in the request. For example, when granting a user permission to upload an object, the bucket owner can require the object be publicly readable by adding the `StringEquals` condition as shown here:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "statement1",
            "Effect": "Allow",
            "Action": [ "s3:PutObject" ],
            "Resource": [ "arn:aws:s3:::examplebucket/*" ],
            "Condition": {
                "StringEquals": {
                    "s3:x-amz-acl": [ "public-read" ]
                }
            }
        }
    ]
}
```

```
    ]  
}
```

The Condition block specifies the StringEquals condition that is applied to the specified key-value pair, "s3:x-amz-acl": ["public-read"]. There is a set of predefined keys you can use in expressing a condition. The example uses the s3:x-amz-acl condition key. This condition requires user to include the x-amz-acl header with value public-read in every PUT object request.

For more information about specifying conditions in an access policy language, go to [Condition in Using IAM](#). This section describes Amazon S3-specific condition keys and provides example policies.

Available Condition Keys

The predefined keys available for specifying conditions in an Amazon S3 access policy can be classified as follows:

- **AWS-wide keys** – AWS provides a set of common keys that are supported by all AWS services that support policies. For a list of these keys, go to [Available Keys](#) in [Using IAM](#).

For example, the following bucket policy allows authenticated users the s3:GetObject action if the request originates from a specific range of IP addresses (192.168.143.*) unless the IP address is 192.168.143.188. In the condition block, the IpAddress and the NotIpAddress are conditions, each condition is provided a key-value pair for evaluation. Both the key-value pairs in this example use the aws:SourceIp key, which is an AWS-wide key as indicated by the prefix "aws" (Amazon S3-specific keys will have the prefix "s3:").

Note

The IPAddress and NotIpAddress key values specified in the condition uses CIDR notation as described in RFC 4632. For more information, go to <http://www.rfc-editor.org/rfc/rfc4632.txt>.

```
{  
    "Version": "2012-10-17",  
    "Id": "S3PolicyId1",  
    "Statement": [  
        {  
            "Sid": "statement1",  
            "Effect": "Allow",  
            "Principal": "*",  
            "Action": ["s3:GetObject"],  
            "Resource": "arn:aws:s3:::examplebucket/*",  
            "Condition": {  
                "IpAddress": {  
                    "aws:SourceIp": "192.168.143.0/24"  
                },  
                "NotIpAddress": {  
                    "aws:SourceIp": "192.168.143.188/32"  
                }  
            }  
        }  
    ]  
}
```

- **Amazon S3-specific keys** – Amazon S3 also defines a set of keys that are applicable only in the context of granting Amazon S3-specific permissions. All these keys have the prefix "s3:".

For example, the following bucket policy allows the s3:PutObject permission for two AWS accounts if the request includes the x-amz-acl header making the object publicly readable.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AddCannedAcl",
            "Effect": "Allow",
            "Principal": {
                "AWS": [ "arn:aws:iam::account1-ID:root", "arn:aws:iam::account2-ID:root" ]
            },
            "Action": [ "s3:PutObject" ],
            "Resource": [ "arn:aws:s3:::examplebucket/*" ],
            "Condition": {
                "StringEquals": {
                    "s3:x-amz-acl": [ "public-read" ]
                }
            }
        }
    ]
}
```

The Condition block uses the StringEquals condition and it is provided a key-value pair, "s3:x-amz-acl": ["public-read"], for evaluation. In the key-value pair, the s3:x-amz-acl is an Amazon S3-specific key, as indicated by the prefix "s3:".

Important

Not all conditions make sense for all actions. For example, it makes sense to include an s3:LocationConstraint condition on a policy that grants the s3:CreateBucket Amazon S3 permission, but not for the s3:GetObject permission. Amazon S3 can test for semantic errors of this type that involve Amazon S3-specific conditions. However, if you are creating a policy for an IAM user and you include a semantically invalid Amazon S3 condition, no error is reported, because IAM cannot validate Amazon S3 conditions.

The following section describes the condition keys that can be used to grant conditional permission for bucket and object operations. In addition, there are condition keys related to Amazon S3 Signature Version 4 authentication. For more information, go to [Amazon S3 Signature Version 4 Authentication Specific Policy Keys](#) in the *Amazon Simple Storage Service API Reference*.

Amazon S3 Condition Keys for Object Operations

The following table shows which Amazon S3 conditions you can use with which Amazon S3 actions. Example policies are provided following the table. Note the following about the Amazon S3-specific condition keys described in the following table.

- The condition key names are preceded by the prefix "s3:". For example, s3:x-amz-acl
- Each condition key maps to the same name request header allowed by the API on which the condition can be set. That is, these condition keys dictate behavior of the same name request headers. For example:
 - The condition key s3:x-amz-acl that you can use to grant condition permission for the s3:PutObject permission defines behavior of the x-amz-acl request header that the PUT Object API supports.
 - The condition key s3:VersionId that you can use to grant conditional permission for the s3:GetObjectVersion permission defines behavior of the versionId query parameter that you set in a GET Object request.

Permission	Applicable Condition Keys (or keywords)	Description
s3:PutObject	<ul style="list-style-type: none"> • s3:x-amz-acl (for canned ACL permissions) • s3:x-amz-grant-permission (for explicit permissions), where <i>permission</i> can be: read, write, read-acp, write-acp, full-control 	<p>The PUT Object operation allows access control list (ACL)-specific headers that you can use to grant ACL-based permissions. Using these keys, the bucket owner can set a condition to require specific access permissions when the user uploads an object.</p> <p>For an example policy, see Example 1: Granting s3:PutObject permission with a condition requiring the bucket owner to get full control (p. 324)</p> <p>For more information about ACLs, see Access Control List (ACL) Overview (p. 363).</p>
	s3:x-amz-copy-source	<p>To copy an object you use the PUT Object API (see PUT Object) and specify the source using the x-amz-copy-source header. Using this key, the bucket owner can restrict the copy source to a specific bucket, a specific folder in the bucket, or a specific object in a bucket.</p> <p>For a policy example, see Example 3: Granting s3:PutObject permission to copy objects with a restriction on the copy source (p. 327).</p>
	s3:x-amz-server-side-encryption	<p>When you upload an object, you can use the x-amz-server-side-encryption header to request Amazon S3 to save the object encrypted, using an envelope encryption key managed either by AWS Key Management Service (KMS) or by Amazon S3 (see Protecting Data Using Server-Side Encryption (p. 381)).</p> <p>When granting the s3:PutObject permission, the bucket owner can add a condition using this key to require the user to specify this header in the request. A bucket owner can grant such conditional permission to ensure that objects the user uploads are saved encrypted.</p> <p>For a policy example, see Example 1: Granting s3:PutObject permission with a condition requiring the bucket owner to get full control (p. 324)</p>
	s3:x-amz-metadata-directive	

Permission	Applicable Condition Keys (or keywords)	Description
		<p>When you copy an object using the PUT Object API (see PUT Object) you can optionally add the <code>x-amz-metadata-directive</code> header to specify whether you want the object metadata copied from the source object or replaced with metadata provided in the request.</p> <p>Using this key bucket, an owner can add a condition to enforce certain behavior when objects are uploaded.</p> <p>Valid values: COPY REPLACE. The default is COPY.</p>
<code>s3:PutObjectAcl</code>	<ul style="list-style-type: none"> • <code>s3:x-amz-acl</code> (for canned ACL permissions) • <code>s3:x-amz-grant-permission</code> (for explicit permissions), where <code>permission</code> can be: <code>read</code>, <code>write</code>, <code>read-acp</code>, <code>write-acp</code>, <code>grant-full-control</code> 	<p>The PUT Object acl API (see PUT Object acl) sets the access control list (ACL) on the specified object. The operation supports ACL-related headers. When granting this permission, the bucket owner can add conditions using these keys to require certain permissions. For more information about ACLs, see Access Control List (ACL) Overview (p. 363).</p> <p>For example, the bucket owner may want to retain control of the object regardless of who owns the object. To accomplish this, the bucket owner can add a condition using one of these keys to require the user to include specific permissions to the bucket owner.</p>
<code>s3:GetObjectVersion</code>	<code>s3:VersionId</code>	<p>This Amazon S3 permission enables the user to perform a set of Amazon S3 API operations (see Amazon S3 Permissions for Object Operations (p. 315)). For a version-enabled bucket, you can specify the object version to retrieve data for.</p> <p>By adding a condition using this key, the bucket owner can restrict the user to accessing data only for a specific version of the object. For an example policy, see 3: Example 4: Granting access to a specific version of an object (p. 328).</p>
<code>s3:GetObjectVersionAcl</code>	<code>s3:VersionId</code>	<p>For a version-enabled bucket, this Amazon S3 permission allows a user to get the ACL for a specific version of the object.</p> <p>The bucket owner can add a condition using the key to restrict the user to a specific version of the object.</p>

Permission	Applicable Condition Keys (or keywords)	Description
s3:PutObjectVersionAcl	s3:VersionId <ul style="list-style-type: none"> • s3:x-amz-acl (for canned ACL permissions) • s3:x-amz-grant-permission (for explicit permissions), where <i>permission</i> can be: read, write, read-acp, write-acp, grant-full-control 	For a version-enabled bucket, you can specify the object version in the PUT Object acl request to set ACL on a specific object version. Using this condition, the bucket owner can restrict the user to setting an ACL only on a specific version of an object. For a version-enabled bucket, this Amazon S3 permission allows use to set an ACL on a specific version of the object. For a description of these condition keys, see the s3:PutObjectACL permission in this table.
s3>DeleteObjectVersion	s3:VersionId	For a version-enabled bucket, this Amazon S3 permission allows the user to delete a specific version of the object. The bucket owner can add a condition using this key to limit the user's ability to delete only a specific version of the object. For an example of using this condition key, see 3: Example 4: Granting access to a specific version of an object (p. 328) . The example is about granting the s3:GetObjectVersion action, but the policy shows the use of this condition key.

Example 1: Granting s3:PutObject permission with a condition requiring the bucket owner to get full control

Suppose Account A owns a bucket and the account administrator wants to grant Dave, a user in Account B, permissions to upload objects. By default, objects that Dave uploads are owned by Account B, and Account A has no permissions on these objects. Because the bucket owner is paying the bills, it wants full permissions on the objects that Dave uploads. The Account A administrator can accomplish this by granting the [s3:PutObject](#) permission to Dave, with a condition that the request include ACL-specific headers, that either grants full permission explicitly or uses a canned ACL (see [PUT Object](#)).

- Require the `x-amz-full-control` header in the request with full control permission to the bucket owner

The following bucket policy grants the [s3:PutObject](#) permission to user Dave with a condition using the `s3:x-amz-grant-full-control` condition key, which requires the request to include the `x-amz-full-control` header.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "statement1",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::AccountB-ID:user/Dave"
            },
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::examplebucket/*",
            "Condition": {
                "StringEquals": {
                    "s3:x-amz-grant-full-control": "id=AccountA-CanonicalUserID"
                }
            }
        }
    ]
}
```

Note

This example is about cross-account permission. However, if Dave, who is getting the permission, belongs to the AWS account that owns the bucket, then this conditional permission is not necessary, because the parent account to which Dave belongs owns objects the user uploads.

The preceding bucket policy grants conditional permission to user Dave in Account B. While this policy is in effect, it is possible for Dave to get the same permission without any condition via some other policy. For example, Dave can belong to a group and you grant the group s3:PutObject permission without any condition. To avoid such permission loopholes, you can write a stricter access policy by adding explicit deny. In this example, we explicitly deny user Dave upload permission if he does not include the necessary headers in the request granting full permissions to the bucket owner. Explicit deny always supersedes any other permission granted. The following is the revised access policy example.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "statement1",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::AccountB-ID:user/AccountBadmin"
            },
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::examplebucket/*",
            "Condition": {
                "StringEquals": {
                    "s3:x-amz-grant-full-control": "id=AccountA-CanonicalUserID"
                }
            }
        },
        {
            "Sid": "statement2",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::examplebucket/*"
        }
    ]
}
```

```

        "Principal": {
            "AWS": "arn:aws:iam::AccountB-ID:user/AccountBadmin"
        },
        "Action": "s3:PutObject",
        "Resource": "arn:aws:s3:::examplebucket/*",
        "Condition": {
            "StringNotEquals": {
                "s3:x-amz-grant-full-control": "id=AccountA-Canonical
alUserID"
            }
        }
    }
}

```

If you have two AWS accounts, you can test the policy using the AWS CLI. You attach the policy and, using Dave's credentials, test the permission using the following AWS CLI `put-object` command. You provide Dave's credentials by adding the `--profile` parameter. You grant full control permission to the bucket owner by adding the `--grant-full-control` parameter. For more information about setting up and using the AWS CLI, see [Setting Up the Tools for the Example Walkthroughs \(p. 284\)](#).

```
aws s3api put-object --bucket examplebucket --key HappyFace.jpg --body
c:\HappyFace.jpg --grant-full-control id="AccountA-CanonicalUserID" --profile
AccountBUserProfile
```

- Require the `x-amz-acl` header with a canned ACL granting full control permission to the bucket owner.

To require the `x-amz-acl` header in the request, you can replace the key-value pair in the `Condition` block and specify the `s3:x-amz-acl` condition key as shown below.

```

"Condition": {
    "StringNotEquals": {
        "s3:x-amz-acl": "bucket-owner-full-control"
    }
}
```

To test the permission using the AWS CLI, you specify the `--acl` parameter. The AWS CLI then adds the `x-amz-acl` header when it sends the request.

```
aws s3api put-object --bucket examplebucket --key HappyFace.jpg --body
c:\HappyFace.jpg --acl "bucket-owner-full-control" --profile AccountBadmin
```

[Example 2: Granting s3:PutObject permission requiring objects stored using server-side encryption](#)

Suppose Account A owns a bucket and the account administrator wants to grant Jane, a user in Account A, permission to upload objects with a condition that Jane always request server-side encryption so that Amazon S3 saves objects encrypted. The Account A administrator can accomplish using the `s3:x-amz-server-side-encryption` condition key as shown. The key-value pair in the `Condition` block specifies the `s3:x-amz-server-side-encryption` key.

```

"Condition": {
    "StringNotEquals": {
```

```
    "s3:x-amz-server-side-encryption": "AES256"
}
```

When testing the permission using AWS CLI, you will need to add the required parameter using the `--server-side-encryption` parameter.

```
aws s3api put-object --bucket examplebucket --key HappyFace.jpg --body c:\HappyFace.jpg --server-side-encryption "AES256" --profile AccountnBadmin
```

Example 3: Granting s3:PutObject permission to copy objects with a restriction on the copy source

In the PUT Object request, when you specify a source object, it is a copy operation (see [PUT Object - Copy](#)). Accordingly, the bucket owner can grant a user permission to copy objects with restrictions on the source. For example:

- allow copying objects only from the `sourcebucket` bucket.
- allow copying objects from the `sourcebucket` bucket, and only the objects whose key name prefix start with `public/f`. For example, `sourcebucket/public/*`
- allow copying only a specific object from the `sourcebucket`. For example, `sourcebucket/example.jpg`.

The following bucket policy grants user Dave `s3:PutObject` permission that allows him to copy only objects with a condition that the request include the `s3:x-amz-copy-source` header and the header value specify the `/examplebucket/public/*` key name prefix.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "cross-account permission to user in your own account",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::AccountA-ID:user/Dave"
            },
            "Action": [ "s3:PutObject" ],
            "Resource": "arn:aws:s3:::examplebucket/*"
        },
        {
            "Sid": "Deny your user permission to upload object if copy source is not /bucket/folder",
            "Effect": "Deny",
            "Principal": {
                "AWS": "arn:aws:iam::AccountA-ID:user/Dave"
            },
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::examplebucket/*",
            "Condition": {
                "StringNotLike": {
                    "s3:x-amz-copy-source": "examplebucket/public/*"
                }
            }
        }
    ]
}
```

```
        ]  
    }
```

You can test the permission using the AWS CLI `copy-object` command. You specify the source by adding the `--copy-source` parameter, the key name prefix must match that the prefix allowed in the policy. You will need to provide user Dave credentials using the `--profile` parameter. For more information about setting up AWS CLI, see [Setting Up the Tools for the Example Walkthroughs \(p. 284\)](#).

```
aws s3api copy-object --bucket examplebucket --key HappyFace.jpg  
--copy-source examplebucket/public/PublicHappyFace1.jpg --profile AccountADave
```

Note that the preceding policy uses the `StringNotLike` condition. To grant permission to copy only a specific object you will need to change the condition from `StringNotLike` to `StringNotEquals` and then specify the exact object key as shown.

```
"Condition": {  
    "StringNotEquals": {  
        "s3:x-amz-copy-source": "examplebucket/public/PublicHappyFace1.jpg"  
    }  
}
```

3: Example 4: Granting access to a specific version of an object

Suppose Account A owns a version-enabled bucket. The bucket has several versions of the `HappyFace.jpg` object. The account administrator now wants to grant its user (Dave) permission to get only a specific version of the object. The account administrator can accomplish this by granting Dave `s3:GetObjectVersion` permission conditionally as shown. The key-value pair in the `Condition` block specifies the `s3:VersionId` condition key.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "statement1",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "arn:aws:iam::AccountA-ID:user/Dave"  
            },  
            "Action": [ "s3:GetObjectVersion" ],  
            "Resource": "arn:aws:s3:::examplebucketversionenabled/HappyFace.jpg"  
        },  
        {  
            "Sid": "statement2",  
            "Effect": "Deny",  
            "Principal": {  
                "AWS": "arn:aws:iam::AccountA-ID:user/Dave"  
            },  
            "Action": [ "s3:GetObjectVersion" ],  
            "Resource": "arn:aws:s3:::examplebucketversionenabled/HappyFace.jpg",  
            "Condition": {  
                "StringNotEquals": {  
                    "s3:VersionId": "AaaHbAQitwil_h47_441R02DDfLlB05e"  
                }  
            }  
        }  
    ]  
}
```

```
        }
    }
}
```

In this case, Dave will need to know the exact object version ID to retrieve the object.

You can test the permissions using the AWS CLI `get-object` command with the `--version-id` parameter identifying the specific object version. The command retrieves the object and saves it to the `OutputFile.jpg` file.

```
aws s3api get-object --bucket examplebucketversionenabled --key HappyFace.jpg
OutputFile.jpg --version-id AaaHbAQitwiL_h47_44lRO2DDfL1B05e --profile AccountADave
```

Amazon S3 Condition Keys for Bucket Operations

The following table shows list of bucket operation-specific permissions you can grant in policies, and for each of the permissions, the available keys you can use in specifying a condition.

Permission	Applicable Condition Keys	Description
s3:CreateBucket	<ul style="list-style-type: none"><code>s3:x-amz-acl</code> (for canned ACL permissions)<code>s3:x-amz-grant-permission</code> (for explicit permissions), where <code>permission</code> can be: <code>read</code>, <code>write</code>, <code>read-acp</code>, <code>write-acp</code>, <code>full-control</code>	The Create Bucket API (see PUT Bucket) supports ACL-specific headers. Using these condition keys, you can require a user to set these headers in the request granting specific permissions.
	<code>s3:LocationConstraint</code>	Using this condition key, you can restrict user to create bucket in a specific region. For a policy example, see Example 1: Allow a user to create a bucket but only in a specific region (p. 332) .

Permission	Applicable Condition Keys	Description
s3>ListBucket	s3:prefix	<p>Using this condition key, you can limit the response of the Get Bucket (List Objects) API (see GET Bucket (List Objects)) to key names with specific prefix.</p> <p>The Get Bucket (List Objects) API returns list of object keys in the specified bucket. This API supports the <code>prefix</code> header to retrieve only the object keys with a specific prefix. This condition key relates to the <code>prefix</code> header.</p> <p>For example, the Amazon S3 console supports the folder concept using key name prefixes. So if you have two objects with key names <code>public/object1.jpg</code> and <code>public/object2.jpg</code>, the console shows the objects under the <code>public</code> folder. If you organize your object keys using such prefixes, you can grant <code>s3>ListBucket</code> permission with the condition that will allow the user to get a list of key names with a specific prefix.</p> <p>For a policy example, see Example 2: Allow a user to get a list of objects in a bucket according to a specific prefix (p. 334).</p>
	s3:delimiter	If you organize your object key names using prefixes and delimiters, you can use this condition key to require the user to specify the <code>delimiter</code> parameter in the Get Bucket (List Objects) request. In this case, the response Amazon S3 returns is a list of object keys with common prefixes grouped together. For an example of using prefixes and delimiters, go to Get Bucket (List Objects) .
	s3:max-keys	<p>Using this condition, you can limit the number of keys Amazon S3 returns in response to the Get Bucket (List Objects) request by requiring the user to specify the <code>max-keys</code> parameter. By default the API returns up to 1000 key names.</p> <p>For a list of numeric conditions you can use, go to Numeric Conditions in <i>Using IAM</i>.</p>

Permission	Applicable Condition Keys	Description
s3>ListBucketVersions	s3:prefix	<p>If your bucket is version-enabled, you can use the GET Bucket Object versions API (see GET Bucket Object versions) to retrieve metadata of all of the versions of objects. For this API, the bucket owner must grant the s3>ListBucketVersions permission in the policy.</p> <p>Using this condition key, you can limit the response of the API to key names with a specific prefix by requiring the user to specify the <code>prefix</code> parameter in the request with a specific value.</p> <p>For example, the Amazon S3 console supports the folder concept of using key name prefixes. If you have two objects with key names <code>public/object1.jpg</code> and <code>public/object2.jpg</code>, the console shows the objects under the <code>public</code> folder. If you organize your object keys using such prefixes, you can grant s3>ListBucket permission with the condition that will allow a use to get a list of key names with a specific prefix.</p> <p>For a policy example, see Example 2: Allow a user to get a list of objects in a bucket according to a specific prefix (p. 334).</p>
	s3:delimiter	If you organize your object key names using prefixes and delimiters, you can use this condition key to require the user to specify the <code>delimiter</code> parameter in the GET Bucket Object versions request. In this case, the response Amazon S3 returns is a list of object keys with common prefixes grouped together.
	s3:max-keys	Using this condition you can limit the number of keys Amazon S3 returns in response to the GET Bucket Object versions request by requiring the user to specify the <code>max-keys</code> parameter. By default, the API returns up to 1000 key names. For a list of numeric conditions you can use, go to Numeric Conditions in Using IAM .

Permission	Applicable Condition Keys	Description
s3:PutBucketAcl	<ul style="list-style-type: none"> • s3:x-amz-acl (for canned ACL permissions) • s3:x-amz-grant-permission (for explicit permissions), where <i>permission</i> can be: read, write, read-acp, write-acp, full-control 	The PUT Bucket acl API (see PUT Bucket) supports ACL-specific headers. You can use these condition keys to require a user to set these headers in the request.

Example 1: Allow a user to create a bucket but only in a specific region

Suppose an AWS account administrator wants to grant its user (Dave), permission to create a bucket in the South America (Sao Paulo) region only. The account administrator can attach the following user policy granting the s3:CreateBucket permission with a condition as shown. The key-value pair in the Condition block specifies the s3:LocationConstraint key and the sa-east-1 region as its value.

Note

In this example, the bucket owner is granting permission to one of its users, so either a bucket policy or a user policy can be used. This example shows a user policy.

For a list of Amazon S3 regions, go to [Regions and Endpoints](#) in the *Amazon Web Services General Reference*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "statement1",
            "Effect": "Allow",
            "Action": [
                "s3:CreateBucket"
            ],
            "Resource": [
                "arn:aws:s3:::*"
            ],
            "Condition": {
                "StringLike": {
                    "s3:LocationConstraint": "sa-east-1"
                }
            }
        }
    ]
}
```

This policy restricts the user from creating a bucket in any other region except sa-east-1. However, it is possible some other policy will grant this user permission to create buckets in another region. For example, if the user belongs to a group, the group may have a policy attached to it allowing all users in the group permission to create buckets in some other region. To ensure the user does not get permission to create buckets in any other region, you can add an explicit deny statement in this policy.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "statement1",  
            "Effect": "Allow",  
            "Action": [  
                "s3:CreateBucket"  
            ],  
            "Resource": [  
                "arn:aws:s3:::*"  
            ],  
            "Condition": {  
                "StringLike": {  
                    "s3:LocationConstraint": "sa-east-1"  
                }  
            }  
        },  
        {  
            "Sid": "statement2",  
            "Effect": "Deny",  
            "Action": [  
                "s3:CreateBucket"  
            ],  
            "Resource": [  
                "arn:aws:s3:::*"  
            ],  
            "Condition": {  
                "StringNotLike": {  
                    "s3:LocationConstraint": "sa-east-1"  
                }  
            }  
        }  
    ]  
}
```

The Deny statement uses the `StringNotLike` condition. That is, a create bucket request will be denied if the location constraint is not "sa-east-1". The explicit deny will not allow the user to create a bucket in any other region, no matter what other permission the user gets.

You can test the policy using the following `create-bucket` AWS CLI command. This example uses the `bucketconfig.txt` file to specify the location constraint. Note the Windows file path. You will need to update the bucket name and path as appropriate. You must provide user credentials using the `--profile` parameter. For more information about setting up and using the AWS CLI, see [Setting Up the Tools for the Example Walkthroughs \(p. 284\)](#).

```
aws s3api create-bucket --bucket examplebucket --profile AccountADave --create-bucket-configuration file://c:/Users/someUser/bucketconfig.txt
```

The `bucketconfig.txt` file specifies the configuration as follows

```
{"LocationConstraint": "sa-east-1"}
```

Example 2: Allow a user to get a list of objects in a bucket according to a specific prefix

A bucket owner can restrict a user to list content of a specific folder in the bucket. This is useful if objects in the bucket are organized by key name prefixes, the Amazon S3 console then uses the prefixes to show a folder hierarchy (only the console supports the concept of folders; the Amazon S3 API supports only buckets and objects).

In this example, the bucket owner and the parent account to which the user belongs are the same. So the bucket owner can use either a bucket policy or a user policy. First, we show a user policy.

The following user policy grants the `s3>ListBucket` permission (see [GET Bucket \(List Objects\)](#)) with a condition that requires the user to specify the prefix in the request with the value `projects`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "statement1",
            "Effect": "Allow",
            "Action": [
                "s3>ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::examplebucket"
            ],
            "Condition": {
                "StringEquals": {
                    "s3:prefix": "projects"
                }
            }
        },
        {
            "Sid": "statement2",
            "Effect": "Deny",
            "Action": [
                "s3>ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::examplebucket"
            ],
            "Condition": {
                "StringNotEquals": {
                    "s3:prefix": "projects"
                }
            }
        }
    ]
}
```

The condition restricts the user to listing object keys with the `projects` prefix. The added explicit deny will deny user request for listing keys with any other prefix no matter what other permissions the user might have. For example, it is possible that the user gets permission to list object keys without any restriction, for example either by updates to the preceding user policy or via a bucket policy. But because explicit deny always supersedes, the user request to list keys other than the `project` prefix will be denied.

The preceding policy is a user policy. If you add the `Principal` element to the policy, identifying the user, you now have a bucket policy as shown.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "statement1",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "arn:aws:iam::BucketOwner-accountID:user/user-name"  
            },  
            "Action": [  
                "s3>ListBucket"  
            ],  
            "Resource": [  
                "arn:aws:s3:::examplebucket"  
            ],  
            "Condition": {  
                "StringEquals": {  
                    "s3:prefix": "examplefolder"  
                }  
            }  
        },  
        {  
            "Sid": "statement2",  
            "Effect": "Deny",  
            "Principal": {  
                "AWS": "arn:aws:iam::BucketOwner-AccountID:user/user-name"  
            },  
            "Action": [  
                "s3>ListBucket"  
            ],  
            "Resource": [  
                "arn:aws:s3:::examplebucket"  
            ],  
            "Condition": {  
                "StringNotEquals": {  
                    "s3:prefix": "examplefolder"  
                }  
            }  
        }  
    ]  
}
```

You can test the policy using the following `list-object` AWS CLI command. In the command, you provide user credentials using the `--profile` parameter. For more information about setting up and using the AWS CLI, see [Setting Up the Tools for the Example Walkthroughs \(p. 284\)](#).

```
aws s3api list-objects --bucket examplebucket --prefix examplefolder --profile AccountADave
```

Now if the bucket is version-enabled, to list the objects in the bucket, instead of `s3>ListBucket` permission, you must grant the `s3>ListBucketVersions` permission in the preceding policy. This permission also supports the `s3:prefix` condition key.

Bucket Policy Examples

This section presents a few examples of typical use cases for bucket policies. The policies use "bucket" and "examplebucket" strings in the resource value. To test these policies, you need to replace these strings with your bucket name. For information about access policy language, see [Access Policy Language Overview \(p. 312\)](#).

You can use the [AWS Policy Generator](#) to create a bucket policy for your Amazon S3 bucket. You can then use the generated document to set your bucket policy by using the [Amazon S3 console](#), by a number of third-party tools, or via your application.

Note

When testing permissions using the Amazon S3 console, you will need to grant additional permissions—`s3>ListAllMyBuckets`, `s3:GetBucketLocation`, and `s3>ListBucket` permissions. These are the additional permissions the console requires. For an example walkthrough that grants permissions to users and tests them using the console, see [An Example Walkthrough: Using user policies to control access to your bucket \(p. 347\)](#).

Topics

- [Granting Permissions to Multiple Accounts with Added Conditions \(p. 336\)](#)
- [Granting Read-Only Permission to an Anonymous User \(p. 337\)](#)
- [Restricting Access to Specific IP Addresses \(p. 337\)](#)
- [Restricting Access to a Specific HTTP Referrer \(p. 338\)](#)
- [Granting Permission to an Amazon CloudFront Origin Identity \(p. 339\)](#)
- [Adding a Bucket Policy to Require MFA Authentication \(p. 339\)](#)
- [Granting Cross-Account Permissions to Upload Objects While Ensuring the Bucket Owner Has Full Control \(p. 341\)](#)

Granting Permissions to Multiple Accounts with Added Conditions

The following example policy grants the `s3:PutObject` and `s3:PutObjectAcl` permissions to multiple AWS accounts and requires that any request for these operations include the `public-read` canned ACL. For more information, see [Specifying Permissions in a Policy \(p. 315\)](#) and [Specifying Conditions in a Policy \(p. 319\)](#).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AddCannedAcl",  
            "Effect": "Allow",  
            "Principal": {"AWS":  
                ["arn:aws:iam::111122223333:root", "arn:aws:iam::444455556666:root"]},  
            "Action": ["s3:PutObject", "s3:PutObjectAcl"],  
            "Resource": ["arn:aws:s3:::examplebucket/*"],  
            "Condition": {"StringEquals": {"s3:x-amz-acl": ["public-read"]}}  
        }  
    ]  
}
```

Granting Read-Only Permission to an Anonymous User

The following example policy grants the `s3:GetObject` permission to any public anonymous users. (For a list of permissions and operations they allow, see [Specifying Permissions in a Policy \(p. 315\)](#).) This permission allows anyone to read the object data, which is useful for when you configure your bucket as a website and want everyone to be able to read objects in the bucket.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AddPerm",  
            "Effect": "Allow",  
            "Principal": "*",  
            "Action": ["s3:GetObject"],  
            "Resource": ["arn:aws:s3:::examplebucket/*"]  
        }  
    ]  
}
```

Restricting Access to Specific IP Addresses

The following example grants permissions to any user to perform any Amazon S3 operations on objects in the specified bucket. However, the request must originate from the range of IP addresses specified in the condition.

The condition in this statement identifies the 54.240.143.* range of allowed IP addresses, with one exception: 54.240.143.188.

The Condition block uses the `IpAddress` and `NotIpAddress` conditions and the `aws:SourceIp` condition key, which is an AWS-wide condition key. For more information, see [Specifying Conditions in a Policy \(p. 319\)](#). Also note that the `aws:sourceIp` values use the CIDR notation described in RFC 2632. For more information, go to <http://www.rfc-editor.org/rfc/rfc4632.txt>.

```
{  
    "Version": "2012-10-17",  
    "Id": "S3PolicyId1",  
    "Statement": [  
        {  
            "Sid": "IPAllow",  
            "Effect": "Allow",  
            "Principal": "*",  
            "Action": "s3:*",  
            "Resource": "arn:aws:s3:::examplebucket/*",  
            "Condition": {  
                "IpAddress": {"aws:SourceIp": "54.240.143.0/24"},  
                "NotIpAddress": {"aws:SourceIp": "54.240.143.188/32"}  
            }  
        }  
    ]  
}
```

Restricting Access to a Specific HTTP Referrer

Suppose you have a website with domain name (`www.example.com` or `example.com`) with links to photos and videos stored in your Amazon S3 bucket, `examplebucket`. By default, all the Amazon S3 resources are private, so only the AWS account that created the resources can access them. To allow read access to these objects from your website, you can add a bucket policy that allows `s3:GetObject` permission with a condition, using the `aws:referer` key, that the get request must originate from specific webpages. The following policy specifies the `StringLike` condition with the `aws:Referer` condition key.

```
{  
    "Version": "2012-10-17",  
    "Id": "http referer policy example",  
    "Statement": [  
        {  
            "Sid": "Allow get requests originated from www.example.com and example.com",  
            "Effect": "Allow",  
            "Principal": "*",  
            "Action": "s3:GetObject",  
            "Resource": "arn:aws:s3:::examplebucket/*",  
            "Condition": {  
                "StringLike": { "aws:Referer": [ "http://www.example.com/*", "http://example.com/*" ] }  
            }  
        }  
    ]  
}
```

Make sure the browsers you use include the `http referer` header in the request.

You can further secure access to objects in the `examplebucket` bucket by adding explicit deny to the bucket policy as shown in the following example. Explicit deny supersedes any permission you might grant to objects in the `examplebucket` bucket using other means such as ACLs or user policies.

```
{  
    "Version": "2012-10-17",  
    "Id": "http referer policy example",  
    "Statement": [  
        {  
            "Sid": "Allow get requests referred by www.mysite.com and mysite.com",  
            "Effect": "Allow",  
            "Principal": "*",  
            "Action": "s3:GetObject",  
            "Resource": "arn:aws:s3:::examplebucket/*",  
            "Condition": {  
                "StringLike": { "aws:Referer": [ "http://www.example.com/*", "http://example.com/*" ] }  
            }  
        },  
        {  
            "Sid": "Explicit deny to ensure requests are allowed only from specific  
            referer.",  
            "Effect": "Deny",  
            "Principal": "*",  
            "Action": "s3:*",  
            "Condition": {  
                "StringNotLike": { "aws:Referer": [ "http://www.mysite.com/*", "http://mysite.com/*" ] }  
            }  
        }  
    ]  
}
```

```
        "Resource": "arn:aws:s3:::examplebucket/*",
        "Condition": {
            "StringNotLike": {"aws:Referer": ["http://www.example.com/*", "http://example.com/*"]}
        }
    ]
}
```

Granting Permission to an Amazon CloudFront Origin Identity

The following example bucket policy grants a CloudFront Origin Identity permission to get (list) all objects in your Amazon S3 bucket. The CloudFront Origin Identity is used to enable CloudFront's private content feature. The policy uses the CanonicalUser prefix, instead of AWS, to specify a Canonical User ID. To learn more about CloudFront's support for serving private content, go to the [Serving Private Content](#) topic in the *Amazon CloudFront Developer Guide*. You must specify the canonical user ID for your CloudFront distribution's origin access identity. For instructions about finding the canonical user ID, see [Specifying a Principal in a Policy \(p. 314\)](#).

```
{
    "Version": "2012-10-17",
    "Id": "PolicyForCloudFrontPrivateContent",
    "Statement": [
        {
            "Sid": " Grant a CloudFront Origin Identity access to support private content",
            "Effect": "Allow",
            "Principal": {"CanonicalUser": "79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be"},
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::example-bucket/*"
        }
    ]
}
```

Adding a Bucket Policy to Require MFA Authentication

Amazon S3 supports MFA-protected API access, a feature that can enforce multi-factor authentication for access to your Amazon S3 resources. Multi-factor authentication provides an extra level of security you can apply to your AWS environment. It is a security feature that requires users to prove physical possession of an MFA device by providing a valid MFA code. For more information, go to [AWS Multi-Factor Authentication](#). You can require MFA authentication for any requests to access your Amazon S3 resources.

You can enforce the MFA authentication requirement using the `aws:MultiFactorAuthAge` key in a bucket policy. IAM users can access Amazon S3 resources by using temporary credentials issued by the AWS Security Token Service (STS). You provide the MFA code at the time of the STS request.

When Amazon S3 receives a request with MFA authentication, the `aws:MultiFactorAuthAge` key provides a numeric value indicating how long ago (in seconds) the temporary credential was created. If the temporary credential provided in the request was not created using an MFA device, this key value is null (absent). In a bucket policy, you can add a condition to check this value, as shown in the following example bucket policy. The policy denies any Amazon S3 operation on the `/taxdocuments` folder in the `examplebucket` bucket if the request is not MFA authenticated. To learn more about MFA authentication, go to [Using Multi-Factor Authentication \(MFA\) Devices with AWS](#).

```
{
    "Version": "2012-10-17",
    "Id": "123",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::examplebucket/taxdocuments/*",
            "Condition": { "Null": { "aws:MultiFactorAuthAge": true } }
        }
    ]
}
```

The Null condition in the Condition block evaluates to true if the `aws:MultiFactorAuthAge` key value is null, indicating that the temporary security credentials in the request were created without the MFA key.

The following bucket policy is an extension of the preceding bucket policy. It includes two policy statements. One statement allows the `s3:GetObject` permission on a bucket (`examplebucket`) to everyone and another statement further restricts access to the `examplebucket/taxdocuments` folder in the bucket by requiring MFA authentication.

```
{
    "Version": "2012-10-17",
    "Id": "123",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::examplebucket/taxdocuments/*",
            "Condition": { "Null": { "aws:MultiFactorAuthAge": true } }
        },
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": "*",
            "Action": ["s3:GetObject"],
            "Resource": "arn:aws:s3:::examplebucket/*"
        }
    ]
}
```

You can optionally use a numeric condition to limit the duration for which the `aws:MultiFactorAuthAge` key is valid, independent of the lifetime of the temporary security credential used in authenticating the request. For example, the following bucket policy, in addition to requiring MFA authentication, also checks how long ago the temporary session was created. The policy denies any operation if the `aws:MultiFactorAuthAge` key value indicates that the temporary session was created more than an hour ago (3,600 seconds).

```
{
    "Version": "2012-10-17",
    "Id": "123",
```

```

"Statement": [
    {
        "Sid": "",
        "Effect": "Deny",
        "Principal": "*",
        "Action": "s3:*",
        "Resource": "arn:aws:s3:::examplebucket/taxdocuments/*",
        "Condition": {"Null": {"aws:MultiFactorAuthAge": true}}
    },
    {
        "Sid": "",
        "Effect": "Deny",
        "Principal": "*",
        "Action": "s3:*",
        "Resource": "arn:aws:s3:::examplebucket/taxdocuments/*",
        "Condition": {"NumericGreaterThanOrEqual": {"aws:MultiFactorAuthAge": 3600}}
    },
    {
        "Sid": "",
        "Effect": "Allow",
        "Principal": "*",
        "Action": ["s3:GetObject"],
        "Resource": "arn:aws:s3:::examplebucket/*"
    }
]
}

```

Granting Cross-Account Permissions to Upload Objects While Ensuring the Bucket Owner Has Full Control

You can allow another AWS account to upload objects to your bucket. However, you may decide that as a bucket owner you must have full control of the objects uploaded to your bucket. The following policy enforces that a specific AWS account (111111111111) be denied the ability to upload objects unless that account grants full-control access to the bucket owner identified by the email address (xyz@amazon.com). The `StringEquals` condition in the policy specifies the `s3:x-amz-grant-full-control` condition key to express the requirement (see [Specifying Conditions in a Policy \(p. 319\)](#)).

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "111",
            "Effect": "Allow",
            "Principal": {"AWS": "111111111111"},
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::examplebucket/*"
        },
        {
            "Sid": "112",
            "Effect": "Deny",
            "Principal": {"AWS": "111111111111"},
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::examplebucket/*",
            "Condition": {
                "StringNotEquals": {"s3:x-amz-grant-full-control": ["emailAd"]

```

```
    dress=xyz@amazon.com" ] }  
    }  
}  
}
```

User Policy Examples

This section shows several IAM user policies for controlling user access to Amazon S3. For information about access policy language, see [Access Policy Language Overview \(p. 312\)](#).

The following example policies will work if you test them programmatically; however, in order to use them with the Amazon S3 console, you will need to grant additional permissions that are required by the console. For information about using policies such as these with the Amazon S3 console, see [An Example Walkthrough: Using user policies to control access to your bucket \(p. 347\)](#).

Topics

- [Example: Allow an IAM user access to one of your buckets \(p. 342\)](#)
- [Example: Allow each IAM user access to a folder in a bucket \(p. 343\)](#)
- [Example: Allow a group to have a shared folder in Amazon S3 \(p. 346\)](#)
- [Example: Allow all your users to read objects in a portion of the corporate bucket \(p. 346\)](#)
- [Example: Allow a partner to drop files into a specific portion of the corporate bucket \(p. 346\)](#)
- [An Example Walkthrough: Using user policies to control access to your bucket \(p. 347\)](#)

Example: Allow an IAM user access to one of your buckets

In this example, you want to grant an IAM user in your AWS account access to one of your buckets, `examplebucket`, and allow the user to add, update, and delete objects.

In addition to granting the `s3:PutObject`, `s3:GetObject`, and `s3:DeleteObject` permissions to the user, the policy also grants the `s3>ListAllMyBuckets`, `s3:GetBucketLocation`, and `s3>ListBucket` permissions. These are the additional permissions required by the console. For an example walkthrough that grants permissions to users and tests them using the console, see [An Example Walkthrough: Using user policies to control access to your bucket \(p. 347\)](#).

```
{  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3>ListAllMyBuckets"  
            ],  
            "Resource": "arn:aws:s3:::*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3>ListBucket",  
                "s3:GetBucketLocation"  
            ],  
            "Resource": "arn:aws:s3:::examplebucket"  
        },  
        {  
    ]  
}
```

```
        "Effect": "Allow",
        "Action": [
            "s3:PutObject",
            "s3:GetObject",
            "s3:DeleteObject"
        ],
        "Resource": "arn:aws:s3:::examplebucket/*"
    }
]
}
```

Example: Allow each IAM user access to a folder in a bucket

In this example, you want two IAM users, Alice and Bob, to have access to your bucket, examplebucket, so they can add, update, and delete objects. However, you want to restrict each user's access to a single folder in the bucket. You might create folders with names that match the user names.

```
examplebucket
Alice/
Bob/
```

To grant each user access only to his or her folder, you can write a policy for each user and attach it individually. For example, you can attach the following policy to user Alice to allow her specific Amazon S3 permissions on the examplebucket/Alice folder.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:GetObjectVersion",
                "s3:DeleteObject",
                "s3:DeleteObjectVersion"
            ],
            "Resource": "arn:aws:s3:::examplebucket/Alice/*"
        }
    ]
}
```

You then attach a similar policy to user Bob, identifying folder Bob in the Resource value.

Instead of attaching policies to individual users, though, you can write a single policy that uses a policy variable and attach the policy to a group. You will first need to create a group and add both Alice and Bob to the group. The following example policy allows a set of Amazon S3 permissions in the examplebucket/\${aws:username} folder. When the policy is evaluated, the policy variable \${aws:username} is replaced by the requester's user name. For example, if Alice sends a request to put an object, the operation is allowed only if Alice is uploading the object to the examplebucket/Alice folder.

```
{
    "Version": "2012-10-17",
```

```
"Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "s3:PutObject",
            "s3:GetObject",
            "s3:GetObjectVersion",
            "s3:DeleteObject",
            "s3:DeleteObjectVersion"
        ],
        "Resource": "arn:aws:s3:::examplebucket/${aws:username}/*"
    }
]
```

Note

When using policy variables you must explicitly specify version 2012-10-17 in the policy. The default version of the access policy language, 2008-10-17, does not support policy variables.

If you want to test the preceding policy on the Amazon S3 console, the console requires permission for additional Amazon S3 permissions, as shown in the following policy. For information about how the console uses these permissions, see [An Example Walkthrough: Using user policies to control access to your bucket \(p. 347\)](#).

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowGroupToSeeBucketListInTheConsole",
            "Action": [ "s3>ListAllMyBuckets", "s3:GetBucketLocation" ],
            "Effect": "Allow",
            "Resource": [ "arn:aws:s3:::*" ]
        },
        {
            "Sid": "AllowRootLevelListingOfTheBucket",
            "Action": [ "s3>ListBucket" ],
            "Effect": "Allow",
            "Resource": [ "arn:aws:s3:::examplebucket" ],
            "Condition": {
                "StringEquals": {
                    "s3:prefix": [ "" ], "s3:delimiter": [ "/" ]
                }
            }
        },
        {
            "Sid": "AllowListBucketOfASpecificUserPrefix",
            "Action": [ "s3>ListBucket" ],
            "Effect": "Allow",
            "Resource": [ "arn:aws:s3:::examplebucket" ],
            "Condition": { "StringLike": { "s3:prefix": [ "${aws:username}/*" ] } }
        },
        {
            "Sid": "AllowUserSpecificActionsOnlyInTheSpecificUserPrefix",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject"
            ]
        }
    ]
}
```

```
        "s3:GetObject",
        "s3:GetObjectVersion",
        "s3:DeleteObject",
        "s3:DeleteObjectVersion"
    ],
    "Resource": "arn:aws:s3:::examplebucket/${aws:username}/*"
}
]
```

Note

In the 2012-10-17 version of the policy, policy variables start with \$. This change in syntax can potentially create a conflict if your object key includes a \$. For example, to include an object key my\$file in a policy, you specify the \$ character with \${\$}, my\${\$}file.

Although IAM user names are friendly, human-readable identifiers, they are not required to be globally unique. For example, if user Bob leaves the organization and another Bob joins, then new Bob could access old Bob's information. Instead of using user names, you could create folders based on user IDs. Each user ID is unique. In this case, you will need to modify the preceding policy to use the \${aws:userid} policy variable. For more information about user identifiers, go to [AWS-Assigned User Identifiers](#) in *Using IAM*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:GetObjectVersion",
                "s3:DeleteObject",
                "s3:DeleteObjectVersion"
            ],
            "Resource": "arn:aws:s3:::my_corporate_bucket/home/${aws:userid}/*"
        }
    ]
}
```

Allow non-IAM users (mobile app users) access to folders in a bucket

Suppose you want to develop a mobile app, a game that stores users' data in an S3 bucket. For each app user, you want to create a folder in your bucket. You also want to limit each user's access to his or her own folder. But you cannot create folders before someone downloads your app and starts playing the game, because you don't have a user ID.

In this case, you can require users to sign in to your app by using public identity providers such as Login with Amazon, Facebook, or Google. After users have signed in to your app through one of these providers, they have a user ID that you can use to create user-specific folders at run time.

You can then use web identity federation in AWS Security Token Service to integrate information from the identity provider with your app and to get temporary security credentials for each user. You can then create IAM policies that allow the app to access your bucket and perform such operations as creating user-specific folders and uploading data. For more information about web identity federation, go to [Creating Temporary Security Credentials for Mobile Apps Using Identity Providers](#) in *Using Temporary Security Credentials*.

Example: Allow a group to have a shared folder in Amazon S3

Attaching the following policy to the group grants everybody in the group access to the following folder in Amazon S3: `my_corporate_bucket/share/marketing`. Group members are allowed to access only the specific Amazon S3 permissions shown in the policy and only for objects in the specified folder.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:PutObject",  
                "s3:GetObject",  
                "s3:GetObjectVersion",  
                "s3:DeleteObject",  
                "s3:DeleteObjectVersion"  
            ],  
            "Resource": "arn:aws:s3:::my_corporate_bucket/share/marketing/*"  
        }  
    ]  
}
```

Example: Allow all your users to read objects in a portion of the corporate bucket

In this example, we create a group called `AllUsers`, which contains all the IAM users that are owned by the AWS account. We then attach a policy that gives the group access to `GetObject` and `GetObjectVersion`, but only for objects in the `my_corporate_bucket/readonly` folder.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:GetObject",  
                "s3:GetObjectVersion"  
            ],  
            "Resource": "arn:aws:s3:::my_corporate_bucket/readonly/*"  
        }  
    ]  
}
```

Example: Allow a partner to drop files into a specific portion of the corporate bucket

In this example, we create a group called `widgetCo` that represents a partner company. We create an IAM user for the specific person or application at the partner company that needs access, and then we put the user in the group.

We then attach a policy that gives the group `PutObject` access to the following folder in the corporate bucket: `my_corporate_bucket/uploads/widgetco`.

We want to prevent the `WidgetCo` group from doing anything else with the bucket, so we add a statement that explicitly denies permission to any Amazon S3 permissions except `PutObject` on any Amazon S3 resource in the AWS account. This step is necessary only if there's a broad policy in use elsewhere in your AWS account that gives users wide access to Amazon S3 resources.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "s3:PutObject",  
            "Resource": "arn:aws:s3:::my_corporate_bucket/uploads/widgetco/*"  
        },  
        {  
            "Effect": "Deny",  
            "NotAction": "s3:PutObject",  
            "Resource": "arn:aws:s3:::my_corporate_bucket/uploads/widgetco/*"  
        },  
        {  
            "Effect": "Deny",  
            "Action": "s3:*",  
            "NotResource": "arn:aws:s3:::my_corporate_bucket/uploads/widgetco/*"  
        }  
    ]  
}
```

An Example Walkthrough: Using user policies to control access to your bucket

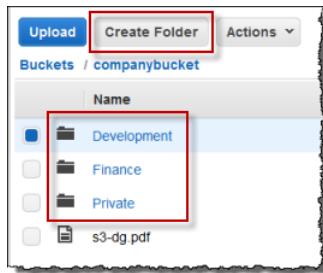
This walkthrough explains how user permissions work with Amazon S3. We will create a bucket with folders, and then we'll create AWS Identity and Access Management users in your AWS account and grant those users incremental permissions on your Amazon S3 bucket and the folders in it.

Topics

- [Background: Basics of Buckets and Folders \(p. 348\)](#)
- [Walkthrough Example \(p. 349\)](#)
- [Step 0: Preparing for the Walkthrough \(p. 349\)](#)
- [Step 1: Create a Bucket \(p. 350\)](#)
- [Step 2: Create IAM Users and a Group \(p. 351\)](#)
- [Step 3: Verify that IAM Users Have No Permissions \(p. 351\)](#)
- [Step 4: Grant Group-Level Permissions \(p. 351\)](#)
- [Step 5: Grant IAM User Alice Specific Permissions \(p. 357\)](#)
- [Step 6: Grant IAM User Bob Specific Permissions \(p. 360\)](#)
- [Step 7: Secure the Private Folder \(p. 360\)](#)
- [Cleanup \(p. 362\)](#)
- [Related Resources \(p. 362\)](#)

Background: Basics of Buckets and Folders

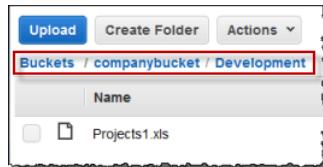
The Amazon S3 data model is a flat structure: you create a bucket, and the bucket stores objects. There is no hierarchy of subbuckets or subfolders; however, you can emulate a folder hierarchy. Tools such as the Amazon S3 Console can present a view of these logical folders and subfolders in your bucket, as shown here:



The console shows that a bucket named **companybucket** has three folders, **Private**, **Development**, and **Finance**, and an object, **s3-dg.pdf**. The console uses the object names (keys) to create a logical hierarchy with folders and subfolders. Consider the following examples:

- When you create the **Development** folder, the console creates an object with the key `Development/`. Note the trailing '/' delimiter.
- When you upload an object named **Projects1.xls** in the **Development** folder, the console uploads the object and gives it the key `Development/Projects1.xls`.

In the key, **Development** is the prefix and '/' is the delimiter. The Amazon S3 API supports prefixes and delimiters in its operations. For example, you can get a list of all objects from a bucket with a specific prefix and delimiter. In the console, when you double-click the **Development** folder, the console lists the objects in that folder. In the following example, the **Development** folder contains one object.



When the console lists the **Development** folder in the `companybucket` bucket, it sends a request to Amazon S3 in which it specifies a prefix of `Development` and a delimiter of '/' in the request. The console's response looks just like a folder list in your computer's file system. The preceding example shows that the bucket `companybucket` has an object with the key `Development/Projects1.xls`.

The console is using object keys to infer a logical hierarchy; Amazon S3 has no physical hierarchy, only buckets that contain objects in a flat file structure. When you create objects by using the Amazon S3 API, you can use object keys that imply a logical hierarchy.

When you create a logical hierarchy of objects, you can manage access to individual folders, as we will do in this walkthrough.

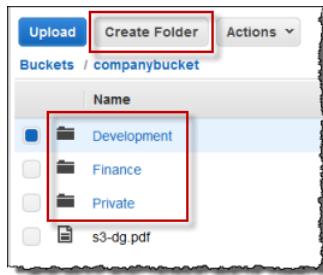
Before going into the walkthrough, you need to familiarize yourself with one more concept, the "root-level" bucket content. Suppose your `companybucket` bucket has the following objects:

`Private/privDoc1.txt`

`Private/privDoc2.zip`

Development/project1.xls
Development/project2.xls
Finance/Tax2011/document1.pdf
Finance/Tax2011/document2.pdf
s3-dg.pdf

These object keys create a logical hierarchy with Private, Development and the Finance as root-level folders and s3-dg.pdf as a root-level object. When you click the bucket name in the Amazon S3 console, the root-level items appear as shown. The console shows the top-level prefixes (Private/, Development/ and Finance/) as root-level folders. The object key s3-dg.pdf has no prefix, and so it appears as a root-level item.



Walkthrough Example

The example for this walkthrough is as follows:

- You create a bucket and then add three folders (Private, Development, and Finance) to it.
- You have two users, Alice and Bob. You want Alice to access only the Development folder and Bob to access only the Finance folder, and you want to keep the Private folder content private. In the walkthrough, you manage access by creating AWS Identity and Access Management (IAM) users (we will use the same user names, Alice and Bob) and grant them the necessary permissions.

IAM also supports creating user groups and granting group-level permissions that apply to all users in the group. This helps you better manage permissions. For this exercise, both Alice and Bob will need some common permissions. So you will also create a group named Consultants and then add both Alice and Bob to the group. You will first grant permissions by attaching a group policy to the group. Then you will add user-specific permissions by attaching policies to specific users.

Note

The walkthrough uses companybucket as the bucket name, Alice and Bob as the IAM users, and Consultants as the group name. Because Amazon S3 requires that bucket names be globally unique, you will need to replace the bucket name with a name that you create.

Step 0: Preparing for the Walkthrough

In this example, you will use your AWS account credentials to create IAM users. Initially, these users have no permissions. You will incrementally grant these users permissions to perform specific Amazon S3 actions. To test these permissions, you will sign in to the console with each user's credentials. As you incrementally grant permissions as an AWS account owner and test permissions as an IAM user, you need to sign in and out, each time using different credentials. You can do this testing with one browser, but the process will go faster if you can use two different browsers: use one browser to connect to the AWS Management Console with your AWS account credentials and another to connect with the IAM user credentials.

To sign into the AWS Management Console with your AWS account credentials, go to <http://aws.amazon.com/console>. An IAM user cannot sign in by using the same link. An IAM user must use an IAM-enabled sign-in page. As the account owner, you can provide this link to your users.

To provide a sign-in link for IAM users

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the **Navigation** pane, click **IAM Dashboard**.
3. Note the URL under **IAM users sign in link**: You will give this link to IAM users to sign in to the console with their IAM user name and password.

For more information about IAM, go to [The AWS Management Console Sign-in Page in AWS Identity and Access Management Using IAM](#).

Step 1: Create a Bucket

In this step, you will sign in to the Amazon S3 console with your AWS account credentials, create a bucket, add folders (Development, Finance, Private) to the bucket, and upload one or two sample documents in each folder.

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Create a bucket.
For step-by-step instructions, go to [Creating a Bucket](#) in the *Amazon Simple Storage Service Console User Guide*.
3. Upload one document to the bucket.
This exercise assumes you have the `s3-dg.pdf` document at the root level of this bucket. If you upload a different document, substitute its file name for `s3-dg.pdf`.
4. Add three folders named Private, Finance, and Development to the bucket.

For step-by-step instructions to create a folder, go to [Creating a Folder](#) in the *Amazon Simple Storage Service Console User Guide*.

5. Upload one or two documents to each folder.

For this exercise, assume you have uploaded a couple of documents in each folder, resulting in the bucket having objects with the following keys:

Private/privDoc1.txt
Private/privDoc2.zip
Development/project1.xls
Development/project2.xls
Finance/Tax2011/document1.pdf
Finance/Tax2011/document2.pdf
s3-dg.pdf

For step-by-step instructions, go to [Uploading Objects into Amazon S3](#) in the *Amazon Simple Storage Service Console User Guide*.

Step 2: Create IAM Users and a Group

Now use the IAM console to add two IAM users, Alice and Bob, to your AWS account. Also create an administrative group named Consultants, and then add both users to the group.

Caution

When you add users and a group, do not attach any policies that grant permissions to these users. At first, these users will not have any permissions. In the following sections, you will incrementally grant permissions. You must first ensure that you have assigned passwords to these IAM users. You will use these user credentials to test Amazon S3 actions and verify that the permissions work as expected.

For step-by-instructions on creating a new IAM user, go to [Adding a New User to Your AWS Account](#) in *Using IAM*.

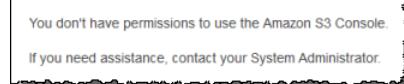
For step-by-step instructions on creating an administrative group, go to [Creating an Admins Group](#) section in the *Using IAM* guide.

Step 3: Verify that IAM Users Have No Permissions

If you are using two browsers, you can now use the second browser to sign into the console using one of the IAM user credentials.

1. Using the IAM user sign-in link (see [To provide a sign-in link for IAM users \(p. 350\)](#)), sign into the AWS console using either of the IAM user credentials.
2. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.

Verify the following console message telling you that you have no permissions.



Now, let's begin granting incremental permissions to the users. First, you will attach a group policy that grants permissions that both users must have.

Step 4: Grant Group-Level Permissions

We want all our users to be able to do the following:

- List all buckets owned by the parent account

To do so, Bob and Alice must have permission for the `s3>ListAllMyBuckets` action.

- List root-level items, folders, and objects, in the `companybucket` bucket.

To do so, Bob and Alice must have permission for the `s3>ListBucket` action on the `companybucket` bucket.

Now we'll create a policy that grants these permissions and then we'll attach it to the Consultants group.

Step 4.1: Grant Permission to List All Buckets

In this step you'll create a managed policy that grants the users minimum permissions to enable them to list all buckets owned by the parent account and then you'll attach the policy to the Consultants group. When you attach the managed policy to a user or a group, you allow the user or group permission to obtain a list of buckets owned by the parent AWS account.

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

Note

Since you'll be granting user permissions sign in with your AWS account credentials, not as an IAM user.

2. Create the managed policy.

- a. In the navigation pane on the left, click **Policies** and then click **Create Policy**.
- b. Next to **Create Your Own Policy**, click **Select**.
- c. Enter `AllowGroupToSeeBucketListInTheConsole` in the **Policy Name** field.
- d. Copy the following access policy and paste it into the **Policy Document** field.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowGroupToSeeBucketListInTheConsole",  
            "Action": ["s3>ListAllMyBuckets"],  
            "Effect": "Allow",  
            "Resource": ["arn:aws:s3:::*"]  
        }  
    ]  
}
```

A policy is a JSON document. In the document, a `Statement` is an array of objects, each describing a permission using a collection of name value pairs. The preceding policy describes one specific permission. The `Action` specifies the type of access. In the policy, the `s3>ListAllMyBuckets` is a predefined Amazon S3 action. This action covers the Amazon S3 GET Service operation, which returns list of all buckets owned by the authenticated sender. The `Effect` element value determine if specific permission is allowed or denied.

3. Attach the `AllowGroupToSeeBucketListInTheConsole` managed policy that you created to the Consultants group.

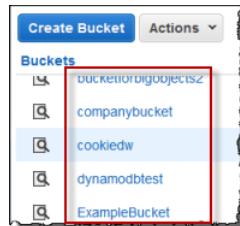
For step-by-step instructions for attaching a managed policy, go to [Managing IAM Policies Using the AWS Management Console](#) in *Using IAM*.

You attach policy documents to IAM users and groups in the IAM console. Because we want both our users to be able to list the buckets, we attach the policy to the group.

4. Test the permission.

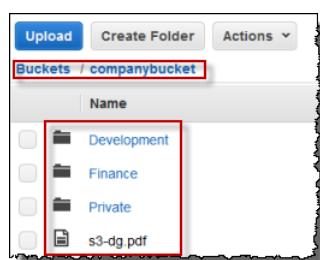
- a. Using the IAM user sign-in link (see [To provide a sign-in link for IAM users \(p. 350\)](#)), sign into the AWS console using any one of IAM user credentials.
- b. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.

The console should now list all the buckets but not the objects in any of the buckets.



Step 4.2: Enable Users to List Root-Level Content of a Bucket

Now let's allow all users to list the root-level companybucket bucket items. When a user clicks the company bucket in the Amazon S3 console, he or she will be able to see the root-level items in the bucket.



Remember, we are using companybucket for illustration. You must use the name of the bucket that you created for this exercise.

To understand what request the console sends to Amazon S3 when you click a bucket name, the response Amazon S3 returns, and how the console interprets the response, it is necessary to take a little deep dive.

When you click a bucket name, the console sends the [GET Bucket \(List Objects\)](#) request to Amazon S3. This request includes the following parameters:

- prefix parameter with an empty string as its value.
- delimiter parameter with / as its value.

The following is an example request:

```
GET ?prefix=&delimiter=/ HTTP/1.1
Host: companybucket.s3.amazonaws.com
Date: Wed, 01 Aug 2012 12:00:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:xQE0diMbLRepdf3YB+FIEXAMPLE=
```

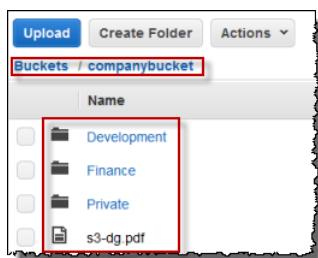
Amazon S3 returns a response that includes the following <ListBucketResult/> element:

```
<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
<Name>companybucket</Name>
<Prefix></Prefix>
<Delimiter></Delimiter>
...
<Contents>
<Key>s3-dg.pdf</Key>
```

```
...
</Contents>
<CommonPrefixes>
  <Prefix>Development/</Prefix>
</CommonPrefixes>
<CommonPrefixes>
  <Prefix>Finance/</Prefix>
</CommonPrefixes>
<CommonPrefixes>
  <Prefix>Private/</Prefix>
</CommonPrefixes>
</ListBucketResult>
```

The key `s3-dg.pdf` does not contain the `'/'` delimiter, and Amazon S3 returns the key in the `<Contents/>` element. However, all other keys in our example bucket contain the `'/'` delimiter. Amazon S3 groups these keys and returns a `<CommonPrefixes/>` element for each of the distinct prefix values `Development/`, `Finance/`, and `/Private` that is a substring from the beginning of these keys to the first occurrence of the specified `'/'` delimiter.

The console interprets this result and displays the root-level items as three folders and one object key.



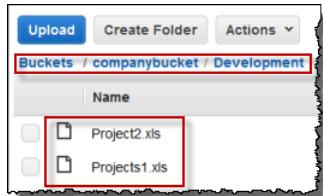
Now, if Bob or Alice double-clicks the **Development** folder, the console sends the [GET Bucket \(List Objects\)](#) request to Amazon S3 with the `prefix` and the `delimiter` parameters set to the following values:

- `prefix` parameter with value `Development/`.
- `delimiter` parameter with `'/'` value.

In response, Amazon S3 returns the object keys that start with the specified prefix.

```
<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Name>companybucket</Name>
  <Prefix>Development/<Prefix>
  <Delimiter>/</Delimiter>
  ...
  <Contents>
    <Key>Project1.xls</Key>
    ...
  </Contents>
  <Contents>
    <Key>Project2.xls</Key>
    ...
  </Contents>
</ListBucketResult>
```

The console shows the object keys:



Now, let's return to granting users permission to list the root-level bucket items. To list bucket content, users need permission to call the `s3>ListBucket` action, as shown in the following policy statement. To ensure that they see only the root-level content, we add a condition that users must specify an empty prefix in the request—that is, they are not allowed to double-click any of our root-level folders. Finally, we will add a condition to require folder-style access by requiring user requests to include the `delimiter` parameter with value `'/'`.

```
{  
    "Sid": "AllowRootLevelListingOfCompanyBucket",  
    "Action": ["s3>ListBucket"],  
    "Effect": "Allow",  
    "Resource": ["arn:aws:s3:::companybucket"],  
    "Condition": {  
        "StringEquals": {  
            "s3:prefix": [""], "s3:delimiter": ["/"]  
        }  
    }  
}
```

When you use the Amazon S3 console, note that when you click a bucket, the console first sends the [GET Bucket location](#) request to find the AWS region where the bucket is deployed. Then the console uses the region-specific endpoint for the bucket to send the [GET Bucket \(List Objects\)](#) request. As a result, if users are going to use the console, you must grant permission for the `s3:GetBucketLocation` action as shown in the following policy statement:

```
{  
    "Sid": "RequiredByS3Console",  
    "Action": ["s3:GetBucketLocation"],  
    "Effect": "Allow",  
    "Resource": ["arn:aws:s3:::*"]  
}
```

To enable users to list root-level bucket content

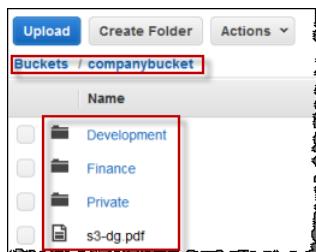
1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
Use your AWS account credentials, not the credentials of an IAM user, to sign in to the console.
2. Replace the existing `AllowGroupToSeeBucketListInTheConsole` managed policy that is attached to the Consultants group with the following policy, which also allows the `s3>ListBucket` action. Remember to replace `companybucket` in the policy `Resource` with the name of your bucket.

For step-by-step instructions, go to [Editing Customer Managed Policies](#) in *Using IAM*. When following the step-by-step instructions, make sure to follow the directions for applying your changes to all principal entities that the policy is attached to.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowGroupToSeeBucketListAndAlsoAllowGetBucketLocationRequiredForListBucket",
            "Action": [ "s3>ListAllMyBuckets", "s3:GetBucketLocation" ],
            "Effect": "Allow",
            "Resource": [ "arn:aws:s3:::*" ]
        },
        {
            "Sid": "AllowRootLevelListingOfCompanyBucket",
            "Action": [ "s3>ListBucket" ],
            "Effect": "Allow",
            "Resource": [ "arn:aws:s3:::companybucket" ],
            "Condition":{
                "StringEquals":{
                    "s3:prefix": [ "" ], "s3:delimiter": [ "/" ]
                }
            }
        }
    ]
}
```

3. Test the updated permissions.

1. Using the IAM user sign-in link (see [To provide a sign-in link for IAM users \(p. 350\)](#)), sign in to the AWS Management Console.
 Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Click the bucket that you created for this exercise, and the console will now show the root-level bucket items. If you click any folders in the bucket, you will not be able to see the folder content, because you have not yet granted those permissions.



This test succeeds when users use the Amazon S3 console because when you click a bucket in the console, the console implementation sends a request that includes the `prefix` parameter with an empty string as its value and the `delimiter` parameter with `'/'` as its value.

Step 4.3: Summary of the Group Policy

The net effect of the group policy that you added is to grant the IAM users Alice and Bob the following minimum permissions:

- List all buckets owned by the parent account.

- See root-level items in the `companybucket` bucket.

However, the users still cannot do much. Let's grant user-specific permissions, as follows:

- Permit Alice to get and put objects in the Development folder.
- Permit Bob to get and put objects in the Finance folder.

For user-specific permissions, you attach a policy to the specific user, not to the group. In the following section, you grant Alice permission to work in the Development folder. You can repeat the steps to grant similar permission to Bob to work in the Finance folder.

Step 5: Grant IAM User Alice Specific Permissions

Now we grant additional permissions to Alice so she can see the content of the Development folder and get and put objects in that folder.

Step 5.1: Grant IAM User Alice Permission to List the Development Folder Content

For Alice to list the Development folder content, you must apply a policy to the Alice user that grants permission for the `s3>ListBucket` action on the `companybucket` bucket, provided the request includes the prefix `Development/`. Because we want this policy to be applied only to the user Alice we'll use an inline policy. For more information about inline policies, go to [Editing Customer Managed Policies in Using IAM](#).

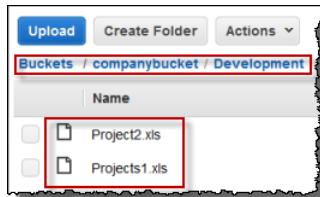
1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
Use your AWS account credentials, not the credentials of an IAM user, to sign in to the console.
2. Create an inline policy to grant the user Alice permission to list the Development folder content.
 - a. In the navigation pane on the left, click **Users**.
 - b. Click the user name Alice.
 - c. Scroll down to the **Permissions** section, and then expand the **Inline Policies** section.
 - d. Click [click here \(or Create User Policy\)](#).
 - e. Click **Custom Policy**, and then click **Select**.
 - f. Enter a name for the policy in the **Policy Name** field.
 - g. Copy the following policy into the **Policy Document** field.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowListBucketIfSpecificPrefixIsIncludedInRequest",  
            "Action": ["s3>ListBucket"],  
            "Effect": "Allow",  
            "Resource": ["arn:aws:s3:::companybucket"],  
            "Condition": { "StringLike": {"s3:prefix": ["Development/*"] } }  
        }  
    ]  
}
```

3. Test the change to Alice's permissions:

- a. Using the IAM user sign in link (see [To provide a sign-in link for IAM users \(p. 350\)](#)), sign in to the AWS Management Console.
- b. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
- c. In the Amazon S3 console, verify that Alice can see the list of objects in the Development/ folder in the bucket.

When the user clicks the /Development folder to see the list of objects in it, the Amazon S3 console sends the `ListObjects` request to Amazon S3 with the prefix /Development. Because the user is granted permission to see the object list with the prefix Development and delimiter '/', Amazon S3 returns the list of objects with the key prefix Development/, and the console displays the list.



Step 5.2: Grant IAM User Alice Permissions to Get and Put Objects in the Development Folder

For Alice to get and put objects in the Development folder, she needs permission to call the `s3:GetObject` and `s3:PutObject` actions. The following policy statements grant these permissions, provided the request includes the `prefix` parameter with a value of `Development/`.

```
{  
    "Sid": "AllowUserToReadWriteObjectData",  
    "Action": [ "s3:GetObject", "s3:PutObject" ],  
    "Effect": "Allow",  
    "Resource": [ "arn:aws:s3:::companybucket/Development/*" ]  
}
```

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
Use your AWS account credentials, not the credentials of an IAM user, to sign in to the console.
2. Edit the inline policy you created in the previous step.
 - a. In the navigation pane on the left, click **Users**.
 - b. Click the user name Alice.
 - c. Scroll down to the **Permissions** section, and then expand the **Inline Policies** section.
 - d. Click **Edit Policy** next to the name of the policy you created in the previous step.
 - e. Copy the following policy into the **Policy Document** field replacing the existing policy.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowListBucketIfSpecificPrefixIsIncludedInRequest",  
            "Action": [ "s3>ListBucket" ],  
            "Effect": "Allow",  
            "Resource": "arn:aws:s3:::companybucket/Development/*"  
        }  
    ]  
}
```

```

        "Resource": [ "arn:aws:s3:::companybucket" ],
        "Condition": {
            "StringLike": { "s3:prefix": [ "Development/*" ] }
        }
    },
{
    "Sid": "AllowUserToReadWriteObjectDataInDevelopmentFolder",
    "Action": [ "s3:GetObject", "s3:PutObject" ],
    "Effect": "Allow",
    "Resource": [ "arn:aws:s3:::companybucket/Development/*" ]
}
]
}

```

3. Test the updated policy:

1. Using the IAM user sign-in link (see [To provide a sign-in link for IAM users \(p. 350\)](#)), sign into the AWS Management Console.
2. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
3. In the Amazon S3 console, verify that Alice can now add an object and download an object in the Development folder.

Step 5.3: Explicitly Deny IAM User Alice Permissions to Any Other Folders in the Bucket

User Alice can now list the root-level content in the companybucket bucket. She can also get and put objects in the Development folder. If you really want to tighten the access permissions, you could explicitly deny Alice access to any other folders in the bucket. If there is any other policy (bucket policy or ACL) that grants Alice access to any other folders in the bucket, this explicit deny overrides those permissions.

You can add the following statement to the user Alice policy that requires all requests that Alice sends to Amazon S3 to include the prefix parameter, whose value can be either Development/* or an empty string.

```
{
    "Sid": "ExplicitlyDenyAnyRequestsForAllOtherFoldersExceptDevelopment",
    "Action": [ "s3>ListBucket" ],
    "Effect": "Deny",
    "Resource": [ "arn:aws:s3:::companybucket" ],
    "Condition": { "StringNotLike": { "s3:prefix": [ "Development/*" ] } },
                    "Null" : { "s3:prefix":false }
    }
}
```

Note that there are two conditional expressions in the Condition block. The result of these conditional expressions is combined by using the logical AND. If both conditions are true, the result of the combined condition is true.

- The Null conditional expression ensures that requests from Alice include the prefix parameter.

The prefix parameter requires folder-like access. If you send a request without the prefix parameter, Amazon S3 returns all the object keys.

If the request includes the `prefix` parameter with a null value, the expression will evaluate to true, and so the entire `Condition` will evaluate to true. You must allow an empty string as value of the `prefix` parameter. You recall from the preceding discussion, allowing the null string allows Alice to retrieve root-level bucket items as the console does in the preceding discussion. For more information, see [Step 4.2: Enable Users to List Root-Level Content of a Bucket \(p. 353\)](#).

- The `StringNotLike` conditional expression ensures that if the value of the `prefix` parameter is specified and is not `Development/*`, the request will fail.

Follow the steps in the preceding section and again update the inline policy you created for user Alice.

Copy the following policy into the **Policy Document** field replacing the existing policy.

```
{  
    "Statement": [  
        {  
            "Sid": "AllowListBucketIfSpecificPrefixIsIncludedInRequest",  
            "Action": ["s3>ListBucket"],  
            "Effect": "Allow",  
            "Resource": ["arn:aws:s3:::companybucket"],  
            "Condition": {  
                "StringLike": {"s3:prefix": ["Development/*"]} } } },  
        {  
            "Sid": "AllowUserToReadWriteObjectDataInDevelopmentFolder",  
            "Action": ["s3GetObject", "s3PutObject"],  
            "Effect": "Allow",  
            "Resource": ["arn:aws:s3:::companybucket/Development/*"] } ,  
        {  
            "Sid": "ExplicitlyDenyAnyRequestsForAllOtherFoldersExceptDevelopment",  
            "Action": ["s3>ListBucket"],  
            "Effect": "Deny",  
            "Resource": ["arn:aws:s3:::companybucket"],  
            "Condition": { "StringNotLike": {"s3:prefix": ["Development/*"]},  
                           "Null": {"s3:prefix": false} } } ] } }
```

Step 6: Grant IAM User Bob Specific Permissions

Now you want to grant Bob permission to the Finance folder. Follow the steps you used earlier to grant permissions to Alice, but replace the Development folder with the Finance folder. For step-by-step instructions, see [Step 5: Grant IAM User Alice Specific Permissions \(p. 357\)](#).

Step 7: Secure the Private Folder

In this example, you have only two users. You granted all the minimum required permissions at the group level and granted user-level permissions only when you really need to permissions at the individual user level. This approach helps minimize the effort of managing permissions. As the number of users increases, managing permissions can become cumbersome. For example, we don't want any of the users in this example to access the content of the Private folder. How do you ensure you don't accidentally grant a

user permission to it? You add a policy that explicitly denies access to the folder. An explicit deny overrides any other permissions. To ensure that the Private folder remains private, you can add the follow two deny statements to the group policy:

- Add the following statement to explicitly deny any action on resources in the `Private` folder (`companybucket/Private/*`).

```
{  
    "Sid": "ExplicitDenyAccessToPrivateFolderToEveryoneInTheGroup",  
    "Action": ["s3:*"],  
    "Effect": "Deny",  
    "Resource": ["arn:aws:s3:::companybucket/Private/*"]  
}
```

- You also deny permission for the list objects action when the request specifies the `Private/` prefix. In the console, if Bob or Alice double-clicks the Private folder, this policy causes Amazon S3 to return an error response.

```
{  
    "Sid": "DenyListBucketOnPrivateFolder",  
    "Action": ["s3>ListBucket"],  
    "Effect": "Deny",  
    "Resource": ["arn:aws:s3:::*"],  
    "Condition": {  
        "StringLike": {"s3:prefix": ["Private/"]} }  
}
```

Replace the Consultants group policy with an updated policy that includes the preceding deny statements. After the updated policy is applied, none of the users in the group will be able to access the Private folder in your bucket.

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
Use your AWS account credentials, not the credentials of an IAM user, to sign in to the console.
2. Replace the existing `AllowGroupToSeeBucketListInTheConsole` managed policy that is attached to the Consultants group with the following policy. Remember to replace `companybucket` in the policy with the name of your bucket.

For instructions, go to [Editing Customer Managed Policies](#) in *Using IAM*. When following the instructions, make sure to follow the directions for applying your changes to all principal entities that the policy is attached to.

```
{  
    "Statement": [  
        {  
            "Sid": "AllowGroupToSeeBucketListAndAlsoAllowGetBucketLocationRequired  
ForListBucket",  
            "Action": ["s3>ListAllMyBuckets", "s3:GetBucketLocation"],  
            "Effect": "Allow",  
            "Resource": ["arn:aws:s3:::*"]  
        },  
        {  
            "Sid": "DenyAccessToPrivateFolder",  
            "Action": ["s3:GetObject"],  
            "Effect": "Deny",  
            "Resource": ["arn:aws:s3:::companybucket/Private/*"]  
        }  
    ]  
}
```

```
    "Sid": "AllowRootLevelListingOfCompanyBucket",
    "Action": [ "s3>ListBucket" ],
    "Effect": "Allow",
    "Resource": [ "arn:aws:s3:::companybucket" ],
    "Condition":{
        "StringEquals":{ "s3:prefix":[ " " ] }
    }
},
{
    "Sid": "RequireFolderStyleList",
    "Action": [ "s3>ListBucket" ],
    "Effect": "Deny",
    "Resource": [ "arn:aws:s3:::*" ],
    "Condition":{
        "StringNotEquals":{ "s3:delimiter":"/" }
    }
},
{
    "Sid": "ExplicitDenyAccessToPrivateFolderToEveryoneInTheGroup",
    "Action": [ "s3:*" ],
    "Effect": "Deny",
    "Resource": [ "arn:aws:s3:::companybucket/Private/*" ]
},
{
    "Sid": "DenyListBucketOnPrivateFolder",
    "Action": [ "s3>ListBucket" ],
    "Effect": "Deny",
    "Resource": [ "arn:aws:s3:::*" ],
    "Condition":{
        "StringLike":{ "s3:prefix":["Private/" ] }
    }
}
]
```

Cleanup

In order to clean up, go to the IAM console and remove the users Alice and Bob. For step-by-step instructions, go to [Deleting a User from Your AWS Account](#) in the *Using IAM* guide.

To ensure that you aren't charged further for storage, you should also delete the objects and the bucket that you created for this exercise .

Related Resources

- [Managing Policies \(AWS Management Console\)](#) in *Using IAM*.

Managing Access with ACLs

Topics

- [Access Control List \(ACL\) Overview \(p. 363\)](#)
- [Managing ACLs \(p. 368\)](#)

Access control lists (ACLs) is one of the resource-based access policy option (see [Overview of Managing Access \(p. 270\)](#)) you can use to manage access to your buckets and objects. You can use ACLs to grant basic read/write permissions to other AWS accounts. There are limits to managing permissions using ACLs. For example, you can grant permissions only to other AWS accounts, you cannot grant permissions to users in your account. You cannot grant conditional permissions, nor can you explicitly deny permissions. ACLs are suitable for specific scenarios. For example, if a bucket owner allows other AWS accounts to upload objects, permissions to these objects can only be managed using object ACL by the AWS account that owns the object. You should read the following introductory topics that explain the basic concepts and options available for you to manage access to your Amazon S3 resources and guidelines for when to use which access policy options.

- [Introduction to Managing Access Permissions to Your Amazon S3 Resources \(p. 269\)](#)
- [Guidelines for Using the Available Access Policy Options \(p. 280\)](#)

Access Control List (ACL) Overview

Topics

- [Who Is a Grantee? \(p. 364\)](#)
- [What Permissions Can I Grant? \(p. 365\)](#)
- [Sample ACL \(p. 366\)](#)
- [Canned ACL \(p. 367\)](#)
- [How to Specify an ACL \(p. 368\)](#)

Amazon S3 Access Control Lists (ACLs) enable you to manage access to buckets and objects. Each bucket and object has an ACL attached to it as a subresource. It defines which AWS accounts or groups are granted access and the type of access. When a request is received against a resource, Amazon S3 checks the corresponding ACL to verify the requester has the necessary access permissions.

When you create a bucket or an object, Amazon S3 creates a default ACL that grants the resource owner full control over the resource as shown in the following sample bucket ACL (the default object ACL has the same structure).

```
<?xml version="1.0" encoding="UTF-8"?>
<AccessControlPolicy xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Owner>
    <ID>*** Owner-Canonical-User-ID ***</ID>
    <DisplayName>owner-display-name</DisplayName>
  </Owner>
  <AccessControlList>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:type="Canonical User">
        <ID>*** Owner-Canonical-User-ID ***</ID>
        <DisplayName>display-name</DisplayName>
      </Grantee>
```

```
<Permission>FULL_CONTROL</Permission>
</Grant>
</AccessControlList>
</AccessControlPolicy>
```

The sample ACL includes an `Owner` element identifying the owner via the AWS account's canonical user ID. The `Grant` element identifies the grantee (either an AWS account or a predefined group), and the permission granted. This default ACL has one `Grant` element for the owner. You grant permissions by adding `Grant` elements, each grant identifying the grantee and the permission.

Note

An ACL can have up to 100 grants.

Who Is a Grantee?

A grantee can be an AWS account or one of the predefined Amazon S3 groups. You grant permission to an AWS account by the email address or the canonical user ID. However, if you provide an email in your grant request, Amazon S3 finds the canonical user ID for that account and adds it to the ACL. The resulting ACLs will always contain the canonical user ID for the AWS account, not the AWS account's email address.

Note

You cannot use an email address to specify a grantee in the EU (Frankfurt), China (Beijing), or AWS GovCloud (US), regions. Also, using an email address to specify a grantee will not be supported for any AWS region created after 12/8/2014.

Finding an AWS Account Canonical User ID

The canonical user ID is associated with your AWS account. You can find this ID by using the following procedure.

To find the canonical user ID for your AWS account

1. Go to <http://aws.amazon.com/console>.
2. Click **Sign into the AWS Management Console** and sign in with your AWS account credentials (not as an IAM user).
3. Go to [Security Credentials](#).
4. Go to the **Account Identifier** section to see the canonical user ID associated with your AWS account.

You can also look up the canonical user ID of an AWS account by reading the ACL of a bucket or an object to which the AWS account has access permissions. When an individual AWS account is granted a permission by a grant request, a grant entry is added to the ACL with the AWS account's canonical user ID. For more information about the canonical user ID, go to [AWS Account Identifiers](#).

Amazon S3 Predefined Groups

Amazon S3 has a set of predefined groups. When granting account access to a group, you specify one of our URIs instead of a canonical user ID. We provide the following predefined groups:

- **Authenticated Users group** – Represented by <http://acs.amazonaws.com/groups/global/AuthenticatedUsers>. This group represents all AWS accounts. Access permission to this group allows any AWS account to access the resource. However, all requests must be signed (authenticated).
- **All Users group** – Represented by <http://acs.amazonaws.com/groups/global/AllUsers>.

Access permission to this group allows anyone to access the resource. The requests can be signed (authenticated) or unsigned (anonymous). Unsigned requests omit the Authentication header in the request.

- **Log Delivery group** – Represented by <http://acs.amazonaws.com/groups/s3/LogDelivery>. WRITE permission on a bucket enables this group to write server access logs (see [Server Access Logging \(p. 530\)](#)) to the bucket.

Note

When using ACLs, a grantee can be an AWS account or one of the predefined Amazon S3 groups. However, the grantee cannot be an Identity and Access Management (IAM) user. For more information about AWS users and permissions within IAM, go to [Using AWS Identity and Access Management](#).

Note

When you grant other AWS accounts access to your resources, be aware that the AWS accounts can delegate their permissions to users under their accounts. This is known as cross-account access. For information about using cross-account access, go to [Enabling Cross-Account Access in Using Identity and Access Management](#).

What Permissions Can I Grant?

The following table lists the set of permissions Amazon S3 supports in an ACL. Note that the set of ACL permissions is same for object ACL and bucket ACL. However, depending on the context (bucket ACL or object ACL), these ACL permissions grant permissions for specific bucket or the object operations. The table lists the permission and describes what they mean in the context of object and bucket permissions.

Permission	When granted on a bucket	When granted on an object
READ	Allows grantee to list the objects in the bucket	Allows grantee to read the object data and its metadata
WRITE	Allows grantee to create, overwrite, and delete any object in the bucket	Not applicable
READ_ACP	Allows grantee to read the bucket ACL	Allows grantee to read the object ACL
WRITE_ACP	Allows grantee to write the ACL for the applicable bucket	Allows grantee to write the ACL for the applicable object
FULL_CONTROL	Allows grantee the READ, WRITE, READ_ACP, and WRITE_ACP permissions on the bucket	Allows grantee the READ, READ_ACP, and WRITE_ACP permissions on the object

Mapping of ACL Permissions and Access Policy Permissions

As shown in the preceding table, ACL allows only a finite set of permissions, compared to the number of permissions you can set in an access policy (see [Specifying Permissions in a Policy \(p. 315\)](#)). Each of these permissions allow one or more Amazon S3 operations. The following table shows how each of the ACL permissions map to the corresponding access policy permissions. As you can see, access policy allows more permissions than ACL does, you use ACL to primarily grant basic read/write permissions, similar to file system permissions. For more information about when to use ACL, see [Guidelines for Using the Available Access Policy Options \(p. 280\)](#).

ACL Permission	Corresponding access policy permissions when the ACL permission is granted on a bucket	Corresponding access policy permissions when the ACL permission is granted on an object
READ	s3:ListBucket, s3>ListBucketVersions, and s3>ListBucketMultipartUploads	s3:GetObject, s3GetObjectVersion, and s3GetObjectTorrent
WRITE	s3:PutObject and s3>DeleteObject. In addition, when the grantee is the bucket owner, granting WRITE permission in a bucket ACL allows the s3>DeleteObjectVersion action to be performed on any version in that bucket.	Not applicable
READ_ACP	s3:GetBucketAcl	s3:GetObjectAcl and s3GetObjectVersionAcl
WRITE_ACP	s3:PutBucketAcl	s3:PutObjectAcl and s3:PutObjectVersionAcl
FULL_CONTROL	It is equivalent to granting READ, READ_ACP, and WRITE_ACP ACL permissions. Accordingly, this ACL permission maps to combination of corresponding access policy permissions.	It is equivalent to granting READ, READ_ACP, and WRITE_ACP ACL permissions. Accordingly, this ACL permission maps to combination of corresponding access policy permissions.

Sample ACL

The following sample ACL on a bucket identifies the resource owner and a set of grants. The format is the XML representation of an ACL in the Amazon S3 REST API. The bucket owner has FULL_CONTROL of the resource. In addition, the ACL shows how permissions are granted on a resource to two AWS accounts, identified by canonical user ID, and two of the predefined Amazon S3 groups discussed in the preceding section.

```

<?xml version="1.0" encoding="UTF-8"?>
<AccessControlPolicy xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Owner>
    <ID>Owner-canonical-user-ID</ID>
    <DisplayName>display-name</DisplayName>
  </Owner>
  <AccessControlList>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="CanonicalUser">
        <ID>Owner-canonical-user-ID</ID>
        <DisplayName>display-name</DisplayName>
      </Grantee>
      <Permission>FULL_CONTROL</Permission>
    </Grant>

    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="CanonicalUser">
        <ID>user1-canonical-user-ID</ID>
      </Grantee>
      <Permission>READ</Permission>
    </Grant>
  </AccessControlList>
</AccessControlPolicy>

```

```

        <DisplayName>display-name</DisplayName>
    </Grantee>
    <Permission>WRITE</Permission>
</Grant>

<Grant>
    <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="CanonicalUser">
        <ID>user2-canonical-user-ID</ID>
        <DisplayName>display-name</DisplayName>
    </Grantee>
    <Permission>READ</Permission>
</Grant>

<Grant>
    <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="Group">
        <URI>http://acs.amazonaws.com/groups/global/AllUsers</URI>
    </Grantee>
    <Permission>READ</Permission>
</Grant>
<Grant>
    <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="Group">
        <URI>http://acs.amazonaws.com/groups/s3/LogDelivery</URI>
    </Grantee>
    <Permission>WRITE</Permission>
</Grant>

</AccessControlList>
</AccessControlPolicy>

```

Canned ACL

Amazon S3 supports a set of predefined grants, known as canned ACLs. Each canned ACL has a predefined a set of grantees and permissions. The following table lists the set of canned ACLs and the associated predefined grants.

Canned ACL	Applies to	Permissions added to ACL
private	Bucket and object	Owner gets FULL_CONTROL. No one else has access rights (default).
public-read	Bucket and object	Owner gets FULL_CONTROL. The AllUsers group (see Who Is a Grantee? (p. 364)) gets READ access.
public-read-write	Bucket and object	Owner gets FULL_CONTROL. The AllUsers group gets READ and WRITE access. Granting this on a bucket is generally not recommended.
authenticated-read	Bucket and object	Owner gets FULL_CONTROL. The AuthenticatedUsers group gets READ access.
bucket-owner-read	Object	Object owner gets FULL_CONTROL. Bucket owner gets READ access. If you specify this canned ACL when creating a bucket, Amazon S3 ignores it.

Canned ACL	Applies to	Permissions added to ACL
bucket-owner-full-control	Object	Both the object owner and the bucket owner get FULL_CONTROL over the object. If you specify this canned ACL when creating a bucket, Amazon S3 ignores it.
log-delivery-write	Bucket	The LogDelivery group gets WRITE and READ_ACP permissions on the bucket. For more information on logs, see (Server Access Logging (p. 530)).

Note

You can specify only one of these canned ACLs in your request.

You specify a canned ACL in your request using the `x-amz-acl` request header. When Amazon S3 receives a request with a canned ACL in the request, it adds the predefined grants to the ACL of the resource.

How to Specify an ACL

Amazon S3 APIs enable you to set an ACL when you create a bucket or an object. Amazon S3 also provides API to set an ACL on an existing bucket or an object. These API provide you with the following methods to set an ACL:

- **Set ACL using request headers**— When you send a request to create a resource (bucket or object), you set an ACL using the request headers. Using these headers, you can either specify a canned ACL or specify grants explicitly (identifying grantee and permissions explicitly).
- **Set ACL using request body**— When you send a request to set an ACL on a existing resource, you can set the ACL either in the request header or in the body.

For more information, see [Managing ACLs \(p. 368\)](#).

Managing ACLs

Topics

- [Managing ACLs in the AWS Management Console \(p. 368\)](#)
- [Managing ACLs Using the AWS SDK for Java \(p. 369\)](#)
- [Managing ACLs Using the AWS SDK for .NET \(p. 373\)](#)
- [Managing ACLs Using the REST API \(p. 378\)](#)

There are several ways you can add grants to your resource ACL. You can use the AWS Management Console, which provides a UI to manage permissions without writing any code. You can use the REST API or use one of the AWS SDKs. These libraries further simplify your programming tasks.

Managing ACLs in the AWS Management Console

AWS Management Console provides a UI for you to grant ACL-based access permissions to your buckets and objects. The **Properties** pane includes the **Permissions** tab where you can grant ACL-based access permissions. The following screen shot shows the **Permissions** for a bucket.



It shows the list of grants found in the bucket ACL. For each grant, it shows the grantee and a set of check boxes showing the permissions granted. The permission names in the console are different than the ACL permission names. The preceding illustration shows the mapping between the two.

The preceding illustration shows a grantee with `FULL_CONTROL` permissions; note that all the check boxes are selected. All the UI components shown, except the **Add bucket policy** link, relate to the ACL-based permissions. The UI allows you to add or remove permissions. To add permissions, click **Add more permissions**, and to delete a permission, highlight the line and click **X** to the right of it. When you are done updating permissions, click **Save** to update the ACL. The console sends the necessary request to Amazon S3 to update the ACL on the specific resource.

For step-by-step instructions, go to [Editing Object Permissions](#) and [Editing Bucket Permissions](#) in the *Amazon Simple Storage Service Console User Guide*.

Managing ACLs Using the AWS SDK for Java

Setting an ACL when Creating a Resource

When creating a resource (buckets and objects), you can grant permissions (see [Access Control List \(ACL\) Overview \(p. 363\)](#)) by adding an `AccessControlList` in your request. For each permission, you explicitly specify the grantee and the permission.

For example, the following Java code snippet sends a `PutObject` request to upload an object. In the request, the code snippet specifies permissions to two AWS accounts and the Amazon S3 `AllUsers` group. The `PutObject` call includes the object data in the request body and the ACL grants in the request headers (see [PUT Object](#)).

```
String bucketName      = "bucket-name";
String keyName         = "object-key";
String uploadFileName = "file-name";

AmazonS3 s3client = new AmazonS3Client(new ProfileCredentialsProvider());

AccessControlList acl = new AccessControlList();
acl.grantPermission(new CanonicalGrantee("d25639fbe9c19cd30a4c0f43fb00e2d3f96400a9aa8dabfbbebe1906Example"),
    Permission.ReadAcp);
acl.grantPermission(GroupGrantee.AllUsers, Permission.Read);
acl.grantPermission(new EmailAddressGrantee("user@email.com"), Permission.WriteAcp);

File file = new File(uploadFileName);
s3client.putObject(new PutObjectRequest(bucketName, keyName, file).withAccessControlList(acl));
```

For more information about uploading objects, see [Working with Amazon S3 Objects \(p. 78\)](#).

In the preceding code snippet, in granting each permission you explicitly identified a grantee and a permission. Alternatively, you can specify a canned (predefined) ACL (see [Canned ACL \(p. 367\)](#)) in your request when creating a resource. The following Java code snippet creates a bucket and specifies a LogDeliveryWrite canned ACL in the request to grant write permission to the Amazon S3 LogDelivery group.

```
String bucketName      = "bucket-name";
AmazonS3 s3client = new AmazonS3Client(new ProfileCredentialsProvider());

s3client.createBucket(new CreateBucketRequest (bucketName).withCannedAcl(CannedAccessControlList.LogDeliveryWrite));
```

For information about the underlying REST API, go to [PUT Bucket](#).

Updating ACL on an Existing Resource

You can set ACL on an existing object or a bucket. You create an instance of the `AccessControlList` class and grant permissions and call the appropriate set ACL method. The following Java code snippet calls the `setObjectAcl` method to set ACL on an existing object.

```
String bucketName      = "bucket-name";
String keyName         = "object-key";

AmazonS3 s3client = new AmazonS3Client(new ProfileCredentialsProvider());

AccessControlList acl = new AccessControlList();
acl.grantPermission(new CanonicalGrantee("d25639fbe9c19cd30a4c0f43fbf00e2d3f96400a9aa8dabfbbebe1906Example"),
    Permission.ReadAcp);
acl.grantPermission(GroupGrantee.AuthenticatedUsers, Permission.Read);
acl.grantPermission(new EmailAddressGrantee("user@email.com"), Permission.WriteAcp);
Owner owner = new Owner();
owner.setId("852b113e7a2f25102679df27bb0ae12b3f85be6f290b936c4393484beExample");
owner.setDisplayName("display-name");
acl.setOwner(owner);

s3client.setObjectAcl(bucketName, keyName, acl);
```

Note

In the preceding code snippet, you can optionally read an existing ACL first, by calling the `getObjectAcl` method, add new grants to it, and then set the revised ACL on the resource.

Instead of granting permissions by explicitly specifying grantees and permissions explicitly, you can also specify a canned ACL in your request. The following Java code snippet sets the ACL on an existing object. In the request, the snippet specifies the canned ACL `AuthenticatedRead` to grant read access to the Amazon S3 Authenticated Users group.

```
String bucketName      = "bucket-name";
String keyName         = "object-key";

AmazonS3 s3client = new AmazonS3Client(new ProfileCredentialsProvider());

s3client.setObjectAcl(bucketName, keyName, CannedAccessControlList.AuthenticatedRead);
```

An Example

The following Java code example first creates a bucket. In the create request, it specifies a public-read canned ACL. It then retrieves the ACL in an `AccessControlList` instance, clears grants, and adds new grants to the `AccessControlList`. Finally, it saves the updated `AccessControlList`, that is, it replaces the bucket ACL subresource.

The following Java code example performs the following tasks:

- Create a bucket. In the request, it specifies a log-delivery-write canned ACL, granting write permission to the LogDelivery Amazon S3 group.
- Read the ACL on the bucket.
- Clear existing permissions and add the new permission to the ACL.
- Call `setBucketAcl` to add the new ACL to the bucket.

Note

To test the following code example, you must update the code and provide your credentials, and also provide the canonical user ID and email address of the accounts that you want to grant permissions to.

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collection;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.AccessControlList;
import com.amazonaws.services.s3.model.Bucket;
import com.amazonaws.services.s3.model.CannedAccessControlList;
import com.amazonaws.services.s3.model.CanonicalGrantee;
import com.amazonaws.services.s3.model.CreateBucketRequest;
import com.amazonaws.services.s3.model.Grant;
import com.amazonaws.services.s3.model.GroupGrantee;
import com.amazonaws.services.s3.model.Permission;
import com.amazonaws.services.s3.model.Region;

public class ACLExample {
    private static String bucketName = "**** Provide bucket name ****";

    public static void main(String[] args) throws IOException {
        AmazonS3 s3Client = new AmazonS3Client(new ProfileCredentialsProvider());

        Collection<Grant> grantCollection = new ArrayList<Grant>();
        try {
            // 1. Create bucket with Canned ACL.
            CreateBucketRequest createBucketRequest =
                new CreateBucketRequest(bucketName, Region.US_Standard).with
CannedAcl(CannedAccessControlList.LogDeliveryWrite);

            Bucket resp = s3Client.createBucket(createBucketRequest);
```

```
// 2. Update ACL on the existing bucket.  
AccessControlList bucketAcl = s3Client.getBucketAcl(bucketName);  
  
// (Optional) delete all grants.  
bucketAcl.getGrants().clear();  
  
// Add grant - owner.  
Grant grant0 = new Grant(  
    new Canonical  
    alGrantee("852b113e7a2f25102679df27bb0ae12b3f85be6f290b936c4393484beExample"),  
    Permission.FullControl);  
grantCollection.add(grant0);  
  
// Add grant using canonical user id.  
Grant grant1 = new Grant(  
    new Canonical  
    alGrantee("d25639fbe9c19cd30a4c0f43fbf00e2d3f96400a9aa8dabfbbebe1906Example"),  
    Permission.Write);  
grantCollection.add(grant1);  
  
// Grant LogDelivery group permission to write to the bucket.  
Grant grant3 = new Grant(GroupGrantee.LogDelivery,  
    Permission.Write);  
grantCollection.add(grant3);  
  
bucketAcl.getGrants().addAll(grantCollection);  
  
// Save (replace) ACL.  
s3Client.setBucketAcl(bucketName, bucketAcl);  
  
} catch (AmazonServiceException ase) {  
    System.out.println("Caught an AmazonServiceException, which" +  
        " means your request made it " +  
        "to Amazon S3, 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 S3, " +  
        "such as not being able to access the network.");  
    System.out.println("Error Message: " + ace.getMessage());  
}  
}  
}
```

Managing ACLs Using the AWS SDK for .NET

Setting an ACL When Creating a Resource

When creating a resource (buckets and objects), you can grant permissions by specifying a collection of Grants (see [Access Control List \(ACL\) Overview \(p. 363\)](#)) in your request. For each Grant, you create an S3Grant object explicitly specifying the grantee and the permission.

For example, the following C# code sample sends a [PUT Bucket](#) request to create a bucket and then a [PutObject](#) request to put a new object in the new bucket. In the request, the code specifies permissions for full control for the owner and WRITE permission for the Amazon S3 **Log Delivery** group. The [PutObject](#) call includes the object data in the request body and the ACL grants in the request headers (see [PUT Object](#)).

```
static string bucketName = "**** Provide existing bucket name ****";
static string newBucketName = "**** Provide a name for a new bucket ****";
static string newKeyName = "**** Provide a name for a new key ****";

IAmazonS3 client;
client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1);

// Retrieve ACL from one of the owner's buckets
S3AccessControlList acl = client.GetACL(new GetACLRequest
{
    BucketName = bucketName,
}).AccessControlList;

// Describe grant for full control for owner.
S3Grant grant1 = new S3Grant
{
    Grantee = new S3Grantee { CanonicalUser = acl.Owner.Id },
    Permission = S3Permission.FULL_CONTROL
};

// Describe grant for write permission for the LogDelivery group.
S3Grant grant2 = new S3Grant
{
    Grantee = new S3Grantee { URI = "http://acs.amazonaws.com/groups/s3/LogDelivery" },
    Permission = S3Permission.WRITE
};

PutBucketRequest request = new PutBucketRequest()
{
    BucketName = newBucketName,
    BucketRegion = S3Region.US,
    Grants = new List<S3Grant> { grant1, grant2 }
};
PutBucketResponse response = client.PutBucket(request);

PutObjectRequest objectRequest = new PutObjectRequest()
{
    ContentBody = "Object data for simple put.",
    BucketName = newBucketName,
    Key = newKeyName,
    Grants = new List<S3Grant> { grant1 }
```

```
};

PutObjectResponse objectResponse = client.PutObject(objectRequest);
```

For more information about uploading objects, see [Working with Amazon S3 Objects \(p. 78\)](#).

In the preceding code sample, for each `S3Grant` you explicitly identify a grantee and permission. Alternatively, you can specify a canned (predefined) ACL (see [Canned ACL \(p. 367\)](#)) in your request when creating a resource. The following C# code sample creates an object and specifies a `LogDeliveryWrite` canned ACL in the request to grant the **Log Delivery** group WRITE and READ_ACP permissions on the bucket.

```
static string newBucketName = "**** Provide existing bucket name ****";
static string keyName = "**** Provide key name ****";

IAmazonS3 client;
client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1);

PutBucketRequest request = new PutBucketRequest()
{
    BucketName = newBucketName,
    BucketRegion = S3Region.US,
    // Add canned ACL.
    CannedACL = S3CannedACL.LogDeliveryWrite
};
PutBucketResponse response = client.PutBucket(request);
```

For information about the underlying REST API, go to [PUT Bucket](#).

Updating ACL on an Existing Resource

You can set an ACL on an existing object or a bucket by calling the `AmazonS3Client.PutACL` method. You create an instance of the `S3AccessControlList` class with a list of ACL grants and include the list in the `PutACL` request.

The following C# code sample reads an existing ACL first, using the `AmazonS3Client.GetACL` method, add new grants to it, and then sets the revised ACL on the object.

```
static string bucketName = "**** Provide existing bucket name ****";
static string keyName = "**** Provide key name ****";

IAmazonS3 client;
client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1);

// Retrieve ACL for object
S3AccessControlList acl = client.GetACL(new GetACLRequest
{
    BucketName = bucketName,
    Key = keyName
}).AccessControlList;

// Retrieve owner
Owner owner = acl.Owner;

// Clear existing grants.
acl.Grants.Clear();
```

```
// First, add grant to reset owner's full permission
// (previous clear statement removed all permissions).
S3Grant grant0 = new S3Grant
{
    Grantee = new S3Grantee { CanonicalUser = acl.Owner.Id }
};
acl.AddGrant(grant0.Grantee, S3Permission.FULL_CONTROL);

// Describe grant for permission using email address.
S3Grant grant1 = new S3Grant
{
    Grantee = new S3Grantee { EmailAddress = emailAddress },
    Permission = S3Permission.WRITE_ACP
};

// Describe grant for permission to the LogDelivery group.
S3Grant grant2 = new S3Grant
{
    Grantee = new S3Grantee { URI = "http://acs.amazonaws.com/groups/s3/LogDe
livery" },
    Permission = S3Permission.WRITE
};

// Create new ACL.
S3AccessControlList newAcl = new S3AccessControlList
{
    Grants = new List<S3Grant> { grant1, grant2 },
    Owner = owner
};

// Set new ACL.
PutACLResponse response = client.PutACL(new PutACLRequest
{
    BucketName = bucketName,
    Key = keyName,
    AccessControlList = newAcl
});
```

Instead of creating `S3Grant` objects and specifying grantee and permission explicitly, you can also specify a canned ACL in your request. The following C# code sample sets a canned ACL on a new bucket. The sample request specifies an `AuthenticatedRead` canned ACL to grant read access to the Amazon S3 Authenticated Users group.

```
static string newBucketName = "**** Provide new bucket name ****";

IAmazonS3 client;
client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1);

PutBucketRequest request = new PutBucketRequest()
{
    BucketName = newBucketName,
    BucketRegion = S3Region.US,
    // Add canned ACL.
    CannedACL = S3CannedACL.AuthenticatedRead
};
PutBucketResponse response = client.PutBucket(request);
```

An Example

The following C# code example performs the following tasks:

- Create a bucket. In the request, it specifies a log-delivery-write canned ACL, granting write permission to the LogDelivery Amazon S3 group.
- Read the ACL on the bucket.
- Clear existing permissions and add new the permission to the ACL.
- Call PutACL request to add the new ACL to the bucket.

For instructions on how to create and test a working example, see [Testing the .NET Code Examples \(p. 547\)](#).

```
using System;
using System.Collections.Specialized;
using System.Configuration;
using Amazon.S3;
using Amazon.S3.Model;
using Amazon.S3.Util;
using System.Collections.Generic;

namespace s3.amazon.com.docsamples
{
    class ManageACLs
    {
        static string bucketName      = "**** Provide existing bucket name ****";
        static string newBucketName  = "**** Provide a name for a new bucket ****";
        static string keyName         = "**** Provide key name ****";
        static string newKeyName     = "**** Provide a name for a new key ****";
        static string emailAddress   = "**** Provide email address ****";

        static IAmazonS3 client;

        public static void Main(string[] args)
        {
            try
            {
                using (client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1))
                {
                    // Add bucket (specify canned ACL).
                    AddBucketWithCannedACL(newBucketName);

                    // Get ACL on a bucket.
                    GetBucketACL(bucketName);

                    // Add (replace) ACL on an object in a bucket.
                    AddACLToExistingObject(bucketName, keyName);

                    Console.WriteLine("Example complete.");
                }
            }
            catch (AmazonS3Exception amazonS3Exception)
            {
                if (amazonS3Exception.ErrorCode != null &&
                    (amazonS3Exception.ErrorCode.Equals("InvalidAccessKeyId") ||
                     amazonS3Exception.ErrorCode.Equals("InvalidSecurity")))

```

```
{  
    Console.WriteLine("Check the provided AWS Credentials.");  
    Console.WriteLine("For service sign up go to ht  
tp://aws.amazon.com/s3");  
}  
else  
{  
    Console.WriteLine(  
        "Error occurred. Message:'{0}' when writing an object"  
        , amazonS3Exception.Message);  
}  
}  
catch (Exception e)  
{  
    Console.WriteLine(e.Message);  
}  
  
Console.WriteLine("Press any key to continue...");  
Console.ReadKey();  
}  
  
static void AddBucketWithCannedACL(string bucketName)  
{  
    PutBucketRequest request = new PutBucketRequest()  
    {  
        BucketName = newBucketName,  
        BucketRegion = S3Region.US,  
        // Add canned ACL.  
        CannedACL = S3CannedACL.LogDeliveryWrite  
    };  
    PutBucketResponse response = client.PutBucket(request);  
}  
  
static void GetBucketACL(string bucketName)  
{  
    GetACLResponse response = client.GetACL(new GetACLRequest  
    {  
        BucketName = bucketName  
    });  
  
    // GetACLResponse response = client.GetACL(request);  
    S3AccessControlList accessControlList = response.AccessControlList;  
    //response.Dispose();  
}  
  
static void AddACLToExistingObject(string bucketName, string keyName)  
{  
    // Retrieve ACL for object  
    S3AccessControlList acl = client.GetACL(new GetACLRequest  
    {  
        BucketName = bucketName,  
        Key = keyName  
    }).AccessControlList;  
  
    // Retrieve owner  
    Owner owner = acl.Owner;  
  
    // Clear existing grants.
```

```
acl.Grants.Clear();

// First, add grant to reset owner's full permission
// (previous clear statement removed all permissions).
S3Grant grant0 = new S3Grant
{
    Grantee = new S3Grantee { CanonicalUser = acl.Owner.Id }
};
acl.AddGrant(grant0.Grantee, S3Permission.FULL_CONTROL);

// Describe grant for permission using email address.
S3Grant grant1 = new S3Grant
{
    Grantee = new S3Grantee { EmailAddress = emailAddress },
    Permission = S3Permission.WRITE_ACP
};

// Describe grant for permission to the LogDelivery group.
S3Grant grant2 = new S3Grant
{
    Grantee = new S3Grantee { URI = "http://acs.amazon
aws.com/groups/s3/LogDelivery" },
    Permission = S3Permission.WRITE
};

// Create new ACL.
S3AccessControlList newAcl = new S3AccessControlList
{
    Grants = new List<S3Grant> { grant1, grant2 },
    Owner = owner
};

// Set new ACL.
PutACLResponse response = client.PutACL(new PutACLRequest
{
    BucketName = bucketName,
    Key = keyName,
    AccessControlList = newAcl
});

// Get and print response.
Console.WriteLine(client.GetACL(new GetACLRequest()
{
    BucketName = bucketName,
    Key = keyName
}));
}
}
}
```

Managing ACLs Using the REST API

For information on the REST API support for managing ACLs, see the following sections in the *Amazon Simple Storage Service API Reference*:

- [GET Bucket acl](#)

- PUT Bucket acl
- GET Object acl
- PUT Object acl
- PUT Object
- PUT Bucket
- PUT Object - Copy
- Initiate Multipart Upload

Protecting Data in Amazon S3

Topics

- [Protecting Data Using Encryption \(p. 380\)](#)
- [Using Reduced Redundancy Storage \(p. 419\)](#)
- [Using Versioning \(p. 422\)](#)

Amazon S3 provides a highly durable storage infrastructure designed for mission-critical and primary data storage. Objects are redundantly stored on multiple devices across multiple facilities in an Amazon S3 region. To help better ensure data durability, Amazon S3 `PUT` and `PUT Object copy` operations synchronously store your data across multiple facilities before returning `SUCCESS`. Once the objects are stored, Amazon S3 maintains their durability by quickly detecting and repairing any lost redundancy.

Amazon S3 also regularly verifies the integrity of data stored using checksums. If Amazon S3 detects data corruption, it is repaired using redundant data. In addition, Amazon S3 calculates checksums on all network traffic to detect corruption of data packets when storing or retrieving data.

Amazon S3's standard storage is:

- Backed with the [Amazon S3 Service Level Agreement](#)
- Designed to provide 99.99999999% durability and 99.99% availability of objects over a given year
- Designed to sustain the concurrent loss of data in two facilities

Amazon S3 further protects your data using versioning. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. By default, requests retrieve the most recently written version. You can retrieve older versions of an object by specifying a version of the object in a request.

Protecting Data Using Encryption

Topics

- [Protecting Data Using Server-Side Encryption \(p. 381\)](#)
- [Protecting Data Using Client-Side Encryption \(p. 408\)](#)

Data protection refers to protecting data while in-transit (as it travels to and from Amazon S3) and at rest (while it is stored on disks in Amazon S3 data centers). You can protect data in transit by using SSL or by using client-side encryption. You have the following options of protecting data at rest in Amazon S3.

- **Use Server-Side Encryption** – You request Amazon S3 to encrypt your object before saving it on disks in its data centers and decrypt it when you download the objects.
- **Use Client-Side Encryption** – You can encrypt data client-side and upload the encrypted data to Amazon S3. In this case, you manage the encryption process, the encryption keys, and related tools.

Protecting Data Using Server-Side Encryption

Server-side encryption is about data encryption at rest—that is, Amazon S3 encrypts your data at the object level as it writes it to disks in its data centers and decrypts it for you when you access it. As long as you authenticate your request and you have access permissions, there is no difference in the way you access encrypted or unencrypted objects. For example, if you share your objects using a presigned URL, that URL works the same way for both encrypted and unencrypted objects.

You have three options depending on how you choose to manage the encryption keys:

- **Use Server-Side Encryption with Amazon S3-Managed Keys (SSE-S3)** – Each object is encrypted with a unique key employing strong multi-factor encryption. As an additional safeguard, it encrypts the key itself with a master key that it regularly rotates. Amazon S3 server-side encryption uses one of the strongest block ciphers available, 256-bit Advanced Encryption Standard (AES-256), to encrypt your data. For more information, see [Protecting Data Using Server-Side Encryption with Amazon S3-Managed Encryption Keys \(SSE-S3\) \(p. 386\)](#).
- **Use Server-Side Encryption with AWS KMS-Managed Keys (SSE-KMS)** – Similar to SSE-S3, but with some additional benefits along with some additional charges for using this service. There are separate permissions for the use of an envelope key (that is, a key that protects your data's encryption key) that provides added protection against unauthorized access of your objects in S3. SSE-KMS also provides you with an audit trail of when your key was used and by whom. Additionally, you have the option to create and manage encryption keys yourself, or use a default key that is unique to you, the service you're using, and the region you're working in. For more information, see [Protecting Data Using Server-Side Encryption with AWS KMS-Managed Keys \(SSE-KMS\) \(p. 381\)](#).
- **Use Server-Side Encryption with Customer-Provided Keys (SSE-C)** – You manage encryption/decryption of your data, the encryption keys, and related tools. For more information, see [Protecting Data Using Server-Side Encryption with Customer-Provided Encryption Keys \(SSE-C\) \(p. 394\)](#).

Note

When you list objects in your bucket, the list API will return a list of all objects, regardless of whether they are encrypted.

Protecting Data Using Server-Side Encryption with AWS KMS-Managed Keys (SSE-KMS)

Server-side encryption is about protecting data at rest. AWS Key Management Service (KMS) is a service that combines secure, highly available hardware and software to provide a key management system scaled for the cloud. AWS KMS uses customer master keys (CMKs) to encrypt your Amazon S3 objects. You use AWS KMS via the Encryption Keys section in the IAM console or via AWS KMS APIs to centrally create encryption keys, define the policies that control how keys can be used, and audit key usage to prove they are being used correctly. You can use these keys to protect your data in Amazon S3 buckets.

The first time you add an SSE-KMS–encrypted object to a bucket in a region, a default CMK is created for you automatically. This key is used for SSE-KMS encryption unless you select a CMK that you created separately using AWS Key Management Service. Creating your own CMK gives you more flexibility,

including the ability to create, rotate, disable, and define access controls, and to audit the encryption keys used to protect your data.

For more information, see [What is AWS Key Management Service?](#) in the *AWS Key Management Service Developer Guide*. If you use AWS KMS, there are additional charges for using AWS-KMS keys. For more information, see [AWS Key Management Service Pricing](#).

Note

If you are uploading or accessing objects encrypted by SSE-KMS, you need to use AWS Signature Version 4 for added security. For more information on how to do this using an AWS SDK, see [Specifying Signature Version in Request Authentication](#).

The highlights of SSE-KMS are:

- You can choose to create and manage encryption keys yourself, or you can to use your default service key uniquely generated on a customer by service by region level.
- The ETag in the response is not the MD5 of the object data.
- The data keys used to encrypt your data are also encrypted and stored alongside the data they protect.
- Auditable master keys can be created, rotated, and disabled from the IAM console.
- The security controls in AWS KMS can help you meet encryption-related compliance requirements.

Amazon S3 supports bucket policies that you can use if you require server-side encryption for all objects that are stored in your bucket. For example, the following bucket policy denies upload object (`s3:PutObject`) permission to everyone if the request does not include the `x-amz-server-side-encryption` header requesting server-side encryption with SSE-KMS.

```
{  
    "Version": "2012-10-17",  
    "Id": "PutObjPolicy",  
    "Statement": [ {  
        "Sid": "DenyUnEncryptedObjectUploads",  
        "Effect": "Deny",  
        "Principal": "*",  
        "Action": "s3:PutObject",  
        "Resource": "arn:aws:s3:::YourBucket/*",  
        "Condition": {  
            "StringNotEquals": {  
                "s3:x-amz-server-side-encryption": "aws:kms"  
            }  
        }  
    }  
]
```

Important

All GET and PUT requests for an object protected by AWS KMS will fail if not made via SSL or by using SigV4.

SSE-KMS encrypts only the object data. Any object metadata is not encrypted.

Using AWS Key Management Service in the Amazon S3 Management Console

For more information about using KMS-Managed Encryption Keys in the Amazon S3 Management Console, go to [Uploading Objects into Amazon S3](#) in the *Amazon Simple Storage Service User Guide*.

API Support for AWS Key Management Service in Amazon S3

The object creation REST APIs (see [Specifying the AWS Key Management Service in Amazon S3 Using the REST API \(p. 385\)](#)) provide a request header, `x-amz-server-side-encryption` that you can use to request SSE-KMS with the value of `aws:kms`. There's also `x-amz-server-side-encryption-aws-kms-key-id`, which specifies the ID of the AWS KMS master encryption key that was used for the object.

The following Amazon S3 APIs support these request headers.

- PUT operation — When uploading data using the PUT API (see [PUT Object](#)), you can specify these request headers.
- Initiate Multipart Upload — When uploading large objects using the multipart upload API, you can specify these headers. You specify these headers in the initiate request (see [Initiate Multipart Upload](#)).
- POST operation — When using a POST operation to upload an object (see [POST Object](#)), instead of the request headers, you provide the same information in the form fields.
- COPY operation — When you copy an object (see [PUT Object - Copy](#)), you have both a source object and a target object. When you pass SSE-KMS headers with the COPY operation, they will be applied only to the target object.

The AWS SDKs also provide wrapper APIs for you to request SSE-KMS with Amazon S3.

Specifying the AWS Key Management Service in Amazon S3 Using the AWS SDKs

Topics

- [AWS SDK for Java \(p. 383\)](#)
- [AWS SDK for .NET \(p. 384\)](#)

When using AWS SDKs, you can request Amazon S3 to use AWS Key Management Service (AWS KMS)-managed encryption keys. This section provides examples of using the AWS SDKs for Java and .NET. For information about other SDKs, go to [Sample Code and Libraries](#).

AWS SDK for Java

This section explains various Amazon S3 operations using the AWS SDK for Java and how you use the AWS KMS-managed encryption keys.

Put Operation

When uploading an object using the AWS SDK for Java, you can request Amazon S3 to use an AWS KMS-managed encryption key by adding the `SSEAwsKeyManagementParams` property as shown in the following request:

```
PutObjectRequest putRequest = new PutObjectRequest(bucketName,  
    keyName, file).withSSEAwsKeyManagementParams(new SSEAwsKeyManagementParams());
```

In this case, Amazon S3 uses the default master key (see [Protecting Data Using Server-Side Encryption with AWS KMS-Managed Keys \(SSE-KMS\) \(p. 381\)](#)). You can optionally create your own key and specify that in the request.

```
PutObjectRequest putRequest = new PutObjectRequest(bucketName,  
    keyName, file).withSSEAwsKeyManagementParams(new SSEAwsKeyManagementParams(key  
ID));
```

For more information about creating keys, go to [Programming the AWS KMS API](#) in the *AWS Key Management Service Developer Guide*.

For working code examples of uploading an object, see the following topics. You will need to update those code examples and provide encryption information as shown in the preceding code fragment.

- For uploading an object in a single operation, see [Upload an Object Using the AWS SDK for Java \(p. 146\)](#)
- For a multipart upload, see the following topics:
 - Using high-level multipart upload API, see [Upload a File \(p. 156\)](#)
 - If you are using the low-level multipart upload API, see [Upload a File \(p. 161\)](#)

Copy Operation

When copying objects, you add the same request properties (`ServerSideEncryptionMethod` and `ServerSideEncryptionKeyManagementServiceKeyId`) to request Amazon S3 to use an AWS KMS–managed encryption key. For more information about copying objects, see [Copying Objects \(p. 199\)](#).

Pre-signed URLs

When creating a pre-signed URL for an object encrypted using an AWS KMS–managed encryption key, you must explicitly specify Signature Version 4:

```
ClientConfiguration clientConfiguration = new ClientConfiguration();
clientConfiguration.setSignerOverride("AWSS3V4SignerType");
AmazonS3Client s3client = new AmazonS3Client(
    new ProfileCredentialsProvider(), clientConfiguration);
...
```

For a code example, see [Generate a Pre-signed Object URL using AWS SDK for Java \(p. 140\)](#).

AWS SDK for .NET

This section explains various Amazon S3 operations using the AWS SDK for .NET and how you use the AWS KMS–managed encryption keys.

Put Operation

When uploading an object using the AWS SDK for .NET, you can request Amazon S3 to use an AWS KMS–managed encryption key by adding the `ServerSideEncryptionMethod` property as shown in the following request:

```
PutObjectRequest putRequest = new PutObjectRequest
{
    BucketName = bucketName,
    Key = keyName,
    // other properties.
    ServerSideEncryptionMethod = ServerSideEncryptionMethod.AWSKMS
};
```

In this case, Amazon S3 uses the default master key (see [Protecting Data Using Server-Side Encryption with AWS KMS–Managed Keys \(SSE-KMS\) \(p. 381\)](#)). You can optionally create your own key and specify that in the request.

```
PutObjectRequest putRequest1 = new PutObjectRequest
{
```

```
BucketName = bucketName,  
Key = keyName,  
// other properties.  
ServerSideEncryptionMethod = ServerSideEncryptionMethod.AWSKMS,  
ServerSideEncryptionKeyManagementServiceKeyId = keyId  
};
```

For more information about creating keys, go to [Programming the AWS KMS API](#) in the *AWS Key Management Service Developer Guide*.

For working code examples of uploading an object, see the following topics. You will need to update these code examples and provide encryption information as shown in the preceding code fragment.

- For uploading an object in a single operation, see [Upload an Object Using the AWS SDK for .NET \(p. 147\)](#)
- For multipart upload see the following topics:
 - Using high-level multipart upload API, see [Upload a File \(p. 166\)](#)
 - Using low-level multipart upload API, see [Upload a File \(p. 175\)](#)

Copy Operation

When copying objects, you add the same request properties (`ServerSideEncryptionMethod` and `ServerSideEncryptionKeyManagementServiceKeyId`) to request Amazon S3 to use an AWS KMS–managed encryption key. For more information about copying objects, see [Copying Objects \(p. 199\)](#).

Pre-signed URLs

When creating a pre-signed URL for an object encrypted using an AWS KMS–managed encryption key, you must explicitly specify Signature Version 4:

```
AWSConfigs.S3Config.UseSignatureVersion4 = true;
```

For a code example, see [Generate a Pre-signed Object URL using AWS SDK for .NET \(p. 143\)](#).

Specifying the AWS Key Management Service in Amazon S3 Using the REST API

At the time of object creation—that is, when you are uploading a new object or making a copy of an existing object—you can specify the use of server-side encryption with AWS KMS–managed encryption keys (SSE-KMS) to encrypt your data by adding the `x-amz-server-side-encryption` header to the request. Set the value of the header to the encryption algorithm `aws:kms`. Amazon S3 confirms that your object is stored using SSE-KMS by returning the response header `x-amz-server-side-encryption`.

The following REST upload APIs accept the `x-amz-server-side-encryption` request header.

- [PUT Object](#)
- [PUT Object - Copy](#)
- [POST Object](#)
- [Initiate Multipart Upload](#)

When uploading large objects using the multipart upload API, you can specify SSE-KMS by adding the `x-amz-server-side-encryption` header to the Initiate Multipart Upload request with the value of `aws:kms`. When copying an existing object, regardless of whether the source object is encrypted or not, the destination object is not encrypted unless you explicitly request server-side encryption.

The response headers of the following REST APIs return the `x-amz-server-side-encryption` header when an object is stored using server-side encryption.

- [PUT Object](#)
- [PUT Object - Copy](#)
- [POST Object](#)
- [Initiate Multipart Upload](#)
- [Upload Part](#)
- [Upload Part - Copy](#)
- [Complete Multipart Upload](#)
- [Get Object](#)
- [Head Object](#)

Note

Encryption request headers should not be sent for `GET` requests and `HEAD` requests if your object uses SSE-KMS or you'll get an HTTP 400 BadRequest error.

Protecting Data Using Server-Side Encryption with Amazon S3-Managed Encryption Keys (SSE-S3)

Server-side encryption is about protecting data at rest. Server-side encryption with Amazon S3-managed encryption keys (SSE-S3) employs strong multi-factor encryption. Amazon S3 encrypts each object with a unique key. As an additional safeguard, it encrypts the key itself with a master key that it regularly rotates. Amazon S3 server-side encryption uses one of the strongest block ciphers available, 256-bit Advanced Encryption Standard (AES-256), to encrypt your data.

Amazon S3 supports bucket policies that you can use if you require server-side encryption for all objects that are stored in your bucket. For example, the following bucket policy denies upload object (`s3:PutObject`) permission to everyone if the request does not include the `x-amz-server-side-encryption` header requesting server-side encryption.

```
{  
    "Version": "2012-10-17",  
    "Id": "PutObjPolicy",  
    "Statement": [  
        {  
            "Sid": "DenyUnEncryptedObjectUploads",  
            "Effect": "Deny",  
            "Principal": "*",  
            "Action": "s3:PutObject",  
            "Resource": "arn:aws:s3:::YourBucket/*",  
            "Condition": {  
                "StringNotEquals": {  
                    "s3:x-amz-server-side-encryption": "AES256"  
                }  
            }  
        }  
    ]  
}
```

Server-side encryption encrypts only the object data. Any object metadata is not encrypted.

API Support for Server-Side Encryption

The object creation REST APIs (see [Specifying Server-Side Encryption Using the REST API \(p. 393\)](#)) provide a request header, `x-amz-server-side-encryption` that you can use to request server-side encryption.

The following Amazon S3 APIs support these headers.

- PUT operation — When uploading data using the PUT API (see [PUT Object](#)), you can specify these request headers.
- Initiate Multipart Upload — When uploading large objects using the multipart upload API, you can specify these headers. You specify these headers in the initiate request (see [Initiate Multipart Upload](#)).
- POST operation — When using a POST operation to upload an object (see [POST Object](#)), instead of the request headers, you provide the same information in the form fields.
- COPY operation — When you copy an object (see [PUT Object - Copy](#)), you have both a source object and a target object.

The AWS SDKs also provide wrapper APIs for you to request server-side encryption. You can also use the AWS Management Console to upload objects and request server-side encryption.

Specifying Server-Side Encryption Using the AWS SDK for Java

When using the AWS SDK for Java to upload an object, you can use the `ObjectMetadata` property of the `PutObjectRequest` to set the `x-amz-server-side-encryption` request header (see [Specifying Server-Side Encryption Using the REST API \(p. 393\)](#)). When you call the `PutObject` method of the `AmazonS3` client as shown in the following Java code sample, Amazon S3 encrypts and saves the data.

```
File file = new File(uploadFileName);
PutObjectRequest putRequest = new PutObjectRequest(
    bucketName, keyName, file);

// Request server-side encryption.
ObjectMetadata objectMetadata = new ObjectMetadata();
objectMetadata.setSSEAlgorithm(ObjectMetadata.AES_256_SERVER_SIDE_ENCRYPTION);

putRequest.setMetadata(objectMetadata);

PutObjectResult response = s3client.putObject(putRequest);
System.out.println("Uploaded object encryption status is " +
    response.getSSEAlgorithm());
```

In response, Amazon S3 returns the encryption algorithm used for encrypting your object data, which you can check using the `getSSEAlgorithm` method.

For a working sample that shows how to upload an object, see [Upload an Object Using the AWS SDK for Java \(p. 146\)](#). For server-side encryption, add the `ObjectMetadata` property to your request.

When uploading large objects using multipart upload API, you can request server-side encryption for the object that you are uploading.

- When using the low-level multipart upload API (see [Upload a File \(p. 161\)](#)) to upload a large object, you can specify server-side encryption when you initiate the multipart upload. That is, you add the `ObjectMetadata` property by calling the `InitiateMultipartUploadRequest.setObjectMetadata` method.

- When using the high-level multipart upload API (see [Using the High-Level Java API for Multipart Upload \(p. 156\)](#)), the `TransferManager` class provides methods to upload objects. You can call any of the upload methods that take `ObjectMetadata` as a parameter.

Determining the Encryption Algorithm Used

To determine the encryption state of an existing object, you can retrieve the object metadata as shown in the following Java code sample.

```
GetObjectMetadataRequest request2 =  
    new GetObjectMetadataRequest(bucketName, keyName);  
  
ObjectMetadata metadata = s3client.getObjectMetadata(request2);  
  
System.out.println("Encryption algorithm used: " +  
    metadata.getSSEAlgorithm());
```

If server-side encryption is not used for the object that is stored in Amazon S3, the method returns null.

Changing Server-Side Encryption of an Existing Object (Copy Operation)

To change the encryption state of an existing object, you make a copy of the object and delete the source object. Note that, by default, the copy API will not encrypt the target, unless you explicitly request server-side encryption. You can request the encryption of the target object by using the `ObjectMetadata` property to specify server-side encryption in the `CopyObjectRequest` as shown in the following Java code sample.

```
CopyObjectRequest copyObjRequest = new CopyObjectRequest(  
    sourceBucket, sourceKey, targetBucket, targetKey);  
  
// Request server-side encryption.  
ObjectMetadata objectMetadata = new ObjectMetadata();  
objectMetadata.setSSEAlgorithm(ObjectMetadata.AES_256_SERVER_SIDE_ENCRYPTION);  
  
copyObjRequest.setNewObjectMetadata(objectMetadata);  
  
CopyObjectResult response = s3client.copyObject(copyObjRequest);  
System.out.println("Copied object encryption status is " +  
    response.getSSEAlgorithm());
```

For a working sample of how to copy an object, see [Copy an Object Using the AWS SDK for Java \(p. 201\)](#). You can specify server-side encryption in the `CopyObjectRequest` object as shown in the preceding code sample.

Specifying Server-Side Encryption Using the AWS SDK for .NET

When using the AWS SDK for .NET to upload an object, you can use the `WithServerSideEncryptionMethod` property of `PutObjectRequest` to set the `x-amz-server-side-encryption` request header (see [Specifying Server-Side Encryption Using the REST API \(p. 393\)](#)). When you call the `PutObject` method of the `AmazonS3` client as shown in the following C# code sample, Amazon S3 encrypts and saves the data.

```
static AmazonS3 client;  
client = new AmazonS3Client(accessKeyID, secretAccessKeyID);
```

```
PutObjectRequest request = new PutObjectRequest();
request.WithContentBody("Object data for simple put.")
    .WithBucketName(bucketName)
    .WithKey(keyName)
    .WithServerSideEncryptionMethod(ServerSideEncryptionMethod.AES256);

S3Response response = client.PutObject(request);

// Check the response header to determine if the object is encrypted.
ServerSideEncryptionMethod destinationObjectEncryptionStatus = response.Server
SideEncryptionMethod;
```

In response, Amazon S3 returns the encryption algorithm that is used to encrypt your object data, which you can check using the `ServerSideEncryptionMethod` property.

For a working sample of how to upload an object, see [Upload an Object Using the AWS SDK for .NET \(p. 147\)](#). For server-side encryption, set the `ServerSideEncryptionMethod` property by calling the `WithServerSideEncryptionMethod` method.

To upload large objects using the multipart upload API, you can specify server-side encryption for the objects that you are uploading.

- When using the low-level multipart upload API (see [Using the Low-Level .NET API for Multipart Upload \(p. 175\)](#)) to upload a large object, you can specify server-side encryption in your `InitiateMultipartUpload` request. That is, you set the `ServerSideEncryptionMethod` property to your `InitiateMultipartUploadRequest` by calling the `WithServerSideEncryptionMethod` method.
- When using the high-level multipart upload API (see [Using the High-Level .NET API for Multipart Upload \(p. 166\)](#)), the `TransferUtility` class provides methods (`Upload` and `UploadDirectory`) to upload objects. In this case, you can request server-side encryption using the `TransferUtilityUploadRequest` and `TransferUtilityUploadDirectoryRequest` objects.

Determining the Encryption Algorithm Used

To determine the encryption state of an existing object, you can retrieve the object metadata as shown in the following C# code sample.

```
AmazonS3 client;
client = new AmazonS3Client(accessKeyID, secretAccessKeyID);

ServerSideEncryptionMethod objectEncryption;

GetObjectMetadataRequest metadataRequest = new GetObjectMetadataRequest()
    .WithBucketName(bucketName)
    .WithKey(keyName);

objectEncryption = client.GetObjectMetadata(metadataRequest)
    .ServerSideEncryptionMethod;
```

The encryption algorithm is specified with an enum. If the stored object is not encrypted (default behavior), then the `ServerSideEncryptionMethod` property of the object will default to `None`.

Changing Server-Side Encryption of an Existing Object (Copy Operation)

To change the encryption state of an existing object, you can make a copy of the object and delete the source object. Note that, by default, the copy API will not encrypt the target, unless you explicitly request server-side encryption of the destination object. The following C# code sample makes a copy of an object. The request explicitly specifies server-side encryption for the destination object.

```
AmazonS3 client;
client = new AmazonS3Client(accessKeyID, secretAccessKeyID);

CopyObjectResponse response = client.CopyObject(new CopyObjectRequest()
    .WithSourceBucket(sourceBucketName)
    .WithSourceKey(sourceObjetKey)
    .WithDestinationBucket(targetBucketName)
    .WithDestinationKey(targetObjectKey)
    .WithServerSideEncryptionMethod(ServerSideEncryptionMethod.AES256)
);
// Check the response header to determine if the object is encrypted.
ServerSideEncryptionMethod destinationObjectEncryptionStatus = response.Server
SideEncryptionMethod;
```

For a working sample of how to copy an object, see [Copy an Object Using the AWS SDK for .NET \(p. 202\)](#). You can specify server-side encryption in the `CopyObjectRequest` object as shown in the preceding code sample.

Specifying Server-Side Encryption Using the AWS SDK for PHP

This topic guides you through using classes from the AWS SDK for PHP to add server-side encryption to objects you are uploading to Amazon S3.

Note

This topic assumes that you are already following the instructions for [Using the AWS SDK for PHP and Running PHP Examples \(p. 547\)](#) and have the AWS SDK for PHP properly installed.

You can use the `Aws\S3\S3Client::putObject()` method to upload an object to Amazon S3. For a working sample of how to upload an object, see [Upload an Object Using the AWS SDK for PHP \(p. 150\)](#).

To add the `x-amz-server-side-encryption` request header (see [Specifying Server-Side Encryption Using the REST API \(p. 393\)](#)) to your upload request, specify the array parameter's `ServerSideEncryption` key with the value `AES256` as shown in the following PHP code sample.

```
use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';
// $filepath should be absolute path to a file on disk
$filepath = '*** Your File Path ***';

// Instantiate the client.
$s3 = S3Client::factory();

// Upload a file with server-side encryption.
$result = $s3->putObject(array(
    'Bucket'          => $bucket,
    'Key'             => $keyname,
    'SourceFile'      => $filepath,
```

```
    'ServerSideEncryption' => 'AES256',
)) ;
```

In response, Amazon S3 returns the `x-amz-server-side-encryption` header with the value of the encryption algorithm used to encrypt your object data.

To upload large objects using the multipart upload API, you can specify server-side encryption for the objects that you are uploading.

- When using the low-level multipart upload API (see [Using the AWS SDK for PHP Low-Level API for Multipart Upload \(p. 184\)](#)), you can specify server-side encryption when you call the `Aws\S3\S3Client::createMultipartUpload()` method. To add the `x-amz-server-side-encryption` request header to your request, specify the `array` parameter's `ServerSideEncryption` key with the value `AES256`.
- When using the high-level multipart upload, you can specify server-side encryption using the `Aws\S3\Model\MultipartUpload\UploadBuilder::setOption()` method like `setOption('ServerSideEncryption', 'AES256')`. For an example of using the `setOption()` method with the high-level `UploadBuilder`, see [Using the AWS SDK for PHP High-Level Abstractions for Multipart Upload \(p. 181\)](#).

Determining Encryption Algorithm Used

To determine the encryption state of an existing object, retrieve the object metadata by calling the `Aws\S3\S3Client::headObject()` method as shown in the following PHP code sample.

```
use Aws\S3\S3Client;

$bucket = '*** Your Bucket Name ***';
$keyname = '*** Your Object Key ***';

// Instantiate the client.
$s3 = S3Client::factory();

// Check which server-side encryption algorithm is used.
$result = $s3->headObject(array(
    'Bucket' => $bucket,
    'Key'     => $keyname,
));
echo $result['ServerSideEncryption'];
```

Changing Server-Side Encryption of an Existing Object (Copy Operation)

To change the encryption state of an existing object, make a copy of the object using the `Aws\S3\S3Client::copyObject()` method and delete the source object. Note that by default `copyObject()` will not encrypt the target, unless you explicitly request server-side encryption of the destination object using the `array` parameter's `ServerSideEncryption` key with the value `AES256`. The following PHP code sample makes a copy of an object and adds server-side encryption to the copied object.

```
use Aws\S3\S3Client;

$sourceBucket = '*** Your Source Bucket Name ***';
$sourceKeyname = '*** Your Source Object Key ***';

$targetBucket = '*** Your Target Bucket Name ***';
$targetKeyname = '*** Your Target Object Key ***';
```

```
// Instantiate the client.  
$s3 = S3Client::factory();  
  
// Copy an object and add server-side encryption.  
$result = $s3->copyObject(array(  
    'Bucket'          => $targetBucket,  
    'Key'             => $targetKeyname,  
    'CopySource'      => "{$sourceBucket}/{${sourceKeyname}}",  
    'ServerSideEncryption' => 'AES256',  
));
```

For a working sample of how to copy an object, see [Copy an Object Using the AWS SDK for PHP \(p. 205\)](#).

Related Resources

- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client Class](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::factory\(\) Method](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::copyObject\(\) Method](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::createMultipartUpload\(\) Method](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::headObject\(\) Method](#)
- [AWS SDK for PHP for Amazon S3 Aws\S3\S3Client::putObject\(\) Method](#)
- [Aws\S3\Model\MultipartUpload\UploadBuilder::setOption\(\) Method](#)
- [AWS SDK for PHP for Amazon S3](#)
- [AWS SDK for PHP Documentation](#)

Specifying Server-Side Encryption Using the AWS SDK for Ruby

When using the AWS SDK for Ruby to upload an object, you can specify that the object be stored at rest encrypted by specifying an options hash `server_side_encryption` in the `#write` instance method. When you read the object back, it is automatically decrypted.

The following Ruby script sample demonstrates how to specify that a file uploaded to Amazon S3 be encrypted at rest.

```
# Upload a file and set server-side encryption.  
key_name = File.basename(file_name)  
s3.buckets[bucket_name].objects[key_name].write(:file => file_name, :server_side_encryption => :aes256)
```

For a working sample that shows how to upload an object, see [Upload an Object Using the AWS SDK for Ruby \(p. 152\)](#).

Determining the Encryption Algorithm Used

To check the encryption algorithm that is used for encrypting an object data at rest, use the `#server_side_encryption` method of the `S3Object` instance. The following code sample demonstrates how to determine the encryption state of an existing object.

```
# Determine server-side encryption of an object.  
enc = s3.buckets[bucket_name].objects[key_name].server_side_encryption  
enc_state = (enc != nil) ? enc : "not set"  
puts "Encryption of #{key_name} is #{enc_state}."
```

If server-side encryption is not used for the object that is stored in Amazon S3, the method returns null.

Changing Server-Side Encryption of an Existing Object (Copy Operation)

To change the encryption state of an existing object, make a copy of the object and delete the source object. The Ruby API `S3Object` class has `#copy_from` and `#copy_to` methods that you can use to copy objects. Note that, by default, the copy methods will not encrypt the target, unless you explicitly request server-side encryption. You can request the encryption of the target object by specifying the `server_side_encryption` value in the options hash argument as shown in the following Ruby code sample. The code sample demonstrates how to use the `#copy_to` method.

```
s3 = AWS::S3.new

# Upload a file and set server-side encryption.
bucket1 = s3.buckets[source_bucket]
bucket2 = s3.buckets[target_bucket]
obj1 = bucket1.objects[source_key]
obj2 = bucket2.objects[target_key]

obj1.copy_to(obj2, :server_side_encryption => :aes256)
```

For a working sample of how to copy an object, see [Copy an Object Using the AWS SDK for Ruby \(p. 208\)](#).

Specifying Server-Side Encryption Using the REST API

At the time of object creation—that is, when you are uploading a new object or making a copy of an existing object—you can specify if you want Amazon S3 to encrypt your data by adding the `x-amz-server-side-encryption` header to the request. Set the value of the header to the encryption algorithm `AES256` that Amazon S3 supports. Amazon S3 confirms that your object is stored using server-side encryption by returning the response header `x-amz-server-side-encryption`.

The following REST upload APIs accept the `x-amz-server-side-encryption` request header.

- [PUT Object](#)
- [PUT Object - Copy](#)
- [POST Object](#)
- [Initiate Multipart Upload](#)

When uploading large objects using the multipart upload API, you can specify server-side encryption by adding the `x-amz-server-side-encryption` header to the Initiate Multipart Upload request. When copying an existing object, regardless of whether the source object is encrypted or not, the destination object is not encrypted unless you explicitly request server-side encryption.

The response headers of the following REST APIs return the `x-amz-server-side-encryption` header when an object is stored using server-side encryption.

- [PUT Object](#)
- [PUT Object - Copy](#)
- [POST Object](#)
- [Initiate Multipart Upload](#)
- [Upload Part](#)
- [Upload Part - Copy](#)
- [Complete Multipart Upload](#)
- [Get Object](#)

- Head Object

Note

Encryption request headers should not be sent for GET requests and HEAD requests if your object uses SSE-S3 or you'll get an HTTP 400 BadRequest error.

Specifying Server-Side Encryption Using the AWS Management Console

When uploading an object using the AWS Management Console, you can specify server-side encryption. For an example of how to upload an object, go to [Uploading Objects into Amazon S3](#).

When you copy an object using the AWS Management Console, the console copies the object as is. That is, if the copy source is encrypted, the target object will be encrypted. For an example of how to copy an object using the console, go to [Copying an Object](#). The console also allows you to update properties of one or more objects. For example, you can select one or more objects and select server-side encryption

Protecting Data Using Server-Side Encryption with Customer-Provided Encryption Keys (SSE-C)

Server-side encryption is about protecting data at rest. Using server-side encryption with customer-provided encryption keys (SSE-C) allows you to set your own encryption keys. With the encryption key you provide as part of your request, Amazon S3 manages both the encryption, as it writes to disks, and decryption, when you access your objects. Therefore, you don't need to maintain any code to perform data encryption and decryption. The only thing you do is manage the encryption keys you provide.

When you upload an object, Amazon S3 uses the encryption key you provide to apply AES-256 encryption to your data and removes the encryption key from memory.

Important

Amazon S3 does not store the encryption key you provide. Instead, we store a randomly salted HMAC value of the encryption key in order to validate future requests. The salted HMAC value cannot be used to derive the value of the encryption key or to decrypt the contents of the encrypted object. That means, if you lose the encryption key, you lose the object.

When you retrieve an object, you must provide the same encryption key as part of your request. Amazon S3 first verifies that the encryption key you provided matches, and then decrypts the object before returning the object data to you.

The highlights of SSE-C are:

- You must use https.

Important

Amazon S3 will reject any requests made over http when using SSE-C. For security considerations, we recommend you consider any key you send erroneously using http to be compromised. You should discard the key, and rotate as appropriate.

- The ETag in the response is not the MD5 of the object data.
- You manage a mapping of which encryption key was used to encrypt which object. Amazon S3 does not store encryption keys. You are responsible for tracking which encryption key you provided for which object.
 - If your bucket is versioning-enabled, each object version you upload using this feature can have its own encryption key. You are responsible for tracking which encryption key was used for which object version.
 - Because you manage encryption keys on the client side, you manage any additional safeguards, such as key rotation, on the client side.

Caution

If you lose the encryption key any GET request for an object without its encryption key will fail, and you lose the object.

Using SSE-C

When using server-side encryption with customer-provided encryption keys (SSE-C), you must provide encryption key information using the following request headers.

Name	Description
<code>x-amz-server-side-encryption-customer-algorithm</code>	Use this header to specify the encryption algorithm. The header value must be "AES256".
<code>x-amz-server-side-encryption-customer-key</code>	Use this header to provide the 256-bit, base64-encoded encryption key for Amazon S3 to use to encrypt or decrypt your data.
<code>x-amz-server-side-encryption-customer-key-MD5</code>	Use this header to provide the base64-encoded 128-bit MD5 digest of the encryption key according to RFC 1321 . Amazon S3 uses this header for a message integrity check to ensure the encryption key was transmitted without error.

You can use AWS SDK wrapper libraries to add these headers to your request. If you need to, you can make the Amazon S3 REST API calls directly in your application.

Note

You cannot use the Amazon S3 console to upload an object and request SSE-C. You also cannot use the console to update (for example, change the storage class or add metadata) an existing object stored using SSE-C.

The following Amazon S3 APIs support these headers.

- GET operation — When retrieving objects using the GET API (see [GET Object](#)), you can specify the request headers. Torrents are not supported for objects encrypted using SSE-C.
- HEAD operation — To retrieve object metadata using the HEAD API (see [HEAD Object](#)), you can specify these request headers.
- PUT operation — When uploading data using the PUT API (see [PUT Object](#)), you can specify these request headers.
- Multipart Upload — When uploading large objects using the multipart upload API, you can specify these headers. You specify these headers in the initiate request (see [Initiate Multipart Upload](#)) and each subsequent part upload request ([Upload Part](#)). For each part upload request, the encryption information must be the same as what you provided in the initiate multipart upload request.
- POST operation — When using a POST operation to upload an object (see [POST Object](#)), instead of the request headers, you provide the same information in the form fields.
- Copy operation — When you copy an object (see [PUT Object - Copy](#)), you have both a source object and a target object. Accordingly, you have the following to consider:
 - If you want the target object encrypted using server-side encryption with AWS-managed keys, you must provide the `x-amz-server-side-encryption` request header.
 - If you want the target object encrypted using SSE-C, you must provide encryption information using the three headers described in the preceding table.
 - If the source object is encrypted using SSE-C, you must provide encryption key information using the following headers so that Amazon S3 can decrypt the object for copying.

Name	Description
<i>x-amz-copy-source-server-side-encryption-customer-algorithm</i>	Include this header to specify the algorithm Amazon S3 should use to decrypt the source object. This value must be <code>AES256</code> .
<i>x-amz-copy-source-server-side-encryption-customer-key</i>	Include this header to provide the base64-encoded encryption key for Amazon S3 to use to decrypt the source object. This encryption key must be the one that you provided Amazon S3 when you created the source object; otherwise, Amazon S3 will not be able to decrypt the object.
<i>x-amz-copy-source-server-side-encryption-customer-key-MD5</i>	Include this header to provide the base64-encoded 128-bit MD5 digest of the encryption key according to RFC 1321 .

Presigned URL and SSE-C

You can generate a presigned URL that can be used for operations such as upload a new object, retrieve an existing object, or object metadata. Presigned URLs support the SSE-C as follows:

- When creating a presigned URL, you must specify the algorithm using the `x-amz-server-side-encryption-customer-algorithm` in the signature calculation.
- When using the presigned URL to upload a new object, retrieve an existing object, or retrieve only object metadata, you must provide all the encryption headers in your client application.

For more information, see the following topics:

- [Specifying Server-Side Encryption with Customer-Provided Encryption Keys Using the AWS Java SDK \(p. 396\)](#)
- [Specifying Server-Side Encryption with Customer-Provided Encryption Keys Using the .NET SDK \(p. 402\)](#)
- [Specifying Server-Side Encryption with Customer-Provided Encryption Keys Using the REST API \(p. 408\)](#)

Specifying Server-Side Encryption with Customer-Provided Encryption Keys Using the AWS Java SDK

The following Java code example illustrates server-side encryption with customer-provided keys (SSE-C) (see [Protecting Data Using Server-Side Encryption with Customer-Provided Encryption Keys \(SSE-C\) \(p. 394\)](#)). The example performs the following operations; each operation shows how you specify SSE-C related headers in the request:

- **Put object** – upload an object requesting server-side encryption using a customer-provided encryption key.
- **Get object** – download the object that you uploaded in the previous step. The example shows that in the Get request you must provide the same encryption information that you provided at the time you uploaded the object, so that Amazon S3 can decrypt the object before returning it.
- **Get object metadata** – The request shows the same encryption information that you specified when creating the object is required to retrieve the object's metadata.
- **Copy object** – This example makes a copy of the previously uploaded object. Because the source object is stored using SSE-C, you must provide the encryption information in your copy request. By

default, the object copy will not be encrypted. But in this example, you request that Amazon S3 store the object copy encrypted by using SSE-C, and therefore you must provide SSE-C encryption information for the target as well.

Note

This example shows how to upload an object in a single operation. When using the multipart upload API to upload large objects, you provide the same encryption information that you provide in your request, as shown in the following example. For multipart upload AWS SDK for Java examples, see [Using the AWS SDK for Java for Multipart Upload \(p. 156\)](#).

The AWS SDK for Java provides the `SSECustomerKey` class for you to add the required encryption information (see [Using SSE-C \(p. 395\)](#)) in your request. You are required to provide only the encryption key. The Java SDK sets the values for the MD5 digest of the encryption key and the algorithm.

For information about how to create and test a working sample, see [Testing the Java Code Examples \(p. 545\)](#).

```
import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.io.InputStreamReader;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;

import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.CopyObjectRequest;
import com.amazonaws.services.s3.model.GetObjectMetadataRequest;
import com.amazonaws.services.s3.model.GetObjectRequest;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.PutObjectRequest;
import com.amazonaws.services.s3.model.S3Object;
import com.amazonaws.services.s3.model.S3ObjectInputStream;
import com.amazonaws.services.s3.model.SSECustomerKey;

public class ServerSideEncryptionUsingClientSideEncryptionKey {
    private static String bucketName      = "**** Provide bucket name ****";
    private static String keyName         = "**** Provide key ****";
    private static String uploadFileName = "**** Provide file name ****";
    private static String targetKeyName  = "**** provide target key ****";
    private static AmazonS3 s3client;

    public static void main(String[] args) throws IOException, NoSuchAlgorithmException {
        s3client = new AmazonS3Client(new ProfileCredentialsProvider());
        try {
            System.out.println("Uploading a new object to S3 from a file\n");
            File file = new File(uploadFileName);
            // Create encryption key.
            SecretKey secretKey = generateSecretKey();
            SSECustomerKey sseKey = new SSECustomerKey(secretKey);
```

```

// 1. Upload object.
uploadObject(file, sseKey);

// 2. Download object.
downloadObject(sseKey);

// 3. Get object metadata (and verify AES256 encryption).
retrieveObjectMetadata(sseKey);

// 4. Copy object (both source and object use SSE-C).
copyObject(sseKey);

} catch (AmazonServiceException ase) {
    System.out.println("Caught an AmazonServiceException, which " +
        "means your request made it " +
        "to Amazon S3, 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 " +
        "an internal error while trying to " +
        "communicate with S3, " +
        "such as not being able to access the network.");
    System.out.println("Error Message: " + ace.getMessage());
}
}

private static void copyObject(SSECustomerKey sseKey) {
    // Create new encryption key for target so it is saved using sse-c
    SecretKey secretKey2 = generateSecretKey();
    SSECustomerKey newSseKey = new SSECustomerKey(secretKey2);

    CopyObjectRequest copyRequest = new CopyObjectRequest(bucketName, keyName,
        bucketName, targetKeyName)
        .withSourceSSECustomerKey(sseKey)
        .withDestinationSSECustomerKey(newSseKey);

    s3client.copyObject(copyRequest);
    System.out.println("Object copied");
}

private static void retrieveObjectMetadata(SSECustomerKey sseKey) {
    GetObjectMetadataRequest getMetadataRequest = new GetObjectMetadataRequest(
        bucketName, keyName)
        .withSSECustomerKey(sseKey);

    ObjectMetadata objectMetadata = s3client.getObjectMetadata(getMetadataRequest);
    System.out.println("object size " + objectMetadata.getContentLength());

    System.out.println("Metadata retrieved");
}

```

```

    private static PutObjectRequest uploadObject(File file, SSECustomerKey
sseKey) {
    // 1. Upload Object.
    PutObjectRequest putObjectRequest = new PutObjectRequest(bucketName,
keyName, file)
        .withSSECustomerKey(sseKey);

    s3client.putObject(putObjectRequest);
    System.out.println("Object uploaded");
    return putObjectRequest;
}

private static void downloadObject(SSECustomerKey sseKey) throws IOException
{
    // Get a range of bytes from an object.
    GetObjectRequest getObjectRequest = new GetObjectRequest(bucketName,
keyName)
        .withSSECustomerKey(sseKey);

    S3Object s3Object = s3client.getObject(getObjectRequest);

    System.out.println("Printing bytes retrieved.");
    displayTextInputStream(s3Object.getObjectContent());
}

private static void displayTextInputStream(S3ObjectInputStream input)
throws IOException {
    // Read one text line at a time and display.
    BufferedReader reader = new BufferedReader(new InputStreamReader(input));

    while (true) {
        String line = reader.readLine();
        if (line == null) break;

        System.out.println("      " + line);
    }
    System.out.println();
}

private static SecretKey generateSecretKey() {
    try {
        KeyGenerator generator = KeyGenerator.getInstance("AES");
        generator.init(256, new SecureRandom());
        return generator.generateKey();
    } catch (Exception e) {
        e.printStackTrace();
        System.exit(-1);
        return null;
    }
}
}

```

Other Amazon S3 Operations and SSE-C

The example in the preceding section shows how to request server-side encryption with customer-provided keys (SSE-C) in the PUT, GET, Head, and Copy operations. This section describes other APIs that support SSE-C.

To upload large objects, you can use multipart upload API (see [Uploading Objects Using Multipart Upload API \(p. 155\)](#)). You can use either high-level or low-level APIs to upload large objects. These APIs support encryption-related headers in the request.

- When using the high-level Transfer-Utility API, you provide the encryption-specific headers in the TransferManager (see [Using the High-Level Java API for Multipart Upload \(p. 156\)](#)).
- When using the low-level API, you provide encryption-related information in the initiate multipart upload request, followed by identical encryption information in the subsequent upload part requests. You do not need to provide any encryption-specific headers in your complete multipart upload request. For examples, see [Using the Low-Level Java API for Multipart Upload \(p. 161\)](#).

The following example uses TransferManager to create objects and shows how to provide SSE-C related information. The example does the following:

- Create an object using the TransferManager.upload method. In the PutObjectRequest instance, you provide encryption key information to request that Amazon S3 store the object encrypted using the customer-provided encryption key.
- Make a copy of the object by calling the TransferManager.copy method. In the CopyObjectRequest, this example requests Amazon S3 to store the object copy also encrypted using a customer-provided encryption key. Because the source object is encrypted using SSE-C, the CopyObjectRequest also provides the encryption key of the source object so Amazon S3 can decrypt the object before it can copy.

```
import java.io.File;
import java.security.SecureRandom;

import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;

import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.model.CopyObjectRequest;
import com.amazonaws.services.s3.model.PutObjectRequest;
import com.amazonaws.services.s3.model.SSECustomerKey;
import com.amazonaws.services.s3.transfer.Copy;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.Upload;

public class ServerSideEncryptionCopyObjectUsingHLwithSSEC {

    public static void main(String[] args) throws Exception {
        String existingBucketName = " *** Provide existing bucket name ***";
        String fileToUpload      = " *** file path ***";
        String keyName           = " *** New object key ***";
        String targetKeyName     = " *** Key name for object copy ***";

        TransferManager tm = new TransferManager(new ProfileCredentialsProvider());

        // 1. first create an object from a file.
        PutObjectRequest putObjectRequest = new PutObjectRequest(existingBucketName, keyName, new File(fileToUpload));

        // we want object stored using SSE-C. So we create encryption key.
        SecretKey secretKey1 = generateSecretKey();
        SSECustomerKey sseCustomerEncryptionKey1 = new SSECustomerKey(secretKey1);
```

```
        putObjectRequest.setSSECustomerKey(sseCustomerEncryptionKey1);
        // now create object.
        //Upload upload = tm.upload(existingBucketName, keyName, new
File(sourceFile));
        Upload upload = tm.upload(putObjectRequest);
        try {
            // Or you can block and wait for the upload to finish
            upload.waitForCompletion();
            //tm.getAmazonS3Client().putObject(putObjectRequest);
            System.out.println("Object created.");
        } catch (AmazonClientException amazonClientException) {
            System.out.println("Unable to upload file, upload was aborted.");
            amazonClientException.printStackTrace();
        }

        // 2. Now make object copy (in the same bucket). Store target using
sse-c.
        CopyObjectRequest copyObjectRequest = new CopyObjectRequest(existing
BucketName, keyName, existingBucketName, targetKeyName);

        SecretKey secretKey2 = generateSecretKey();
        SSECustomerKey sseTargetObjectEncryptionKey = new SSECustomer
Key(secretKey2);

        copyObjectRequest.setSourceSSECustomerKey(sseCustomerEncryptionKey1);

        copyObjectRequest.setDestinationSSECustomerKey(sseTargetObjectEncryp
tionKey);

        // TransferManager processes all transfers asynchronously,
        // so this call will return immediately.
        Copy copy = tm.copy(copyObjectRequest);
        try {
            // Or you can block and wait for the upload to finish
            copy.waitForCompletion();
            System.out.println("Copy complete.");
        } catch (AmazonClientException amazonClientException) {
            System.out.println("Unable to upload file, upload was aborted.");
            amazonClientException.printStackTrace();
        }
    }

    private static SecretKey generateSecretKey() {
        KeyGenerator generator;
        try {
            generator = KeyGenerator.getInstance("AES");
            generator.init(256, new SecureRandom());
            return generator.generateKey();
        } catch (Exception e) {
            e.printStackTrace();
            System.exit(-1);
            return null;
        }
    }
}
```

Specifying Server-Side Encryption with Customer-Provided Encryption Keys Using the .NET SDK

The following C# code example illustrates server-side encryption with customer-provided keys (SSE-C) (see [Protecting Data Using Server-Side Encryption with Customer-Provided Encryption Keys \(SSE-C\) \(p. 394\)](#)). The example performs the following operations, each operation shows how you specify SSE-C-related headers in the request:

- **Put object** – upload an object requesting server-side encryption using customer-provided encryption keys.
- **Get object** – download the object uploaded in the previous step. It shows that the request must provide the same encryption information for Amazon S3 to decrypt the object so that it can return it to you.
- **Get object metadata** – The request shows that the same encryption information you specified when creating the object is required to retrieve the object metadata.
- **Copy object** – This example makes a copy of the previously uploaded object. Because the source object is stored using SSE-C, you must provide encryption information in your copy request. By default, the object copy will not be encrypted. But in this example, you request that Amazon S3 store the object copy encrypted using SSE-C, and therefore you provide encryption-related information for the target as well.

Note

When using multipart upload API to upload large objects, you provide the same encryption information that you provide in your request as shown in the following example. For multipart upload .NET SDK examples, see [Using the AWS SDK for .NET for Multipart Upload \(p. 166\)](#).

For information about how to create and test a working sample, see [Testing the .NET Code Examples \(p. 547\)](#).

```
using System;
using System.IO;
using System.Security.Cryptography;
using Amazon.S3;
using Amazon.S3.Model;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace s3.amazon.com.docsamples
{
    class SSEClientEncryptionKeyObjectOperations
    {
        static string bucketName      = "**** bucket name ****";
        static string keyName         = "**** object key name for new object ****";
        static string copyTargetKeyName = "**** copy operation target object key name ****";

        static IAmazonS3 client;

        public static void Main(string[] args)
        {
            using (client = new AmazonS3Client(Amazon.RegionEndpoint.USWest2))
            {
                try
                {
                    // Create encryption key.
                    Aes aesEncryption = Aes.Create();
```

```
        aesEncryption.KeySize = 256;
        aesEncryption.GenerateKey();
        string base64Key = Convert.ToBase64String(aesEncryption.Key);

        // 1. Upload object.
        PutObjectRequest putObjectRequest = UploadObject(base64Key);

        // 2. Download object (and also verify content is same as
what you uploaded).
        DownloadObject(base64Key, putObjectRequest);
        // 3. Get object metadata (and also verify AES256 encryption).
        GetObjectMetadata(base64Key);
        // 4. Copy object (both source and target objects use
server-side encryption with
        //      customer-provided encryption key.
        CopyObject(aesEncryption, base64Key);
    }
catch (AmazonS3Exception amazonS3Exception)
{
    if (amazonS3Exception.ErrorCode != null &&
        (amazonS3Exception.ErrorCode.Equals("InvalidAccessKeyId"))

        ||
        amazonS3Exception.ErrorCode.Equals("InvalidSecurity")))

    {
        Console.WriteLine("Check the provided AWS Credentials.");

        Console.WriteLine(
            "For service sign up go to ht
tp://aws.amazon.com/s3");
    }
    else
    {
        Console.WriteLine(
            "Error occurred. Message:'{0}' when writing an ob
ject"
            , amazonS3Exception.Message);
    }
}

Console.WriteLine("Press any key to continue...");
Console.ReadKey();
}

private static void CopyObject(Aes aesEncryption, string base64Key)
{
    aesEncryption.GenerateKey();
    string copyBase64Key = Convert.ToBase64String(aesEncryption.Key);

    CopyObjectRequest copyRequest = new CopyObjectRequest
    {
        SourceBucket = bucketName,
        SourceKey = keyName,
        DestinationBucket = bucketName,
```

```
        DestinationKey = copyTargetKeyName,
        // Source object encryption information.
        CopySourceServerSideEncryptionCustomerMethod = ServerSideEncryptionCustomerMethod.AES256,
        CopySourceServerSideEncryptionCustomerProvidedKey = base64Key,

        // Target object encryption information.
        ServerSideEncryptionCustomerMethod = ServerSideEncryptionCustomerMethod.AES256,
        ServerSideEncryptionCustomerProvidedKey = copyBase64Key
    };
    client.CopyObject(copyRequest);
}

private static void DownloadObject(string base64Key, PutObjectRequest putObjectRequest)
{
    GetObjectRequest getObjectRequest = new GetObjectRequest
    {
        BucketName = bucketName,
        Key = keyName,
        // Provide encryption information of the object stored in S3.
        ServerSideEncryptionCustomerMethod = ServerSideEncryptionCustomerMethod.AES256,
        ServerSideEncryptionCustomerProvidedKey = base64Key
    };

    using (GetObjectResponse getResponse = client.GetObject(getObjectRequest))
    using (StreamReader reader = new StreamReader(getResponse.ResponseStream))
    {
        string content = reader.ReadToEnd();
        Assert.AreEqual(putObjectRequest.ContentBody, content);
        Assert.AreEqual(ServerSideEncryptionCustomerMethod.AES256, getResponse.ServerSideEncryptionCustomerMethod);
    }
}

private static void GetObjectMetadata(string base64Key)
{
    GetObjectMetadataRequest getObjectMetadataRequest = new GetObjectMetadataRequest
    {
        BucketName = bucketName,
        Key = keyName,

        // Object stored in S3 is encrypted. So provide necessary encryption information.
        ServerSideEncryptionCustomerMethod = ServerSideEncryptionCustomerMethod.AES256,
        ServerSideEncryptionCustomerProvidedKey = base64Key
    };

    GetObjectMetadataResponse getObjectMetadataResponse = client.GetObjectMetadata(getObjectMetadataRequest);
    Assert.AreEqual(ServerSideEncryptionCustomerMethod.AES256, getObjectMetadataResponse.ServerSideEncryptionCustomerMethod);
```

```

        }

        private static PutObjectRequest UploadObject(string base64Key)
        {
            PutObjectRequest putObjectRequest = new PutObjectRequest
            {
                BucketName = bucketName,
                Key = keyName,
                ContentBody = "sample text",
                ServerSideEncryptionCustomerMethod = ServerSideEncryptionCustomer
Method.AES256,
                ServerSideEncryptionCustomerProvidedKey = base64Key
            };
            PutObjectResponse putObjectResponse = client.PutObject(putObjectRe
quest);
            return putObjectRequest;
        }
    }
}

```

Other Amazon S3 Operations and SSE-C

The example in the preceding section shows how to request server-side encryption with customer-provided key (SSE-C) in the PUT, GET, Head, and Copy operations. This section describes other APIs that support SSE-C.

To upload large objects, you can use multipart upload API (see [Uploading Objects Using Multipart Upload API \(p. 155\)](#)). You can use either high-level or low-level APIs to upload large objects. These APIs support encryption-related headers in the request.

- When using high-level Transfer-Utility API, you provide the encryption-specific headers in the `TransferUtilityUploadRequest` as shown. For code examples, see [Using the High-Level .NET API for Multipart Upload \(p. 166\)](#).

```

TransferUtilityUploadRequest request = new TransferUtilityUploadRequest()
{
    FilePath = filePath,
    BucketName = existingBucketName,
    Key = keyName,
    // Provide encryption information.
    ServerSideEncryptionCustomerMethod = ServerSideEncryptionCustomerMeth
od.AES256,
    ServerSideEncryptionCustomerProvidedKey = base64Key,
};

```

- When using the low-level API, you provide encryption-related information in the initiate multipart upload request, followed by identical encryption information in the subsequent upload part requests. You do not need to provide any encryption-specific headers in your complete multipart upload request. For examples, see [Using the Low-Level .NET API for Multipart Upload \(p. 175\)](#).

The following is a low-level multipart upload example that makes a copy of an existing large object. In the example, the object to be copied is stored in Amazon S3 using SSE-C, and you want to save the target object also using SSE-C. In the example, you do the following:

- Initiate a multipart upload request by providing an encryption key and related information.
- Provide source and target object encryption keys and related information in the `CopyPartRequest`.
- Obtain the size of the source object to be copied by retrieving the object metadata.

- Upload the objects in 5 MB parts.

```
using System;
using System.Collections.Generic;
using System.Security.Cryptography;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples
{
    class SSECLowLevelMPUcopyObject
    {
        static string existingBucketName      = "**** bucket name ****";
        static string sourceKeyName          = "**** key name ****";
        static string targetKeyName          = "**** key name ****";

        static void Main(string[] args)
        {
            IAmazonS3 s3Client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1);
            List<CopyPartResponse> uploadResponses = new List<CopyPartResponse>();

            Aes aesEncryption = Aes.Create();
            aesEncryption.KeySize = 256;
            aesEncryption.GenerateKey();
            string base64Key = Convert.ToBase64String(aesEncryption.Key);

            // 1. Initialize.
            InitiateMultipartUploadRequest initiateRequest = new InitiateMultipartUploadRequest
            {
                BucketName = existingBucketName,
                Key = targetKeyName,
                ServerSideEncryptionCustomerMethod = ServerSideEncryptionCustomerMethod.AES256,
                ServerSideEncryptionCustomerProvidedKey = base64Key,
            };

            InitiateMultipartUploadResponse initResponse =
                s3Client.InitiateMultipartUpload(initiateRequest);

            // 2. Upload Parts.
            long partSize = 5 * (long)Math.Pow(2, 20); // 5 MB
            long firstByte = 0;
            long lastByte = partSize;

            try
            {
                // First find source object size. Because object is stored
                // encrypted with
                // customer provided key you need to provide encryption information
                // in your request.
                GetObjectMetadataRequest getObjectMetadataRequest = new GetObjectMetadataRequest()
                {
                    BucketName = existingBucketName,
```

```

        Key = sourceKeyName,
        ServerSideEncryptionCustomerMethod = ServerSideEncryption
CustomerMethod.AES256,
        ServerSideEncryptionCustomerProvidedKey = "****source object
encryption key ****"
    };

    GetObjectMetadataResponse getObjectMetadataResponse = s3Cli
ent.GetObjectMetadata(getObjectMetadataRequest);

    long filePosition = 0;
    for (int i = 1; filePosition < getObjectMetadataResponse.Con
tentLength; i++)
    {
        CopyPartRequest copyPartRequest = new CopyPartRequest
        {
            UploadId = initResponse.UploadId,
            // Source.
            SourceBucket = existingBucketName,
            SourceKey = sourceKeyName,
            // Source object is stored using SSE-C. Provide encryp
tion information.
            CopySourceServerSideEncryptionCustomerMethod =
ServerSideEncryptionCustomerMethod.AES256,
            CopySourceServerSideEncryptionCustomerProvidedKey =
"****source object encryption key ****",
            FirstByte = firstByte,
            // If the last part is smaller then our normal part
size then use the remaining size.
            LastByte = lastByte > getObjectMetadataResponse.Con
tentLength ?
                getObjectMetadataResponse.ContentLength - 1 :
lastByte,

            // Target.
            DestinationBucket = existingBucketName,
            DestinationKey = targetKeyName,
            PartNumber = i,
            // Encryption information for the target object.
            ServerSideEncryptionCustomerMethod = ServerSideEncryp
tionCustomerMethod.AES256,
            ServerSideEncryptionCustomerProvidedKey = base64Key
        };
        uploadResponses.Add(s3Client.CopyPart(copyPartRequest));
        filePosition += partSize;
        firstByte += partSize;
        lastByte += partSize;
    }

    // Step 3: complete.
    CompleteMultipartUploadRequest completeRequest = new Com
pleteMultipartUploadRequest
    {
        BucketName = existingBucketName,
        Key = targetKeyName,
        UploadId = initResponse.UploadId,
    };
    completeRequest.AddPartETags(uploadResponses);
}

```

```
        CompleteMultipartUploadResponse completeUploadResponse =
            s3Client.CompleteMultipartUpload(completeRequest);
    }
    catch (Exception exception)
    {
        Console.WriteLine("Exception occurred: {0}", exception.Message);
        AbortMultipartUploadRequest abortMPUREquest = new AbortMultipartUploadRequest
        {
            BucketName = existingBucketName,
            Key = targetKeyName,
            UploadId = initResponse.UploadId
        };
        s3Client.AbortMultipartUpload(abortMPUREquest);
    }
}
}
```

Specifying Server-Side Encryption with Customer-Provided Encryption Keys Using the REST API

The following Amazon S3 REST APIs support headers related to server-side encryption with customer-provided encryption keys. For more information about these headers, see [Using SSE-C \(p. 395\)](#).

- [GET Object](#)
- [HEAD Object](#)
- [PUT Object](#)
- [PUT Object - Copy](#)
- [POST Object](#)
- [Initiate Multipart Upload](#)
- [Upload Part](#)
- [Upload Part - Copy](#)

Protecting Data Using Client-Side Encryption

Client-side encryption refers to encrypting data before sending it to Amazon S3. You have the following two options for using data encryption keys:

- Use an AWS KMS-managed customer master key
- Use a client-side master key

Option 1: Using an AWS KMS–Managed Customer Master Key (CMK)

When using an AWS KMS-managed customer master key for client-side data encryption, you don't have to worry about providing any encryption keys to the Amazon S3 encryption client (for example, the

`AmazonS3EncryptionClient` in the AWS SDK for Java). Instead, you provide only an AWS KMS customer master key ID (CMK ID), and the client does the rest. This is how it works:

- **When uploading an object** – Using the CMK ID, the client first sends a request to AWS KMS for a key that it can use to encrypt your object data. In response, AWS KMS returns a randomly generated data encryption key. In fact, AWS KMS returns two versions of the data encryption key:
 - A plain text version that the client uses to encrypt the object data.
 - A cipher blob of the same data encryption key that the client uploads to Amazon S3 as object metadata.

Note

The client obtains a unique data encryption key for each object it uploads.

For a working example, see [Example: Client-Side Encryption \(Option 1: Using an AWS KMS–Managed Customer Master Key \(AWS SDK for Java\)\)](#) (p. 410).

- **When downloading an object** – The client first downloads the encrypted object from Amazon S3 along with the cipher blob version of the data encryption key stored as object metadata. The client then sends the cipher blob to AWS KMS to get the plain text version of the same, so that it can decrypt the object data.

For more information about AWS KMS, go to [What is the AWS Key Management Service?](#) in the *AWS Key Management Service Developer Guide*.

Option 2: Using a Client-Side Master Key

This section shows how to provide your client-side master key in the client-side data encryption process.

Important

Your client-side master keys and your unencrypted data are never sent to AWS; therefore, it is important that you safely manage your encryption keys. If you lose them, you won't be able to decrypt your data.

This is how it works:

- **When uploading an object** – You provide a client-side master key to the Amazon S3 encryption client (for example, `AmazonS3EncryptionClient` when using the AWS SDK for Java). The client uses this master key only to encrypt the data encryption key that it generates randomly. The process works like this:
 1. The Amazon S3 encryption client locally generates a one-time-use symmetric key, a.k.a. a data encryption key (or data key). It uses this data key to encrypt the data of a single S3 object (for each object, the client generates a separate data key).
 2. The client encrypts the data encryption key using the master key you provide.

The client uploads the encrypted data key and its material description as part of the object metadata. The material description helps the client later determine which client-side master key to use for decryption (when you download the object, the client decrypts it).

3. The client then uploads the encrypted data to Amazon S3 and also saves the encrypted data key as object metadata (`x-amz-meta-x-amz-key`) in Amazon S3 by default.

- **When downloading an object** – The client first downloads the encrypted object from Amazon S3 along with the metadata. Using the material description in the metadata, the client first determines which master key to use to decrypt the encrypted data key. Using that master key, the client decrypts the data key and uses it to decrypt the object.

The client-side master key you provide can be either a symmetric key or a public/private key pair. For examples, see [Examples: Client-Side Encryption \(Option 2: Using a Client-Side Master Key \(AWS SDK for Java\)\)](#) (p. 411).

For more information, see the [Client-Side Data Encryption with the AWS SDK for Java and Amazon S3](#) article.

The following AWS SDKs support client-side encryption:

- AWS SDK for Java
- AWS SDK for .NET
- AWS SDK for Ruby

Example: Client-Side Encryption (Option 1: Using an AWS KMS–Managed Customer Master Key (AWS SDK for Java))

The following Java code example uploads an object to Amazon S3. The example uses a KMS-managed customer master key (CMK) to encrypt data on the client-side before uploading to Amazon S3. You will need the CMK ID in the code.

For more information about how client-side encryption using a KMS-managed CMK works, see [Option 1: Using an AWS KMS–Managed Customer Master Key \(CMK\) \(p. 408\)](#).

For instructions on how to create and test a working sample, see [Testing the Java Code Examples \(p. 545\)](#). You will need to update the code by providing your bucket name and a CMK ID.

```
import java.io.ByteArrayInputStream;
import java.util.Arrays;

import junit.framework.Assert;

import org.apache.commons.io.IOUtils;

import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Region;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3EncryptionClient;
import com.amazonaws.services.s3.model.CryptoConfiguration;
import com.amazonaws.services.s3.model.KMSEncryptionMaterialsProvider;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.PutObjectRequest;
import com.amazonaws.services.s3.model.S3Object;

public class testKMSkeyUploadObject {

    private static AmazonS3EncryptionClient encryptionClient;

    public static void main(String[] args) throws Exception {
        String bucketName = "***bucket name***";
        String objectKey   = "ExampleKMSEncryptedObject";
        String kms_cmk_id = "***AWS KMS customer master key ID***";

        KMSEncryptionMaterialsProvider materialProvider = new KMSEncryptionMaterialsProvider(kms_cmk_id);

        encryptionClient = new AmazonS3EncryptionClient(new ProfileCredentialsProvider(),
            new CryptoConfiguration().withKmsRegion(Regions.US_EAST_1)
            .withRegion(Region.getRegion(Regions.US_EAST_1)));
    }
}
```

```
// Upload object using the encryption client.  
byte[] plaintext = "Hello World, S3 Client-side Encryption Using Asym  
metric Master Key!"  
.getBytes();  
System.out.println("plaintext's length: " + plaintext.length);  
encryptionClient.putObject(new PutObjectRequest(bucketName, objectKey,  
  
new ByteArrayInputStream(plaintext), new ObjectMetadata()));  
  
// Download the object.  
S3Object downloadedObject = encryptionClient.getObject(bucketName,  
objectKey);  
byte[] decrypted = IOUtils.toByteArray(downloadedObject  
.getObjectContent());  
  
// Verify same data.  
Assert.assertTrue(Arrays.equals(plaintext, decrypted));  
}  
}
```

Examples: Client-Side Encryption (Option 2: Using a Client-Side Master Key (AWS SDK for Java))

This section provides code examples of client-side encryption. As described in the overview (see [Protecting Data Using Client-Side Encryption \(p. 408\)](#)) the client-side master key you provide can be either a symmetric key or a public/private key pair. This section provides examples of both types of master keys, symmetric master key (256-bit Advanced Encryption Standard (AES) secret key) and asymmetric master key (1024-bit RSA key pair).

Topics

- [Example 1: Encrypt and Upload a File Using a Client-Side Symmetric Master Key \(p. 411\)](#)
- [Example 2: Encrypt and Upload a File to Amazon S3 Using a Client-Side Asymmetric Master Key \(p. 415\)](#)

Note

If you get a cipher encryption error message when you use the encryption API for the first time, your version of the JDK may have a Java Cryptography Extension (JCE) jurisdiction policy file that limits the maximum key length for encryption and decryption transformations to 128 bits.

The AWS SDK requires a maximum key length of 256 bits. To check your maximum key length, use the `getMaxAllowedKeyLength` method of the `javax.crypto.Cipher` class. To remove the key length restriction, install the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files at the [Java SE download page](#).

Example 1: Encrypt and Upload a File Using a Client-Side Symmetric Master Key

This section provides example code using the AWS SDK for Java to do the following:

- First create a 256-bit AES symmetric master key and save it to a file.
- Upload an object to Amazon S3 using an S3 encryption client that first encrypts sample data on the client-side. The example also downloads the object and verifies that the data is the same.

Example 1a: Creating a Symmetric Master Key

Run this code to first generate a 256-bit AES symmetric master key for encrypted uploads to Amazon S3. The example saves the master key to a file (`secret.key`) in a temp directory (on Windows, it is the `c:\Users\<username>\AppData\Local\Tmp` folder).

For instructions on how to create and test a working sample, see [Using the AWS SDK for Java \(p. 544\)](#).

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.X509EncodedKeySpec;
import java.util.Arrays;

import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;

import org.junit.Assert;

public class GenerateSymmetricMasterKey {

    private static final String keyDir = System.getProperty("java.io.tmpdir");

    private static final String keyName = "secret.key";

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

        //Generate symmetric 256 bit AES key.
        KeyGenerator symKeyGenerator = KeyGenerator.getInstance("AES");
        symKeyGenerator.init(256);
        SecretKey symKey = symKeyGenerator.generateKey();

        //Save key.
        saveSymmetricKey(keyDir, symKey);

        //Load key.
        SecretKey symKeyLoaded = loadSymmetricAESKey(keyDir, "AES");

        Assert.assertTrue(Arrays.equals(symKey.getEncoded(), symKeyLoaded.getEncoded()));
    }

    public static void saveSymmetricKey(String path, SecretKey secretKey)
            throws IOException {
        X509EncodedKeySpec x509EncodedKeySpec = new X509EncodedKeySpec(
                secretKey.getEncoded());
        FileOutputStream keyfos = new FileOutputStream(path + "/" + keyName);
        keyfos.write(x509EncodedKeySpec.getEncoded());
        keyfos.close();
    }

    public static SecretKey loadSymmetricAESKey(String path, String algorithm)
```

```
        throws IOException, NoSuchAlgorithmException, InvalidKeySpecException,
        InvalidKeyException{
    //Read private key from file.
    File keyFile = new File(path + "/" + keyName);
    FileInputStream keyfis = new FileInputStream(keyFile);
    byte[] encodedPrivateKey = new byte[(int)keyFile.length()];
    keyfis.read(encodedPrivateKey);
    keyfis.close();

    //Generate secret key.
    return new SecretKeySpec(encodedPrivateKey, "AES");
}
}
```

This code example is for demonstration purposes only. For production use, you should consult your security engineer on how to obtain or generate the client-side master key.

Example 1b: Uploading a File to Amazon S3 Using a Symmetric Key

Run this code to encrypt sample data using a symmetric master key created by the preceding code example. The example uses an S3 encryption client to encrypt the data on the client-side and then upload it to Amazon S3.

For instructions on how to create and test a working sample, see [Using the AWS SDK for Java \(p. 544\)](#).

```
import java.io.ByteArrayInputStream;
import java.util.Arrays;
import java.util.Iterator;
import java.util.UUID;

import javax.crypto.SecretKey;

import org.apache.commons.io.IOUtils;
import org.joda.time.DateTime;
import org.joda.time.format.DateTimeFormat;
import org.junit.Assert;

import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3EncryptionClient;
import com.amazonaws.services.s3.model.EncryptionMaterials;
import com.amazonaws.services.s3.model.ListVersionsRequest;
import com.amazonaws.services.s3.model.ObjectListing;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.PutObjectRequest;
import com.amazonaws.services.s3.model.S3Object;
import com.amazonaws.services.s3.model.S3ObjectSummary;
import com.amazonaws.services.s3.model.S3VersionSummary;
import com.amazonaws.services.s3.model.StaticEncryptionMaterialsProvider;
import com.amazonaws.services.s3.model.VersionListing;

public class S3ClientSideEncryptionWithSymmetricMasterKey {
    private static final String masterKeyDir = System.getProperty("java.io.tmp
dir");
    private static final String bucketName = UUID.randomUUID() + "-"
+ DateTimeFormat.forPattern("yyMMdd-hhmmss").print(new DateTime());
}
```

```

private static final String objectKey = UUID.randomUUID().toString();

public static void main(String[] args) throws Exception {
    SecretKey mySymmetricKey = GenerateSymmetricMasterKey
        .loadSymmetricAESKey(masterKeyDir, "AES");

    EncryptionMaterials encryptionMaterials = new EncryptionMaterials(
        mySymmetricKey);

    AmazonS3EncryptionClient encryptionClient = new AmazonS3EncryptionClient(
        new ProfileCredentialsProvider(),
        new StaticEncryptionMaterialsProvider(encryptionMaterials));
    // Create the bucket
    encryptionClient.createBucket(bucketName);

    // Upload object using the encryption client.
    byte[] plaintext = "Hello World, S3 Client-side Encryption Using Asym-
metric Master Key!"
        .getBytes();
    System.out.println("plaintext's length: " + plaintext.length);
    encryptionClient.putObject(new PutObjectRequest(bucketName, objectKey,
        new ByteArrayInputStream(plaintext), new ObjectMetadata()));

    // Download the object.
    S3Object downloadedObject = encryptionClient.getObject(bucketName,
        objectKey);
    byte[] decrypted = IOUtils.toByteArray(downloadedObject
        .getObjectContent());

    // Verify same data.
    Assert.assertTrue(Arrays.equals(plaintext, decrypted));
    deleteBucketAndAllContents(encryptionClient);
}

private static void deleteBucketAndAllContents(AmazonS3 client) {
    System.out.println("Deleting S3 bucket: " + bucketName);
    ObjectListing objectListing = client.listObjects(bucketName);

    while (true) {
        for ( Iterator<?> iterator = objectListing.getObjectSummaries().iter-
ator(); iterator.hasNext(); ) {
            S3ObjectSummary objectSummary = (S3ObjectSummary) iterator
            .next();
            client.deleteObject(bucketName, objectSummary.getKey());
        }

        if (objectListing.isTruncated()) {
            objectListing = client.listNextBatchOfObjects(objectListing);
        } else {
            break;
        }
    }
    VersionListing list = client.listVersions(new ListVersionsRequest().with
    BucketName(bucketName));
    for ( Iterator<?> iterator = list.getVersionSummaries().iterator();
    iterator.hasNext(); ) {

```

```
        S3VersionSummary s = (S3VersionSummary)iterator.next();
        client.deleteVersion(bucketName, s.getKey(), s.getVersionId());
    }
    client.deleteBucket(bucketName);
}
}
```

Example 2: Encrypt and Upload a File to Amazon S3 Using a Client-Side Asymmetric Master Key

This section provides example code using the AWS SDK for Java to first create a 1024-bit RSA key pair. The example then uses that key pair as the client-side master key for the purpose of encrypting and upload a file.

This is how it works:

- First create a 1024-bit RSA key pair (asymmetric master key) and save it to a file.
- Upload an object to Amazon S3 using an S3 encryption client that encrypts sample data on the client-side. The example also downloads the object and verifies that the data is the same.

Example 2a: Creating a 1024-bit RSA Key Pair

Run this code to first generate a 1024-bit key pair (asymmetric master key). The example saves the master key to a file (`secret.key`) in a temp directory (on Windows, it is the `c:\Users\<username>\AppData\Local\Tmp` folder).

For instructions on how to create and test a working sample, see [Using the AWS SDK for Java \(p. 544\)](#).

```
import static org.junit.Assert.assertTrue;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.security.KeyFactory;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.SecureRandom;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;
import java.util.Arrays;

public class GenerateAsymmetricMasterKey {
    private static final String keyDir = System.getProperty("java.io.tmpdir");

    private static final SecureRandom srand = new SecureRandom();

    public static void main(String[] args) throws Exception {
        // Generate RSA key pair of 1024 bits
        KeyPair keypair = genKeyPair("RSA", 1024);
        // Save to file system
    }
}
```

```
    saveKeyPair(keyDir, keypair);
    // Loads from file system
    KeyPair loaded = loadKeyPair(keyDir, "RSA");
    // Sanity check
    assertTrue(Arrays.equals(keypair.getPublic().getEncoded(), loaded
        .getPublic().getEncoded()));
    assertTrue(Arrays.equals(keypair.getPrivate().getEncoded(), loaded
        .getPrivate().getEncoded()));
}

public static KeyPair genKeyPair(String algorithm, int bitLength)
    throws NoSuchAlgorithmException {
    KeyPairGenerator keyGenerator = KeyPairGenerator.getInstance(algorithm);

    keyGenerator.initialize(1024, srandom);
    return keyGenerator.generateKeyPair();
}

public static void saveKeyPair(String dir, KeyPair keyPair)
    throws IOException {
    PrivateKey privateKey = keyPair.getPrivate();
    PublicKey publicKey = keyPair.getPublic();

    X509EncodedKeySpec x509EncodedKeySpec = new X509EncodedKeySpec(
        publicKey.getEncoded());
    FileOutputStream fos = new FileOutputStream(dir + "/public.key");
    fos.write(x509EncodedKeySpec.getEncoded());
    fos.close();

    PKCS8EncodedKeySpec pkcs8EncodedKeySpec = new PKCS8EncodedKeySpec(
        privateKey.getEncoded());
    fos = new FileOutputStream(dir + "/private.key");
    fos.write(pkcs8EncodedKeySpec.getEncoded());
    fos.close();
}

public static KeyPair loadKeyPair(String path, String algorithm)
    throws IOException, NoSuchAlgorithmException,
    InvalidKeySpecException {
    // read public key from file
    File filePublicKey = new File(path + "/public.key");
    FileInputStream fis = new FileInputStream(filePublicKey);
    byte[] encodedPublicKey = new byte[(int) filePublicKey.length()];
    fis.read(encodedPublicKey);
    fis.close();

    // read private key from file
    File filePrivateKey = new File(path + "/private.key");
    fis = new FileInputStream(filePrivateKey);
    byte[] encodedPrivateKey = new byte[(int) filePrivateKey.length()];
    fis.read(encodedPrivateKey);
    fis.close();

    // Convert them into KeyPair
    KeyFactory keyFactory = KeyFactory.getInstance(algorithm);
    X509EncodedKeySpec publicKeySpec = new X509EncodedKeySpec(
        encodedPublicKey);
    PublicKey publicKey = keyFactory.generatePublic(publicKeySpec);
```

```
        PKCS8EncodedKeySpec privateKeySpec = new PKCS8EncodedKeySpec(  
            encodedPrivateKey);  
        PrivateKey privateKey = keyFactory.generatePrivate(privateKeySpec);  
  
        return new KeyPair(publicKey, privateKey);  
    }  
}
```

This code example is for demonstration purposes only. For production use, you should consult your security engineer on how to obtain or generate the client-side master key.

Example 2b: Uploading a File to Amazon S3 Using a Key Pair

Run this code to encrypt sample data using a symmetric master key created by the preceding code example. The example uses an S3 encryption client to encrypt the data on the client-side and then upload it to Amazon S3.

For instructions on how to create and test a working sample, see [Using the AWS SDK for Java \(p. 544\)](#).

```
import java.io.ByteArrayInputStream;  
import java.io.File;  
import java.security.KeyFactory;  
import java.security.KeyPair;  
import java.security.PrivateKey;  
import java.security.PublicKey;  
import java.security.spec.PKCS8EncodedKeySpec;  
import java.security.spec.X509EncodedKeySpec;  
import java.util.Arrays;  
import java.util.Iterator;  
import java.util.UUID;  
  
import org.apache.commons.io.FileUtils;  
import org.apache.commons.io.IOUtils;  
import org.joda.time.DateTime;  
import org.joda.time.format.DateTimeFormat;  
import org.junit.Assert;  
  
import com.amazonaws.auth.profile.ProfileCredentialsProvider;  
import com.amazonaws.services.s3.AmazonS3;  
import com.amazonaws.services.s3.AmazonS3EncryptionClient;  
import com.amazonaws.services.s3.model.EncryptionMaterials;  
import com.amazonaws.services.s3.model.ListVersionsRequest;  
import com.amazonaws.services.s3.model.ObjectListing;  
import com.amazonaws.services.s3.model.ObjectMetadata;  
import com.amazonaws.services.s3.model.PutObjectRequest;  
import com.amazonaws.services.s3.model.S3Object;  
import com.amazonaws.services.s3.model.S3ObjectSummary;  
import com.amazonaws.services.s3.model.S3VersionSummary;  
import com.amazonaws.services.s3.model.StaticEncryptionMaterialsProvider;  
import com.amazonaws.services.s3.model.VersionListing;  
  
public class S3ClientSideEncryptionAsymmetricMasterKey {  
    private static final String keyDir = System.getProperty("java.io.tmpdir");  
  
    private static final String bucketName = UUID.randomUUID() + "-"  
        + DateTimeFormat.forPattern("yyMMdd-hhmmss").print(new DateTime());
```

```
private static final String objectKey = UUID.randomUUID().toString();

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

    // 1. Load keys from files
    byte[] bytes = FileUtils.readFileToByteArray(new File(
        keyDir + "private.key"));
    KeyFactory kf = KeyFactory.getInstance("RSA");
    PKCS8EncodedKeySpec ks = new PKCS8EncodedKeySpec(bytes);
    PrivateKey pk = kf.generatePrivate(ks);

    bytes = FileUtils.readFileToByteArray(new File(keyDir + "public.key"));

    PublicKey publicKey = KeyFactory.getInstance("RSA").generatePublic(
        new X509EncodedKeySpec(bytes));

    KeyPair loadedKeyPair = new KeyPair(publicKey, pk);

    // 2. Construct an instance of AmazonS3EncryptionClient.
    EncryptionMaterials encryptionMaterials = new EncryptionMaterials(
        loadedKeyPair);
    AmazonS3EncryptionClient encryptionClient = new AmazonS3EncryptionClient(
        new ProfileCredentialsProvider(),
        new StaticEncryptionMaterialsProvider(encryptionMaterials));
    // Create the bucket
    encryptionClient.createBucket(bucketName);
    // 3. Upload the object.
    byte[] plaintext = "Hello World, S3 Client-side Encryption Using Asym-
metric Master Key!";
    .getBytes();
    System.out.println("plaintext's length: " + plaintext.length);
    encryptionClient.putObject(new PutObjectRequest(bucketName, objectKey,
        new ByteArrayInputStream(plaintext), new ObjectMetadata()));

    // 4. Download the object.
    S3Object downloadedObject = encryptionClient.getObject(bucketName, ob-
jectKey);
    byte[] decrypted = IOUtils.toByteArray(downloadedObject
        .getObjectContent());
    Assert.assertTrue(Arrays.equals(plaintext, decrypted));
    deleteBucketAndAllContents(encryptionClient);
}

private static void deleteBucketAndAllContents(AmazonS3 client) {
    System.out.println("Deleting S3 bucket: " + bucketName);
    ObjectListing objectListing = client.listObjects(bucketName);

    while (true) {
        for ( Iterator<?> iterator = objectListing.getObjectSummaries().iter-
ator(); iterator.hasNext(); ) {
            S3ObjectSummary objectSummary = (S3ObjectSummary) iterator
            .next();
            client.deleteObject(bucketName, objectSummary.getKey());
        }
    }
}
```

```
        if (objectListing.isTruncated()) {
            objectListing = client.listNextBatchOfObjects(objectListing);
        } else {
            break;
        }
    }
    VersionListing list = client.listVersions(new ListVersionsRequest().with
BucketName(bucketName));
    for ( Iterator<?> iterator = list.getVersionSummaries().iterator();
iterator.hasNext(); ) {
        S3VersionSummary s = (S3VersionSummary)iterator.next();
        client.deleteVersion(bucketName, s.getKey(), s.getVersionId());
    }
    client.deleteBucket(bucketName);
}
}
```

Using Reduced Redundancy Storage

Topics

- [Setting the Storage Class of an Object You Upload \(p. 420\)](#)
- [Changing the Storage Class of an Object in Amazon S3 \(p. 420\)](#)

Amazon S3 stores objects according to their storage class. It assigns the storage class to an object when it is written to Amazon S3. You can assign objects a specific storage class (standard or reduced redundancy) only when you write the objects to an Amazon S3 bucket or when you copy objects that are already stored in Amazon S3. Standard is the default storage class. For information about storage classes, see [Object Key and Metadata \(p. 79\)](#).

In order to reduce storage costs, you can use reduced redundancy storage for noncritical, reproducible data at lower levels of redundancy than Amazon S3 provides with standard storage. The lower level of redundancy results in less durability and availability, but in many cases, the lower costs can make reduced redundancy storage an acceptable storage solution. For example, it can be a cost-effective solution for sharing media content that is durably stored elsewhere. It can also make sense if you are storing thumbnails and other resized images that can be easily reproduced from an original image.

Reduced redundancy storage is designed to provide 99.99% durability of objects over a given year. This durability level corresponds to an average annual expected loss of 0.01% of objects. For example, if you store 10,000 objects using the RRS option, you can, on average, expect to incur an annual loss of a single object per year (0.01% of 10,000 objects).

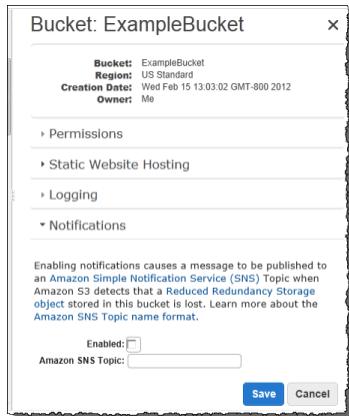
Note

This annual loss represents an expected average and does not guarantee the loss of less than 0.01% of objects in a given year.

Reduced redundancy storage stores objects on multiple devices across multiple facilities, providing 400 times the durability of a typical disk drive, but it does not replicate objects as many times as Amazon S3 standard storage. In addition, reduced redundancy storage is designed to sustain the loss of data in a single facility.

If an object in reduced redundancy storage has been lost, Amazon S3 will return a 405 error on requests made to that object. Amazon S3 also offers notifications for reduced redundancy storage object loss: you can configure your bucket so that when Amazon S3 detects the loss of an RRS object, a notification will be sent through Amazon Simple Notification Service (Amazon SNS). You can then replace the lost object.

To enable notifications, you can use the Amazon S3 console to set the **Notifications** property of your bucket.



Latency and throughput for reduced redundancy storage are the same as for standard storage. For pricing information, go to [Pricing](#) on the *Amazon S3 product details page*.

Setting the Storage Class of an Object You Upload

To set the storage class of an object you upload to RRS, you set `x-amz-storage-class` to `REDUCED_REDUNDANCY` in a `PUT` request.

How to Set the Storage Class of an Object You're Uploading to RRS

- Create a `PUT` Object request setting the `x-amz-storage-class` request header to `REDUCED_REDUNDANCY`.

You must have the correct permissions on the bucket to perform the `PUT` operation. The default value for the storage class is `STANDARD` (for regular Amazon S3 storage).

The following example sets the storage class of `my-image.jpg` to RRS.

```
PUT /my-image.jpg HTTP/1.1
Host: myBucket.s3.amazonaws.com
Date: Wed, 12 Oct 2009 17:50:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:xQE0diMbLRepdf3YB+FIEXAMPLE=
Content-Type: image/jpeg
Content-Length: 11434
Expect: 100-continue
x-amz-storage-class: REDUCED_REDUNDANCY
```

Changing the Storage Class of an Object in Amazon S3

Topics

- [Return Code for Lost Data \(p. 422\)](#)

You can also change the storage class of an object that is already stored in Amazon S3 by copying it to the same key name in the same bucket. To do that, you use the following request headers in a `PUT` Object copy request:

- `x-amz-metadata-directive` set to `COPY`
- `x-amz-storage-class` set to `STANDARD` or `REDUCED_REDUNDANCY`

Important

To optimize the execution of the copy request, do not change any of the other metadata in the `PUT` Object copy request. If you need to change metadata other than the storage class, set `x-amz-metadata-directive` to `REPLACE` for better performance.

How to Rewrite the Storage Class of an Object in Amazon S3

- Create a `PUT` Object copy request and set the `x-amz-storage-class` request header to `REDUCED_REDUNDANCY` (for RRS) or `STANDARD` (for regular Amazon S3 storage), and make the target name the same as the source name.

You must have the correct permissions on the bucket to perform the copy operation.

The following example sets the storage class of `my-image.jpg` to RRS.

```
PUT /my-image.jpg HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
x-amz-copy-source: /bucket/my-image.jpg
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
x-amz-storage-class: REDUCED_REDUNDANCY
x-amz-metadata-directive: COPY
```

The following example sets the storage class of `my-image.jpg` to standard.

```
PUT /my-image.jpg HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
x-amz-copy-source: /bucket/my-image.jpg
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
x-amz-storage-class: STANDARD
x-amz-metadata-directive: COPY
```

Note

If you copy an RRS object and fail to include the `x-amz-storage-class` request header, the storage class of the target object defaults to `STANDARD`.

It is not possible to change the storage class of a specific version of an object. When you copy it, Amazon S3 gives it a new version ID.

Note

When an object is written in a copy request, the entire object is rewritten in order to apply the new storage class.

For more information about versioning, see [Using Versioning \(p. 422\)](#).

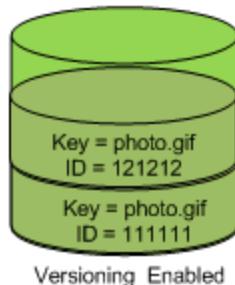
Return Code for Lost Data

If Amazon S3 detects that an object has been lost, any subsequent GET, or HEAD operations, or PUT Object copy operation that uses the lost object as the source object, will result in a 405 Method Not Allowed error. Once an object is marked lost, Amazon S3 will never be able to recover the object. In this situation, you can either delete the key, or upload a copy of the object.

Using Versioning

Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures.

In one bucket, for example, you can have two objects with the same key, but different version IDs, such as `photo.gif` (version 111111) and `photo.gif` (version 121212).



Versioning-enabled buckets enable you to recover objects from accidental deletion or overwrite. For example:

- If you delete an object, instead of removing it permanently, Amazon S3 inserts a delete marker, which becomes the current object version. You can always restore the previous version. For more information, see [Deleting Object Versions \(p. 436\)](#).
- If you overwrite an object, it results in a new object version in the bucket. You can always restore the previous version.

Buckets can be in one of three states: unversioned (the default), versioning-enabled, or versioning-suspended.

Important

Once you version-enable a bucket, it can never return to an unversioned state. You can, however, suspend versioning on that bucket.

The versioning state applies to all (never some) of the objects in that bucket. The first time you enable a bucket for versioning, objects in it are thereafter always versioned and given a unique version ID. Note the following:

- Objects stored in your bucket before you set the versioning state have a version ID of `null`. When you enable versioning, existing objects in your bucket do not change. What changes is how Amazon S3 handles the objects in future requests. For more information, see [Managing Objects in a Versioning-Enabled Bucket \(p. 427\)](#).
- The bucket owner (or any user with appropriate permissions) can suspend versioning to stop accruing object versions. When you suspend versioning, existing objects in your bucket do not change. What changes is how Amazon S3 handles objects in future requests. For more information, see [Managing Objects in a Versioning-Suspended Bucket \(p. 443\)](#).

How to Configure Versioning on a Bucket

You can configure bucket versioning using any of the following methods:

- Configure versioning using the Amazon S3 console.
- Configure versioning programmatically using the AWS SDKs

Both the console and the SDKs call the REST API Amazon S3 provides to manage versioning.

Note

If you need to, you can also make the Amazon S3 REST API calls directly from your code. However, this can be cumbersome because it requires you to write code to authenticate your requests.

Each bucket you create has a *versioning* subresource (see [Bucket Configuration Options \(p. 57\)](#)) associated with it. By default, your bucket is unversioned, and accordingly the versioning subresource stores empty versioning configuration.

```
<VersioningConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
</VersioningConfiguration>
```

To enable versioning, you send a request to Amazon S3 with a versioning configuration that includes a status.

```
<VersioningConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Status>Enabled</Status>
</VersioningConfiguration>
```

To suspend versioning, you set the status value to Suspended.

The bucket owner, an AWS account that created the bucket (root account), and authorized users can configure the versioning state of a bucket. For more information about permissions, see [Managing Access Permissions to Your Amazon S3 Resources \(p. 269\)](#).

For an example of configuring versioning, see [Examples of Enabling Bucket Versioning \(p. 424\)](#).

MFA Delete

You can optionally add another layer of security by configuring a bucket to enable MFA (Multi-Factor Authentication) Delete, which requires additional authentication for either of the following operations.

- Change the versioning state of your bucket
- Permanently delete an object version

MFA Delete requires two forms of authentication together:

- Your security credentials
- The concatenation of a valid serial number, a space, and the six-digit code displayed on an approved authentication device

MFA Delete thus provides added security in the event, for example, your security credentials are compromised.

To enable or disable MFA delete, you use the same API that you use to configure versioning on a bucket. Amazon S3 stores the MFA Delete configuration in the same *versioning* subresource that stores the bucket's versioning status.

```
<VersioningConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Status>VersioningState</Status>
  <MfaDelete>MfaDeleteState</MfaDelete>
</VersioningConfiguration>
```

To use MFA Delete, you can use either a hardware or virtual MFA device to generate an authentication code. The following example shows a generated authentication code displayed on a hardware device.



Note

MFA Delete and MFA-protected API access are features intended to provide protection for different scenarios. You configure MFA Delete on a bucket to ensure that data in your bucket cannot be accidentally deleted. MFA-protected API access is used to enforce another authentication factor (MFA code) when accessing sensitive Amazon S3 resources. You can require any operations against these Amazon S3 resources be done with temporary credentials created using MFA. For an example, see [Adding a Bucket Policy to Require MFA Authentication \(p. 339\)](#).

For more information on how to purchase and activate an authentication device, go to <http://aws.amazon.com/mfa/>.

Note

The bucket owner, the AWS account that created the bucket (root account), and all authorized IAM users can enable versioning, but only the bucket owner (root account) can enable MFA delete.

Related Topics

For more information, see the following topics:

[Examples of Enabling Bucket Versioning \(p. 424\)](#)

[Managing Objects in a Versioning-Enabled Bucket \(p. 427\)](#)

[Managing Objects in a Versioning-Suspended Bucket \(p. 443\)](#)

Examples of Enabling Bucket Versioning

Topics

- [Using the Amazon S3 Console \(p. 425\)](#)
- [Using the AWS SDK for Java \(p. 425\)](#)
- [Using the AWS SDK for .NET \(p. 426\)](#)
- [Using Other AWS SDKs \(p. 427\)](#)

This section provides examples of enabling versioning on a bucket. The examples first enable versioning on a bucket and then retrieve versioning status. For an introduction, see [Using Versioning \(p. 422\)](#).

Using the Amazon S3 Console

For more information about enabling versioning on a bucket using the Amazon S3 console, go to [Enable Versioning](#) in the *Amazon Simple Storage Service Console User Guide*.

Using the AWS SDK for Java

For instructions on how to create and test a working sample, see [Testing the Java Code Examples \(p. 545\)](#).

```
import java.io.IOException;

import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.regions.Region;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.AmazonS3Exception;
import com.amazonaws.services.s3.model.BucketVersioningConfiguration;
import com.amazonaws.services.s3.model.SetBucketVersioningConfigurationRequest;

public class BucketVersioningConfigurationExample {
    public static String bucketName = "*** bucket name ***";
    public static AmazonS3Client s3Client;

    public static void main(String[] args) throws IOException {
        s3Client = new AmazonS3Client(new ProfileCredentialsProvider());
        s3Client.setRegion(Region.getRegion(Regions.US_EAST_1));
        try {

            // 1. Enable versioning on the bucket.
            BucketVersioningConfiguration configuration =
                new BucketVersioningConfiguration().withStatus("Enabled");

            SetBucketVersioningConfigurationRequest setBucketVersioningConfigurationRe-
quest =
                new SetBucketVersioningConfigurationRequest(bucketName, configuration);

            s3Client.setBucketVersioningConfiguration(setBucketVersioningConfiguration
Request);

            // 2. Get bucket versioning configuration information.
            BucketVersioningConfiguration conf = s3Client.getBucketVersioningConfigura-
tion(bucketName);
            System.out.println("bucket versioning configuration status: " +
conf.getStatus());

        } catch (AmazonS3Exception amazonS3Exception) {
            System.out.format("An Amazon S3 error occurred. Exception: %s",
amazonS3Exception.toString());
        } catch (Exception ex) {
            System.out.format("Exception: %s", ex.toString());
        }
    }
}
```

Using the AWS SDK for .NET

For information about how to create and test a working sample, see [Testing the .NET Code Examples \(p. 547\)](#).

```
using System;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazonaws.com.docsamples
{
    class BucketVersioningConfiguration
    {
        static string bucketName = "**** bucket name ****";

        public static void Main(string[] args)
        {
            using (var client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1))
            {
                try
                {
                    EnableVersioningOnBucket(client);
                    string bucketVersioningStatus = RetrieveBucketVersioningConfiguration(client);
                }
                catch (AmazonS3Exception amazonS3Exception)
                {
                    if (amazonS3Exception.ErrorCode != null &&
                        (amazonS3Exception.ErrorCode.Equals("InvalidAccessKeyId") ||
                         amazonS3Exception.ErrorCode.Equals("InvalidSecurity")))
                    {
                        Console.WriteLine("Check the provided AWS Credentials.");

                        Console.WriteLine(
                            "To sign up for service, go to http://aws.amazon.com/s3");
                    }
                    else
                    {
                        Console.WriteLine(
                            "Error occurred. Message:'{0}' when listing objects",
                            amazonS3Exception.Message);
                    }
                }
            }

            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }

        static void EnableVersioningOnBucket(IAmazonS3 client)
        {
```

```
quest          PutBucketVersioningRequest request = new PutBucketVersioningRe
{
    BucketName = bucketName,
    VersioningConfig = new S3BucketVersioningConfig
    {
        Status = VersionStatus.Enabled
    }
};

PutBucketVersioningResponse response = client.PutBucketVersion
ing(request);
}

static string RetrieveBucketVersioningConfiguration(IAmazonS3 client)
{
    GetBucketVersioningRequest request = new GetBucketVersioningRe
quest
{
    BucketName = bucketName
};

GetBucketVersioningResponse response = client.GetBucketVersion
ing(request);
return response.VersioningConfig.Status;
}
}
```

Using Other AWS SDKs

For information about using other AWS SDKs, go to [Sample Code and Libraries](#).

Managing Objects in a Versioning-Enabled Bucket

Topics

- [Adding Objects to Versioning-Enabled Buckets \(p. 428\)](#)
- [Listing Objects in a Versioning-Enabled Bucket \(p. 429\)](#)
- [Retrieving Object Versions \(p. 434\)](#)
- [Deleting Object Versions \(p. 436\)](#)
- [Transitioning Object Versions \(p. 441\)](#)
- [Restoring Previous Versions \(p. 441\)](#)
- [Versioned Object Permissions \(p. 442\)](#)

Objects stored in your bucket before you set the versioning state have a version ID of null. When you enable versioning, existing objects in your bucket do not change. What changes is how Amazon S3 handles the objects in future requests. The topics in this section explain various object operations in a versioning-enabled bucket.

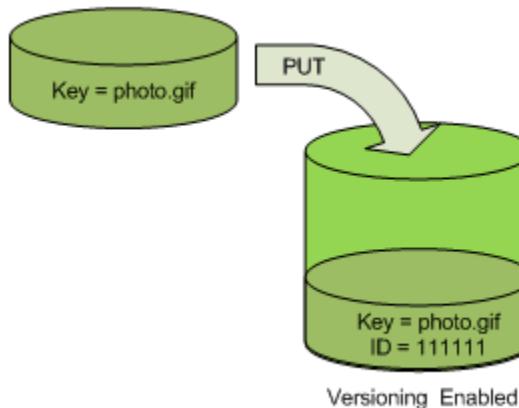
Adding Objects to Versioning-Enabled Buckets

Topics

- [Using the Console \(p. 428\)](#)
- [Using the AWS SDKs \(p. 428\)](#)
- [Using the REST API \(p. 428\)](#)

Once you enable versioning on a bucket, Amazon S3 automatically adds a unique version ID to every object stored (using `PUT`, `POST`, or `COPY`) in the bucket.

The following figure shows that Amazon S3 adds a unique version ID to an object when it is added to a versioning-enabled bucket.



Using the Console

For instructions, go to [Uploading Objects into Amazon S3](#) in the Amazon Simple Storage Service Console User Guide.

Using the AWS SDKs

For examples of uploading objects using the AWS SDKs for Java, .NET, and PHP, see [Uploading Objects \(p. 145\)](#). The examples for uploading objects in nonversioned and versioning-enabled buckets are the same, although in the case of versioning-enabled buckets, Amazon S3 assigns a version number. Otherwise, the version number is null.

For information about using other AWS SDKs, go to [Sample Code and Libraries](#).

Using the REST API

Adding Objects to Versioning-Enabled Buckets

1	Enable versioning on a bucket using a <code>PUT Bucket versioning</code> request. For more information, go to PUT Bucket versioning .
2	Send a <code>PUT</code> , <code>POST</code> , or <code>COPY</code> request to store an object in the bucket.

When you add an object to a versioning-enabled bucket, Amazon S3 returns the version ID of the object in the `x-amz-versionid` response header, for example:

```
x-amz-version-id: 3/L4kqtJlcpXroDTDmJ+rmSpXd3dIbrHY
```

Note

Normal Amazon S3 rates apply for every version of an object stored and transferred. Each version of an object is the entire object; it is not just a diff from the previous version. Thus, if you have three versions of an object stored, you are charged for three objects.

Note

The version ID values that Amazon S3 assigns are URL safe (can be included as part of a URI).

Listing Objects in a Versioning-Enabled Bucket

Topics

- [Using the Console \(p. 429\)](#)
- [Using the AWS SDKs \(p. 429\)](#)
- [Using the REST API \(p. 432\)](#)

This section provides an example of listing object versions from a versioning-enabled bucket. Amazon S3 stores object version information in the *versions* subresource (see [Bucket Configuration Options \(p. 57\)](#)) associated with the bucket.

Using the Console

If your bucket is versioning-enabled, the console provides buttons for you to optionally show or hide object versions. If you hide object versions, the console shows only the list of the latest object versions.

Using the AWS SDKs

The code examples in this section retrieve an object listing from a version-enabled bucket. Each request returns up to 1000 versions. If you have more, you will need to send a series of requests to retrieve a list of all versions. To illustrate how pagination works, the code examples limit the response to two object versions. If there are more than two object versions in the bucket, the response returns the `IsTruncated` element with the value "true" and also includes the `NextKeyMarker` and `NextVersionIdMarker` elements whose values you can use to retrieve the next set of object keys. The code example includes these values in the subsequent request to retrieve the next set of objects.

For information about using other AWS SDKs, go to [Sample Code and Libraries](#).

Using the AWS SDK for Java

For information about how to create and test a working sample, see [Testing the Java Code Examples \(p. 545\)](#).

```
import java.io.IOException;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.ListVersionsRequest;
import com.amazonaws.services.s3.model.S3VersionSummary;
import com.amazonaws.services.s3.model.VersionListing;

public class ListKeysVersionEnabledBucket {
```

```

private static String bucketName = "**** bucket name ****";

public static void main(String[] args) throws IOException {
    AmazonS3 s3client = new AmazonS3Client(new ProfileCredentialsProvider());

    try {
        System.out.println("Listing objects");

        ListVersionsRequest request = new ListVersionsRequest()
            .withBucketName(bucketName)
            .withMaxResults(2);
        // you can specify .withPrefix to obtain version list for a
        specific object or objects with
        // the specified key prefix.

        VersionListing versionListing;
        do {
            versionListing = s3client.listVersions(request);
            for (S3VersionSummary objectSummary :
                versionListing.getVersionSummaries()) {
                System.out.println(" - " + objectSummary.getKey() + " " +
                    "(size = " + objectSummary.getSize() + ")" +
                    "(versionID= " + objectSummary.getVersionId() + ")");
            }
            request.setKeyMarker(versionListing.getNextKeyMarker());
            request.setVersionIdMarker(versionListing.getNextVersionIdMark
er());
        } while (versionListing.isTruncated());
    } catch (AmazonServiceException ase) {
        System.out.println("Caught an AmazonServiceException, " +
            "which means your request made it " +
            "to Amazon S3, 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 " +
            "an internal error while trying to communicate" +
            " with S3, " +
            "such as not being able to access the network.");
        System.out.println("Error Message: " + ace.getMessage());
    }
}
}

```

Using the AWS SDK for .NET

For information about how to create and test a working sample, see [Testing the .NET Code Examples \(p. 547\)](#).

```
using System;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazonaws.com.docsamples
{
    class ListObjectsVersioningEnabledBucket
    {
        static string bucketName = "**** bucket name ****";

        public static void Main(string[] args)
        {
            using (var client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1))
            {
                Console.WriteLine("Listing objects stored in a bucket");

                GetObjectListWithAllVersions(client);
            }

            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }

        static void GetObjectListWithAllVersions(IAmazonS3 client)
        {
            try
            {
                ListVersionsRequest request = new ListVersionsRequest()
                {
                    BucketName = bucketName,
                    // You can optionally specify key name prefix in the request

                    // if you want list of object versions of a specific object.

                    // For this example we limit response to return list of 2
                    MaxKeys = 2
                };
                do
                {
                    ListVersionsResponse response = client.ListVersions(request);

                    // Process response.
                    foreach (S3ObjectVersion entry in response.Versions)
                    {
                        Console.WriteLine("key = {0} size = {1}",
                            entry.Key, entry.Size);
                    }

                    // If response is truncated, set the marker to get the next
                    // set of keys.
                    if (response.IsTruncated)
                    {
                        request.KeyMarker = response.NextKeyMarker;
                        request.VersionIdMarker = response.NextVersionIdMarker;
                    }
                }
            }
        }
    }
}
```

```
        }
        else
        {
            request = null;
        }
    } while (request != null);

}
catch (AmazonS3Exception amazonS3Exception)
{
    if (amazonS3Exception.ErrorCode != null &&
        (amazonS3Exception.ErrorCode.Equals("InvalidAccessKeyId")
        ||
        amazonS3Exception.ErrorCode.Equals("InvalidSecurity")))
    {
        Console.WriteLine("Check the provided AWS Credentials.");
        Console.WriteLine(
            "To sign up for service, go to http://aws.amazon.com/s3");
    }
    else
    {
        Console.WriteLine(
            "Error occurred. Message:{0} when listing objects",
            amazonS3Exception.Message);
    }
}
}
```

Using the REST API

To list all of the versions of all of the objects in a bucket, you use the `versions` subresource in a GET Bucket request. Amazon S3 can retrieve only a maximum of 1000 objects, and each object version counts fully as an object. Therefore, if a bucket contains two keys (e.g., `photo.gif` and `picture.jpg`), and the first key has 990 versions and the second key has 400 versions; a single request would retrieve all 990 versions of `photo.gif` and only the most recent 10 versions of `picture.jpg`.

Amazon S3 returns object versions in the order in which they were stored, with the most recently stored returned first.

To list all object versions in a bucket

- In a GET Bucket request, include the `versions` sub-resource.

```
GET /?versions HTTP/1.1
Host: bucketName.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 +0000
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQF4/cRonhpabX5sCYVf1bNRuU=
```

Retrieving a Subset of Objects in a Bucket

This section discusses the following two example scenarios:

- You want to retrieve a subset of all object versions in a bucket, for example, retrieve all versions of a specific object.
- The number of object versions in the response exceeds the value for *max-key* (1000 by default), so that you have to submit a second request to retrieve the remaining object versions.

To retrieve a subset of object versions, you use the request parameters for GET Bucket. For more information, go to [GET Bucket](#).

Example 1: Retrieving All Versions of Only a Specific Object

You can retrieve all versions of an object using the *versions* subresource and the *prefix* request parameter using the following process. For more information about *prefix*, go to [GET Bucket](#).

Retrieving All Versions of a Key

- | | |
|---|--|
| 1 | Set the <i>prefix</i> parameter to the key of the object you want to retrieve. |
| 2 | Send a GET Bucket request using the <i>versions</i> subresource and <i>prefix</i> .
GET /?versions&prefix=objectName HTTP/1.1 |

Example Retrieving Objects Using a Prefix

The following example retrieves objects whose key is or begins with *myObject*.

```
GET /?versions&prefix=myObject HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

You can use the other request parameters to retrieve a subset of all versions of the object. For more information, go to [GET Bucket](#).

Example 2: Retrieving a Listing of Additional Objects if the Response Is Truncated

If the number of objects that could be returned in a GET request exceeds the value of *max-keys*, the response contains <*isTruncated*>true</*isTruncated*>, and includes the first key (in *NextKeyMarker*) and the first version ID (in *NextVersionIdMarker*) that satisfy the request, but were not returned. You use those returned values as the starting position in a subsequent request to retrieve the additional objects that satisfy the GET request.

Use the following process to retrieve additional objects that satisfy the original GET Bucket *versions* request from a bucket. For more information about *key-marker*, *version-id-marker*, *NextKeyMarker*, and *NextVersionIdMarker*, go to [GET Bucket](#).

Retrieving Additional Responses that Satisfy the Original GET Request

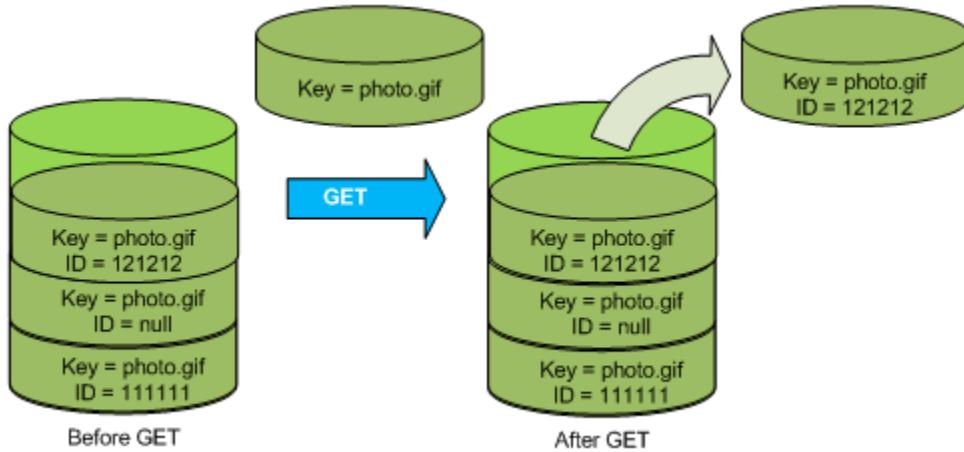
- | | |
|---|--|
| 1 | Set the value of <i>key-marker</i> to the key returned in <i>NextKeyMarker</i> in the previous response. |
| 2 | Set the value of <i>version-id-marker</i> to the version ID returned in <i>NextVersionIdMarker</i> in the previous response. |
| 3 | Send a GET Bucket <i>versions</i> request using <i>key-marker</i> and <i>version-id-marker</i> . |

Example Retrieving Objects Starting with a Specified Key and Version ID

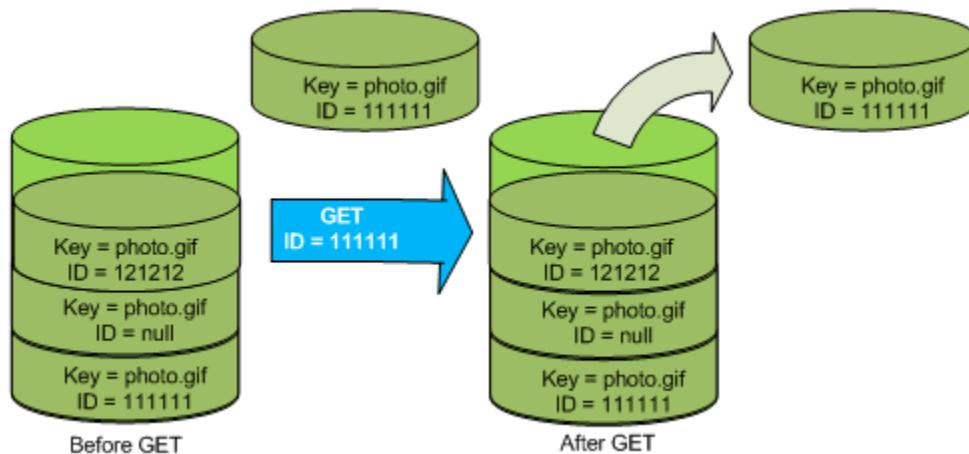
```
GET /?versions&key-marker=myObject&version-id-marker=298459348571 HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

Retrieving Object Versions

A simple GET request retrieves the current version of an object. The following figure shows how GET returns the current version of the object, photo.gif.



To retrieve a specific version, you have to specify its version ID. The following figure shows that a GET versionId request retrieves the specified version of the object (not necessarily the current one).



Using the Console

For instructions go to, [Downloading an Object](#) in the Amazon Simple Storage Service Console User Guide. You will need to click the **Show** button in the console to list all object versions.

Using the AWS SDKs

For examples of uploading objects using AWS SDKs for Java, .NET, and PHP, see [Getting Objects \(p. 130\)](#). The examples for uploading objects in a nonversioned and versioning-enabled buckets are the same, although in the case of versioning-enabled buckets, Amazon S3 assigns a version number. Otherwise, the version number is null.

For information about using other AWS SDKs, go to [Sample Code and Libraries](#).

Using REST

To retrieve a specific object version

1. Set `versionId` to the ID of the version of the object you want to retrieve.
2. Send a GET Object `versionId` request.

Example Retrieving a Versioned Object

The following request retrieves version L4kqtJlcpXroDTDmpUMLUo of `my-image.jpg`.

```
GET /my-image.jpg?versionId=L4kqtJlcpXroDTDmpUMLUo HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

Related Topics

[Retrieving the Metadata of an Object Version \(p. 435\)](#)

Retrieving the Metadata of an Object Version

If you only want to retrieve the metadata of an object (and not its content), you use the HEAD operation. By default, you get the metadata of the most recent version. To retrieve the metadata of a specific object version, you specify its version ID.

To retrieve the metadata of an object version

1. Set `versionId` to the ID of the version of the object whose metadata you want to retrieve.
2. Send a HEAD Object `versionId` request.

Example Retrieving the Metadata of a Versioned Object

The following request retrieves the metadata of version 3HL4kqCxf3vjVBH40Nrjfk of `my-image.jpg`.

```
HEAD /my-image.jpg?versionId=3HL4kqCxf3vjVBH40Nrjfk HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

The following shows a sample response.

```
HTTP/1.1 200 OK
x-amz-id-2: ef8yU9AS1ed4OpIszj7UDNEHGran
x-amz-request-id: 318BC8BC143432E5
```

```
x-amz-version-id: 3HL4kqtJlcpXroDTDmjVBH40Nrjfkd
Date: Wed, 28 Oct 2009 22:32:00 GMT
Last-Modified: Sun, 1 Jan 2006 12:00:00 GMT
ETag: "fba9dede5f27731c9771645a39863328"
Content-Length: 434234
Content-Type: text/plain
Connection: close
Server: AmazonS3
```

Deleting Object Versions

You can delete object versions whenever you want. In addition, you can also define lifecycle configuration rules for objects that have a well-defined lifecycle to request Amazon S3 to expire current object versions or permanently remove noncurrent object versions. When your bucket is version-enabled or versioning is suspended, the lifecycle configuration actions work as follows:

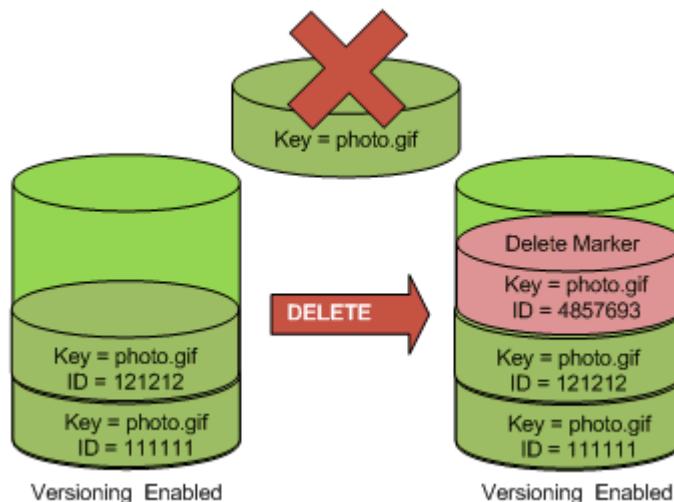
- The `Expiration` action applies to the current object version and instead of deleting the current object version, Amazon S3 retains the current version as a noncurrent version by adding a delete marker, which then becomes the current version.
- The `NoncurrentVersionExpiration` action applies to noncurrent object versions, and Amazon S3 permanently removes these object versions. You cannot recover permanently removed objects.

For more information, see [Object Lifecycle Management \(p. 86\)](#).

A `DELETE` request has the following use cases:

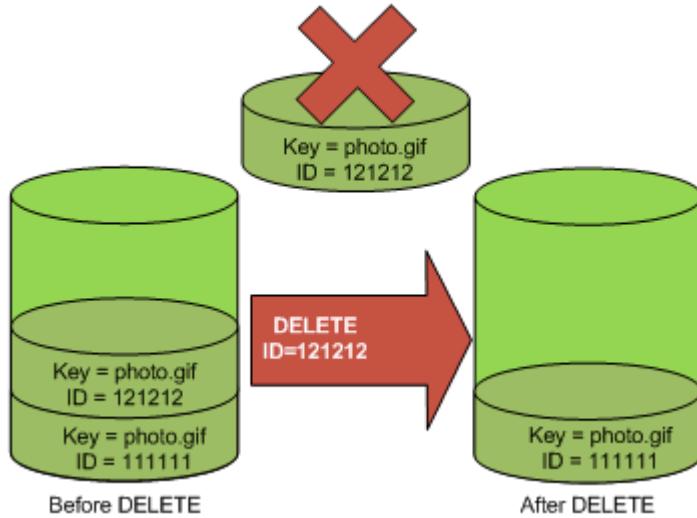
- When versioning is enabled, a simple `DELETE` cannot permanently delete an object. Instead, Amazon S3 inserts a delete marker in the bucket, and that marker becomes the current version of the object with a new ID. When you try to `GET` an object whose current version is a delete marker, Amazon S3 behaves as though the object has been deleted (even though it has not been erased) and returns a 404 error.

The following figure shows that a simple `DELETE` does not actually remove the specified object. Instead, Amazon S3 inserts a delete marker.



- To permanently delete versioned objects, you must use `DELETE Object versionId`.

The following figure shows that deleting a specified object version permanently removes that object.



Using the Console

For instructions go to, [Downloading an Object](#) in the Amazon Simple Storage Service Console User Guide. You will need to click the **Show** button in the console to list all object versions.

Using the AWS SDKs

For examples of uploading objects using the AWS SDKs for Java, .NET, and PHP, see [Deleting Objects \(p. 228\)](#). The examples for uploading objects in nonversioned and versioning-enabled buckets are the same, although in the case of versioning-enabled buckets, Amazon S3 assigns a version number. Otherwise, the version number is null.

For information about using other AWS SDKs, go to [Sample Code and Libraries](#).

Using REST

To delete a specific version of an object

- In a `DELETE`, specify a version ID.

Example Deleting a Specific Version

The following example shows how to delete version UIORUnfnd89493jJFJ of `photo.gif`.

```
DELETE /photo.gif?versionId=UIORUnfnd89493jJFJ HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 12 Oct 2009 17:50:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:xQE0diMbLRepdf3YB+FIEXAMPLE=
Content-Type: text/plain
Content-Length: 0
```

Related Topics

[Using MFA Delete \(p. 438\)](#)

[Working with Delete Markers \(p. 438\)](#)

[Removing Delete Markers \(p. 440\)](#)

[Using Versioning \(p. 422\)](#)

Using MFA Delete

If a bucket's versioning configuration is MFA Delete–enabled, the bucket owner must include the `x-amz-mfa` request header in requests to permanently delete an object version or change the versioning state of the bucket. Requests that include `x-amz-mfa` must use HTTPS. The header's value is the concatenation of your authentication device's serial number, a space, and the authentication code displayed on it. If you do not include this request header, the request fails.

For more information about authentication devices, go to <http://aws.amazon.com/mfa/>.

Example Deleting an Object from an MFA Delete Enabled Bucket

The following example shows how to delete `my-image.jpg` (with the specified version), which is in a bucket configured with MFA Delete enabled. Note the space between `[SerialNumber]` and `[AuthenticationCode]`. For more information, go to [DELETE Object](#).

```
DELETE /my-image.jpg?versionId=3HL4kqCxf3vjVBH40Nrjfkd HTTPS/1.1
Host: bucketName.s3.amazonaws.com
x-amz-mfa: 20899872 301749
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

For more information about enabling MFA delete, see [MFA Delete \(p. 423\)](#).

Working with Delete Markers

A delete marker is a placeholder (marker) for a versioned object that was named in a simple `DELETE` request. Because the object was in a versioning-enabled bucket, the object was not deleted. The delete marker, however, makes Amazon S3 behave as if it had been deleted.

A delete marker has a key name (or key) and version ID like any other object. However, a delete marker differs from other objects in the following ways:

- It does not have data associated with it.
- It is not associated with an access control list (ACL) value.
- It does not retrieve anything from a `GET` request because it has no data; you get a 404 error.
- The only operation you can use on a delete marker is `DELETE`, and only the bucket owner can issue such a request.

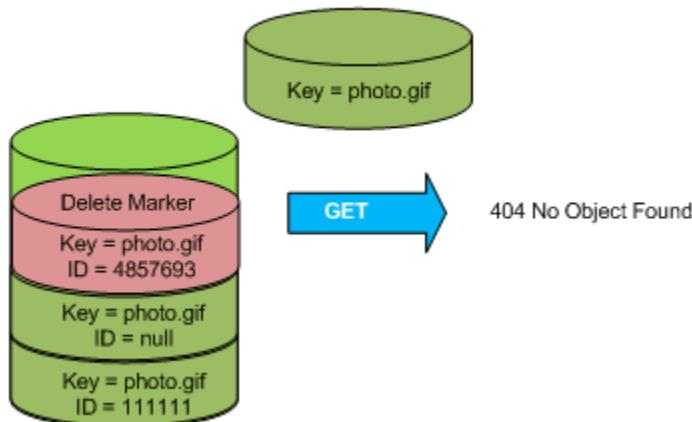
Delete markers accrue a nominal charge for storage in Amazon S3. The storage size of a delete marker is equal to the size of the key name of the delete marker. A key name is a sequence of Unicode characters. The UTF-8 encoding adds from 1 to 4 bytes of storage to your bucket for each character in the name. For more information about key names, see [Object Keys \(p. 79\)](#). For information about deleting a delete marker, see [Removing Delete Markers \(p. 440\)](#).

Only Amazon S3 can create a delete marker, and it does so whenever you send a `DELETE Object` request on an object in a versioning-enabled or suspended bucket. The object named in the `DELETE` request is not actually deleted. Instead, the delete marker becomes the current version of the object. (The object's key name (or key) becomes the key of the delete marker.) If you try to get an object and its current version is a delete marker, Amazon S3 responds with:

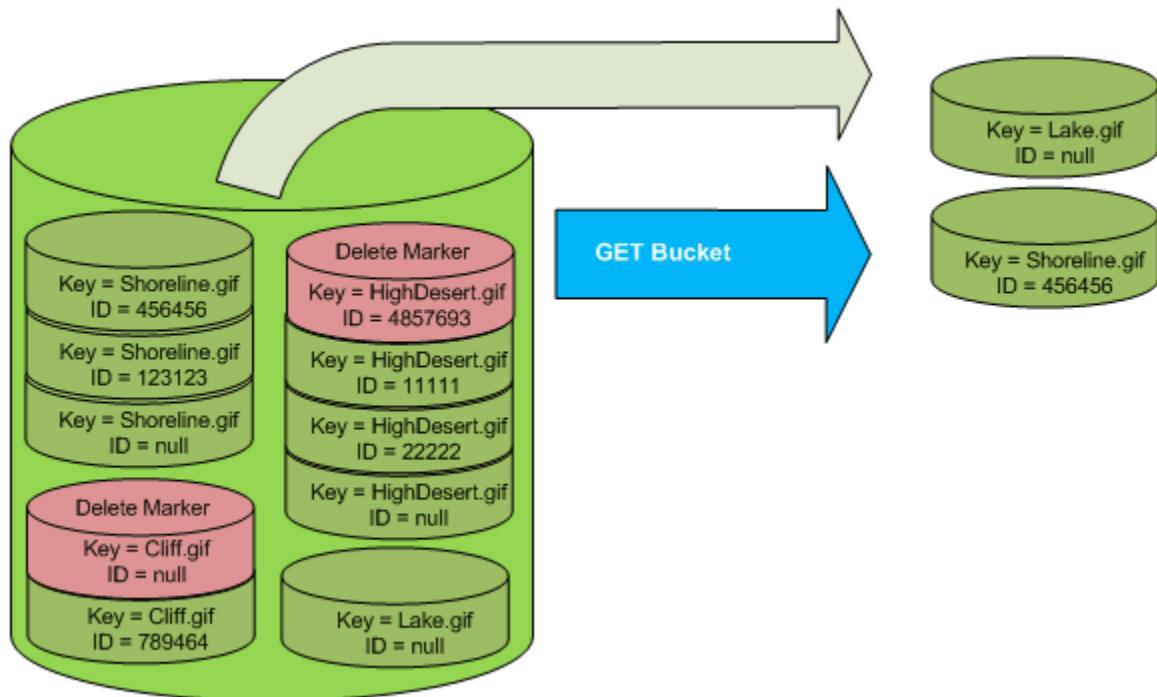
- A 404 (Object not found) error
- A response header, x-amz-delete-marker: true

The response header tells you that the object accessed was a delete marker. This response header never returns `false`; if the value is `false`, Amazon S3 does not include this response header in the response.

The following figure shows how a simple `GET` on an object, whose current version is a delete marker, returns a 404 No Object Found error.



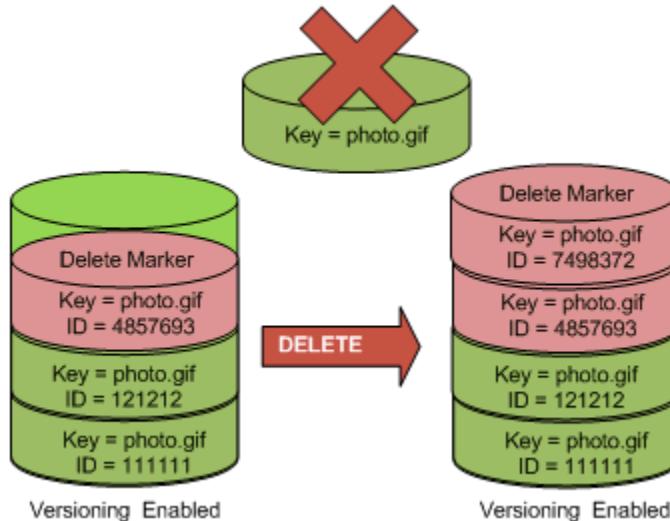
The only way to list delete markers (and other versions of an object) is by using the `versions` subresource in a `GET Bucket versions` request. A simple `GET` does not retrieve delete marker objects. The following figure shows that a `GET Bucket` request does not return objects whose current version is a delete marker.



Removing Delete Markers

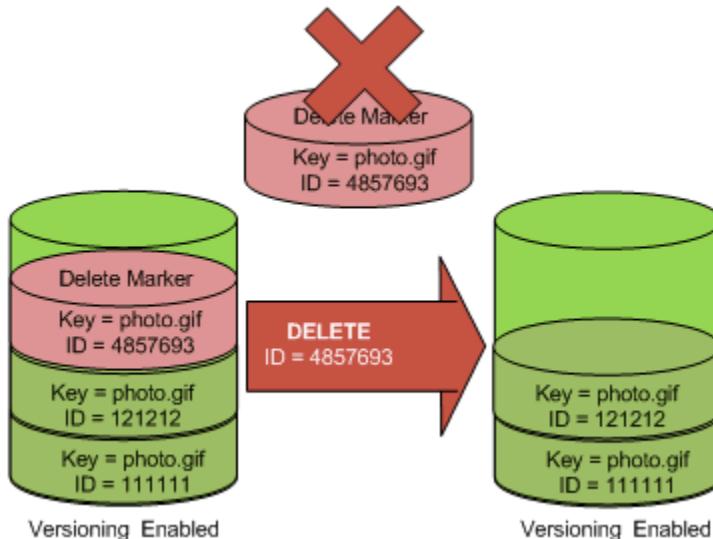
To delete a delete marker, you must specify its version ID in a `DELETE Object versionId` request. If you use a `DELETE` request to delete a delete marker (without specifying the version ID of the delete marker), Amazon S3 does not delete the delete marker, but instead, inserts another delete marker.

The following figure shows how a simple `DELETE` on a delete marker removes nothing, but adds a new delete marker to a bucket.



In a versioning-enabled bucket, this new delete marker would have a unique version ID. So, it's possible to have multiple delete markers of the same object in one bucket.

To permanently delete a delete marker, you must include its version ID in a `DELETE Object versionId` request. The following figure shows how a `DELETE Object versionId` request permanently removes a delete marker. Only the owner of a bucket can permanently remove a delete marker.



The effect of removing the delete marker is that a simple `GET` request will now retrieve the current version (121212) of the object.

To permanently remove a delete marker

1. Set *versionId* to the ID of the version to the delete marker you want to remove.
2. Send a `DELETE Object` *versionId* request.

Example Removing a Delete Marker

The following example removes the delete marker for `photo.gif` version 4857693.

```
DELETE /photo.gif?versionId=4857693 HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
```

When you delete a delete marker, Amazon S3 includes in the response:

```
204 NoContent
x-amz-version-id: versionID
x-amz-delete-marker: true
```

Transitioning Object Versions

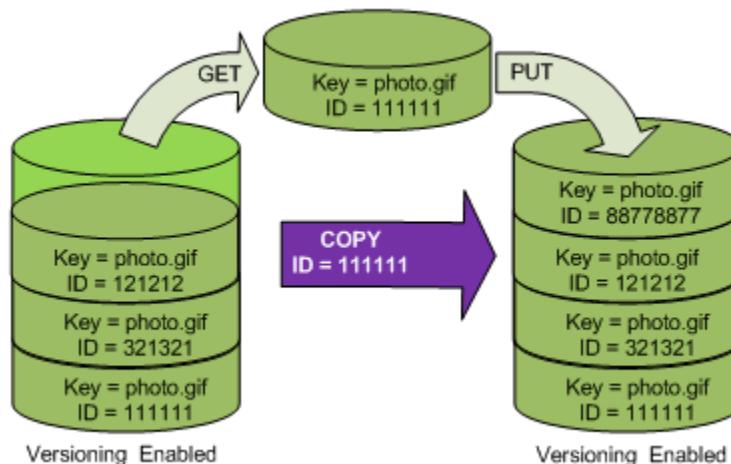
You can define lifecycle configuration rules for objects that have a well-defined lifecycle to transition object versions to the `GLACIER` storage class at a specific time in the object's lifetime. For more information, see [Object Lifecycle Management \(p. 86\)](#).

Restoring Previous Versions

One of the value propositions of versioning is the ability to retrieve previous versions of an object. There are two approaches to doing so:

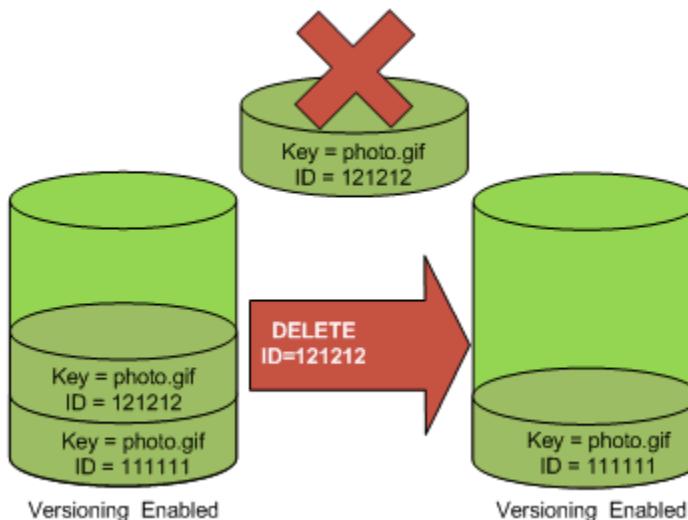
- Copy a previous version of the object into the same bucket
The copied object becomes the current version of that object and all object versions are preserved.
- Permanently delete the current version of the object
When you delete the current object version, you, in effect, turn the previous version into the current version of that object.

Because all object versions are preserved, you can make any earlier version the current version by copying a specific version of the object into the same bucket. In the following figure, the source object (*ID* = 111111) is copied into the same bucket. Amazon S3 supplies a new ID (88778877) and it becomes the current version of the object. So, the bucket has both the original object version (111111) and its copy (88778877).



A subsequent GET will retrieve version 88778877.

The following figure shows how deleting the current version (121212) of an object, which leaves the previous version (111111) as the current object.



A subsequent GET will retrieve version 111111.

Versioned Object Permissions

Permissions are set at the version level. Each version has its own object owner; an AWS account that creates the object version is the owner. So, you can set different permissions for different versions of the same object. To do so, you must specify the version ID of the object whose permissions you want to set in a `PUT Object versionId acl` request. For a detailed description and instructions on using ACLs, see [Managing Access Permissions to Your Amazon S3 Resources \(p. 269\)](#).

Example Setting Permissions for an Object Version

The following request sets the permission of the grantee, BucketOwner@amazon.com, to FULL_CONTROL on the key, my-image.jpg, version ID, 3HL4kqtJvjVBH40NrjfkD.

```
PUT /my-image.jpg?acl&versionId=3HL4kqtJvjVBH40NrjfkD HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU=
Content-Length: 124

<AccessControlPolicy>
  <Owner>
    <ID>75cc57f09aa0c8caeab4f8c24e99d10f8e7faebf76c078efc7c6caea54ba06a</ID>
    <DisplayName>mtd@amazon.com</DisplayName>
  </Owner>
  <AccessControlList>
    <Grant>
      <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="CanonicalUser">
        <ID>a9a7b886d6fd24a52fe8ca5bef65f89a64e0193f23000e241bf9b1c61be666e9</ID>

        <DisplayName>BucketOwner@amazon.com</DisplayName>
      </Grantee>
      <Permission>FULL_CONTROL</Permission>
    </Grant>
  </AccessControlList>
</AccessControlPolicy>
```

Likewise, to get the permissions of a specific object version, you must specify its version ID in a GET Object `versionId acl` request. You need to include the version ID because, by default, GET Object `acl` returns the permissions of the current version of the object.

Example Retrieving the Permissions for a Specified Object Version

In the following example, Amazon S3 returns the permissions for the key, my-image.jpg, version ID, DVBH40Nr8X8gUMLUo.

```
GET /my-image.jpg?versionId=DVBH40Nr8X8gUMLUo&acl HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: AWS AKIAIOSFODNN7EXAMPLE:0RQf4/cRonhpaBX5sCYVf1bNRuU
```

For more information, go to [GET Object acl](#).

Managing Objects in a Versioning-Suspended Bucket

Topics

- [Adding Objects to Versioning-Suspended Buckets \(p. 444\)](#)
- [Retrieving Objects from Versioning-Suspended Buckets \(p. 445\)](#)
- [Deleting Objects from Versioning-Suspended Buckets \(p. 445\)](#)

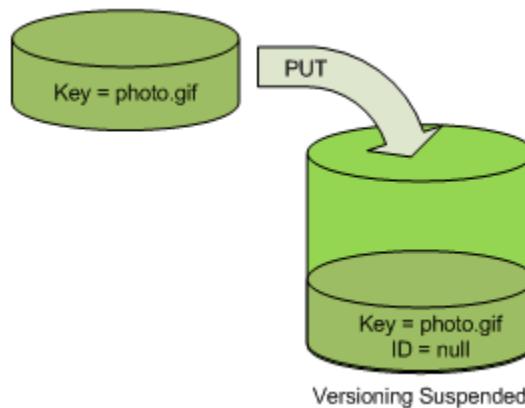
You suspend versioning to stop accruing new versions of the same object in a bucket. You might do this because you only want a single version of an object in a bucket, or you might not want to accrue charges for multiple versions.

When you suspend versioning, existing objects in your bucket do not change. What changes is how Amazon S3 handles objects in future requests. The topics in this section explain various object operations in a versioning-suspended bucket.

Adding Objects to Versioning-Suspended Buckets

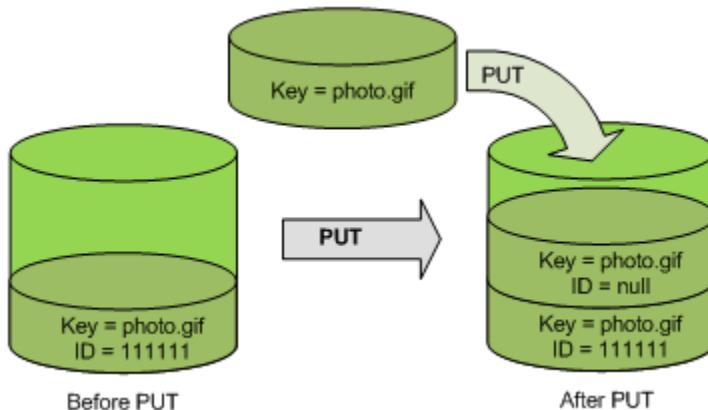
Once you suspend versioning on a bucket, Amazon S3 automatically adds a `null` version ID to every subsequent object stored thereafter (using `PUT`, `POST`, or `COPY`) in that bucket.

The following figure shows how Amazon S3 adds the version ID of `null` to an object when it is added to a version-suspended bucket.

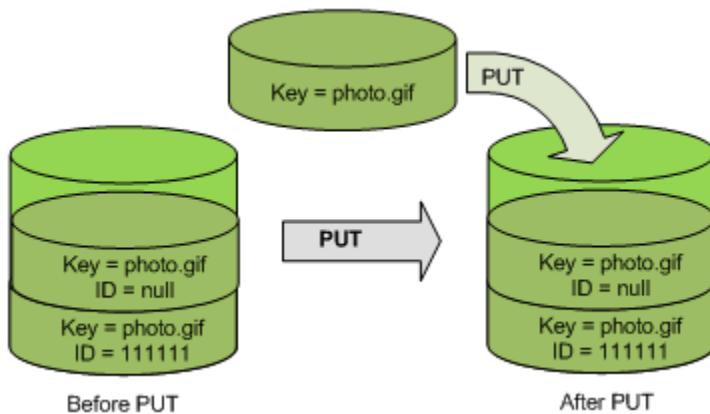


If a null version is already in the bucket and you add another object with the same key, the added object overwrites the original null version.

If there are versioned objects in the bucket, the version you `PUT` becomes the current version of the object. The following figure shows how adding an object to a bucket that contains versioned objects does not overwrite the object already in the bucket. In this case, version 111111 was already in the bucket. Amazon S3 attaches a version ID of `null` to the object being added and stores it in the bucket. Version 111111 is not overwritten.



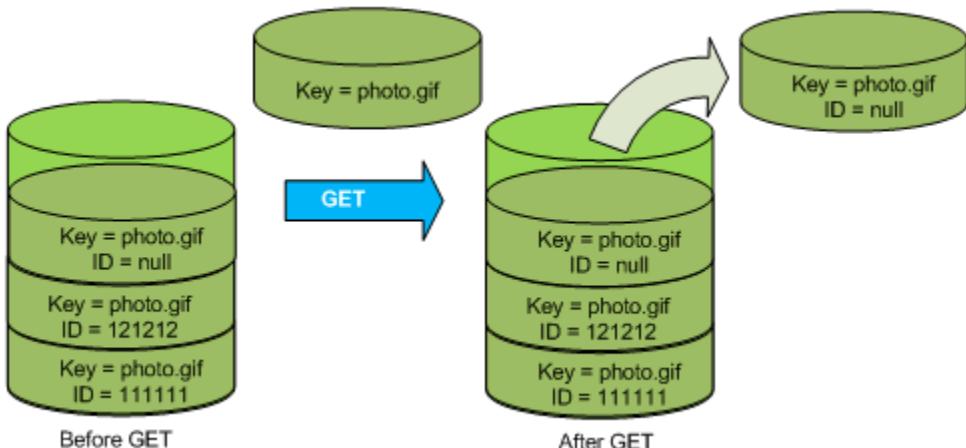
If a null version already exists in a bucket, the null version is overwritten, as shown in the following figure.



Note that although the key and version ID (`null`) of null version are the same before and after the `PUT`, the contents of the null version originally stored in the bucket is replaced by the contents of the object `PUT` into the bucket.

Retrieving Objects from Versioning-Suspended Buckets

A `GET` Object request returns the current version of an object whether you've enabled versioning on a bucket or not. The following figure shows how a simple `GET` returns the current version of an object.

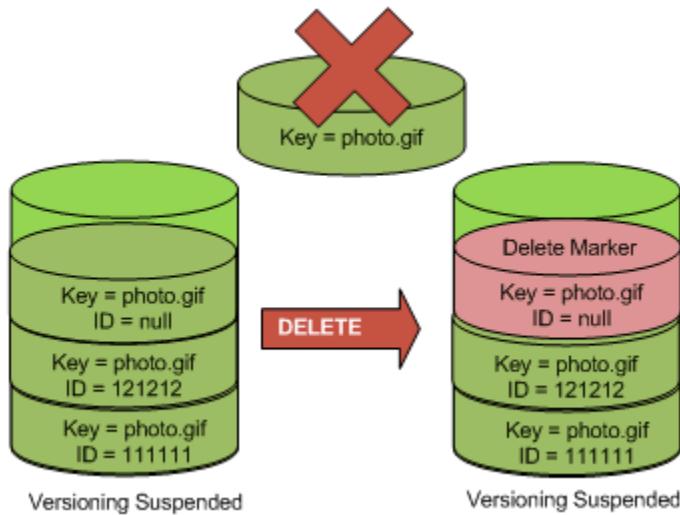


Deleting Objects from Versioning-Suspended Buckets

If versioning is suspended, a `DELETE` request:

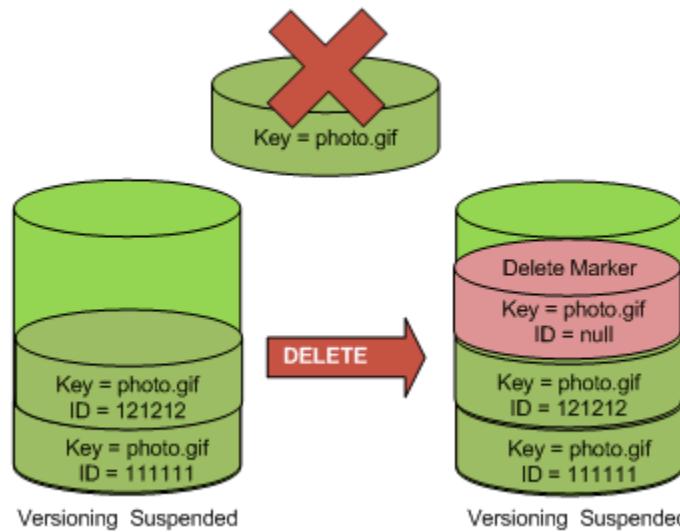
- Can only remove an object whose version ID is `null`
 Doesn't remove anything if there isn't a null version of the object in the bucket.
- Inserts a delete marker into the bucket.

The following figure shows how a simple `DELETE` removes a null version and Amazon S3 inserts a delete marker in its place with a version ID of `null`.

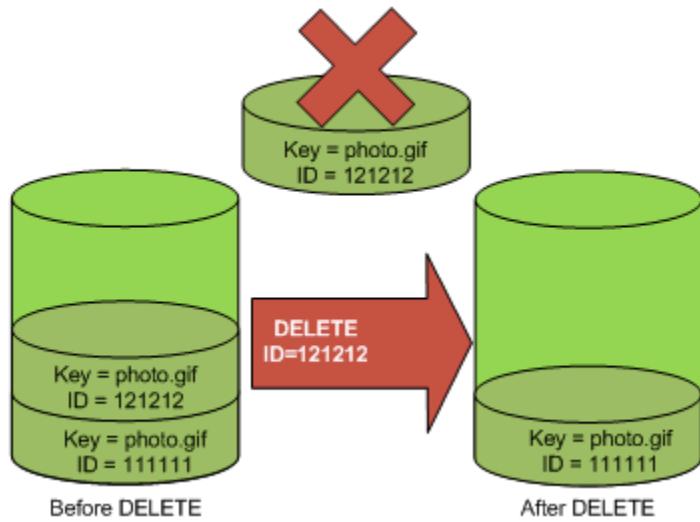


Remember that a delete marker doesn't have content, so you lose the content of the null version when a delete marker replaces it.

The following figure shows a bucket that doesn't have a null version. In this case, the `DELETE` removes nothing; Amazon S3 just inserts a delete marker.



Even in a versioning-suspended bucket, the bucket owner can permanently delete a specified version. The following figure shows that deleting a specified object version permanently removes that object. Only the bucket owner can delete a specified object version.



Hosting a Static Website on Amazon S3

Topics

- [Website Endpoints \(p. 449\)](#)
- [Configure a Bucket for Website Hosting \(p. 451\)](#)
- [Example Walkthroughs - Hosting Websites On Amazon S3 \(p. 462\)](#)

You can host a static website on Amazon S3. On a static website, individual web pages include static content. They may also contain client-side scripts. By contrast, a dynamic website relies on server-side processing, including server-side scripts such as PHP, JSP, or ASP.NET. Amazon S3 does not support server-side scripting.

Note

Amazon Web Services (AWS) has resources for hosting dynamic websites. To learn more about website hosting on AWS, go to [Websites and Website Hosting](#).

To host your static website, you configure an Amazon S3 bucket for website hosting and then upload your website content to the bucket. The website is then available at the region-specific website endpoint of the bucket:

```
<bucket-name>.s3-website-<AWS-region>.amazonaws.com
```

For a list of region specific website endpoints for Amazon S3, see [Website Endpoints \(p. 449\)](#). For example, suppose you create a bucket called `examplebucket` in the US Standard region and configure it as a website. The following example URLs provide access to your website content:

- This URL returns a default index document that you configured for the website.

```
http://examplebucket.s3-website-us-east-1.amazonaws.com/
```

- This URL requests the `photo.jpg` object, which is stored at the root level in the bucket.

```
http://examplebucket.s3-website-us-east-1.amazonaws.com/photo.jpg
```

- This URL requests the `docs/doc1.html` object in your bucket.

```
http://examplebucket.s3-website-us-east-1.amazonaws.com/docs/doc1.html
```

Using Your Own Domain

Instead of accessing the website by using an Amazon S3 website endpoint, you can use your own domain, such as `example.com` to serve your content. Amazon S3, in conjunction with Amazon Route 53, supports hosting a website at the root domain. For example if you have the root domain `example.com` and you host your website on Amazon S3, your website visitors can access the site from their browser by typing either `http://www.example.com` or `http://example.com`. For an example walkthrough, see [Example: Setting Up a Static Website Using a Custom Domain \(p. 464\)](#).

To configure a bucket for website hosting, you add website configuration to the bucket. For more information, see [Configure a Bucket for Website Hosting \(p. 451\)](#).

Website Endpoints

Topics

- [Key Differences Between the Amazon Website and the REST API Endpoint \(p. 450\)](#)

When you configure a bucket for website hosting, the website is available via the region-specific website endpoint. Website endpoints are different from the endpoints where you send REST API requests. For more information about the endpoints, see [Request Endpoints \(p. 13\)](#).

The general form of an Amazon S3 website endpoint is as follows:

```
bucket-name.s3-website-region.amazonaws.com
```

For example, if your bucket is named `example-bucket` and it resides in the US Standard region, the website is available at the following Amazon S3 website endpoint:

```
http://example-bucket.s3-website-us-east-1.amazonaws.com/
```

The following table lists Amazon S3 regions and the corresponding website endpoints.

Note

The website endpoints do not support https.

Region	Website endpoint
US Standard	<code>bucket-name.s3-website-us-east-1.amazonaws.com</code>
US West (Oregon) region	<code>bucket-name.s3-website-us-west-2.amazonaws.com</code>
US West (Northern California) region	<code>bucket-name.s3-website-us-west-1.amazonaws.com</code>

Region	Website endpoint
EU (Ireland) region	<code>bucket-name.s3-website-eu-west-1.amazonaws.com</code>
EU (Frankfurt) Region	<code>bucket-name.s3-website.eu-central-1.amazonaws.com</code>
Asia Pacific (Singapore) Region	<code>bucket-name.s3-website-ap-southeast-1.amazonaws.com</code>
Asia Pacific (Tokyo) region	<code>bucket-name.s3-website-ap-northeast-1.amazonaws.com</code>
Asia Pacific (Sydney) Region	<code>bucket-name.s3-website-ap-southeast-2.amazonaws.com</code>
South America (Sao Paulo) Region	<code>bucket-name.s3-website-sa-east-1.amazonaws.com</code>

In order for your customers to access content at the website endpoint, you must make all your content publicly readable. To do so, you can use a bucket policy or an ACL on an object to grant the necessary permissions.

Note

Requester Pays buckets or DevPay buckets do not allow access through the website endpoint. Any request to such a bucket will receive a 403 Access Denied response. For more information, see [Requester Pays Buckets \(p. 73\)](#).

If you have a registered domain, you can add a DNS CNAME entry to point to the Amazon S3 website endpoint. For example, if you have registered domain, `www.example-bucket.com`, you could create a bucket `www.example-bucket.com`, and add a DNS CNAME record that points to `www.example-bucket.com.s3-website-<region>.amazonaws.com`. All requests to `http://www.example-bucket.com` will be routed to `www.example-bucket.com.s3-website-<region>.amazonaws.com`. For more information, see [Virtual Hosting of Buckets \(p. 48\)](#).

Key Differences Between the Amazon Website and the REST API Endpoint

The website endpoint is optimized for access from a web browser. The following table describes the key differences between the Amazon REST API endpoint and the website endpoint.

Key Difference	REST API Endpoint	Website Endpoint
Access control	Supports both public and private content.	Supports only publicly readable content.
Error message handling	Returns an XML-formatted error response.	Returns an HTML document.
Redirection support	Not applicable	Supports both object-level and bucket-level redirects.

Key Difference	REST API Endpoint	Website Endpoint
Requests supported	Supports all bucket and object operations	Supports only GET and HEAD requests on objects.
Responses to GET and HEAD requests at the root of a bucket	Returns a list of the object keys in the bucket.	Returns the index document that is specified in the website configuration.
Secure Sockets Layer (SSL) support	Supports SSL connections.	Does not support SSL connections.

Configure a Bucket for Website Hosting

Topics

- [Overview \(p. 451\)](#)
- [Syntax for Specifying Routing Rules \(p. 453\)](#)
- [Index Document Support \(p. 457\)](#)
- [Custom Error Document Support \(p. 458\)](#)
- [Configuring a Web Page Redirect \(p. 460\)](#)
- [Permissions Required for Website Access \(p. 462\)](#)

Overview

To configure a bucket for static website hosting, you add a website configuration to your bucket. The configuration includes the following information:

- Index document

When you type a URL such as `http://example.com` you are not requesting a specific page. In this case the web server serves a default page, for the directory where the requested website content is stored. This default page is referred as index document, and most of the time it is named `index.html`. When you configure a bucket for website hosting, you must specify an index document. Amazon S3 returns this index document when requests are made to the root domain or any of the subfolders. For more information, see [Index Documents and Folders \(p. 457\)](#).

- Error document

If an error occurs, Amazon S3 returns an HTML error document. For 4XX class errors, you can optionally provide your own custom error document, in which you can provide additional guidance to your users. For more information, see [Custom Error Document Support \(p. 458\)](#).

- Redirects all requests

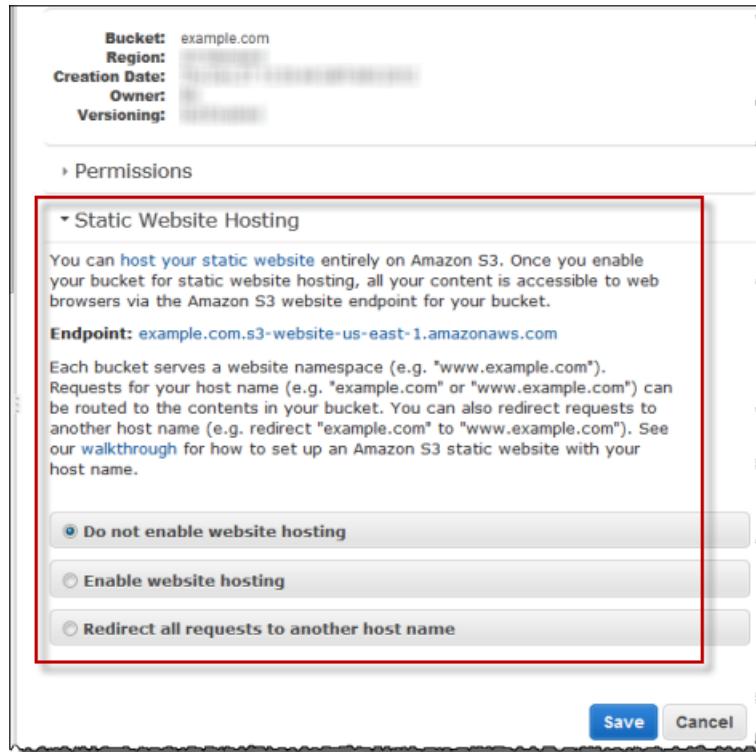
If your root domain is `example.com` and you want to serve requests for both `http://example.com` and `http://www.example.com`, you can create two buckets named `example.com` and `www.example.com`, maintain website content in only one bucket, say, `example.com`, and configure the other bucket to redirect all requests to the `example.com` bucket.

- Advanced conditional redirects

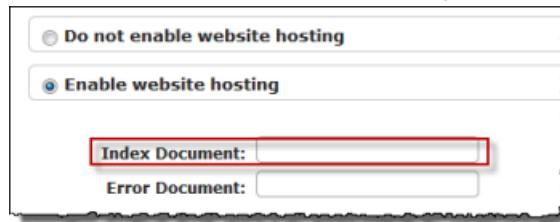
You can conditionally route requests according to specific object key names or prefixes in the request, or according to the response code. For example, suppose that you delete or rename an object in your bucket. You can add a routing rule that redirects the request to another object. Suppose that you want

to make a folder unavailable. You can add a routing rule to redirect the request to another page, which explains why the folder is no longer available. You can also add a routing rule to handle an error condition by routing requests that return the error to another domain, where the error will be processed.

You can manage your buckets website configuration using the [Amazon S3 console](#). The bucket **Properties** panel in the console enables you to specify the website configuration.



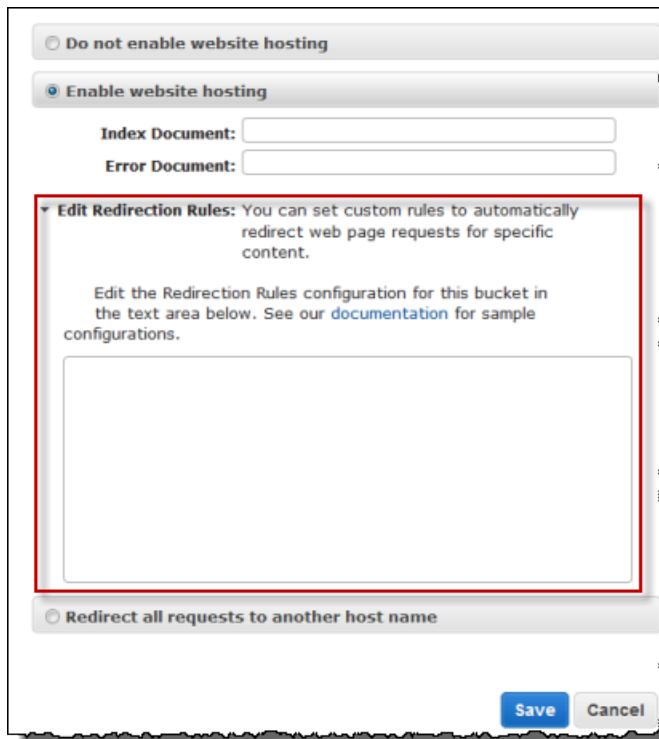
To host a static website on Amazon S3, you need only provide the name of the index document.



To redirect all requests to the bucket's website endpoint to another host, you only need to provide host name.



However, when configuring bucket for website hosting, you can optionally specify advanced redirection rules.



You describe the rules using XML. The following section provides general syntax and examples of specifying redirection rules.

Syntax for Specifying Routing Rules

The following is a general syntax for defining the routing rules in a website configuration:

```
<RoutingRules> =
  <RoutingRules>
    <RoutingRule>...</RoutingRule>
    [<RoutingRule>...</RoutingRule>
     ...]
```

```

        </RoutingRules>

<RoutingRule> =
    <RoutingRule>
        [ <Condition>...</Condition> ]
        <Redirect>...</Redirect>
    </RoutingRule>

<Condition> =
    <Condition>
        [ <KeyPrefixEquals>...</KeyPrefixEquals> ]
        [ <HttpErrorCodeReturnedEquals>...</HttpErrorCodeReturnedEquals> ]
    </Condition>
    Note: <Condition> must has at least one child element.

<Redirect> =
    <Redirect>
        [ <HostName>...</HostName> ]
        [ <Protocol>...</Protocol> ]
        [ <ReplaceKeyPrefixWith>...</ReplaceKeyPrefixWith> ]
        [ <ReplaceKeyWith>...</ReplaceKeyWith> ]
        [ <HttpRedirectCode>...</HttpRedirectCode> ]
    </Redirect>
    Note: <Redirect> must has at least one child element.
        Also, you can have either ReplaceKeyPrefix with or ReplaceKeyWith,
        but not both.

```

The following table describes the elements in the routing rule.

Name	Description
<i>RoutingRules</i>	Container for a collection of <i>RoutingRule</i> elements.
<i>RoutingRule</i>	A rule that identifies a condition and the redirect that is applied when the condition is met. Condition: A <i>RoutingRules</i> , container must contain at least one routing rule.
<i>Condition</i>	Container for describing a condition that must be met for the specified redirect to be applied. If the routing rule does not include a condition, the rule is applied to all requests.
<i>KeyPrefixEquals</i>	The object key name prefix from which requests will be redirected. <i>KeyPrefixEquals</i> is required if <i>HttpErrorCodeReturnedEquals</i> is not specified. If both <i>KeyPrefixEquals</i> and <i>HttpErrorCodeReturnedEquals</i> are specified, both must be true for the condition to be met.
<i>HttpErrorCodeReturnedEquals</i>	The HTTP error code that must match for the redirect to apply. In the event of an error, if the error code meets this value, then specified redirect applies. <i>HttpErrorCodeReturnedEquals</i> is required if <i>KeyPrefixEquals</i> is not specified. If both <i>KeyPrefixEquals</i> and <i>HttpErrorCodeReturnedEquals</i> are specified, both must be true for the condition to be met.

Name	Description
<i>Redirect</i>	Container element that provides instructions for redirecting the request. You can redirect requests to another host, or another page, or you can specify another protocol to use. A <code>RoutingRule</code> must have a <code>Redirect</code> element. A <code>Redirect</code> element must contain at least one of the following elements: <code>Protocol</code> , <code>HostName</code> , <code>ReplaceKeyPrefixWith</code> , <code>ReplaceKeyWith</code> or <code>HttpRedirectCode</code> .
<i>Protocol</i>	The protocol, http or https, to be used in the Location header that is returned in the response. <i>Protocol</i> is not required if one of its siblings is supplied.
<i>HostName</i>	The host name to be used in the Location header that is returned in the response. <i>HostName</i> is not required if one of its siblings is supplied.
<i>ReplaceKeyPrefixWith</i>	The object key name prefix that will replace the value of <code>KeyPrefixEquals</code> in the redirect request. <i>ReplaceKeyPrefixWith</i> is not required if one of its siblings is supplied. It can be supplied only if <code>ReplaceKeyWith</code> is not supplied.
<i>ReplaceKeyWith</i>	The object key to be used in the Location header that is returned in the response. <i>ReplaceKeyWith</i> is not required if one of its siblings is supplied. It can be supplied only if <code>ReplaceKeyPrefixWith</code> is not supplied.
<i>HttpRedirectCode</i>	The HTTP redirect code to be used in the Location header that is returned in the response. <i>HttpRedirectCode</i> is not required if one of its siblings is supplied.

The following are some of the examples:

Example 1: Redirect after renaming a key prefix

Suppose your bucket contained the following objects:

```
index.html  
docs/article1.html  
docs/article2.html
```

Now you decided to rename the folder from `docs/` to `documents/`. After you make this change, you will need to redirect requests for prefix `/docs` to `documents/`. For example, request for `docs/article1.html` will need to be redirected to `documents/article1.html`.

In this case you add the following routing rule to the website configuration:

```
<RoutingRules>  
  <RoutingRule>  
    <Condition>  
      <KeyPrefixEquals>docs/</KeyPrefixEquals>  
    </Condition>  
    <Redirect>  
      <ReplaceKeyPrefixWith>documents/</ReplaceKeyPrefixWith>  
    </Redirect>  
  </RoutingRule>  
</RoutingRules>
```

Example 2: Redirect requests for a deleted folder to a page

Suppose you delete the `images/` folder (that is, you delete all objects with key prefix `images/`). You can add a routing rule that redirects requests for any object with the key prefix `images/` to a page named `folderdeleted.html`.

```
<RoutingRules>  
  <RoutingRule>  
    <Condition>  
      <KeyPrefixEquals>images/</KeyPrefixEquals>  
    </Condition>  
    <Redirect>  
      <ReplaceKeyWith>folderdeleted.html</ReplaceKeyWith>  
    </Redirect>  
  </RoutingRule>  
</RoutingRules>
```

Example 3: Redirect for an HTTP error

Suppose that when a requested object is not found, you want to redirect requests to an Amazon EC2 instance. You can add a redirection rule so that when an HTTP status code 404 (Not Found) is returned the site visitor is redirected to an EC2 instance that will handle the request. The following example also inserts the object key prefix `report-404/` in the redirect. For example, if you request a page `ExamplePage.html` and it results in a HTTP 404 error, the request is redirected to a page `report-404/ExamplePage.html` on the specified EC2 instance. If there is no routing rule and the HTTP error 404 occurs, the error document specified in the configuration is returned.

```
<RoutingRules>
  <RoutingRule>
    <Condition>
      <HttpErrorCodeReturnedEquals>404</HttpErrorCodeReturnedEquals>
    </Condition>
    <Redirect>
      <HostName>ec2-11-22-333-44.compute-1.amazonaws.com</HostName>
      <ReplaceKeyPrefixWith>report-404/</ReplaceKeyPrefixWith>
    </Redirect>
  </RoutingRule>
</RoutingRules>
```

Index Document Support

An index document is a webpage that is returned when a request is made to the root of a website or any subfolder. For example, if a user enters `http://www.example.com` in the browser, the user is not requesting any specific page. In that case, Amazon S3 serves up the index document, which is sometimes referred to as the default page.

When you configure your bucket as a website, you should provide the name of the index document. You must upload an object with this name and configure it to be publicly readable. For information about configuring a bucket as a website, see [Example: Setting Up a Static Website \(p. 463\)](#).

The trailing slash at the root-level URL is optional. For example, if you configure your website with `index.html` as the index document, either of the following two URLs will return `index.html`.

```
http://example-bucket.s3-website-region.amazonaws.com/
http://example-bucket.s3-website-region.amazonaws.com
```

For more information about Amazon S3 website endpoints, see [Website Endpoints \(p. 449\)](#).

Index Documents and Folders

In Amazon S3, a bucket is a flat container of objects; it does not provide any hierarchical organization as the file system on your computer does. You can create a logical hierarchy by using object key names that imply a folder structure. For example, consider a bucket with three objects and the following key names.

`sample1.jpg`

`photos/2006/Jan/sample2.jpg`

`photos/2006/Feb/sample3.jpg`

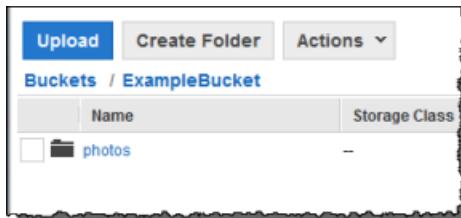
Although these are stored with no physical hierarchical organization, you can infer the following logical folder structure from the key names.

sample1.jpg object is at the root of the bucket

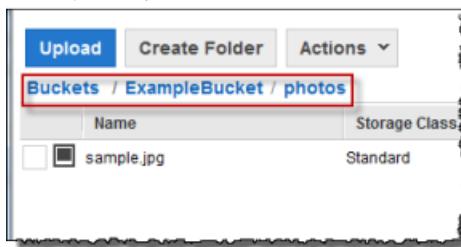
sample2.jpg object is in the photos/2006/Jan subfolder, and

sample3.jpg object is in photos/2006/Feb subfolder.

The folder concept that Amazon S3 console supports is based on object key names. To continue the previous example, the console displays the ExampleBucket with a photos folder.



You can upload objects to the bucket or to the photos folder within the bucket. If you add the object sample.jpg to the bucket, the key name is sample.jpg. If you upload the object to the photos folder, the object key name is photos/sample.jpg.



If you create such a folder structure in your bucket, you must have an index document at each level. When a user specifies a URL that resembles a folder lookup, the presence or absence of a trailing slash determines the behavior of the website. For example, the following URL, with a trailing slash, returns the photos/index.html index document.

```
http://example-bucket.s3-website-region.amazonaws.com/photos/
```

However, if you exclude the trailing slash from the preceding URL, Amazon S3 first looks for an object photos in the bucket. If the photos object is not found, then it searches for an index document, photos/index.html. If that document is found, Amazon S3 returns a 302 Found message and points to the photos/ key. For subsequent requests to photos/, Amazon S3 returns photos/index.html. If the index document is not found, Amazon S3 returns an error.

Custom Error Document Support

The following table lists the subset of HTTP response codes that Amazon S3 returns when an error occurs.

HTTP Error Code	Description
301 Moved Permanently	When a user sends a request directly to the Amazon S3 website endpoints (<code>http://s3-website-<region>.amazonaws.com/</code>), Amazon S3 returns a 301 Moved Permanently response and redirects those requests to <code>http://aws.amazon.com/s3</code> .

HTTP Error Code	Description
302 Found	When Amazon S3 receives a request for a key <code>x</code> , <code>http://<bucket>.s3-website-<region>.amazonaws.com/x</code> , without a trailing slash, it first looks for the object with the keyname <code>x</code> . If the object is not found, Amazon S3 determines that the request is for subfolder <code>x</code> and redirects the request by adding a slash at the end, and returns 302 Found .
304 Not Modified	Amazon S3 users request headers <code>If-Modified-Since</code> , <code>If-Unmodified-Since</code> , <code>If-Match</code> and/or <code>If-None-Match</code> to determine whether the requested object is same as the cached copy held by the client. If the object is the same, the website endpoint returns a 304 Not Modified response.
400 Malformed Request	The website endpoint responds with a 400 Malformed Request when a user attempts to access a bucket through the incorrect regional endpoint.
403 Forbidden	The website endpoint responds with a 403 Forbidden when a user request translates to an object that is not publicly readable. The object owner must make the object publicly readable using a bucket policy or an ACL.
404 Not Found	<p>The website endpoint responds with 404 Not Found for the following reasons:</p> <ul style="list-style-type: none"> • Amazon S3 determines the website URL refers to an object key that does not exist • Amazon infers the request is for an index document that does not exist • A bucket specified in the URL does not exist • A bucket specified in the URL exists, however, it is not configured as a website <p>You can create a custom document that is returned for 404 Not Found. Make sure the document is uploaded to the bucket configured as a website, and that the website hosting configuration is set to use the document.</p> <p>For information on how Amazon S3 interprets the URL as a request for an object or an index document, see Index Document Support (p. 457).</p>
500 Service Error	The website endpoint responds with a 500 Service Error when an internal server error occurs.
503 Service Unavailable	The website endpoint responds with a 503 Service Unavailable when Amazon S3 determines that you need to reduce your request rate.

For each of these errors, Amazon S3 returns a predefined HTML as shown in the following sample HTML returned for **403 Forbidden** response.

403 Forbidden

- Code: AccessDenied
- Message: Access Denied
- RequestId: 873CA367A51F7EC7
- HostId: DdQezl9vkuw5huD5HKsFaTDm9KH4PZzCPRkW3igimILbTu1DiYlvXjgyd7pVxq32

An Error Occurred While Attempting to Retrieve a Custom Error Document

- Code: AccessDenied
- Message: Access Denied

You can optionally provide a custom error document with a user-friendly error message and with additional help. You provide this custom error document as part of adding website configuration to your bucket. Amazon S3 returns your custom error document for only the HTTP 4XX class of error codes.

Error Documents and Browser Behavior

When an error occurs, Amazon S3 returns an HTML error document. If you have configured your website with a custom error document, Amazon S3 returns that error document. However, note that when an error occurs, some browsers display their own error message, ignoring the error document Amazon S3 returns. For example, when an HTTP 404 Not Found error occurs, Chrome might display its own error ignoring the error document that Amazon S3 returns.

Configuring a Web Page Redirect

If your Amazon S3 bucket is configured for website hosting, you can redirect requests for an object to another object in the same bucket or to an external URL. You set the redirect by adding the `x-amz-website-redirect-location` property to the object metadata. The website then interprets the object as 301 redirect. To redirect a request to another object, you set the redirect location to the key of the target object. To redirect a request to an external URL, you set the redirect location to the URL that you want. For more information about object metadata, see [System-Defined Metadata \(p. 81\)](#).

A bucket configured for website hosting has both the website endpoint and the REST endpoint. A request for a page that is configured as a 301 redirect has the following possible outcomes, depending on the endpoint of the request:

- **Region-specific website endpoint** – Amazon S3 redirects the page request according to the value of the `x-amz-website-redirect-location` property.
- **REST endpoint** – Amazon S3 does not redirect the page request. It returns the requested object.

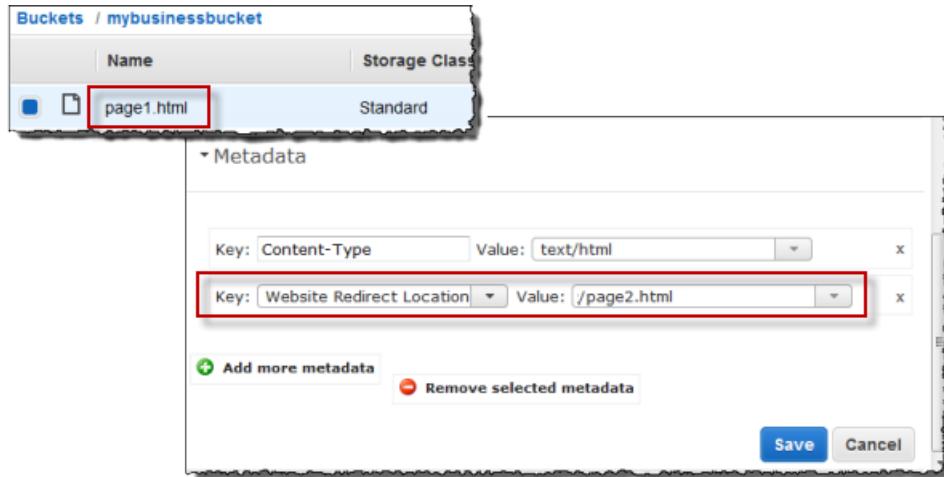
For more information about the endpoints, see [Key Differences Between the Amazon Website and the REST API Endpoint \(p. 450\)](#).

You can set a page redirect from the Amazon S3 console or by using the Amazon S3 REST API

Page Redirect Support in the Amazon S3 Console

You can use the Amazon S3 console to set the website redirect location in the metadata of the object. When you set a page redirect, you can either keep or delete the source object content. For example, suppose you have a `page1.html` object in your bucket. To redirect any requests for this page to another object, `page2.html`, you can do one of the following:

- To keep the content of the `page1.html` object and only redirect page requests, under **Properties** for `page1.html`, click the **Metadata** tab. Add Website Redirect Location to the metadata, as shown in the following example, and set its value to `/page2.html`. The `/` prefix in the value is required.



You can also set the value to an external URL, such as `http://www.example.com`.

- To delete the content of the `page1.html` object and redirect requests, you can upload a new zero-byte object with the same key, `page1.html`, to replace the existing object, and then specify Website Redirect Location for `page1.html` in the upload process. For information about uploading an object, go to [Uploading Objects into Amazon S3](#) in the *Amazon Simple Storage Service Console User Guide*.

Setting a Page Redirect from the REST API

The following Amazon S3 API actions support the `x-amz-website-redirect-location` header in the request. Amazon S3 stores the header value in the object metadata as `x-amz-website-redirect-location`.

- [PUT Object](#)
- [Initiate Multipart Upload](#)
- [POST Object](#)
- [PUT Object - Copy](#)

When setting a page redirect you can either keep or delete the object content. For example, suppose you have a `page1.html` object in your bucket.

- To keep the content of `page1.html` and only redirect page requests, you can submit a [PUT Object - Copy](#) request to create a new `page1.html` object that uses the existing `page1.html` object as the source. In your request, you set the `x-amz-website-redirect-location` header. When the request is complete, you have the original page with its content unchanged, but Amazon S3 redirects any requests for the page to the redirect location that you specify.
- To delete the content of the `page1.html` object and redirect requests for the page, you can send a PUT Object request to upload a zero-byte object that has the same object key, `page1.html`. In the PUT request, you set `x-amz-website-redirect-location` for `page1.html` to the new object. When the request is complete, `page1.html` has no content, and any requests will be redirected to the location that is specified by `x-amz-website-redirect-location`.

When you retrieve the object using the [GET Object](#) action, along with other object metadata, Amazon S3 returns the `x-amz-website-redirect-location` header in the response.

Permissions Required for Website Access

When you configure a bucket as a website, you must make the objects that you want to serve publicly readable. To do so, you write a bucket policy that grants everyone `s3:GetObject` permission. On the website endpoint, if a user requests an object that does not exist, Amazon S3 returns HTTP response code 404 (Not Found). If the object exists but you have not granted read permission on the object, the website endpoint returns HTTP response code 403 (Access Denied). The user can use the response code to infer if a specific object exists or not. If you do not want this behavior, you should not enable website support for your bucket.

The following sample bucket policy grants everyone access to the objects in the specified folder. For more information on bucket policies, see [Using Bucket Policies and User Policies \(p. 312\)](#).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "PublicReadGetObject",  
            "Effect": "Allow",  
            "Principal": "*",  
            "Action": ["s3:GetObject"],  
            "Resource": ["arn:aws:s3:::example-bucket/*"]  
        }  
    ]  
}
```

Note

The bucket policy applies only to objects owned by the bucket owner. If your bucket contains objects not owned by the bucket owner, then public READ permission on those objects should be granted using the object ACL.

You can grant public read permission to your objects by using either a bucket policy or an object ACL. To make an object publicly readable using an ACL, you grant READ permission to the AllUsers group as shown in the following grant element. You add this grant element to the object ACL. For information on managing ACLs, see [Managing Access with ACLs \(p. 363\)](#).

```
<Grant>  
    <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
             xsi:type="Group">  
        <URI>http://acs.amazonaws.com/groups/global/AllUsers</URI>  
    </Grantee>  
    <Permission>READ</Permission>  
</Grant>
```

Example Walkthroughs - Hosting Websites On Amazon S3

Topics

- [Example: Setting Up a Static Website \(p. 463\)](#)

- Example: Setting Up a Static Website Using a Custom Domain (p. 464)

This section provides two examples. In the first example, you configure a bucket for website hosting, upload a sample index document, and test the website using the Amazon S3 website endpoint for the bucket. The second example shows how you can use your own domain such as example.com, instead of the Amazon S3 bucket website endpoint, and serve content from an Amazon S3 bucket configured as a website. The example also shows how Amazon S3 offers the root domain support.

Example: Setting Up a Static Website

You can configure an Amazon S3 bucket to function like a website. This example walks you through the steps of hosting a website on Amazon S3. In the following procedure, you will use the AWS Management Console to perform the necessary tasks:

1. Create an Amazon S3 bucket and configure it as a website (see [To create a bucket and configure it as a website \(p. 463\)](#)).
2. Add a bucket policy that make the bucket content public (see [To add a bucket policy that makes your bucket content publicly available \(p. 463\)](#)).

The content that you serve at the website endpoint must be publicly readable. You can grant the necessary permissions by adding a bucket policy or using Access Control List (ACL). Here we describe adding a bucket policy.

3. Upload an index document (see [To upload an index document \(p. 464\)](#)).
4. Test your website using the Amazon S3 bucket website endpoint ([Test your website \(p. 464\)](#)).

To create a bucket and configure it as a website

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Create a bucket.

For step-by-step instructions, go to [Create a bucket](#) in *Amazon Simple Storage Service Console User Guide*.

For bucket naming guidelines, see [Bucket Restrictions and Limitations \(p. 59\)](#). If you have your registered domain name, for additional information about bucket naming, see [Customizing Amazon S3 URLs with CNAMEs \(p. 51\)](#).

3. Open the bucket **Properties** panel, click **Static Website Hosting**, and do the following:
 1. Select the **Enable website hosting**.
 2. In the **Index Document** box, add the name of your index document. This name is typically `index.html`.
 3. Click **Save** to save the website configuration.
 4. Note down the **Endpoint**.

This is the Amazon S3-provided website endpoint for your bucket. You will use this endpoint in the following steps to test your website.

To add a bucket policy that makes your bucket content publicly available

1. In bucket **Properties** panel, click the **Permissions**.
2. Click **Add Bucket Policy**.

3. Copy the following bucket policy, and then paste it in the Bucket Policy Editor.

```
{  
    "Version": "2012-10-17",  
    "Statement": [ {  
        "Sid": "PublicReadForGetBucketObjects",  
        "Effect": "Allow",  
        "Principal": "*",  
        "Action": [ "s3:GetObject" ],  
        "Resource": [ "arn:aws:s3:::example-bucket/*"  
        ]  
    }  
}
```

4. In the policy, replace *example-bucket* with the name of your bucket.
5. Click **Save**.

To upload an index document

1. Create a document. The file name must be same as the name that you provided for the index document earlier.
2. Using the console, upload the index document to your bucket.
For instructions, go to [Uploading Objects into Amazon S3](#) in the *Amazon Simple Storage Service Console User Guide*.

Test your website

- Enter the following URL in the browser, replacing *example-bucket* with the name of your bucket and *website-region* with the name of the region where you deployed your bucket. For information about region names, see [Website Endpoints \(p. 449\)](#).

```
http://example-bucket.s3-website-region.amazonaws.com
```

If your browser displays your `index.html` page, the website was successfully deployed.

Note

HTTPS access to the website is not supported.

You now have a website hosted on Amazon S3. This website is available at the Amazon S3 website endpoint. However, you might have a domain such as `example.com` that you want to use to serve the content from the website you created. You might also want to use Amazon S3's root domain support to serve requests for both the `http://www.example.com` and `http://example.com`. This requires additional steps. For an example, see [Example: Setting Up a Static Website Using a Custom Domain \(p. 464\)](#).

Example: Setting Up a Static Website Using a Custom Domain

Topics

- [Before You Begin \(p. 465\)](#)

- [Step 1: Register a Domain \(p. 465\)](#)
- [Step 2: Create and Configure Buckets and Upload Data \(p. 466\)](#)
- [Step 3: Create and Configure Amazon Route 53 Hosted Zone \(p. 469\)](#)
- [Step 4: Switch to Amazon Route 53 as Your DNS Provider \(p. 470\)](#)
- [Step 5: Testing \(p. 471\)](#)

Suppose you want to host your static website on Amazon S3. You have registered a domain, for example, `example.com`, and you want requests for `http://www.example.com` and `http://example.com` to be served from your Amazon S3 content.

Whether you have an existing static website that you now want to host on Amazon S3 or you are starting from scratch, this example will help you host websites on Amazon S3.

Before You Begin

As you walk through the steps in this example, note that you will work with the following services:

Domain registrar of your choice— If you do not already have a registered domain name, such as `example.com`, you will need to create and register one with a registrar of your choice. You can typically register a domain for a small yearly fee. For procedural information about registering a domain name, see the web site of the registrar

Amazon S3— You will use Amazon S3 to create buckets, upload a sample website page, configure permissions so everyone can see the content, and then configure the buckets for website hosting. In this example, because you want to allow requests for both `http://www.example.com` and `http://example.com`, you will create two buckets; however, you will host content in only one bucket. You will configure the other Amazon S3 bucket to redirect requests to the bucket that hosts the content.

Amazon Route 53— You will configure Amazon Route 53 as your DNS provider. You will create a hosted zone in Amazon Route 53 for your domain and configure applicable DNS records. If you are switching from an existing DNS provider, you will need to ensure that you have transferred all of the DNS records for your domain.

As you walk through this example, a basic familiarity with domains, Domain Name System (DNS), CNAME records, and A records would be helpful. A detailed explanation of these concepts is beyond the scope of this guide, but your domain registrar should provide any basic information that you need.

Note

All the steps in this example use `example.com` as a domain name. You will need to replace this domain name with the one you registered.

Step 1: Register a Domain

If you already have a registered domain, you can skip this step. If you are new to hosting a website, your first step is to register a domain, such as `example.com`, with a registrar of your choice.

After you have chosen a registrar, you will register your domain name according to the instructions at the registrar's website. For a list of registrar web sites that you can use to register your domain name, go to [ICANN.org](#).

When you have a registered domain name, your next task is to create and configure Amazon S3 buckets for website hosting and to upload your website content.

Step 2: Create and Configure Buckets and Upload Data

In this example, to support requests from both the root domain such as `example.com` and subdomain such as `www.example.com`, you will create two buckets. One bucket will contain the content and you will configure the other bucket to redirect requests. You perform the following tasks in Amazon S3 console to create and configure your website:

1. Create two buckets.
2. Configure these buckets for website hosting.
3. Test the Amazon S3 provided bucket website endpoint.

Step 2.1: Create Two Buckets

The bucket names must match the names of the website that you are hosting. For example, to host your `example.com` website on Amazon S3, you would create a bucket named `example.com`. To host a website under `www.example.com`, you would name the bucket `www.example.com`. In this example, your website will support requests from both `example.com` and `www.example.com`.

In this step, you will sign in to the Amazon S3 console with your AWS account credentials and create the following two buckets.

- `example.com`
- `www.example.com`

Note

To create the buckets for this example, follow these steps. As you walk through this example, substitute the domain name that you registered for `example.com`.

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Create two buckets that match your domain name. One of the bucket names should include the subdomain www. For instance, `example.com` and `www.example.com`.
For step-by-step instructions, go to [Creating a Bucket](#) in the *Amazon Simple Storage Service Console User Guide*.
3. Upload your website data to the `example.com` bucket.

You will host your content out of the root domain bucket (`example.com`), and you will redirect requests for `www.example.com` to the root domain bucket. Note that you can store content in either bucket. For this example you will host content in `example.com` bucket. The content can be text files, family photos, videos—whatever you want. If you have not yet created a website, then you only need one file for this example. You can upload any file. For example, you can create a file using the following HTML and upload it to the bucket. The file name of the home page of a website is typically `index.html`, but you can give it any name. In a later step, you will provide this file name as the index document name for your website.

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
    <title>My Website Home Page</title>
</head>
<body>
    <h1>Welcome to my website</h1>
    <p>Now hosted on Amazon S3!</p>
</body>
</html>
```

For step-by-step instructions, go to [Uploading Objects into Amazon S3](#) in the *Amazon Simple Storage Service Console User Guide*.

4. Configure permissions for your objects to make them publicly accessible.

Attach the following bucket policy to the `example.com` bucket substituting the name of your bucket for `example.com`. For step-by-step instructions to attach a bucket policy, go to [Editing Bucket Permissions](#) in the *Amazon Simple Storage Service Console User Guide*.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AddPerm",  
            "Effect": "Allow",  
            "Principal": "*",  
            "Action": [ "s3:GetObject" ],  
            "Resource": [ "arn:aws:s3:::example.com/*"  
            ]  
        }  
    ]  
}
```

You now have two buckets, `example.com` and `www.example.com`, and you have uploaded your website content to the `example.com` bucket. In the next step, you will configure `www.example.com` to redirect requests to your `example.com` bucket. By redirecting requests you can maintain only one copy of your website content and both visitors who specify “www” in their browsers and visitors that only specify the root domain will both be routed to the same website content in your `example.com` bucket.

Step 2.2: Configure Buckets for Website Hosting

When you configure a bucket for website hosting, you can access the website using the Amazon S3 assigned bucket website endpoint.

In this step, you will configure both buckets for website hosting. First, you will configure `example.com` as a website and then you'll configure `www.example.com` to redirect all requests to the `example.com` bucket.

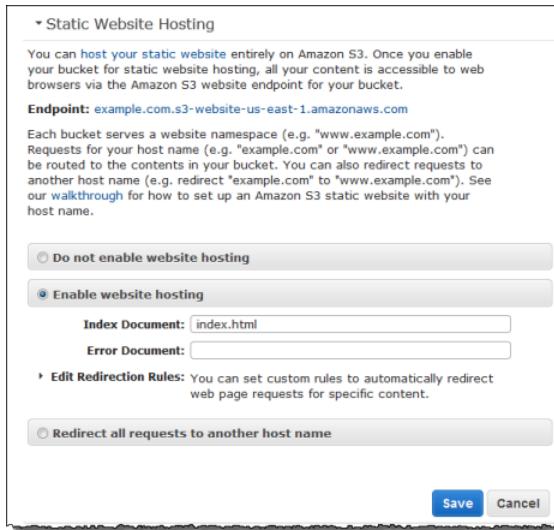
To configure `example.com` bucket for website hosting

1. Configure `example.com` bucket for website hosting. In the **Index Document** box, type the name that you gave your index page.

For step-by-step-instructions, go to [Managing Bucket Website Configuration](#) in the *Amazon Simple Storage Service Console User Guide*. Make a note of the URL for the website endpoint. You will need it later.

Amazon Simple Storage Service Developer Guide

Example: Setting Up a Static Website Using a Custom Domain

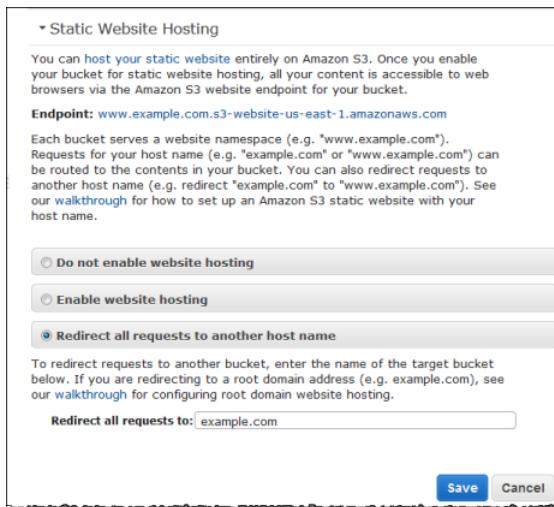


2. To test the website, enter the **Endpoint URL** in your browser.

Your browser will display the index document page. Next, you will configure `www.example.com` bucket to redirect all requests for `www.example.com` to `example.com`.

To redirect requests from `www.example.com` to `example.com`

1. In the Amazon S3 console, in the Buckets list, right-click `www.example.com` and then click **Properties**.
2. Under **Static Website Hosting**, click **Redirect all requests to another host name**. In the **Redirect all requests** box, type `example.com`.



3. To test the website, enter the **Endpoint URL** in your browser.

Your request will be redirected and the browser will display the index document for `example.com`.

The following Amazon S3 bucket website endpoints are accessible to any internet user:

`example.com.s3-website-us-east-1.amazonaws.com`

`http://www.example.com.s3-website-us-east-1.amazonaws.com`

Now you will do additional configuration to serve requests from the domain you registered in the preceding step. For example, if you registered a domain `example.com`, you want to serve requests from the following URLs :

`http://example.com`

`http://www.example.com`

In the next step, we will use Amazon Route 53 to enable customers to use the URLs above to navigate to your site.

Step 3: Create and Configure Amazon Route 53 Hosted Zone

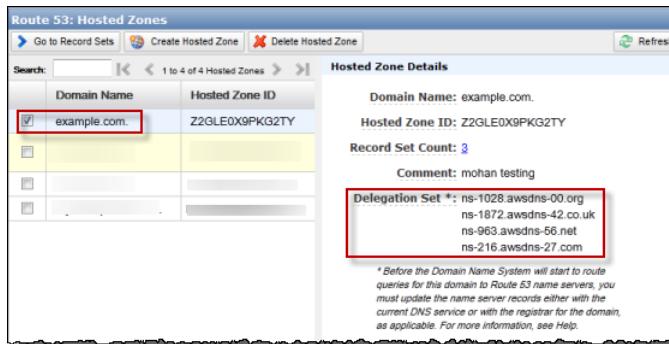
Now you will configure Amazon Route 53 as your Domain Name System (DNS) provider. You must use Amazon Route 53 if you want to serve content from your root domain, such as `example.com`. You will create a hosted zone, which holds the DNS records associated with your domain:

- An alias record that maps the domain `example.com` to the `example.com` bucket. This is the bucket that you configured as a website endpoint in step 2.2.
- Another alias record that maps the subdomain `www.example.com` to the `www.example.com` bucket. You configured this bucket to redirect requests to the `example.com` bucket in step 2.2.

Step 3.1: Create a Hosted Zone for Your Domain

Go to the Amazon Route 53 console at <https://console.aws.amazon.com/route53> and then create a hosted zone for your domain. For instructions, go to [Creating a Hosted Zone](#) in the <http://docs.aws.amazon.com/Route53/latest/DeveloperGuide/>.

The following example shows the hosted zone created for the `example.com` domain. Write down the Amazon Route 53 name servers (NS) for this domain. You will need them later.

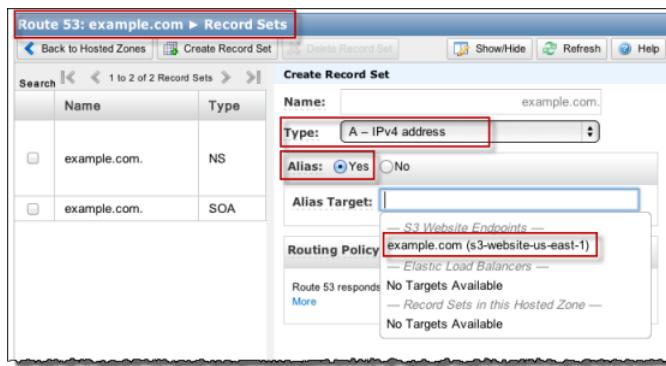


Step 3.2: Add Alias Records for example.com and www.example.com

The alias records that you add to the hosted zone for your domain will map `example.com` and `www.example.com` to the corresponding Amazon S3 buckets. Instead of using IP addresses, the alias records use the Amazon S3 website endpoints. Amazon Route 53 maintains a mapping between the alias records and the IP addresses where the Amazon S3 buckets reside.

For step-by-step instructions, go to [Creating an Alias Resource Record Set](#) in the [Amazon Route 53 Developer Guide](#).

The following screenshot shows the alias record for `example.com` as an illustration. You will also need to create an alias record for `www.example.com`.



To enable this hosted zone, you must use Amazon Route 53 as the DNS server for your domain [example.com](#). Before you switch, if you are moving an existing website to Amazon S3, you must transfer DNS records associated with your domain [example.com](#) to the hosted zone that you created in Amazon Route 53 for your domain. If you are creating a new website, you can go directly to step 4.

Note

Creating, changing, and deleting resource record sets take time to propagate to the Route 53 DNS servers. Changes generally propagate to all Route 53 name servers in a couple of minutes. In rare circumstances, propagation can take up to 30 minutes.

Step 3.3: Transfer Other DNS Records from Your Current DNS Provider to Amazon Route 53

Before you switch to Amazon Route 53 as your DNS provider, you must transfer any remaining DNS records from your current DNS provider, including MX records, CNAME records, and A records, to Amazon Route 53. You don't need to transfer the following records:

- NS records— Instead of transferring these, you replace their values with the name server values that are provided by Amazon Route 53.
- SOA record— Amazon Route 53 provides this record in the hosted zone with a default value.

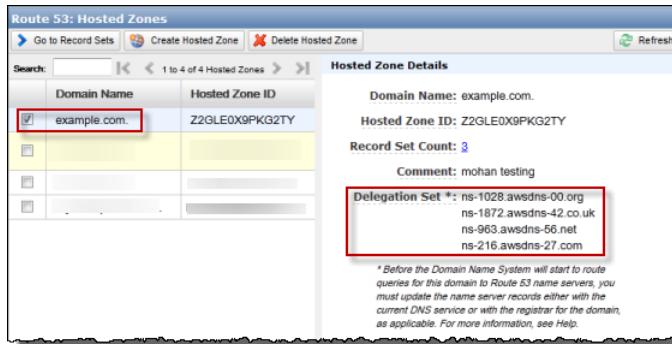
Migrating required DNS records is a critical step to ensure the continued availability of all the existing services hosted under the domain name.

Step 4: Switch to Amazon Route 53 as Your DNS Provider

To switch to Amazon Route 53 as your DNS provider, you must go to your current DNS provider and update the name server (NS) record to use the name servers in your delegation set in Amazon Route 53.

Go to your DNS provider site and update the NS record with the delegation set values of the hosted zone as shown in the following Amazon Route 53 console screenshot. For more information, go to [Updating Your DNS Service's Name Server Records in Amazon Route 53 Developer Guide](#).

Amazon Simple Storage Service Developer Guide
Example: Setting Up a Static Website Using a Custom Domain



When the transfer to Amazon Route 53 is complete, there are tools that you can use to verify the name server for your domain has indeed changed. On a Linux computer, you can use the `dig` DNS lookup utility. For example, this `dig` command:

```
dig +recurse +trace www.example.com any
```

returns the following output (only partial output is shown). The output shows the same four name servers the name servers on Amazon Route 53 hosted zone you created for `example.com` domain.

```
...
example.com.    172800   IN      NS      ns-9999.awsdns-99.com.
example.com.    172800   IN      NS      ns-9999.awsdns-99.org.
example.com.    172800   IN      NS      ns-9999.awsdns-99.co.uk.
example.com.    172800   IN      NS      ns-9999.awsdns-99.net.

www.example.com. 300      IN      CNAME   www.example.com.s3-website-us-east-
1.amazonaws.com.
...
```

Step 5: Testing

To verify that the website is working correctly, in your browser, try the following URLs:

- `http://example.com` - Displays the index document in the `example.com` bucket.
- `http://www.example.com` - Redirects your request to `http://example.com`.

In some cases, you may need to clear the cache to see the expected behavior.

Configuring Amazon S3 Event Notifications

The Amazon S3 notification feature enables you to receive notifications when certain events happen in your bucket. To enable notifications, you must first add a notification configuration identifying the events you want Amazon S3 to publish, and the destinations where you want Amazon S3 to send the event notifications. You store this configuration in the *notification* subresource (see [Bucket Configuration Options \(p. 57\)](#)) associated with a bucket. Amazon S3 provides an API for you to manage this subresource.

Currently, Amazon S3 can publish the following events:

- A new object created event—Amazon S3 supports multiple APIs to create objects. You can request notification when only a specific API is used (e.g., `s3:ObjectCreated:Put`) or you can use a wildcard (e.g., `s3:ObjectCreated:*`) to request notification when an object is created regardless of the API used.
- A Reduced Redundancy Storage (RRS) object lost event—Amazon S3 sends a notification message when it detects that an object of the RRS storage class has been lost.

For a list of supported event types, see [Supported Event Types \(p. 475\)](#).

Amazon S3 supports the following destinations where it can publish events:

- Amazon Simple Notification Service (Amazon SNS) topic

Amazon SNS is a flexible, fully managed push messaging service. Using this service, you can push messages to mobile devices or distributed services. With SNS you can publish a message once, and deliver it one or more times. An SNS topic is an "access point" for allowing recipients to dynamically subscribe to for event notification. For more information about SNS, go to the [Amazon SNS product detail page](#).

- Amazon Simple Queue Service (Amazon SQS) queue

Amazon SQS is a scalable and fully managed message queuing service. You can use SQS to transmit any volume of data without requiring other services to be always available. In your notification configuration you can request that Amazon S3 publish events to an SQS queue. For more information about SQS, go to [Amazon SQS product detail page](#).

- AWS Lambda

AWS Lambda is a compute service that makes it easy for you to build applications that respond quickly to new information. AWS Lambda runs your code in response to events such as image uploads, in-app activity, website clicks, or outputs from connected devices. You can use AWS Lambda to extend other AWS services with custom logic, or create your own back-end that operates at AWS scale, performance, and security. With AWS Lambda, you can easily create discrete, event-driven applications that execute only when needed and scale automatically from a few requests per day to thousands per second.

AWS Lambda can run custom code in response to Amazon S3 bucket events. You upload your custom code to AWS Lambda and create what is called a Lambda function. When Amazon S3 detects an event of a specific type (for example, an object created event), it can publish the event to AWS Lambda and invoke your function in Lambda. In response, AWS Lambda executes your function. For more information, go to [AWS Lambda product detail page](#).

The following sections offer more detail about how to enable event notifications on a bucket. The subtopics also provide example walkthroughs to help you explore the notification feature. For more information, see:

- [Example Walkthrough 1: Configure a Bucket for Notifications \(Message Destination: SNS Topic and SQS Queue\) \(p. 478\)](#)
- [Example Walkthrough 2: Configure a Bucket for Notifications \(Message Destination: AWS Lambda\) \(p. 484\)](#)

How to Enable Event Notifications

Enabling notifications is a bucket-level operation; that is, you store notification configuration information in the *notification* subresource associated with a bucket. You can use any of the following methods to manage notification configuration:

- Using the Amazon S3 console

The console UI enables you to set a notification configuration on a bucket without having to write any code. For instruction, go to [Enabling Event Notifications](#) in the *Amazon Simple Storage Service Console User Guide*.

- Programmatically using the AWS SDKs

Note

If you need to, you can also make the Amazon S3 REST API calls directly from your code. However, this can be cumbersome because it requires you to write code to authenticate your requests.

Internally, both the console and the SDKs call the Amazon S3 REST API to manage *notification* subresources associated with the bucket. For notification configuration using AWS SDK examples, see the walkthrough link provided in the preceding section.

Regardless of the method you use, Amazon S3 stores the notification configuration as XML in the *notification* subresource associated with a bucket. For information about bucket subresources, see [Bucket Configuration Options \(p. 57\)](#)). By default, notifications are not enabled for any type of event. Therefore, initially the *notification* subresource stores an empty configuration.

```
<NotificationConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
</NotificationConfiguration>
```

To enable notifications for events of specific types, you replace the XML with the appropriate configuration identifying the event types you want Amazon S3 to publish and the destination where you want the events published. For each destination, you add a corresponding XML configuration. For example:

- Publish event messages to an SQS queue—To set an SQS queue as the notification destination for one or more event types, you add the `QueueConfiguration`.

```
<NotificationConfiguration>
  <QueueConfiguration>
    <Id>optional-id-string</Id>
    <Queue>sqs-queue-arn</Queue>
    <Event>event-type</Event>
    <Event>event-type</Event>
    ...
  </QueueConfiguration>
  ...
</NotificationConfiguration>
```

- Publish event messages to an SNS topic—To set an SNS topic as the notification destination for specific event types, you add the `TopicConfiguration`.

```
<NotificationConfiguration>
  <TopicConfiguration>
    <Id>optional-id-string</Id>
    <Topic>sns-topic-arn</Topic>
    <Event>event-type</Event>
    <Event>event-type</Event>
    ...
  </TopicConfiguration>
  ...
</NotificationConfiguration>
```

- Invoke AWS Lambda function and provide event message as an argument—To set a Lambda function as the notification destination for specific event types, you add the `CloudFunctionConfiguration`.

```
<NotificationConfiguration>
  <CloudFunctionConfiguration>
    <Id>optional-id-string</Id>
    <CloudFunction>cloud-function-arn</CloudFunction>
    <Event>event-type</Event>
    <Event>event-type</Event>
    ...
  </CloudFunctionConfiguration>
  ...
</NotificationConfiguration>
```

Your configuration can include multiple destinations, but an event type may appear in at most one destination. For example, the following notification configuration includes all destinations, each requesting Amazon S3 to publish events of specific types to those destinations.

```
<NotificationConfiguration>
  <QueueConfiguration>
    <Queue>sqs-queue-arn</Queue>
    <Event>event-type1</Event>
    <Event>event-type2</Event>
    ...
  </QueueConfiguration>
  ...
</NotificationConfiguration>
```

```

<TopicConfiguration>
  <Topic>sns-topic-arn</Topic>
  <Event>event-type3</Event>
  ...
</TopicConfiguration>
<CloudFunctionConfiguration>
  <CloudFunction>cloud-function-arn</CloudFunction>
  <Event>event-type4</Event>
</CloudFunctionConfiguration>
</NotificationConfiguration>

```

One of the event types (`s3:ObjectCreated:*`), explained in the following section, uses a wildcard, which counts as all the object created events, and therefore it precludes inclusion of any other event types.

To remove all notifications configured on a bucket, you save an empty `<NotificationConfiguration/>` element in the *notification* subresource.

When Amazon S3 detects an event of the specific type, it publishes a message with the event information (see [Event Message Structure \(p. 484\)](#)).

Supported Event Types

Amazon S3 can publish events of the following types. You specify these event types in the notification configuration.

Event types	Description
<code>s3:ObjectCreated:*</code>	Amazon S3 APIs such as PUT, POST, and COPY can create an object. Using these event types, you can enable notification when an object is created using a specific API, or you can use the <code>s3:ObjectCreated:*</code> event type to request notification regardless of the API that was used to create an object.
<code>s3:ObjectCreated:Put</code>	
<code>s3:ObjectCreated:Post</code>	
<code>s3:ObjectCreated:Copy</code>	
<code>s3:ObjectCreated:CompleteMultipartUpload</code>	
<code>s3:ReducedRedundancyLostObject</code>	You can use this event type to request Amazon S3 to send a notification message when Amazon S3 detects that an object of the RRS storage class is lost.

Supported Destinations

Amazon S3 can send event notification messages to the following destinations. You specify the ARN value of these destinations in the notification configuration.

- Publish event messages to an Amazon Simple Notification Service (Amazon SNS) topic
- Publish event messages to an Amazon Simple Queue Service (Amazon SQS) queue
- Publish event messages to AWS Lambda by invoking a Lambda function and provide the event message as an argument

You must grant Amazon S3 permissions to post messages to an SNS topic or an SQS queue. You must also grant Amazon S3 permission to invoke an AWS Lambda function on your behalf.

Granting Permissions to Publish Event Notification Messages to a Destination

Before Amazon S3 can publish messages to a destination, you must grant the Amazon S3 principal the necessary permissions to call the relevant API to publish messages to an SNS topic, an SQS queue, or a Lambda function.

Granting Permissions to Invoke an AWS Lambda Function

Amazon S3 publishes event messages to AWS Lambda by invoking a Lambda function and providing the event message as an argument.

When you use the Amazon S3 console to configure event notifications on an Amazon S3 bucket for a Lambda function, the Amazon S3 console will set up the necessary permissions on the Lambda function so that Amazon S3 has permissions to invoke the function from the bucket. For more information, see [Enabling Event Notifications](#) in the *Amazon Simple Storage Service Console User Guide*.

You can also grant Amazon S3 permissions from AWS Lambda to invoke your Lambda function. For more information, see [Configure Amazon S3 to Publish Events](#), which is part of the [AWS Lambda Walkthrough 2: Handling Amazon S3 Events \(Using the AWS CLI\)](#) in the *AWS Lambda Developer Guide*.

Granting Permissions to Publish Messages to an SNS Topic or an SQS Queue

You attach an IAM policy to the destination SNS topic or SQS queue to grant Amazon S3 permissions to publish messages to the SNS topic or SQS queue.

Example of an IAM policy that you attach to the destination SNS topic.

```
{  
    "Version": "2008-10-17",  
    "Id": "example-ID",  
    "Statement": [  
        {  
            "Sid": "example-statement-ID",  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "s3.amazonaws.com"  
            },  
            "Action": [  
                "SNS:Publish"  
            ],  
            "Resource": "SNS-ARN",  
            "Condition": {  
                "ArnLike": {  
                    "aws:SourceArn": "arn:aws:s3:::*::bucket-name"  
                }  
            }  
        }  
    ]  
}
```

```
        }
    ]
}
```

Example of an IAM policy that you attach to the destination SQS queue.

```
{
  "Version": "2008-10-17",
  "Id": "example-ID",
  "Statement": [
    {
      "Sid": "example-statement-ID",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": [
        "SQS:SendMessage"
      ],
      "Resource": "SQS-ARN",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:s3::*:*:bucket-name"
        }
      }
    ]
}
```

Note that for both the Amazon SNS and Amazon SQS IAM policies, you can specify the `StringLike` condition in the policy, instead of the `ArnLike` condition.

```
"Condition": {
  "StringLike": {
    "aws:SourceArn": "arn:aws:s3::*:*:bucket-name"
  }
}
```

For an example of how to attach a policy to a SNS topic or an SQS queue, see [Example Walkthrough 1: Configure a Bucket for Notifications \(Message Destination: SNS Topic and SQS Queue\) \(p. 478\)](#).

For more information about permissions, see the following topics:

- [Example Cases for Amazon SNS Access Control](#) in the *Amazon Simple Notification Service Developer Guide*
- [Access Control Using AWS Identity and Access Management \(IAM\)](#) in the *Amazon Simple Queue Service Developer Guide*

Related Topics

- [Example Walkthrough 1: Configure a Bucket for Notifications \(Message Destination: SNS Topic and SQS Queue\) \(p. 478\)](#)

- Example Walkthrough 2: Configure a Bucket for Notifications (Message Destination: AWS Lambda) (p. 484)
- Event Message Structure (p. 484)

Example Walkthrough 1: Configure a Bucket for Notifications (Message Destination: SNS Topic and SQS Queue)

Topics

- [Walkthrough Summary \(p. 478\)](#)
- [Step 1: Create an Amazon SNS Topic \(p. 479\)](#)
- [Step 2: Create an Amazon SQS Queue \(p. 479\)](#)
- [Step 3: Add a Notification Configuration to Your Bucket \(p. 480\)](#)
- [Step 4: Test the Setup \(p. 484\)](#)

Walkthrough Summary

In this walkthrough you add notification configuration on a bucket requesting Amazon S3 to:

- Publish events of the `s3:ObjectCreated:*` type to an Amazon SQS topic.
- Publish events of the `s3:ReducedRedundancyLostObject` type to an Amazon SNS topic.

For information about notification configuration, see [Configuring Amazon S3 Event Notifications \(p. 472\)](#).

You can do all these steps using the console, without writing any code. In addition, code examples, using AWS SDKs for Java and .NET are also provided so you can add notification configuration programmatically.

You will do the following in this walkthrough:

1. Create an Amazon SNS topic.

Using the Amazon SNS console, you create an SNS topic and subscribe to the topic so that any events posted to it are delivered to you. You will specify email as the communications protocol. After you create a topic, Amazon SNS will send an email. You must click a link in the email to confirm the topic subscription.

You will attach an access policy to the topic to grant Amazon S3 permission to post messages.

2. Create an Amazon SQS queue.

Using the Amazon SQS console, you create an SQS queue. You can access any messages Amazon S3 sends to the queue programmatically. But for this walkthrough, you will verify notification messages in the console.

You will attach an access policy to the topic to grant Amazon S3 permission to post messages.

3. Add notification configuration to a bucket.

Step 1: Create an Amazon SNS Topic

Follow the steps to create and subscribe to an Amazon Simple Notification Service (Amazon SNS) topic.

1. Using Amazon SNS console create a topic. For instructions, go to [Create a Topic](#) in the *Amazon Simple Notification Service Developer Guide*.
2. Subscribe to the topic. For this exercise, use email as the communications protocol. For instructions, go to [Subscribe to a Topic](#) in the *Amazon Simple Notification Service Developer Guide*.
You will get email requesting you to confirm your subscription to the topic. Confirm the subscription.
3. Replace the access policy attached to the topic by the following policy. You must update the policy by providing the your SNS topic ARN and bucket name.:

```
{  
    "Version": "2008-10-17",  
    "Id": "example-ID",  
    "Statement": [  
        {  
            "Sid": "example-statement-ID",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "*"  
            },  
            "Action": [  
                "SNS:Publish"  
            ],  
            "Resource": "SNS-topic-ARN",  
            "Condition": {  
                "ArnLike": {  
                    "aws:SourceArn": "arn:aws:s3:::bucket-name"  
                }  
            }  
        }  
    ]  
}
```

4. Note the topic ARN.

The SNS topic you created is another resource in your AWS account, and it has a unique Amazon Resource Name (ARN). You will need this ARN in the next step. The ARN will be of the following format:

```
arn:aws:sns:aws-region:account-id:topic-name
```

Step 2: Create an Amazon SQS Queue

Follow the steps to create and subscribe an Amazon Simple Queue Service (Amazon SQS) topic.

1. Using the Amazon SQS console, create a topic. For instructions, go to [Create a Queue](#) in the *Amazon Simple Queue Service Getting Started Guide*.
2. Replace the access policy attached to the queue by the following policy (in the SQS console, you select the queue, and in the **Permissions** tab, click **Edit Policy Document (Advanced)**).

```
{  
    "Version": "2008-10-17",  
    "Id": "example-ID",  
    "Statement": [  
        {  
            "Sid": "example-statement-ID",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "*"  
            },  
            "Action": [  
                "SQS:SendMessage"  
            ],  
            "Resource": "SQS-topic-ARN",  
            "Condition": {  
                "ArnLike": {  
                    "aws:SourceArn": "arn:aws:s3:*:*:bucket-name"  
                }  
            }  
        }  
    ]  
}
```

3. Note the queue ARN

The SQS queue you created is another resource in your AWS account, and it has a unique Amazon Resource Name (ARN). You will need this ARN in the next step. The ARN will be of the following format:

```
arn:aws:sns:aws-region:account-id:topic-name
```

Step 3: Add a Notification Configuration to Your Bucket

You can enable bucket notifications either by using the Amazon S3 console or programmatically by using AWS SDKs. Choose any one of the options to configure notifications on your bucket. This section provides code examples using the AWS SDKs for Java and .NET.

Step 3 (option a): Enable Notifications on a Bucket Using the Console

Using the Amazon S3 console, add a notification configuration requesting Amazon S3 to:

- Publish events of the s3:ObjectCreated:* type to your Amazon SQS queue.
- Publish events of the s3:ReducedRedundancyLostObject type to your Amazon SNS topic.

After you save the notification configuration, Amazon S3 will post a test message, which you will get via email.

For instructions, go to [Enabling Notifications in the Amazon Simple Storage Service Console User Guide](#).

Step 3 (option b): Enable Notifications on a Bucket Using the AWS SDK for .NET

The following C# code example provides a complete code listing that adds a notification configuration to a bucket. You will need to update the code and provide your bucket name and SNS topic ARN. For information about how to create and test a working sample, see [Testing the .NET Code Examples \(p. 547\)](#).

```
using System;
using System.Collections.Generic;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazonaws.com.docsamples
{
    class EnableNotifications
    {
        static string bucketName = "****bucket name****";
        static string snsTopic = "****SNS topic ARN****";
        static string sqsQueue = "****SQS queue ARN****";

        static string putEventType = "s3:ObjectCreated:Put";
        static string rrsObjectLostType = "s3:ObjectCreated:Copy";

        public static void Main(string[] args)
        {
            using (var client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1))
            {
                Console.WriteLine("Enabling Notification on a bucket");
                EnableNotification(client);
            }

            Console.WriteLine("Press any key to continue... ");
            Console.ReadKey();
        }

        static void EnableNotification(IAmazonS3 client)
        {
            try
            {
                List<Amazon.S3.Model.TopicConfiguration> topicConfigurations =
new List<TopicConfiguration>();

                topicConfigurations.Add(new TopicConfiguration()
                {
                    Event = rrsObjectLostType,
                    Topic = snsTopic
                });

                List<Amazon.S3.Model.QueueConfiguration> queueConfigurations =
new List<QueueConfiguration>();
                queueConfigurations.Add(new QueueConfiguration()
                {
                    Events = new List<string> { putEventType },
                    Queue = sqsQueue
                });
            }
        }
    }
}
```

```
        PutBucketNotificationRequest request = new PutBucketNotifica
tionRequest
    {
        BucketName = bucketName,
        TopicConfigurations = topicConfigurations,
        QueueConfigurations = queueConfigurations
    };

    PutBucketNotificationResponse response = client.PutBucketNoti
fication(request);
}
catch (AmazonS3Exception amazonS3Exception)
{
    if (amazonS3Exception.ErrorCode != null &&
        (amazonS3Exception.ErrorCode.Equals("InvalidAccessKeyId")
        ||
        amazonS3Exception.ErrorCode.Equals("InvalidSecurity")))
    {
        Console.WriteLine("Check the provided AWS Credentials.");
        Console.WriteLine(
            "To sign up for service, go to http://aws.amazon.com/s3");
    }
    else
    {
        Console.WriteLine(
            "Error occurred. Message:'{0}' when enabling notifica
tions.",
            amazonS3Exception.Message);
    }
}
}
}
```

Step 3 (option c): Enable Notifications on a Bucket Using the AWS SDK for Java

The following Java code example provides a complete code listing that adds a notification configuration to a bucket. You will need to update the code and provide your bucket name and SNS topic ARN. For instructions on how to create and test a working sample, see [Testing the Java Code Examples \(p. 545\)](#).

```
import java.io.IOException;
import java.util.Collection;
import java.util.EnumSet;
import java.util.LinkedList;

import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.AmazonS3Exception;
import com.amazonaws.services.s3.model.BucketNotificationConfiguration;
import com.amazonaws.services.s3.model.TopicConfiguration;
import com.amazonaws.services.s3.model.QueueConfiguration;
```

```
import com.amazonaws.services.s3.model.S3Event;
import com.amazonaws.services.s3.model.SetBucketNotificationConfigurationRequest;

public class NotificationConfigurationOnABucket {
    private static String bucketName = "**** bucket name ****";
    private static String snsTopicARN = "**** SNS Topic ARN ****";
    private static String sqsQueueARN = "**** SQS Queue ARN ****";

    public static void main(String[] args) throws IOException {
        AmazonS3 s3client = new AmazonS3Client(new ProfileCredentialsProvider());

        try {
            System.out.println("Setting notification configuration on a bucket.\n");

            BucketNotificationConfiguration notificationConfiguration = new
                BucketNotificationConfiguration();
            notificationConfiguration.addConfiguration(
                "snsTopicConfig",
                new TopicConfiguration(snsTopicARN, EnumSet
                    .of(S3Event.ReducedRedundancyLostObject)));

            notificationConfiguration.addConfiguration(
                "sqsQueueConfig",
                new QueueConfiguration(sqsQueueARN, EnumSet
                    .of(S3Event.ObjectCreated)));

            SetBucketNotificationConfigurationRequest request =
                new SetBucketNotificationConfigurationRequest(bucketName,
                    notificationConfiguration);

            s3client.setBucketNotificationConfiguration(request);

        } catch (AmazonS3Exception ase) {
            System.out.println("Caught an AmazonServiceException, which "
                + "means your request made it "
                + "to Amazon S3, 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());
            System.out.println("Error XML" + ase.getErrorResponseXml());
        } catch (AmazonClientException ace) {
            System.out.println("Caught an AmazonClientException, which "
                + "means the client encountered "
                + "an internal error while trying to "
                + "communicate with S3, "
                + "such as not being able to access the network.");
            System.out.println("Error Message: " + ace.getMessage());
        }
    }
}
```

Step 4: Test the Setup

Now you can test the setup by uploading an object to your bucket and verify the event notification in the Amazon SQS console. For instructions, go to [Receiving a Message](#) in the *Amazon Simple Queue Service Getting Started Guide*.

Example Walkthrough 2: Configure a Bucket for Notifications (Message Destination: AWS Lambda)

For the example, go to [AWS Lambda Walkthrough 2: Handling Amazon S3 Events \(Using the AWS CLI\)](#) in the *AWS Lambda Developer Guide*.

Event Message Structure

The notification message Amazon S3 sends to publish an event is a JSON message with the following structure. Note the following:

- The `responseElements` key value is useful if you want to trace the request by following up with Amazon S3 support. Both `x-amz-request-id` and `x-amz-id-2` help Amazon S3 to trace the individual request. These values are the same as those that Amazon S3 returned in the response to your original PUT request, which initiated the event.
- The `s3` key provides information about the bucket and object involved in the event. Note that the object `keyname` value is URL encoded. For example "red flower.jpg" becomes "red+flower.jpg".

```
{  
    "Records": [  
        {  
            "eventVersion": "2.0",  
            "eventSource": "aws:s3",  
            "awsRegion": "us-east-1",  
            "eventTime": The time, in ISO-8601 format, for example, 1970-01-01T00:00:00.000Z, when S3 finished processing the request,  
            "eventName": "event-type",  
            "userIdentity": {  
                "principalId": "user-who-caused-the-event"  
            },  
            "requestParameters": {  
                "sourceIPAddress": "ip-address-where-request-came-from"  
            },  
            "responseElements": {  
                "x-amz-request-id": "Amazon S3 generated request ID",  
                "x-amz-id-2": "Amazon S3 host that processed the request"  
            },  
            "s3": {  
                "s3SchemaVersion": "1.0",  
                "configurationId": "ID found in the bucket notification configuration",  
                "bucket": {  
                    "name": "my-bucket",  
                    "objectKey": "my-object-key"  
                }  
            }  
        }  
    ]  
}
```

```

        "name": "bucket-name" ,
        "ownerIdentity": {
            "principalId": "Amazon S3 specific bucket owner ID"
        },
        "arn": "bucket-ARN"
    },
    "object": {
        "key": "object-key" ,
        "size": object-size,
        "eTag": "object eTag" ,
        "versionId": "object version if bucket is versioning-enabled,
otherwise null" }
    }
},
{
    // Additional events
}
]
}

```

The following are example messages:

- Test message—When you configure an event notification on a bucket, Amazon S3 sends the following test message:

```
{
    "Service": "Amazon S3",
    "Event": "s3:TestEvent",
    "Time": "2014-10-13T15:57:02.089Z",
    "Bucket": "bucketname" ,
    "RequestId": "5582815E1AEA5ADF" ,
    "HostId": "8cLeGAmw098X5cv4Zkwcmo8vvZa3eH3eKxsPzbB9wrR+YstdA6Knx4Ip8EXAMPLE"
}
```

- Example message when an object is created using the PUT request—The following message is an example of a message Amazon S3 sends to publish an *s3:ObjectCreated:Put* event:

```
{
    "Records": [
        {
            "eventVersion": "2.0",
            "eventSource": "aws:s3",
            "awsRegion": "us-east-1",
            "eventTime": "1970-01-01T00:00:00.000Z",
            "eventName": "ObjectCreated:Put",
            "userIdentity": {
                "principalId": "AIDAJDPLRKLG7UEEXAMPLE"
            },
            "requestParameters": {
                "sourceIPAddress": "127.0.0.1"
            },
            "responseElements": {
                "x-amz-request-id": "C3D13FE58DE4C810",
                "x-amz-id-2": "FMyUVURIY8/IgAtTv8xRjskZQpcIZ9KG4V5Wp6S7S/JRWeUWer
MUE5JgHvANOjpD"
            }
        }
    ]
}
```

```
        } ,
    "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "testConfigRule",
        "bucket": {
            "name": "mybucket",
            "ownerIdentity": {
                "principalId": "A3NL1KOZZKExample"
            },
            "arn": "arn:aws:s3:::mybucket"
        },
        "object": {
            "key": "HappyFace.jpg",
            "size": 1024,
            "eTag": "d41d8cd98f00b204e9800998ecf8427e",
            "versionId": "096fKKXTRTt13on89fVO.nfljtsv6qko"
        }
    }
}
```

Cross-Region Replication

Cross-region replication is automatic, asynchronous copying of objects across buckets in different AWS regions. To activate this bucket-level feature, you add a replication configuration to your source bucket. In the configuration, you provide information such as the destination bucket where you want objects replicated to. You can request Amazon S3 to replicate all or a subset of objects with specific key name prefixes. For example, you can configure cross-region replication to replicate only objects with the key name prefix "Tax/". This causes Amazon S3 to replicate objects with a key such as "Tax/doc1" or "Tax/doc2", but not an object with the key "Legal/doc3".

The object replicas in the destination bucket are exact replicas of the objects in the source bucket. They have the same key names and the same metadata—for example, creation time, owner, user-defined metadata, version ID, storage class, and ACL. Amazon S3 encrypts all data transfer in transit across AWS regions using SSL.

Use-case Scenarios

You might configure cross-region replication on a bucket for various reasons, including these:

- Compliance requirements – Although, by default, Amazon S3 stores your data across multiple geographically distant Availability Zones, compliance requirements might dictate that you store data at even further distances. Cross-region replication allows you to replicate data between distant AWS regions to satisfy these compliance requirements.
- Minimize latency – Your customers are in two geographic locations. To minimize latency in accessing objects, you can maintain object copies in AWS regions that are geographically closer to your users.
- Operational reasons – You have compute clusters in two different regions that analyze the same set of objects. You might choose to maintain object copies in those regions.

Requirements

Requirements for cross-region replication:

- The source and destination buckets must be versioning-enabled (see [Using Versioning \(p. 422\)](#)).
- The source and destination buckets must be in different AWS regions. For a list of AWS regions where you can create a bucket, go to Regions and Endpoints in [AWS General Reference](#).
- You can replicate objects from a source bucket to only one destination bucket.

- Amazon S3 must have permission to replicate objects from that source bucket to the destination bucket on your behalf.

You can grant these permissions by creating an IAM role that Amazon S3 can assume (see [Create an IAM Role \(p. 490\)](#)). You must grant this role permissions for S3 actions so that when Amazon S3 assumes this role, it can perform replication tasks.

- If the source bucket owner also owns the object, then the bucket owner has full permissions to replicate the object. If not, the source bucket owner must have permission for the S3 actions `s3:GetObjectVersion` and `s3:GetObjectVersionACL` to read the object and object ACL (see [Specifying Permissions in a Policy \(p. 315\)](#)). For information about resources and ownership, see [Amazon S3 Resources \(p. 270\)](#).
- If you are setting up cross-region replication in a cross-account scenario (where the source and destination buckets are owned by different AWS accounts), the source bucket owner must have permission to replicate objects in the destination bucket.

The destination bucket owner will need to grant these permissions via a bucket policy. For an example, see [Walkthrough 2: Configure Cross-Region Replication \(Source and Destination Buckets Owned by Different AWS Accounts\) \(p. 496\)](#).

What Is and Is Not Replicated

This section explains what Amazon S3 will replicate and what it will not, after you add a replication configuration on a bucket.

What Is Replicated

Amazon S3 will replicate the following:

- Any new objects created after you add a replication configuration, with exceptions described in the next section.
- Objects created with server-side encryption using the Amazon S3-managed encryption key. The replicated copy of the object will also be encrypted using server-side encryption using the Amazon S3-managed encryption key.
- Amazon S3 will replicate only objects in the source bucket for which the bucket owner has permission to read objects and read ACLs (see [About the Resource Owner \(p. 270\)](#)).
- Any object ACL updates are replicated, although there can be some delay before Amazon S3 can bring the two in sync. This applies only to objects created after you add a replication configuration to the bucket.

Delete Operation and Cross-Region Replication

If you delete an object from the source bucket, the cross-region replication behavior is as follows:

- If a DELETE request is made without specifying an object version ID, Amazon S3 adds a delete marker, which cross-region replication will replicate to the destination bucket. For more information about versioning and delete markers, see [Using Versioning \(p. 422\)](#).
- If a DELETE request specifies a particular object version ID to delete, Amazon S3 will delete that object version in the source bucket, but it will not replicate the deletion in the destination bucket (in other words, it will not delete the same object version from the destination bucket). This behavior protects data from malicious deletions.

What Is Not Replicated

Amazon S3 will not replicate the following:

- Amazon S3 will not retroactively replicate objects that existed before you added replication configuration.
- Objects created with server-side encryption using either customer-provided (SSE-C) or AWS KMS-managed encryption (SSE-KMS) keys are not replicated. For more information see, [Protecting Data Using Server-Side Encryption \(p. 381\)](#).

Amazon S3 does not keep the encryption keys you provide after the object is created in the source bucket so it cannot decrypt the object for replication, and therefore will not replicate the object.

- Amazon S3 will not replicate objects in the source bucket for which the bucket owner does not have permissions (see [Granting Cross-Account Permissions to Upload Objects While Ensuring the Bucket Owner Has Full Control \(p. 341\)](#) if object owner is different from the bucket owner).
- Updates to bucket-level subresources (see [Amazon S3 Resources \(p. 270\)](#)) are not replicated. This allows you to have different bucket configurations on the source and destination buckets.
- Only customer actions are replicated and not actions performed by lifecycle configuration (see [Object Lifecycle Management \(p. 86\)](#)).

For example, if lifecycle configuration is enabled only on your source bucket, Amazon S3 will create delete markers for expired objects, but will not replicate those markers. However, you can have the same lifecycle configuration on both the source and destination buckets if you want the same lifecycle actions to happen to both buckets.

- Objects in the source bucket that are replicas, created by another cross-region replication, will not be replicated.

Suppose you configure cross-region replication where bucket A is the source and bucket B is the destination. Now suppose you add another cross-region replication where bucket B is the source and bucket C is the destination. In this case, objects in bucket B that are replicas of objects in bucket A will not be replicated to bucket C.

Related Topics

For more information, see the following topics:

[Cross-Region Replication \(p. 487\)](#)

[How to Set Up Cross-Region Replication \(p. 490\)](#)

[How to Find Replication Status of an Object \(p. 504\)](#)

Related Topics

For more information, see the following topics:

[What Is and Is Not Replicated \(p. 488\)](#)

[How to Set Up Cross-Region Replication \(p. 490\)](#)

[How to Find Replication Status of an Object \(p. 504\)](#)

[Cross-Region Replication and Other Bucket Configurations \(p. 507\)](#)

[Walkthrough 1: Configure Cross-Region Replication \(Source and Destination Buckets Owned by the Same AWS Account\) \(p. 494\)](#)

[Walkthrough 2: Configure Cross-Region Replication \(Source and Destination Buckets Owned by Different AWS Accounts\) \(p. 496\)](#)

How to Set Up Cross-Region Replication

To set up cross-region replication, you need two buckets (source and destination). These buckets must be versioning-enabled and in different AWS regions. For a list of AWS regions where you can create a bucket, go to Regions and Endpoints in [AWS General Reference](#).

You can replicate objects from a source bucket to only one destination bucket. If both the buckets are owned by the same AWS account, you do the following to set up cross-region replication from the source to the destination bucket:

- Create an IAM role to grant Amazon S3 permission to replicate objects on your behalf.
- Add a replication configuration on the source bucket.

In addition, if the source and destination buckets are owned by two different AWS accounts, the destination bucket owner must also add a bucket policy to grant the source bucket owner permissions to perform replication actions.

Create an IAM Role

By default, all Amazon S3 resources—buckets, objects, and related subresources—are private: only the resource owner can access the resource. So Amazon S3 will need permission to read objects from the source bucket and replicate them to the destination bucket. You grant this permission by creating an IAM role. When you create an IAM role, you attach the following role policies:

- A trust policy in which you trust Amazon S3 to assume the role as shown:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "s3.amazonaws.com"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

Note the `Principal` in the policy identifies Amazon S3. For more information about IAM roles, go to [Roles \(Delegation and Federation\)](#) in [Using IAM](#).

- An access policy in which you grant the role permission to perform the replication task on your behalf. The following access policy grants these permissions:
 - The `s3:GetReplicationConfiguration` and `s3>ListBucket` permissions on the source bucket so Amazon S3 can retrieve replication configuration and list bucket (the current permission model requires the `s3>ListBucket` permission to access the delete markers).

- The `s3:GetObjectVersion` and `s3:GetObjectVersionAcl` permissions on all objects in the versioning-enabled source bucket. This allows Amazon S3 to get a specific object version and ACL on it.
- The `s3:ReplicateObject` and `s3:ReplicateDelete` permissions on objects in the destination bucket so Amazon S3 can replicate objects or delete markers from the destination bucket. For information about delete markers, see [Delete Operation and Cross-Region Replication \(p. 488\)](#).

For a list of Amazon S3 actions, see [Specifying Permissions in a Policy \(p. 315\)](#).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:GetReplicationConfiguration",  
                "s3>ListBucket"  
            ],  
            "Resource": [  
                "arn:aws:s3:::source-bucket"  
            ]  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:GetObjectVersion",  
                "s3:GetObjectVersionAcl"  
            ],  
            "Resource": [  
                "arn:aws:s3:::source-bucket/*"  
            ]  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:ReplicateObject",  
                "s3:ReplicateDelete"  
            ],  
            "Resource": "arn:aws:s3:::destination-bucket/*"  
        }  
    ]  
}
```

Add Replication Configuration

When you add a replication configuration to a bucket, Amazon S3 stores the configuration as XML. The following are example configurations. For more information about the XML structure, go to [PUT Bucket replication](#) in the *Amazon Simple Storage Service API Reference*.

Example 1: Replication configuration with one rule requesting

The following replication configuration has one rule. It requests Amazon S3 to replicate all objects to the specified destination bucket. The rule specifies an empty prefix indicating all objects. The configuration also specifies an IAM role Amazon S3 can assume to replicate objects on your behalf.

```
<?xml version="1.0" encoding="UTF-8"?>
<ReplicationConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Role>arn:aws:iam::account-id:role/role-name</Role>
  <Rule>
    <Status>Enabled</Status>
    <Prefix></Prefix>
    <Destination><Bucket>arn:aws:s3:::destinationbucket</Bucket></Destination>
  </Rule>
</ReplicationConfiguration>
```

Example 2: Replication configuration with two rules, each specifying a key name prefix

The following replication configuration specifies two rules. The first rule requests Amazon S3 to replicate objects with the key name prefix "TaxDocs/". The second rule requests Amazon S3 to replicate objects with key name prefix "ProjectDocs/". Amazon S3 will replicate, for example, objects with key names "TaxDocs/doc1.pdf" and "ProjectDocs/project1.txt". But it will not replicate any object with the key name "PersonalDoc/documentA". Note that both rules specify the same destination bucket.

```
<?xml version="1.0" encoding="UTF-8"?>
<ReplicationConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
    <Role>arn:aws:iam::account-id:role/role-name</Role>
    <Rule>
        <Prefix>TaxDocs</Prefix>
        ...
    </Rule>
    <Rule>
        <Prefix>ProjectDocs</Prefix>
        ...
    </Rule>
</ReplicationConfiguration>
```

Note that you cannot specify overlapping prefixes. The following example configuration has two rules specifying overlapping prefixes "TaxDocs/" and "TaxDocs/2015", which is not allowed.

```
<ReplicationConfiguration>
    <Role>arn:aws:iam::account-id:role/role-name</Role>
    <Rule>
        <Prefix>TaxDocs</Prefix>
        <Status>Enabled</Status>
        <Destination>
            <Bucket>arn:aws:s3:::destinationbucket</Bucket>
        </Destination>
    </Rule>
    <Rule>
        <Prefix>TaxDocs/2015</Prefix>
        <Status>Enabled</Status>
        <Destination>
            <Bucket>arn:aws:s3:::destinationbucket</Bucket>
        </Destination>
    </Rule>
</ReplicationConfiguration>
```

When adding replication configuration to a bucket you have two scenarios to consider depending on who owns the source and destination buckets.

Scenario 1: Buckets Owned by the Same AWS Account

When both the source and destination buckets are owned by same AWS account, you can use the Amazon S3 console to set up cross-region replication. Assuming you have source and destination buckets that are both versioning enabled, you can use the console to add replication configuration on the source bucket. For more information, see the following topics:

- [Walkthrough 1: Configure Cross-Region Replication \(Source and Destination Buckets Owned by the Same AWS Account\) \(p. 494\)](#)
- [Enabling Cross-Region Replication in the Amazon Simple Storage Service Console User Guide](#).

Scenario 2: Buckets Owned by Different AWS Accounts

When the source and destination buckets are owned by two different AWS accounts, you cannot add replication configuration using the console, since there is no way in the console for you to specify that a destination bucket is owned by another AWS account. So you add replication configuration programmatically using AWS SDKs or the AWS Command Line Interface. You specify a replication configuration as XML. The following is an example replication configuration:

```
<?xml version="1.0" encoding="UTF-8"?>
<ReplicationConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
    <Role>arn:aws:iam::46173example:role/CrrRoleName</Role>
    <Rule>
        <Status>Enabled</Status>
        <Prefix>TaxDocs</Prefix>
        <Destination><Bucket>arn:aws:s3:::destinationbucket</Bucket></Destination>
    </Rule>
</ReplicationConfiguration>
```

The configuration requests Amazon S3 to replicate objects with the key prefix "TaxDocs/" to the destinationbucket. The configuration also specifies an IAM role that Amazon S3 can assume to replicate objects on your behalf. For more information about the XML structure, go to [PUT Bucket replication](#) in the *Amazon Simple Storage Service API Reference*.

Because the destination bucket is owned by another AWS account, the destination bucket owners must also grant the source bucket owner permissions to replicate (replicate and delete) objects as shown:

```
{
    "Version": "2008-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::AWS account ID that owns the source bucket:root"
            },
            "Action": [ "s3:ReplicateObject", "s3:ReplicateDelete" ],
            "Resource": "arn:aws:s3:::destination bucket/*"
        }
    ]
}
```

This bucket policy on the destination bucket grants source bucket owner permissions for the Amazon S3 object operations, s3:ReplicateObject and s3:ReplicateDelete, on the destination bucket.

For an example walkthrough, see [Walkthrough 2: Configure Cross-Region Replication \(Source and Destination Buckets Owned by Different AWS Accounts\)](#) (p. 496).

Walkthrough 1: Configure Cross-Region Replication (Source and Destination Buckets Owned by the Same AWS Account)

In this section, you create two buckets (*source* and *destination*) in different AWS regions, enable versioning on both the buckets, and then configure cross-region replication on the *source* bucket.

1. Create two buckets.
 - a. Create a *source* bucket in an AWS region, say, Oregon (us-west-2).
For instructions, go to [Creating a Bucket](#) in the *Amazon Simple Storage Service Console User Guide*.
 - b. Create a *destination* bucket in another AWS region, say, US Standard (us-east-1).
2. Enable versioning on both buckets.
For instructions, go to [Enabling Bucket Versioning](#) in the *Amazon Simple Storage Service Console User Guide*.
3. Enable cross-region replication on the *source* bucket. You decide if you want to replicate all objects or only objects with a specific prefix (when using the console, think of this as deciding if you want to replicate only objects from a specific folder).
For instructions, go to [Enabling Cross-Region Replication](#) in the *Amazon Simple Storage Service Console User Guide*.
4. Test the setup as follows:
 - Create objects in the source bucket and verify that Amazon S3 replicated the objects in the destination bucket. Time it takes for Amazon S3 to replicate an object depends on the object size. For information about finding replication status, see [How to Find Replication Status of an Object \(p. 504\)](#).
 - Update the object's ACL in the source bucket and verify that changes appear in the destination bucket.

For instruction, go to [Editing Object Permissions](#) in the *Amazon Simple Storage Service Console User Guide*.

 - Update the object's metadata and verify that the changes appear in the destination bucket.

For instructions, go to [Editing Object Metadata](#) in the *Amazon Simple Storage Service Console User Guide*.

Remember, the replicas are exact copies of the objects in the source bucket.

Related Topics

For more information, see the following topics:

[Cross-Region Replication \(p. 487\)](#)

[Walkthrough 2: Configure Cross-Region Replication \(Source and Destination Buckets Owned by Different AWS Accounts\) \(p. 496\)](#)

[What Is and Is Not Replicated \(p. 488\)](#)

[How to Find Replication Status of an Object \(p. 504\)](#)

Walkthrough 2: Configure Cross-Region Replication (Source and Destination Buckets Owned by Different AWS Accounts)

In this walkthrough, you set up cross-region replication on the source bucket owned by one account to replicate objects in a destination bucket owned by another account.

The process is the same as setting up cross-region replication when both buckets are owned by the same account, except that you do one extra step; the destination bucket owner must create a bucket policy granting the source bucket owner permission for replication actions.

In this exercise, you will do all the steps using the console except, creating an IAM role, and setting a replication configuration on the source bucket. You will do this using either the AWS CLI or the AWS SDK for Java.

1. Create two buckets.

- a. Create a *source* bucket in an AWS region, say, Oregon (us-west-2) in *Account A*.

For instructions, go to [Creating a Bucket](#) in the *Amazon Simple Storage Service Console User Guide*.

- b. Create a *destination* bucket in another AWS region, say, US Standard (us-east-1) in *Account B*.

2. Enable versioning on both the buckets.

For instructions, go to [Enabling Bucket Versioning](#) in the *Amazon Simple Storage Service Console User Guide*.

3. Add the following bucket policy on the destination bucket to allow the source bucket owner permission for replication actions:

```
{  
    "Version": "2008-10-17",  
    "Id": "",  
    "Statement": [  
        {  
            "Sid": "Stmt123",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "arn:aws:iam::AWS-ID-Account-A:root"  
            },  
            "Action": [ "s3:ReplicateObject", "s3:ReplicateDelete" ],  
            "Resource": "arn:aws:s3:::destination-bucket/*"  
        }  
    ]  
}
```

For instructions, go to [Editing Bucket Permissions](#) in the *Amazon Simple Storage Service Console User Guide*.

4. Create an IAM role in *Account A*. Account A will then specify this role when adding replication configuration on the source bucket in the following step.

You will use the AWS CLI to create this IAM role. For instructions to setup the AWS CLI, see [Setting Up the Tools for the Example Walkthroughs \(p. 284\)](#).

- a. Copy the following policy and save it to a file called `s3-role-trust-policy.json`. The policy grants Amazon S3 permission to assume the role.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "s3.amazonaws.com"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

- b. Copy the following policy and save it to a file called `s3-role-permissions-policy.json`. This access policy grants permission for various Amazon S3 bucket and object actions. In the following step you will add the policy to the IAM role you are creating.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:GetObjectVersion",  
                "s3:GetObjectVersionAcl"  
            ],  
            "Resource": [  
                "arn:aws:s3:::source-bucket/*"  
            ]  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3>ListBucket",  
                "s3:GetReplicationConfiguration"  
            ],  
            "Resource": [  
                "arn:aws:s3:::source-bucket"  
            ]  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:ReplicateObject",  
                "s3:ReplicateDelete"  
            ],  
            "Resource": "arn:aws:s3:::destination-bucket/*"  
        }  
    ]  
}
```

- c. Run the following CLI command to create a role:

```
$ aws iam create-role \
--role-name RoleForS3CrossAccountCrossRegionReplication \
--assume-role-policy-document file://S3-role-trust-policy.json
```

- d. Run the following CLI command to create a policy:

```
$ aws iam create-policy \
--policy-name PolicyForS3CrossAccountCrossRegionReplication \
--policy-document file://S3-role-permissions-policy.json
```

- e. Write down the policy ARN that is returned in the output by the preceding command.

- f. Run the following CLI command to attach the policy to the role:

```
$ aws iam attach-role-policy \
--role-name RoleForS3CrossAccountCrossRegionReplication \
--policy-arn
```

Now Account A has created a role that the necessary Amazon S3 actions so it can replicate objects.

5. Enable cross-region replication on the *source* bucket in *Account A*. In the replication configuration you add one rule requesting Amazon S3 to replicate objects with the key name prefix "Tax/" to the specified destination bucket. Amazon S3 saves the replication configuration as XML as shown in the following example:

```
<ReplicationConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Role>arn:aws:iam::account-id:role/rolename</Role>
  <Rule>
    <Status>Enabled</Status>
    <Prefix>Tax</Prefix>
    <Destination><Bucket>arn:aws:s3:::destinationbucket</Bucket></Destination>
  </Rule>
</ReplicationConfiguration>
```

You can add the replication configuration to your source bucket using either the AWS CLI or AWS SDK.

- Using AWS CLI.

The AWS CLI requires you to specify the configuration as JSON. Save the following JSON in a file (replication.json). You will need to provide your bucket name and IAM role ARN.

```
{
  "Role": "arn:aws:iam::account A-id:role/role-name",
  "Rules": [
    {
      "Prefix": "Tax",
      "Status": "Enabled",
      "Destination": {
        "Bucket": "arn:aws:s3:::destination-bucket"
      }
    }
  ]
}
```

```
        ]  
    }
```

Then run the CLI command to add replication configuration to your source bucket:

```
$ aws s3api put-bucket-replication \  
--bucket source-bucket \  
--replication-configuration file://replication.json
```

For instructions to set up the AWS CLI, see [Setting Up the Tools for the Example Walkthroughs \(p. 284\)](#).

Account A can use the `get-bucket-replication` command to retrieve the replication configuration:

```
$ aws s3api get-bucket-replication \  
--bucket source-bucket
```

- Using the AWS SDK for Java.

For a code example, see [How to Set Up Cross-Region Replication Using the AWS SDK for Java \(p. 500\)](#).

6. Test the setup as follows:

- Using *Account A* credentials create objects in the `source` bucket and verify that Amazon S3 replicated the objects in the `destination` bucket owned by *Account B*. Time it takes for Amazon S3 to replicate an object depends on the object size. For information about finding replication status, see [How to Find Replication Status of an Object \(p. 504\)](#).

Note that when you upload objects in the source bucket the object key name must have a "Tax" prefix (for example, "Tax/document.pdf"). Accordingly to the replication configuration Account A added to the source bucket, Amazon S3 will only replicate objects with the "Tax" prefix.

- Update an object's ACL in the `source` bucket and verify that changes appear in the `destination` bucket.

For instructions, go to [Editing Object Permissions](#) in the *Amazon Simple Storage Service Console User Guide*.

- Update the object's metadata and verify that the changes appear in the destination bucket.

For instructions, go to [Editing Object Metadata](#) in the *Amazon Simple Storage Service Console User Guide*.

Remember, the replicas are exact copies of the objects in the source bucket.

Related Topics

For more information, see the following topics:

[Cross-Region Replication \(p. 487\)](#)

[What Is and Is Not Replicated \(p. 488\)](#)

[How to Find Replication Status of an Object \(p. 504\)](#)

[Walkthrough 1: Configure Cross-Region Replication \(Source and Destination Buckets Owned by the Same AWS Account\) \(p. 494\)](#)

How to Set Up Cross-Region Replication Using the Console

When both the source and destination buckets are owned by the same AWS account, you can add replication configuration on the source bucket using the Amazon S3 console. For more information, see the following topics:

- [Walkthrough 1: Configure Cross-Region Replication \(Source and Destination Buckets Owned by the Same AWS Account\) \(p. 494\)](#)
- [Enabling Cross-Region Replication in the *Amazon Simple Storage Service Console User Guide*.](#)

Related Topics

For more information, see the following topics:

[Cross-Region Replication \(p. 487\)](#)

[How to Set Up Cross-Region Replication \(p. 490\)](#)

How to Set Up Cross-Region Replication Using the AWS SDK for Java

When the source and destination buckets are owned by two different AWS accounts, you can use either the AWS CLI or one of the AWS SDKs to add replication configuration on the source bucket. You cannot use the console to add the replication configuration because there is no way for you to specify a destination bucket owned by another AWS account at the time you add replication configuration on a source bucket. For more information, see [How to Set Up Cross-Region Replication \(p. 490\)](#).

The following AWS SDK for Java code example first adds replication configuration to a bucket and then retrieves it. You will need to update the code by providing your bucket names and IAM role ARN. For instructions on how to create and test a working sample, see [Testing the Java Code Examples \(p. 545\)](#).

```
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.BucketReplicationConfiguration;
import com.amazonaws.services.s3.model.ReplicationDestinationConfig;
import com.amazonaws.services.s3.model.ReplicationRule;
import com.amazonaws.services.s3.model.ReplicationRuleStatus;

public class CrossRegionReplicationComplete {
    private static String sourceBucketName = "source-bucket";
```

```
private static String roleARN = "arn:aws:iam::account-id:role/role-name";  
  
private static String destinationBucketArn = "arn:aws:s3:::destination-  
bucket";  
  
public static void main(String[] args) throws IOException {  
    AmazonS3 s3Client = new AmazonS3Client(new ProfileCredentialsProvider());  
  
    try {  
        Map<String, ReplicationRule> replicationRules = new HashMap<String,  
ReplicationRule>();  
        replicationRules.put(  
            "a-sample-rule-id",  
            new ReplicationRule()  
                .withPrefix("Tax/")  
                .withStatus(ReplicationRuleStatus.Enabled)  
                .withDestinationConfig(  
                    new ReplicationDestinationConfig().withBucket  
ARN(destinationBucketArn)  
                )  
        );  
        s3Client.setBucketReplicationConfiguration(  
            sourceBucketName,  
            new BucketReplicationConfiguration()  
                .withRoleARN(roleARN)  
                .withRules(replicationRules)  
        );  
        BucketReplicationConfiguration replicationConfig = s3Client.getBucket  
etReplicationConfiguration(sourceBucketName);  
  
        ReplicationRule rule = replicationConfig.getRule("a-sample-rule-  
id");  
  
        System.out.println("Destination Bucket ARN : " + rule.getDestina  
tionConfig().getBucketARN());  
        System.out.println("Prefix : " + rule.getPrefix());  
        System.out.println("Status : " + rule.getStatus());  
  
    } catch (AmazonServiceException ase) {  
        System.out.println("Caught an AmazonServiceException, which" +  
            " means your request made it " +  
            "to Amazon S3, 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 Amazon S3, " +  
            "such as not being able to access the network.");  
        System.out.println("Error Message: " + ace.getMessage());  
    }  
}
```

```
}
```

Related Topics

For more information, see the following topics:

[Cross-Region Replication \(p. 487\)](#)

[How to Set Up Cross-Region Replication \(p. 490\)](#)

How to Set Up Cross-Region Replication Using the AWS SDK for .NET

When the source and destination buckets are owned by two different AWS accounts, you can use either the AWS CLI or one of the AWS SDKs to add replication configuration on the source bucket. You cannot use the console to add the replication configuration because there is no way for you to specify a destination bucket owned by another AWS account at the time you add replication configuration on a source bucket. For more information, see [How to Set Up Cross-Region Replication \(p. 490\)](#).

The following AWS SDK for .NET code example first adds replication configuration to a bucket and then retrieves it. You will need to update the code by providing your bucket names and IAM role ARN. For instructions on how to create and test a working sample, see [Testing the .NET Code Examples \(p. 547\)](#).

```
using System;
using System.Collections.Generic;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazonaws.com.docsamples
{
    class CrossRegionReplication
    {
        static string sourceBucket          = "source-bucket";
        static string destinationBucketArn = "arn:aws:s3:::destination-bucket";

        static string roleArn              = "arn:aws:iam::account-id:role/role-
name";

        public static void Main(string[] args)
        {
            try
            {
                using (var client = new AmazonS3Client(Amazon.RegionEnd
point.USEast1))
                {
                    EnableReplication(client);
                    RetrieveReplicationConfiguration(client);
                }

                Console.WriteLine("Press any key to continue...");
                Console.ReadKey();
            }
            catch (AmazonS3Exception amazonS3Exception)
```

```
{  
    if (amazonS3Exception.ErrorCode != null &&  
        (amazonS3Exception.ErrorCode.Equals("InvalidAccessKeyId")  
        ||  
        amazonS3Exception.ErrorCode.Equals("InvalidSecurity")))  
    {  
        Console.WriteLine("Check the provided AWS Credentials.");  
        Console.WriteLine(  
            "To sign up for service, go to http://aws.amazon.com/s3");  
  
    }  
    else  
    {  
        Console.WriteLine(  
            "Error occurred. Message:'{0}' when enabling notifica  
tions.",  
            amazonS3Exception.Message);  
    }  
}  
  
}  
  
static void EnableReplication(IAmazonS3 client)  
{  
    ReplicationConfiguration replConfig = new ReplicationConfiguration  
    {  
        Role = roleArn,  
        Rules =  
        {  
            new ReplicationRule  
            {  
                Prefix = "Tax",  
                Status = ReplicationRuleStatus.Enabled,  
                Destination = new ReplicationDestination  
                {  
                    BucketArn = destinationBucketArn  
                }  
            }  
        };  
    };  
  
    PutBucketReplicationRequest putRequest = new PutBucketReplica  
Request  
    {  
        BucketName = sourceBucket,  
        Configuration = replConfig  
    };  
  
    PutBucketReplicationResponse putResponse = client.PutBucketReplica  
tion(putRequest);  
}  
  
private static void RetrieveReplicationConfiguration(IAmazonS3 client)  
{  
    // Retrieve the configuration.  
    GetBucketReplicationRequest getRequest = new GetBucketReplica
```

```
Request
{
    BucketName = sourceBucket
};
GetBucketReplicationResponse getResponse = client.GetBucketReplica
tion(getRequest);
// Print.
Console.WriteLine("Printing replication configuration informa
tion...");
Console.WriteLine("Role ARN: {0}", getResponse.Configuration.Role);

foreach (var rule in getResponse.Configuration.Rules)
{
    Console.WriteLine("ID: {0}", rule.Id);
    Console.WriteLine("Prefix: {0}", rule.Prefix);
    Console.WriteLine("Status: {0}", rule.Status);
}
}
```

Related Topics

For more information, see the following topics:

[Cross-Region Replication \(p. 487\)](#)

[How to Set Up Cross-Region Replication \(p. 490\)](#)

Related Topics

For more information, see the following topics:

[Cross-Region Replication \(p. 487\)](#)

[What Is and Is Not Replicated \(p. 488\)](#)

[Walkthrough 1: Configure Cross-Region Replication \(Source and Destination Buckets Owned by the Same AWS Account\) \(p. 494\)](#)

[Walkthrough 2: Configure Cross-Region Replication \(Source and Destination Buckets Owned by Different AWS Accounts\) \(p. 496\)](#)

[How to Find Replication Status of an Object \(p. 504\)](#)

[Troubleshooting Cross-Region Replication \(p. 506\)](#)

How to Find Replication Status of an Object

In cross-region replication, you have a source bucket on which you configure replication and a destination bucket where Amazon S3 replicates objects. When you request an object (GET Object) or object metadata (HEAD Object) from these buckets, Amazon S3 will return the `x-amz-replication-status` header in the response as follow:

- If requesting an object from the source bucket — Amazon S3 will return the `x-amz-replication-status` header if the object in your request is eligible for replication.

For example, suppose in your replication configuration you specify the object prefix "TaxDocs" requesting Amazon S3 to replicate objects with the key name prefix "TaxDocs". Then any objects you upload with this key name prefix—for example, "TaxDocs/document1.pdf"—are eligible for replication. For any object request with this key name prefix, Amazon S3 will return the `x-amz-replication-status` header with the value PENDING, COMPLETED, or FAILED indicating the object replication status.

- If requesting an object from the destination bucket — Amazon S3 will return the `x-amz-replication-status` header with value REPLICA if the object in your request is a replica that Amazon S3 created.

You can find the object replication state in the console, using the AWS CLI, or programmatically using AWS SDK.

- In the console, you select the object and click **Properties** to view object properties including the replication status.
- You can use the `head-object` AWS CLI command as shown to retrieve object metadata information:

```
aws s3api head-object --bucket source-bucket --key object-key --version-id object-version-id
```

The command returns object metadata information including the `ReplicationStatus` as shown in the following example response:

```
{  
    "AcceptRanges": "bytes",  
    "ContentType": "image/jpeg",  
    "LastModified": "Mon, 23 Mar 2015 21:02:29 GMT",  
    "ContentLength": 3191,  
    "ReplicationStatus": "COMPLETED",  
    "VersionId": "jfnW.HIMOfYid_9rGbSkmroXsFj3fqZ.",  
    "ETag": "\"6805f2cfc46c0f04559748bb039d69ae\"",  
    "Metadata": {}  
}
```

- You can use the AWS SDKs to retrieve replication state of an object. Following are code fragments using AWS SDK for Java and AWS SDK for .NET.
- AWS SDK for Java

```
GetObjectMetadataRequest metadataRequest = new GetObjectMetadataRequest(bucketName, bucketName);  
metadataRequest.setKey(key);  
ObjectMetadata metadata = s3Client.getObjectMetadata(metadataRequest);  
  
System.out.println("Replication Status : " + metadata.getRawMetadataValue(Headers.OBJECT_REPLICATION_STATUS));
```

- AWS SDK for .NET

```
GetObjectMetadataRequest getmetadataRequest = new GetObjectMetadataRequest
{
    BucketName = sourceBucket,
    Key        = objectKey
};

GetObjectMetadataResponse getmetadataResponse = client.GetObjectMetadata(get
metadataRequest);
Console.WriteLine("Object replication status: {0}", getmetadataResponse.Rep
licationStatus);
```

Note

If you decide to delete an object from a source bucket that has replication enabled, you should check the replication status of the object before deletion to ensure the object has been replicated. If lifecycle configuration is enabled on the source bucket, Amazon S3 will put on hold any lifecycle actions until it marks the objects status as either "COMPLETED" or "FAILED".

Related Topics

[Cross-Region Replication \(p. 487\)](#)

Troubleshooting Cross-Region Replication

If, after configuring cross-region replication, you don't see the object replica created in the destination bucket, check out the following troubleshooting hints:

- The time it takes for Amazon S3 to replicate an object depends on the object size. For large objects, it can take up to several hours. If the object in question is large, check to see if the replicated object appears in the destination bucket again at a later time.
- In the replication configuration on the source bucket:
 - Verify the destination bucket ARN is correct.
 - Verify the key name prefix. For example, if you set the configuration to replicate objects with the prefix "Tax", then only objects with key names such as "Tax/document1" or "Tax/document2" are replicated. An object with the key name "document3" will not be replicated.
 - Verify the status is "enabled."
- If the destination bucket is owned by another AWS account, verify that the bucket owner has a bucket policy on the destination bucket that allows the source bucket owner to replicate objects.
- If an object replica does not appear in the destination bucket, note the following:
 - An object in a source bucket that is itself a replica created by another replication configuration, Amazon S3 will not replicate the replica. For example, if you set replication configuration from bucket A to bucket B to bucket C, Amazon S3 will not replicate objects replicas in bucket B.
 - A bucket owner can grant other AWS accounts permission to upload objects. By default, the bucket owner does not have any permissions on the objects created by the other account. And the replication configuration will replicate only the objects for which the bucket owner has access permissions. The bucket owner can grant other AWS accounts permissions to create objects conditionally requiring explicit access permissions on those objects. For an example policy, see [Granting Cross-Account Permissions to Upload Objects While Ensuring the Bucket Owner Has Full Control \(p. 341\)](#).

Related Topics

[Cross-Region Replication \(p. 487\)](#)

Cross-Region Replication and Other Bucket Configurations

In addition to replication configuration, Amazon S3 supports several other bucket configuration options including:

- Configure versioning on a bucket (see [Using Versioning \(p. 422\)](#))
- Configure a bucket for website hosting (see [Hosting a Static Website on Amazon S3 \(p. 448\)](#))
- Configure bucket access via a bucket policy (see [Using Bucket Policies and User Policies \(p. 312\)](#)) or ACL (see [Managing Access with ACLs \(p. 363\)](#))
- Configure a bucket to store access logs ([Server Access Logging \(p. 530\)](#))
- Configure the lifecycle for objects in the bucket (see [Object Lifecycle Management \(p. 86\)](#)).

This section explains how bucket replication configuration influences behavior of other bucket configurations:

Lifecycle Configuration and Object Replicas

The time it takes for Amazon S3 to replicate an object depends on object size. For large objects, it can take several hours. Even though it might take some time before a replica is available in the destination bucket, creation time of the replica remains the same as the corresponding object in the source bucket. Therefore, if you have a lifecycle policy on the destination bucket, note that lifecycle rules honor the original creation time of the object, not when the replica became available in the destination bucket.

Versioning Configuration and Replication Configuration

Both the source and destination buckets must be versioning-enabled when you configure replication on a bucket. After you enable versioning on both the source and destination buckets, and configure replication on the source bucket, note that:

- If you attempt to disable versioning on the source bucket, Amazon S3 will return an error. You must remove the replication configuration before you can disable versioning on the source bucket.
- If you disable versioning on the destination bucket, Amazon S3 stops replication.

Logging Configuration and Replication Configuration

If you have logging enabled on any bucket and Amazon S3 is delivering logs to your source bucket where you also have replication enabled, then Amazon S3 will replicate the log objects.

Related Topics

[Cross-Region Replication \(p. 487\)](#)

Request Routing

Topics

- [Request Redirection and the REST API \(p. 509\)](#)
- [DNS Considerations \(p. 513\)](#)

Programs that make requests against buckets created using the <CreateBucketConfiguration> API must support redirects. Additionally, some clients that do not respect DNS TTLs might encounter issues.

This section describes routing and DNS issues to consider when designing your service or application for use with Amazon S3.

Request Redirection and the REST API

Overview

Amazon S3 uses the Domain Name System (DNS) to route requests to facilities that can process them. This system works very effectively. However, temporary routing errors can occur.

If a request arrives at the wrong Amazon S3 location, Amazon S3 responds with a temporary redirect that tells the requester to resend the request to a new endpoint.

If a request is incorrectly formed, Amazon S3 uses permanent redirects to provide direction on how to perform the request correctly.

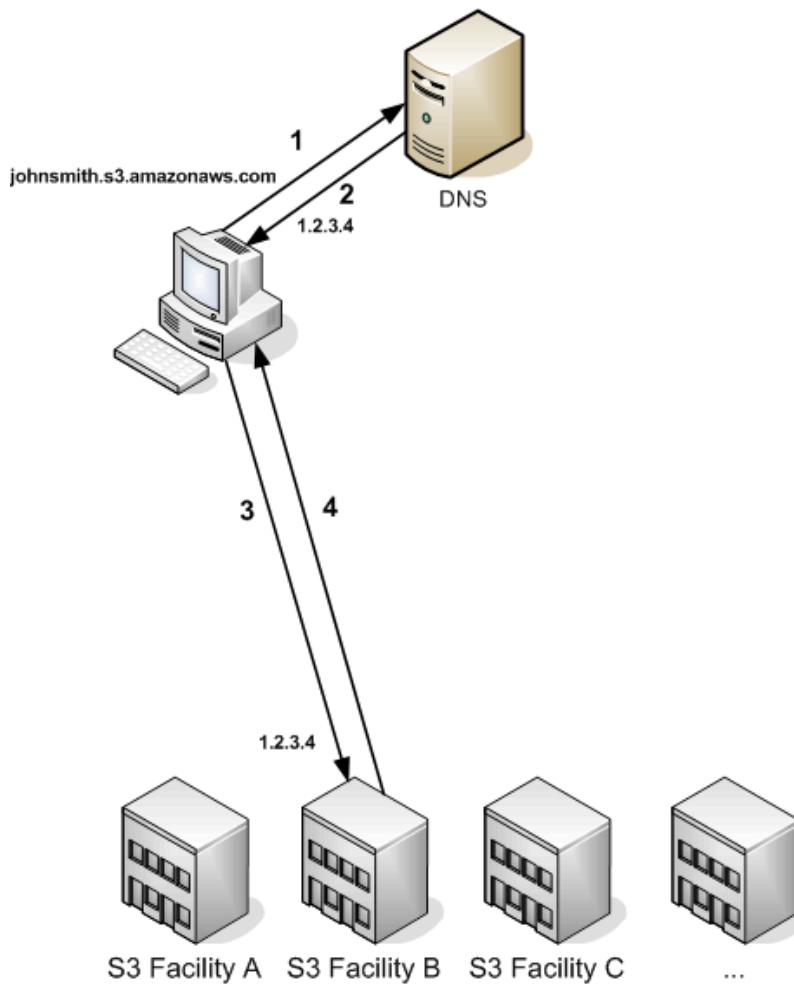
Important

Every Amazon S3 program must be designed to handle redirect responses. The only exception is for programs that work exclusively with buckets that were created without <CreateBucketConfiguration>. For more information on location constraints, see [Accessing a Bucket \(p. 57\)](#).

DNS Routing

DNS routing routes requests to appropriate Amazon S3 facilities.

The following figure shows an example of DNS routing.



1	The client makes a DNS request to get an object stored on Amazon S3.
2	The client receives one or more IP addresses for facilities that can process the request.
3	The client makes a request to Amazon S3 Facility B.
4	Facility B returns a copy of the object.

Temporary Request Redirection

A temporary redirect is a type of error response that signals to the requester that he should resend his request to a different endpoint.

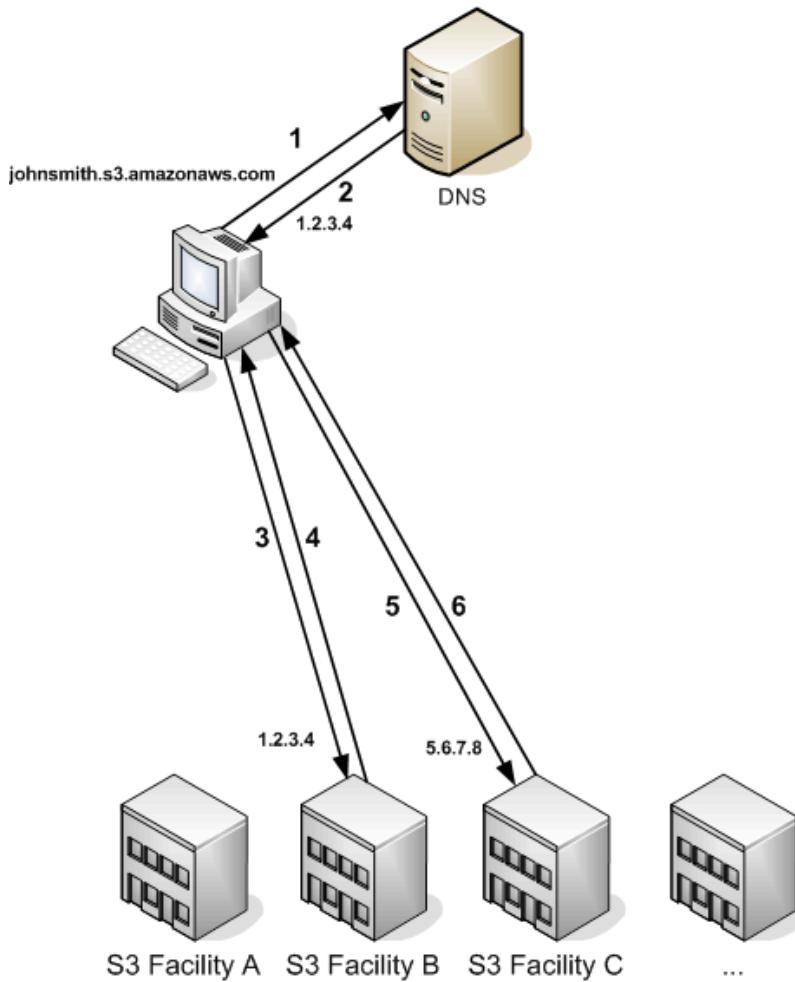
Due to the distributed nature of Amazon S3, requests can be temporarily routed to the wrong facility. This is most likely to occur immediately after buckets are created or deleted. For example, if you create a new bucket and immediately make a request to the bucket, you might receive a temporary redirect, depending on the location constraint of the bucket. If you created the bucket in the US Standard region (`s3.amazonaws.com` endpoint) you will not see the redirect because this is also the default endpoint. However, if bucket is created in any other region, any requests for the bucket will go to the default endpoint while the bucket's DNS entry is propagated. The default endpoint will redirect the request to the correct endpoint with a HTTP 302 response.

Temporary redirects contain a URI to the correct facility which you can use to immediately resend the request.

Important

Do not reuse an endpoint provided by a previous redirect response. It might appear to work (even for long periods of time), but might provide unpredictable results and will eventually fail without notice.

The following figure shows an example of a temporary redirect.



1	The client makes a DNS request to get an object stored on Amazon S3.
2	The client receives one or more IP addresses for facilities that can process the request.
3	The client makes a request to Amazon S3 Facility B.
4	Facility B returns a redirect indicating the object is available from Location C.
5	The client resends the request to Facility C.
6	Facility C returns a copy of the object.

Permanent Request Redirection

A permanent redirect indicates that your request addressed a resource inappropriately. For example, permanent redirects occur if you use a path-style request to access a bucket that was created using <CreateBucketConfiguration>. For more information, see [Accessing a Bucket \(p. 57\)](#).

To help you find these errors during development, this type of redirect does not contain a Location HTTP header that allows you to automatically follow the request to the correct location. Consult the resulting XML error document for help using the correct Amazon S3 endpoint.

Example REST API Redirect

```
HTTP/1.1 307 Temporary Redirect
Location: http://johnsmith.s3-gztb4pa9sq.amazonaws.com/photos/puppy.jpg?rk=e2c69a31
Content-Type: application/xml
Transfer-Encoding: chunked
Date: Fri, 12 Oct 2007 01:12:56 GMT
Server: AmazonS3

<?xml version="1.0" encoding="UTF-8"?>
<Error>
  <Code>TemporaryRedirect</Code>
  <Message>Please re-send this request to the specified temporary endpoint.
  Continue to use the original request endpoint for future requests.</Message>

  <Endpoint>johnsmith.s3-gztb4pa9sq.amazonaws.com</Endpoint>
</Error>
```

Example SOAP API Redirect

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

```
<soapenv:Body>
  <soapenv:Fault>
    <Faultcode>soapenv:Client.TemporaryRedirect</Faultcode>
    <Faultstring>Please re-send this request to the specified temporary endpoint.

    Continue to use the original request endpoint for future requests.</Faultstring>
    <Detail>
      <Bucket>images</Bucket>
      <Endpoint>s3-gztb4pa9sq.amazonaws.com</Endpoint>
    </Detail>
  </soapenv:Fault>
</soapenv:Body>
```

DNS Considerations

One of the design requirements of Amazon S3 is extremely high availability. One of the ways we meet this requirement is by updating the IP addresses associated with the Amazon S3 endpoint in DNS as needed. These changes are automatically reflected in short-lived clients, but not in some long-lived clients. Long-lived clients will need to take special action to re-resolve the Amazon S3 endpoint periodically to benefit from these changes. For more information about virtual machines (VMs), refer to the following:

- For Java, Sun's JVM caches DNS lookups forever by default; go to the "InetAddress Caching" section of [the InetAddress documentation](#) for information on how to change this behavior.
- For PHP, the persistent PHP VM that runs in the most popular deployment configurations caches DNS lookups until the VM is restarted. Go to [the getHostByName PHP docs](#).

Performance Optimization

Topics

- Request Rate and Performance Considerations (p. 514)
- TCP Window Scaling (p. 517)
- TCP Selective Acknowledgement (p. 518)

Amazon S3 provides new features that support high performance networking. These include TCP window scaling and selective acknowledgements.

Note

For more information on high performance tuning, go to <http://www.psc.edu/networking/projects/tcptune/>.

Request Rate and Performance Considerations

Topics

- Workloads with a Mix of Request Types (p. 515)
- GET-Intensive Workloads (p. 517)

Amazon S3 scales to support very high request rates. If your workload in an Amazon S3 bucket routinely exceeds 100 PUT/LIST/DELETE requests per second or more than 300 GET requests per second, follow the guidelines in this topic to ensure the best performance and scalability. If your request rate grows steadily, Amazon S3 automatically partitions your buckets as needed to support higher request rates. However, if you expect a rapid increase in the request rate for a bucket to more than 300 PUT/LIST/DELETE requests per second or more than 800 GET requests per second, we recommend that you open a support case to prepare for the workload and avoid any temporary limits on your request rate. To open a support case, go to [Contact Us](#).

This topic discusses two types of workloads:

- **Workloads that include a mix of request types** – If your requests are typically a mix of GET, PUT, DELETE, or GET Bucket (list objects), choosing appropriate key names for your objects will ensure better performance by providing low-latency access to the Amazon S3 index (discussed in the following section). It will also ensure scalability regardless of the number of requests you send per second.
- **Workloads that are GET-intensive** – If the bulk of your workload consists of GET requests, we recommend using the Amazon CloudFront content delivery service.

Note

The guidelines in this section apply if you are routinely processing 100 or more requests per second. If your typical workload involves only occasional bursts of 100 requests per second and less than 800 requests per second, you don't need to follow the guidelines in this section.

Workloads with a Mix of Request Types

When uploading a large number of objects, customers sometimes use sequential numbers or date and time values as part of their key names. For example, you might choose key names that use some combination of the date and time, as shown in the following example, where the prefix includes a timestamp:

```
examplebucket/2013-26-05-15-00-00/cust1234234/photo1.jpg
examplebucket/2013-26-05-15-00-00/cust3857422/photo2.jpg
examplebucket/2013-26-05-15-00-00/cust1248473/photo2.jpg
examplebucket/2013-26-05-15-00-00/cust8474937/photo2.jpg
examplebucket/2013-26-05-15-00-00/cust1248473/photo3.jpg
...
examplebucket/2013-26-05-15-00-01/cust1248473/photo4.jpg
examplebucket/2013-26-05-15-00-01/cust1248473/photo5.jpg
examplebucket/2013-26-05-15-00-01/cust1248473/photo6.jpg
examplebucket/2013-26-05-15-00-01/cust1248473/photo7.jpg
...
```

The sequence pattern in the key names introduces a performance problem. To understand the issue, let's look at how Amazon S3 stores key names.

Amazon S3 maintains an index of object key names in each AWS region. Object keys are stored lexicographically across multiple partitions in the index. That is, Amazon S3 stores key names in alphabetical order. The key name dictates which partition the key is stored in. Using a sequential prefix, such as timestamp or an alphabetical sequence, increases the likelihood that Amazon S3 will target a specific partition for a large number of your keys, overwhelming the I/O capacity of the partition. If you introduce some randomness in your key name prefixes, the key names, and therefore the I/O load, will be distributed across more than one partition.

If you anticipate that your workload will consistently exceed 100 requests per second, you should avoid sequential key names. If you must use sequential numbers or date and time patterns in key names, add a random prefix to the key name. The randomness of the prefix more evenly distributes key names across multiple index partitions. Examples of introducing randomness are provided later in this topic.

Note

The guidelines provided for the key name prefixes in the following section also apply to the bucket name. When Amazon S3 stores a key name in the index, it stores the bucket names as part of the key name (for example, examplebucket/object.jpg).

Example 1: Add a Hex Hash Prefix to Key Name

One way to introduce randomness to key names is to add a hash string as prefix to the key name. For example, you can compute an MD5 hash of the character sequence that you plan to assign as the key name. From the hash, pick a specific number of characters, and add them as the prefix to the key name. The following example shows key names with a four-character hash.

Note

A hashed prefix of three or four characters should be sufficient. We strongly recommend using a hexadecimal hash as the prefix.

```
examplebucket/232a-2013-26-05-15-00-00/cust1234234/photo1.jpg
examplebucket/7b54-2013-26-05-15-00-00/cust3857422/photo2.jpg
examplebucket/921c-2013-26-05-15-00-00/cust1248473/photo2.jpg
examplebucket/ba65-2013-26-05-15-00-00/cust8474937/photo2.jpg
examplebucket/8761-2013-26-05-15-00-00/cust1248473/photo3.jpg
examplebucket/2e4f-2013-26-05-15-00-01/cust1248473/photo4.jpg
examplebucket/9810-2013-26-05-15-00-01/cust1248473/photo5.jpg
examplebucket/7e34-2013-26-05-15-00-01/cust1248473/photo6.jpg
examplebucket/c34a-2013-26-05-15-00-01/cust1248473/photo7.jpg
...
```

Note that this randomness does introduce some interesting challenges. Amazon S3 provides a [GET Bucket \(List Objects\)](#) operation, which returns an alphabetical list of key names. Here are some side-effects:

- Because of the hashed prefixes, however, the alphabetical listing will appear randomly ordered.
- The problem gets compounded if you want to list object keys with specific date in the key name. The preceding example uses 4 character hex hash, so there are 65536 possible character combinations (4 character prefix, and each character can be any of the hex characters 0-f). So you will be sending 65536 List Bucket requests each with a specific prefix that is a combination of 4-digit hash and the date. For example, suppose you want to find all keys with 2013-26-05 in the key name. Then you will send List Bucket requests with prefixes such [0-f][0-f][0-f][0-f]2013-26-05.

You can optionally add more prefixes in your key name, before the hash string, to group objects. The following example adds `animations/` and `videos/` prefixes to the key names.

```
examplebucket/animations/232a-2013-26-05-15-00-00/cust1234234/animation1.obj
examplebucket/animations/7b54-2013-26-05-15-00-00/cust3857422/animation2.obj
examplebucket/animations/921c-2013-26-05-15-00-00/cust1248473/animation3.obj
examplebucket/videos/ba65-2013-26-05-15-00-00/cust8474937/video2.mpg
examplebucket/videos/8761-2013-26-05-15-00-00/cust1248473/video3.mpg
examplebucket/videos/2e4f-2013-26-05-15-00-01/cust1248473/video4.mpg
examplebucket/videos/9810-2013-26-05-15-00-01/cust1248473/video5.mpg
examplebucket/videos/7e34-2013-26-05-15-00-01/cust1248473/video6.mpg
examplebucket/videos/c34a-2013-26-05-15-00-01/cust1248473/video7.mpg
...
```

In this case, the ordered list returned by the GET Bucket (List Objects) operation will be grouped by the prefixes `animations` and `videos`.

Note

Again, the prefixes you add to group objects should not have sequences, or you will again overwhelm a single index partition.

Example 2: Reverse the Key Name String

Suppose your application uploads objects with key names whose prefixes include an increasing sequence of application IDs.

```
examplebucket/2134857/data/start.png
examplebucket/2134857/data/resource.rsrc
examplebucket/2134857/data/results.txt
examplebucket/2134858/data/start.png
examplebucket/2134858/data/resource.rsrc
examplebucket/2134858/data/results.txt
```

```
examplebucket/2134859/data/start.png  
examplebucket/2134859/data/resource.rsrc  
examplebucket/2134859/data/results.txt
```

In this key naming scheme, write operations will overwhelm a single index partition. If you reverse the application ID strings, however, you have the key names with random prefixes:

```
examplebucket/7584312/data/start.png  
examplebucket/7584312/data/resource.rsrc  
examplebucket/7584312/data/results.txt  
examplebucket/8584312/data/start.png  
examplebucket/8584312/data/resource.rsrc  
examplebucket/8584312/data/results.txt  
examplebucket/9584312/data/start.png  
examplebucket/9584312/data/resource.rsrc  
examplebucket/9584312/data/results.txt
```

Reversing the key name string lays the groundwork for Amazon S3 to start with the following partitions, one for each distinct first character in the key name. The examplebucket refers to the name of the bucket where you upload application data.

```
examplebucket/7  
examplebucket/8  
examplebucket/9
```

This example illustrate how Amazon S3 can use the first character of the key name for partitioning, but for very large workloads (more than 2000 requests per seconds or for bucket that contain billions of objects), Amazon S3 can use more characters for the partitioning scheme. Amazon S3 can automatically split these partitions further as the key count and request rate increase over time.

GET-Intensive Workloads

If your workload is mainly sending GET requests, in addition to the preceding guidelines, you should consider using Amazon CloudFront for performance optimization.

Integrating Amazon CloudFront with Amazon S3, you can distribute content to your users with low latency and a high data transfer rate. You will also send fewer direct requests to Amazon S3, which will reduce your costs.

For example, suppose you have a few objects that are very popular. Amazon CloudFront will fetch those objects from Amazon S3 and cache them. Amazon CloudFront can then serve future requests for the objects from its cache, reducing the number of GET requests it sends to Amazon S3. For more information, go to the [Amazon CloudFront](#) product detail page.

TCP Window Scaling

TCP window scaling allows you to improve network throughput performance between your operating system and application layer and Amazon S3 by supporting window sizes larger than 64 KB. At the start of the TCP session, a client advertises its supported receive window WSCALE factor, and Amazon S3 responds with its supported receive window WSCALE factor for the upstream direction.

Although TCP window scaling can improve performance, it can be challenging to set correctly. Make sure to adjust settings at both the application and kernel level. For more information about TCP window scaling, refer to your operating system's documentation and go to [RFC 1323](#).

TCP Selective Acknowledgement

TCP selective acknowledgement is designed to increase recovery time after a large number of packet losses. TCP selective acknowledgement is supported by most newer operating systems, but might have to be enabled. For more information about TCP selective acknowledgements, refer to the documentation that accompanied your operating system and go to [RFC 2018](#).

Using BitTorrent with Amazon S3

Topics

- [How You are Charged for BitTorrent Delivery \(p. 519\)](#)
- [Using BitTorrent to Retrieve Objects Stored in Amazon S3 \(p. 520\)](#)
- [Publishing Content Using Amazon S3 and BitTorrent \(p. 521\)](#)

BitTorrent is an open, peer-to-peer protocol for distributing files. You can use the BitTorrent protocol to retrieve any publicly-accessible object in Amazon S3. This section describes why you might want to use BitTorrent to distribute your data out of Amazon S3 and how to do so.

Amazon S3 supports the BitTorrent protocol so that developers can save costs when distributing content at high scale. Amazon S3 is useful for simple, reliable storage of any data. The default distribution mechanism for Amazon S3 data is via client/server download. In client/server distribution, the entire object is transferred point-to-point from Amazon S3 to every authorized user who requests that object. While client/server delivery is appropriate for a wide variety of use cases, it is not optimal for everybody. Specifically, the costs of client/server distribution increase linearly as the number of users downloading objects increases. This can make it expensive to distribute popular objects.

BitTorrent addresses this problem by recruiting the very clients that are downloading the object as distributors themselves: Each client downloads some pieces of the object from Amazon S3 and some from other clients, while simultaneously uploading pieces of the same object to other interested "peers." The benefit for publishers is that for large, popular files the amount of data actually supplied by Amazon S3 can be substantially lower than what it would have been serving the same clients via client/server download. Less data transferred means lower costs for the publisher of the object.

Note

You can get torrent only for objects that are less than 5 GB in size.

How You are Charged for BitTorrent Delivery

There is no extra charge for use of BitTorrent with Amazon S3. Data transfer via the BitTorrent protocol is metered at the same rate as client/server delivery. To be precise, whenever a downloading BitTorrent client requests a "piece" of an object from the Amazon S3 "seeder," charges accrue just as if an anonymous request for that piece had been made using the REST or SOAP protocol. These charges will appear on your Amazon S3 bill and usage reports in the same way. The difference is that if a lot of clients are requesting the same object simultaneously via BitTorrent, then the amount of data Amazon S3 must serve

to satisfy those clients will be lower than with client/server delivery. This is because the BitTorrent clients are simultaneously uploading and downloading amongst themselves.

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

The data transfer savings achieved from use of BitTorrent can vary widely depending on how popular your object is. Less popular objects require heavier use of the "seeder" to serve clients, and thus the difference between BitTorrent distribution costs and client/server distribution costs might be small for such objects. In particular, if only one client is ever downloading a particular object at a time, the cost of BitTorrent delivery will be the same as direct download.

Using BitTorrent to Retrieve Objects Stored in Amazon S3

Any object in Amazon S3 that can be read anonymously can also be downloaded via BitTorrent. Doing so requires use of a BitTorrent client application. Amazon does not distribute a BitTorrent client application, but there are many free clients available. The Amazon S3BitTorrent implementation has been tested to work with the official BitTorrent client (go to <http://www.bittorrent.com/>).

The starting point for a BitTorrent download is a .torrent file. This small file describes for BitTorrent clients both the data to be downloaded and where to get started finding that data. A .torrent file is a small fraction of the size of the actual object to be downloaded. Once you feed your BitTorrent client application an Amazon S3 generated .torrent file, it should start downloading immediately from Amazon S3 and from any "peer" BitTorrent clients.

Retrieving a .torrent file for any publicly available object is easy. Simply add a "?torrent" query string parameter at the end of the REST GET request for the object. No authentication is required. Once you have a BitTorrent client installed, downloading an object using BitTorrent download might be as easy as opening this URL in your web browser.

There is no mechanism to fetch the .torrent for an Amazon S3 object using the SOAP API.

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

Example

This example retrieves the Torrent file for the "Nelson" object in the "quotes" bucket.

Sample Request

```
GET /quotes/Nelson?torrent HTTP/1.0
Date: Wed, 25 Nov 2009 12:00:00 GMT
```

Sample Response

```
HTTP/1.1 200 OK
x-amz-request-id: 7CD745EBB7AB5ED9
Date: Wed, 25 Nov 2009 12:00:00 GMT
Content-Disposition: attachment; filename=Nelson.torrent;
Content-Type: application/x-bittorrent
Content-Length: 537
Server: AmazonS3

<body: a Bencoded dictionary as defined by the BitTorrent specification>
```

Publishing Content Using Amazon S3 and BitTorrent

Every anonymously readable object stored in Amazon S3 is automatically available for download using BitTorrent. The process for changing the ACL on an object to allow anonymous READ operations is described in [Managing Access Permissions to Your Amazon S3 Resources \(p. 269\)](#).

You can direct your clients to your BitTorrent accessible objects by giving them the .torrent file directly or by publishing a link to the ?torrent URL of your object. One important thing to note is that the .torrent file describing an Amazon S3 object is generated on-demand, the first time it is requested (via the REST ?torrent resource). Generating the .torrent for an object takes time proportional to the size of that object. For large objects, this time can be significant. Therefore, before publishing a ?torrent link, we suggest making the first request for it yourself. Amazon S3 might take several minutes to respond to this first request, as it generates the .torrent file. Unless you update the object in question, subsequent requests for the .torrent will be fast. Following this procedure before distributing a ?torrent link will ensure a smooth BitTorrent downloading experience for your customers.

To stop distributing a file using BitTorrent, simply remove anonymous access to it. This can be accomplished by either deleting the file from Amazon S3, or modifying your access control policy to prohibit anonymous reads. After doing so, Amazon S3 will no longer act as a "seeder" in the BitTorrent network for your file, and will no longer serve the .torrent file via the ?torrent REST API. However, after a .torrent for your file is published, this action might not stop public downloads of your object that happen exclusively using the BitTorrent peer to peer network.

Using Amazon DevPay with Amazon S3

Topics

- [Amazon S3 Customer Data Isolation \(p. 522\)](#)
- [Amazon DevPay Token Mechanism \(p. 523\)](#)
- [Amazon S3 and Amazon DevPay Authentication \(p. 523\)](#)
- [Amazon S3 Bucket Limitation \(p. 524\)](#)
- [Amazon S3 and Amazon DevPay Process \(p. 525\)](#)
- [Additional Information \(p. 525\)](#)

Amazon DevPay enables you to charge customers for using your Amazon S3 product through Amazon's authentication and billing infrastructure. You can charge any amount for your product including usage charges (storage, transactions, and bandwidth), monthly fixed charges, and a one-time charge.

Once a month, Amazon bills your customers for you. AWS then deducts the fixed Amazon DevPay transaction fee and pays you the difference. AWS then separately charges you for the Amazon S3 usage costs incurred by your customers and the percentage-based Amazon DevPay fee.

If your customers do not pay their bills, AWS turns off access to Amazon S3 (and your product). AWS handles all payment processing.

Amazon S3 Customer Data Isolation

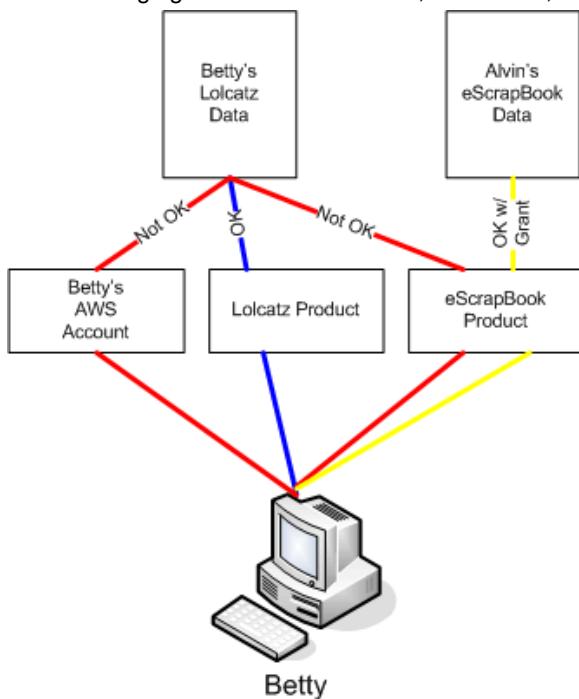
Amazon DevPay requests store and access data on behalf of the users of your product. The resources created by your application are owned by your users; unless you modify the ACL, you cannot read or modify the user's data.

Data stored by your product is isolated from other Amazon DevPay products and general Amazon S3 access. Customers that *store* data in Amazon S3 through your product can only *access* that data through your product. The data cannot be accessed through other Amazon DevPay products or through a personal AWS account.

Two users of a product can only access each others data if your application explicitly grants access through the ACL.

Example

The following figure illustrates allowed, disallowed, and conditional (discretionary) data access.



Betty's access is limited as follows:

- She can access Lolcatz data through the Lolcatz product. If she attempts to access her Lolcatz data through another product or a personal AWS account, her requests will be denied.
- She can access Alvin's eScrapBook data through the eScrapBook product if access is explicitly granted.

Amazon DevPay Token Mechanism

To enable you to make requests on behalf of your customers and ensure that your customers are billed for use of your application, your application must send two tokens with each request: the product token and the user token.

The product token identifies your product; you must have one product token for each Amazon DevPay product that you provide. The user token identifies a user in relationship to your product; you must have a user token for each user/product combination. For example, if you provide two products and a user subscribes to each, you must obtain a separate user token for each product.

For information on obtaining product and user tokens, refer to the *Amazon DevPay Getting Started Guide*.

Amazon S3 and Amazon DevPay Authentication

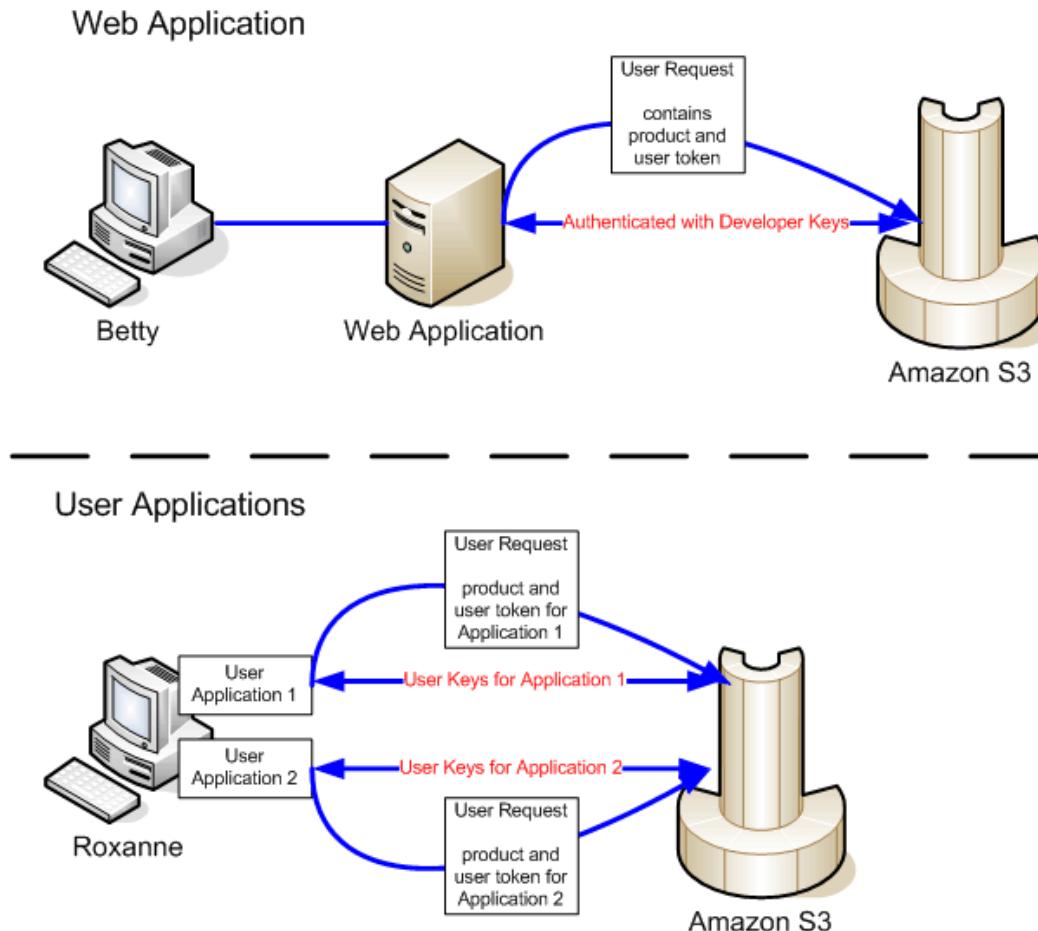
Although the token mechanism uniquely identifies a customer and product, it does not provide authentication.

Normally, your applications communicate directly with Amazon S3 using your Access Key ID and Secret Access Key. For Amazon DevPay, Amazon S3 authentication works a little differently.

If your Amazon DevPay product is a web application, you securely store the Secret Access Key on your servers and use the user token to specify the customer for which requests are being made.

However, if your Amazon S3 application is installed on your customers' computers, your application must obtain an Access Key ID and a Secret Access Key for each installation and must use those credentials when communicating with Amazon S3.

The following figure shows the differences between authentication for web applications and user applications.



Amazon S3 Bucket Limitation

Each of your customers can have up to 100 buckets for each Amazon DevPay product that you sell. For example, if a customer uses three of your products, the customer can have up to 300 buckets ($100 * 3$) plus any buckets outside of your Amazon DevPay products (i.e., buckets in Amazon DevPay products from other developers and the customer's personal AWS account).

Amazon S3 and Amazon DevPay Process

Following is a high-level overview of the Amazon DevPay process.

Launch Process

1	A customer signs up for your product through Amazon.
2	The customer receives an activation key.
3	The customer enters the activation key into your application.
4	Your application communicates with Amazon and obtains the user's token. If your application is installed on the user's computer, it also obtains an Access Key ID and Secret Access Key on behalf of the customer.
5	Your application provides the customer's token and the application product token when making Amazon S3 requests on behalf of the customer. If your application is installed on the customer's computer, it authenticates with the customer's credentials.
6	Amazon uses the customer's token and your product token to determine who to bill for the Amazon S3 usage.
7	Once a month, Amazon processes usage data and bills your customers according to the terms you defined.
8	AWS deducts the fixed Amazon DevPay transaction fee and pays you the difference. AWS then separately charges you for the Amazon S3 usage costs incurred by your customers and the percentage-based Amazon DevPay fee.

Additional Information

For information about using, setting up, and integrating with Amazon DevPay, go to [Amazon DevPay](#)

Handling REST and SOAP Errors

Topics

- [The REST Error Response \(p. 526\)](#)
- [The SOAP Error Response \(p. 528\)](#)
- [Amazon S3 Error Best Practices \(p. 528\)](#)

This section describes REST and SOAP errors and how to handle them.

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

The REST Error Response

Topics

- [Response Headers \(p. 527\)](#)
- [Error Response \(p. 527\)](#)

If a REST request results in an error, the HTTP reply has:

- An XML error document as the response body
- Content-Type: application/xml
- An appropriate 3xx, 4xx, or 5xx HTTP status code

Following is an example of a REST Error Response.

```
<?xml version="1.0" encoding="UTF-8"?>
<Error>
  <Code>NoSuchKey</Code>
  <Message>The resource you requested does not exist</Message>
  <Resource>/mybucket/myfoto.jpg</Resource>
  <RequestId>4442587FB7D0A2F9</RequestId>
</Error>
```

For more information about Amazon S3 errors, go to [ErrorCodeList](#).

Response Headers

Following are response headers returned by all operations:

- *x-amz-request-id*: A unique ID assigned to each request by the system. In the unlikely event that you have problems with Amazon S3, Amazon can use this to help troubleshoot the problem.
- *x-amz-id-2*: A special token that will help us to troubleshoot problems.

Error Response

Topics

- [Error Code \(p. 527\)](#)
- [Error Message \(p. 527\)](#)
- [Further Details \(p. 527\)](#)

When an Amazon S3 request is in error, the client receives an error response. The exact format of the error response is API specific: For example, the REST error response differs from the SOAP error response. However, all error responses have common elements.

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

Error Code

The error code is a string that uniquely identifies an error condition. It is meant to be read and understood by programs that detect and handle errors by type. Many error codes are common across SOAP and REST APIs, but some are API-specific. For example, `NoSuchKey` is universal, but `UnexpectedContent` can occur only in response to an invalid REST request. In all cases, SOAP fault codes carry a prefix as indicated in the table of error codes, so that a `NoSuchKey` error is actually returned in SOAP as `Client.NoSuchKey`.

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

Error Message

The error message contains a generic description of the error condition in English. It is intended for a human audience. Simple programs display the message directly to the end user if they encounter an error condition they don't know how or don't care to handle. Sophisticated programs with more exhaustive error handling and proper internationalization are more likely to ignore the error message.

Further Details

Many error responses contain additional structured data meant to be read and understood by a developer diagnosing programming errors. For example, if you send a Content-MD5 header with a REST PUT request that doesn't match the digest calculated on the server, you receive a `BadDigest` error. The error response also includes as detail elements the digest we calculated, and the digest you told us to expect.

During development, you can use this information to diagnose the error. In production, a well-behaved program might include this information in its error log.

The SOAP Error Response

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

In SOAP, an error result is returned to the client as a SOAP fault, with the HTTP response code 500. If you do not receive a SOAP fault, then your request was successful. The Amazon S3 SOAP fault code is comprised of a standard SOAP 1.1 fault code (either "Server" or "Client") concatenated with the Amazon S3-specific error code. For example: "Server.InternalError" or "Client.NoSuchBucket". The SOAP fault string element contains a generic, human readable error message in English. Finally, the SOAP fault detail element contains miscellaneous information relevant to the error.

For example, if you attempt to delete the object "Fred", which does not exist, the body of the SOAP response contains a "NoSuchKey" SOAP fault.

Example

```
<soapenv:Body>
  <soapenv:Fault>
    <Faultcode>soapenv:Client.NoSuchKey</Faultcode>
    <Faultstring>The specified key does not exist.</Faultstring>
    <Detail>
      <Key>Fred</Key>
    </Detail>
  </soapenv:Fault>
</soapenv:Body>
```

For more information about Amazon S3 errors, go to [ErrorCodeList](#).

Amazon S3 Error Best Practices

When designing an application for use with Amazon S3, it is important to handle Amazon S3 errors appropriately. This section describes issues to consider when designing your application.

Retry InternalErrors

Internal errors are errors that occur within the Amazon S3 environment.

Requests that receive an InternalError response might not have processed. For example, if a PUT request returns InternalError, a subsequent GET might retrieve the old value or the updated value.

If Amazon S3 returns an InternalError response, retry the request.

Tune Application for Repeated SlowDown errors

As with any distributed system, S3 has protection mechanisms which detect intentional or unintentional resource over-consumption and react accordingly. SlowDown errors can occur when a high request rate triggers one of these mechanisms. Reducing your request rate will decrease or eliminate errors of this

type. Generally speaking, most users will not experience these errors regularly; however, if you would like more information or are experiencing high or unexpected SlowDown errors, please post to our Amazon S3 developer forum <https://forums.aws.amazon.com/> or sign up for AWS Premium Support <http://aws.amazon.com/premiumsupport/>.

Isolate Errors

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

Amazon S3 provides a set of error codes that are used by both the SOAP and REST API. The SOAP API returns standard Amazon S3 error codes. The REST API is designed to look like a standard HTTP server and interact with existing HTTP clients (e.g., browsers, HTTP client libraries, proxies, caches, and so on). To ensure the HTTP clients handle errors properly, we map each Amazon S3 error to an HTTP status code.

HTTP status codes are less expressive than Amazon S3 error codes and contain less information about the error. For example, the `NoSuchKey` and `NoSuchBucket` Amazon S3 errors both map to the `HTTP 404 Not Found` status code.

Although the HTTP status codes contain less information about the error, clients that understand HTTP, but not the Amazon S3 API, will usually handle the error correctly.

Therefore, when handling errors or reporting Amazon S3 errors to end users, use the Amazon S3 error code instead of the HTTP status code as it contains the most information about the error. Additionally, when debugging your application, you should also consult the human readable `<Details>` element of the XML error response.

Server Access Logging

Overview

In order to track requests for access to your bucket, you can enable access logging. Each access log record provides details about a single access request, such as the requester, bucket name, request time, request action, response status, and error code, if any. Access log information can be useful in security and access audits. It can also help you learn about your customer base and understand your Amazon S3 bill.

Note

There is no extra charge for enabling server access logging on an Amazon S3 bucket; however, any log files the system delivers to you will accrue the usual charges for storage. (You can delete the log files at any time.) No data transfer charges will be assessed for log file delivery, but access to the delivered log files is charged the same as any other data transfer.

By default, logging is disabled. To enable access logging, you must do the following:

- Turn on the log delivery by adding logging configuration on the bucket for which you want Amazon S3 to deliver access logs. We will refer to this bucket as the *source bucket*.
- Grant the Amazon S3 Log Delivery group write permission on the bucket where you want the access logs saved. We will refer to this bucket as the *target bucket*.

To turn on log delivery, you provide the following logging configuration information:

- Name of the target bucket name where you want Amazon S3 to save the access logs as objects. You can have logs delivered to any bucket that you own, including the source bucket. We recommend that you save access logs in a different bucket so you can easily manage the logs. If you choose to save access logs in the same bucket as the source bucket, we recommend you specify a prefix to all log object keys so that you can easily identify the log objects.

Note

Both the source and target buckets must be owned by the same AWS account.

- (Optional) A prefix for Amazon S3 to assign to all log object keys. The prefix will make it simpler for you to locate the log objects.

For example, if you specify the prefix value `logs/`, each log object that Amazon S3 creates will begin with the `logs/` prefix in its key, as in this example:

```
logs/2013-11-01-21-32-16-E568B2907131C0C0
```

The key prefix can help when you delete the logs. For example, you can set a lifecycle configuration rule for Amazon S3 to delete objects with a specific key prefix. For more information, see [Deleting Log Files \(p. 541\)](#).

- (Optional) Permissions so that others can access the generated logs. By default, the bucket owner always has full access to the log objects. You can optionally grant access to other users.

Log Object Key Format

Amazon S3 uses the following object key format for the log objects it uploads in the target bucket:

```
TargetPrefixYYYY-mm-DD-HH-MM-SS-UniqueString
```

In the key, YYYY, mm, DD, HH, MM and SS are the digits of the year, month, day, hour, minute, and seconds (respectively) when the log file was delivered.

A log file delivered at a specific time can contain records written at any point before that time. There is no way to know whether all log records for a certain time interval have been delivered or not.

The UniqueString component of the key is there to prevent overwriting of files. It has no meaning, and log processing software should ignore it.

How are Logs Delivered?

Amazon S3 periodically collects access log records, consolidates the records in log files, and then uploads log files to your target bucket as log objects. If you enable logging on multiple source buckets that identify the same target bucket, the target bucket will have access logs for all those source buckets, but each log object will report access log records for a specific source bucket.

Amazon S3 uses a special log delivery account, called the Log Delivery group, to write access logs. These writes are subject to the usual access control restrictions. You will need to grant the Log Delivery group write permission on the target bucket by adding a grant entry in the bucket's access control list (ACL). If you use the Amazon S3 console to enable logging on a bucket, the console will both enable logging on the source bucket and update the ACL on the target bucket to grant write permission to the Log Delivery group.

Best Effort Server Log Delivery

Server access log records are delivered on a best effort basis. Most requests for a bucket that is properly configured for logging will result in a delivered log record, and most log records will be delivered within a few hours of the time that they were recorded.

The completeness and timeliness of server logging, however, is not guaranteed. The log record for a particular request might be delivered long after the request was actually processed, or it might not be delivered at all. The purpose of server logs is to give you an idea of the nature of traffic against your bucket. It is not meant to be a complete accounting of all requests. It is rare to lose log records, but server logging is not meant to be a complete accounting of all requests.

It follows from the best-effort nature of the server logging feature that the usage reports available at the AWS portal (Billing and Cost Management reports on the [AWS Management Console](#)) might include one or more access requests that do not appear in a delivered server log.

Bucket Logging Status Changes Take Effect Over Time

Changes to the logging status of a bucket take time to actually affect the delivery of log files. For example, if you enable logging for a bucket, some requests made in the following hour might be logged, while others might not. If you change the target bucket for logging from bucket A to bucket B, some logs for the next hour might continue to be delivered to bucket A, while others might be delivered to the new target bucket B. In all cases, the new settings will eventually take effect without any further action on your part.

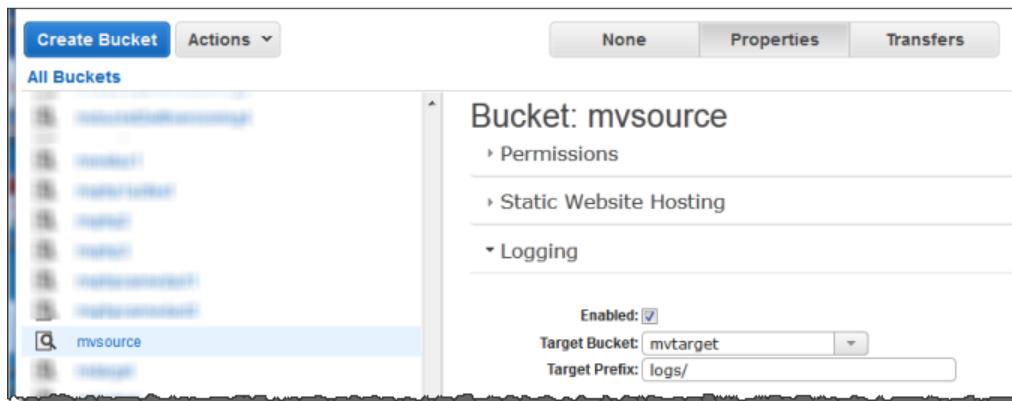
Related Topics

For more information about server access logging, see the following topics.

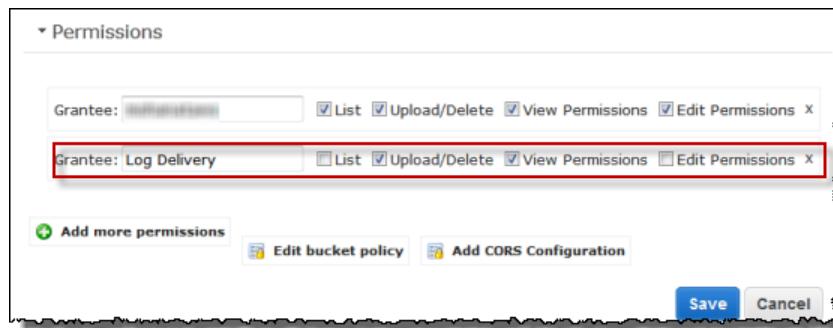
- [Enabling Logging Using the Console \(p. 532\)](#)
- [Enabling Logging Programmatically \(p. 534\)](#)
- [Server Access Log Format \(p. 537\)](#)
- [Deleting Log Files \(p. 541\)](#)

Enabling Logging Using the Console

To enable logging (see [Server Access Logging \(p. 530\)](#)) the [Amazon S3 console](#) provides a **Logging** section in the bucket **Properties**:



When you enable logging on a bucket the console will both enable logging on the source bucket and add a grant in the target bucket's ACL granting write permission to the Log Delivery group.



To enable logging on a bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Under **All Buckets**, click the bucket for which access requests will be logged.
3. In the Details pane, click **Properties**
4. Under **Logging**, do the following:



- Select the **Enabled** check box
 - In the **Target Bucket** box, click the name of the bucket that will receive the log objects.
 - (optional) To specify a key prefix for log objects, in the **Target Prefix** box, type the prefix that you want.
5. Click **Save**.

To disable logging on a bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Under **All Buckets**, click the bucket for which access requests will be logged.
3. In the Details pane, click **Properties** Under **Logging**, clear the **Enabled** check box.
4. Click **Save**.

For information about enable logging programmatically, see [Enabling Logging Programmatically \(p. 534\)](#).

For information about the log record format, including the list of fields and their descriptions, see [Server Access Log Format \(p. 537\)](#).

Enabling Logging Programmatically

Topics

- [Enabling logging \(p. 534\)](#)
- [Granting the Log Delivery Group WRITE and READ_ACP Permissions \(p. 534\)](#)
- [Example: AWS SDK for .NET \(p. 535\)](#)

You can enable or disable logging programmatically by using either the Amazon S3 API or the AWS SDKs. To do so, you both enable logging on the bucket and grant the Log Delivery group permission to write logs to the target bucket.

Enabling logging

To enable logging, you submit a [PUT Bucket logging](#) request to add the logging configuration on source bucket. The request specifies the target bucket and, optionally, the prefix to be used with all log object keys. The following example identifies `logbucket` as the target bucket and `logs/` as the prefix.

```
<BucketLoggingStatus xmlns="http://doc.s3.amazonaws.com/2006-03-01">
  <LoggingEnabled>
    <TargetBucket>logbucket</TargetBucket>
    <TargetPrefix>logs/</TargetPrefix>
  </LoggingEnabled>
</BucketLoggingStatus>
```

The log objects are written and owned by the Log Delivery account and the bucket owner is granted full permissions on the log objects. In addition, you can optionally grant permissions to other users so that they may access the logs. For more information, see [PUT Bucket logging](#).

Amazon S3 also provides the [GET Bucket logging](#) API to retrieve logging configuration on a bucket. To delete logging configuration you send the PUT Bucket logging request with empty `<BucketLoggingStatus>` empty.

```
<BucketLoggingStatus xmlns="http://doc.s3.amazonaws.com/2006-03-01">
</BucketLoggingStatus>
```

You can use either the Amazon S3 API or the AWS SDK wrapper libraries to enable logging on a bucket.

Granting the Log Delivery Group WRITE and READ_ACP Permissions

Amazon S3 writes the log files to the target bucket as a member of the predefined Amazon S3 group Log Delivery. These writes are subject to the usual access control restrictions. You will need to grant `s3:GetObjectAcl` and `s3:PutObject` permissions to this group by adding grants to the access control list (ACL) of the target bucket. The Log Delivery group is represented by the following URL.

```
http://acs.amazonaws.com/groups/s3/LogDelivery
```

To grant WRITE and READ_ACP permissions, you have to add the following grants. For information about ACLs, see [Managing Access with ACLs \(p. 363\)](#).

```
<Grant>
    <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="Group">
        <URI>http://acs.amazonaws.com/groups/s3/LogDelivery</URI>
    </Grantee>
    <Permission>WRITE</Permission>
</Grant>
<Grant>
    <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="Group">
        <URI>http://acs.amazonaws.com/groups/s3/LogDelivery</URI>
    </Grantee>
    <Permission>READ_ACP</Permission>
</Grant>
```

For examples of adding ACL grants programmatically using AWS SDKs, see [Managing ACLs Using the AWS SDK for Java \(p. 369\)](#) and [Managing ACLs Using the AWS SDK for .NET \(p. 373\)](#).

Example: AWS SDK for .NET

The following C# example enables logging on a bucket. You will need to create two buckets, source bucket and target bucket. The example first grants the Log Delivery group necessary permission to write logs to the target bucket and then enable logging on the source bucket. For more information, see [Enabling Logging Programmatically \(p. 534\)](#). For instructions on how to create and test a working sample, see [Testing the .NET Code Examples \(p. 547\)](#).

```
using System;
using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazonaws.com.docsamples
{
    class ServerAccessLogging
    {
        static string sourceBucket = "**** Provide bucket name ****"; // On which
        to enable logging.
        static string targetBucket = "**** Provide bucket name ****"; // Where
        access logs can be stored.
        static string logObjectKeyPrefix = "Logs";
        static IAmazonS3 client;

        public static void Main(string[] args)
        {
            using (client = new AmazonS3Client(Amazon.RegionEndpoint.USEast1))
            {
                Console.WriteLine("Enabling logging on source bucket...");
                try
                {
                    // Step 1 - Grant Log Delivery group permission to write
                    log to the target bucket.
                    GrantLogDeliveryPermissionToWriteLogsInTargetBucket();
                    // Step 2 - Enable logging on the source bucket.
                    EnableDisableLogging();
                }
                catch (AmazonS3Exception amazonS3Exception)
```

```

    {
        if (amazonS3Exception.ErrorCode != null &&
            (amazonS3Exception.ErrorCode.Equals("InvalidAccessKeyId") ||
             amazonS3Exception.ErrorCode.Equals("InvalidSecurity")))

        {
            Console.WriteLine("Check the provided AWS Credentials.");

            Console.WriteLine(
                "To sign up for service, go to ht
tp://aws.amazon.com/s3");
        }
        else
        {
            Console.WriteLine(
                "Error occurred. Message:'{0}' when enabling logging",
                amazonS3Exception.Message);
        }
    }

    Console.WriteLine("Press any key to continue...");  

    Console.ReadKey();
}

static void GrantLogDeliveryPermissionToWriteLogsInTargetBucket()
{
    S3AccessControlList bucketACL = new S3AccessControlList();
    GetACLResponse aclResponse = client.GetACL(new GetACLRequest {
        BucketName = targetBucket });
    bucketACL = aclResponse.AccessControlList;
    bucketACL.AddGrant(new S3Grantee { URI = "http://acs.amazon
aws.com/groups/s3/LogDelivery" }, S3Permission.WRITE);
    bucketACL.AddGrant(new S3Grantee { URI = "http://acs.amazon
aws.com/groups/s3/LogDelivery" }, S3Permission.READ_ACP);
    PutACLRequest setACLRequest = new PutACLRequest
    {
        AccessControlList = bucketACL,
        BucketName = targetBucket
    };
    client.PutACL(setACLRequest);
}

static void EnableDisableLogging()
{
    S3BucketLoggingConfig loggingConfig = new S3BucketLoggingConfig
    {
        TargetBucketName = targetBucket,
        TargetPrefix = logObjectKeyPrefix
    };

    // Send request.
    PutBucketLoggingRequest putBucketLoggingRequest = new PutBucketLog
gingRequest
    {
}

```

```
        BucketName = sourceBucket,
        LoggingConfig = loggingConfig
    };
    PutBucketLoggingResponse response = client.PutBucketLogging(putBucketLoggingRequest);
}
}
```

Server Access Log Format

The server access log files consist of a sequence of new-line delimited log records. Each log record represents one request and consists of space delimited fields. The following is an example log consisting of six log records.

```
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be mybucket
[06/Feb/2014:00:00:38 +0000] 192.0.2.3 79a59df900b949e55d96a1e698fbaced
fd6e09d98eacf8f8d5218e7cd47ef2be 3E57427F3EXAMPLE REST.GET.VERSIONING - "GET
/mybucket?versioning HTTP/1.1" 200 - 113 - 7 - "-" "S3Console/0.4" -
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be mybucket
[06/Feb/2014:00:00:38 +0000] 192.0.2.3 79a59df900b949e55d96a1e698fbaced
fd6e09d98eacf8f8d5218e7cd47ef2be 891CE47D2EXAMPLE REST.GET.LOGGING_STATUS -
"GET /mybucket?logging HTTP/1.1" 200 - 242 - 11 - "-" "S3Console/0.4" -
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be mybucket
[06/Feb/2014:00:00:38 +0000] 192.0.2.3 79a59df900b949e55d96a1e698fbaced
fd6e09d98eacf8f8d5218e7cd47ef2be A1206F460EXAMPLE REST.GET.BUCKETPOLICY -
"GET /mybucket?policy HTTP/1.1" 404 NoSuchBucketPolicy 297 - 38 - "-" "S3Console/0.4"
-
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be mybucket
[06/Feb/2014:00:01:00 +0000] 192.0.2.3 79a59df900b949e55d96a1e698fbaced
fd6e09d98eacf8f8d5218e7cd47ef2be 7B4A0FABBEXAMPLE REST.GET.VERSIONING - "GET
/mybucket?versioning HTTP/1.1" 200 - 113 - 33 - "-" "S3Console/0.4" -
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be mybucket
[06/Feb/2014:00:01:57 +0000] 192.0.2.3 79a59df900b949e55d96a1e698fbaced
fd6e09d98eacf8f8d5218e7cd47ef2be DD6CC733AEXAMPLE REST.PUT.OBJECT s3-dg.pdf
"PUT /mybucket/s3-dg.pdf HTTP/1.1" 200 - 4406583 41754 28 "-" "S3Console/0.4"
-
79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be mybucket
[06/Feb/2014:00:03:21 +0000] 192.0.2.3 79a59df900b949e55d96a1e698fbaced
fd6e09d98eacf8f8d5218e7cd47ef2be BC3C074D0EXAMPLE REST.GET.VERSIONING - "GET
/mybucket?versioning HTTP/1.1" 200 - 113 - 28 - "-" "S3Console/0.4" -
```

Note

Any field can be set to "-" to indicate that the data was unknown or unavailable, or that the field was not applicable to this request.

The following table describes the log record fields.

Field Name	Example Entry	Notes
Bucket Owner	79a59df900b949e55d96a1e698fbaced fd6e09d98eacf8f8d5218e7cd47ef2be	The canonical user id of the owner of the source bucket.
Bucket	mybucket	The name of the bucket that the request was processed against. If the system receives a malformed request and cannot determine the bucket, the request will not appear in any server access log.
Time	[06/Feb/2014:00:00:38 +0000]	The time at which the request was received. The format, using <code>strftime()</code> terminology, is [%d/%b/%Y:%H:%M:%S %z]
Remote IP	192.0.2.3	The apparent Internet address of the requester. Intermediate proxies and firewalls might obscure the actual address of the machine making the request.
Requester	79a59df900b949e55d96a1e698fbaced fd6e09d98eacf8f8d5218e7cd47ef2be	The canonical user id of the requester, or the string "Anonymous" for unauthenticated requests. This identifier is the same one used for access control purposes.
Request ID	3E57427F33A59F07	The request ID is a string generated by Amazon S3 to uniquely identify each request.
Operation	REST.PUT.OBJECT	Either SOAP. <i>operation</i> , REST. <i>HTTP_method.re-source_type</i> or WEBSITE. <i>HTTP_method.re-source_type</i>
Key	/photos/2014/08/puppy.jpg	The "key" part of the request, URL encoded, or "-" if the operation does not take a key parameter.
Request-URI	"GET /mybucket/photos/2014/08/puppy.jpg?x-foo=bar"	The Request-URI part of the HTTP request message.
HTTP status	200	The numeric HTTP status code of the response.
Error Code	NoSuchBucket	The Amazon S3 Error Code (p. 527) , or "-" if no error occurred.
Bytes Sent	2662992	The number of response bytes sent, excluding HTTP protocol overhead, or "-" if zero.
Object Size	3462992	The total size of the object in question.

Field Name	Example Entry	Notes
Total Time	70	The number of milliseconds the request was in flight from the server's perspective. This value is measured from the time your request is received to the time that the last byte of the response is sent. Measurements made from the client's perspective might be longer due to network latency.
Turn-Around Time	10	The number of milliseconds that Amazon S3 spent processing your request. This value is measured from the time the last byte of your request was received until the time the first byte of the response was sent.
Referrer	"http://www.amazon.com/web services"	The value of the HTTP Referrer header, if present. HTTP user-agents (e.g. browsers) typically set this header to the URL of the linking or embedding page when making a request.
User-Agent	"curl/7.15.1"	The value of the HTTP User-Agent header.
Version Id	3HL4kqtJvjbVH40Nrjfkd	The version ID in the request, or "-" if the operation does not take a <i>versionId</i> parameter.

Custom Access Log Information

You can include custom information to be stored in the access log record for a request by adding a custom query-string parameter to the URL for the request. Amazon S3 will ignore query-string parameters that begin with "x-", but will include those parameters in the access log record for the request, as part of the Request-URI field of the log record. For example, a GET request for "s3.amazonaws.com/mybucket/photos/2014/08/puppy.jpg?x-user=johndoe" will work the same as the same request for "s3.amazonaws.com/mybucket/photos/2014/08/puppy.jpg", except that the "x-user=johndoe" string will be included in the Request-URI field for the associated log record. This functionality is available in the REST interface only.

Programming Considerations for Extensible Server Access Log Format

From time to time, we might extend the access log record format by adding new fields to the end of each line. Code that parses server access logs must be written to handle trailing fields that it does not understand.

Additional Logging for Copy Operations

A copy operation involves a `GET` and a `PUT`. For that reason, we log two records when performing a copy operation. The previous table describes the fields related to the `PUT` part of the operation. The following table describes the fields in the record that relate to the `GET` part of the copy operation.

Field Name	Example Entry	Notes
Bucket Owner	79a59df900b949e55d96a1e698fbaced fd6e09d98eacf8f8d5218e7cd47ef2be	The canonical user id of the bucket that stores the object being copied.
Bucket	mybucket	The name of the bucket that stores the object being copied.
Time	[06/Feb/2014:00:00:38 +0000]	The time at which the request was received. The format, using <code>strftime()</code> terminology, is [%d/%B/%Y:%H:%M:%S %z]
Remote IP	192.0.2.3	The apparent Internet address of the requester. Intermediate proxies and firewalls might obscure the actual address of the machine making the request.
Requester	79a59df900b949e55d96a1e698fbaced fd6e09d98eacf8f8d5218e7cd47ef2be	The canonical user id of the requester, or the string "Anonymous" for unauthenticated requests. This identifier is the same one used for access control purposes.
Request ID	3E57427F33A59F07	The request ID is a string generated by Amazon S3 to uniquely identify each request.
Operation	REST.COPY.OBJECT_GET	Either <code>SOAP.operation</code> , <code>REST.HTTP_method.re-source_type</code> or <code>WEBSITE.HTTP_method.re-source_type</code>
Key	/photos/2014/08/puppy.jpg	The "key" of the object being copied or "-" if the operation does not take a key parameter.
Request-URI	"GET /mybucket/photos/2014/08/puppy.jpg?x-foo=bar"	The Request-URI part of the HTTP request message.
HTTP status	200	The numeric HTTP status code of the GET portion of the copy operation.
Error Code	NoSuchBucket	The Amazon S3 Error Code (p. 527) , of the GET portion of the copy operation or "-" if no error occurred.
Bytes Sent	2662992	The number of response bytes sent, excluding HTTP protocol overhead, or "-" if zero.
Object Size	3462992	The total size of the object in question.

Field Name	Example Entry	Notes
Total Time	70	The number of milliseconds the request was in flight from the server's perspective. This value is measured from the time your request is received to the time that the last byte of the response is sent. Measurements made from the client's perspective might be longer due to network latency.
Turn-Around Time	10	The number of milliseconds that Amazon S3 spent processing your request. This value is measured from the time the last byte of your request was received until the time the first byte of the response was sent.
Referrer	"http://www.amazon.com/web services"	The value of the HTTP Referrer header, if present. HTTP user-agents (e.g. browsers) typically set this header to the URL of the linking or embedding page when making a request.
User-Agent	"curl/7.15.1"	The value of the HTTP User-Agent header.
Version Id	3HL4kqtJvjVBH40Nrjfkd	The version ID of the object being copied or "-" if the x-amz-copy-source header didn't specify a <i>versionId</i> parameter as part of the copy source.

Deleting Log Files

A logging enabled bucket (see [Server Access Logging \(p. 530\)](#)) can have many server log objects created over time. Your application might need these access logs for a specific period after creation, and after that you may want to delete them. You can use Amazon S3 lifecycle configuration to set rules so that Amazon S3 automatically queues these objects for deletion at the end of their life.

If you specified a prefix in your logging configuration, you can set lifecycle configuration rule to delete log objects with that prefix. For example, if you log objects have prefix `logs/` after a specified time, you can set lifecycle configuration rule to delete objects with prefix `/logs`. For more information about lifecycle configuration, see [Object Lifecycle Management \(p. 86\)](#).

Using the AWS SDKs and Explorers

Topics

- [Specifying Signature Version in Request Authentication \(p. 543\)](#)
- [Using the AWS SDK for Java \(p. 544\)](#)
- [Using the AWS SDK for .NET \(p. 546\)](#)
- [Using the AWS SDK for PHP and Running PHP Examples \(p. 547\)](#)
- [Using the AWS SDK for Ruby \(p. 549\)](#)
- [Using the AWS SDK for Python \(Boto\) \(p. 550\)](#)

You can use the AWS SDKs when developing applications with Amazon S3. The AWS SDKs simplify your programming tasks by wrapping the underlying REST API. Mobile SDKs are also available for building connected mobile applications using AWS. This section provides an overview of using AWS SDKs for developing Amazon S3 applications. This section also describes how you can test the AWS SDK code samples provided in this guide.

In addition to the AWS SDKs, AWS Explorers are available for Visual Studio and Eclipse for Java IDE. In this case, the SDKs and the explorers are available bundled together as AWS Toolkits.

You can also use the AWS Command Line Interface (CLI) to manage Amazon S3 buckets and objects.

AWS Toolkit for Eclipse

The AWS Toolkit for Eclipse includes both the AWS SDK for Java and AWS Explorer for Eclipse. The AWS Explorer for Eclipse is an open source plug-in for Eclipse for Java IDE that makes it easier for developers to develop, debug, and deploy Java applications using AWS. The easy to use GUI interface enables you to access and administer your AWS infrastructure including Amazon S3. You can perform common operations such as manage your buckets and objects, set IAM policies, while developing applications, all from within the context of Eclipse for Java IDE. For set up instructions, go to [Setting Up the AWS Toolkit for Eclipse](#). For examples of using Amazon S3 using the explorer, go to [Viewing and Editing Amazon S3 Buckets](#).

AWS Toolkit for Visual Studio

AWS Explorer for Visual Studio is an extension for Microsoft Visual Studio that makes it easier for developers to develop, debug, and deploy .NET applications using Amazon Web Services. The easy-to-use

GUI enables you to access and administer your AWS infrastructure including Amazon S3. You can perform common operations such as managing your buckets and objects or setting IAM policies, while developing applications, all from within the context of Visual Studio. For set up instructions, go to [Setting Up the AWS Toolkit for Visual Studio](#). For examples of using Amazon S3 using the explorer, go to [Using Amazon S3 from AWS Explorer](#).

AWS SDKs

You can download only the SDKs. For information about downloading the SDK libraries, go to [Sample Code Libraries](#).

AWS CLI

The AWS Command Line Interface (CLI) is a unified tool to manage your AWS services, including Amazon S3. For information about downloading the AWS CLI, go to [AWS Command Line Interface](#).

Specifying Signature Version in Request Authentication

In the China (Beijing) and EU (Frankfurt) regions, Amazon S3 supports only Signature Version 4, and AWS SDKs use this signature version to authenticate requests.

In all other regions, Amazon S3 supports both Signature Version 4 and Signature Version 2. For these regions, AWS SDKs use Signature Version 2 by default to authenticate requests. You can optionally request the SDKs to use Signature Version 4 as shown in the following table:

SDK	Requesting Signature Version 4 for Request Authentication
AWS CLI	<p>For the default profile, run the following command.</p> <pre>\$ aws configure set default.s3.signature_version s3v4</pre> <p>For a custom profile, run the following command.</p> <pre>\$ aws configure set profile.your_profile_name.s3.signature_version s3v4</pre>
Java SDK	<p>Add the following in your code.</p> <pre>System.setProperty(SDKGlobalConfiguration.ENABLE_S3_SIGV4_SYSTEM_PROPERTY, "true");</pre> <p>Or, on the command line, specify the following.</p> <pre>-Dcom.amazonaws.services.s3.enableV4</pre>

SDK	Requesting Signature Version 4 for Request Authentication
JavaScript SDK	<p>Set the <code>signatureVersion</code> parameter to <code>v4</code> when constructing the client.</p> <pre>var s3 = new AWS.S3({signatureVersion: 'v4'});</pre>
PHP SDK	<p>Set the <code>signature</code> parameter to <code>v4</code> when constructing the Amazon S3 service client.</p> <pre><?php \$s3 = \Aws\S3\S3Client::factory(array('signature' => 'v4'));</pre>
Python-Boto SDK	<p>Specify the following in the <code>.boto</code>, default config file.</p> <pre>[s3] use-sigv4 = True</pre>
Ruby SDK	<p>Ruby SDK - Version 1: Set the <code>:s3_signature_version</code> parameter to <code>:v4</code> when constructing the client.</p> <pre>s3 = AWS::S3::Client.new(:s3_signature_version => :v4)</pre> <p>Ruby SDK - Version 2: Set the <code>signature_version</code> parameter to <code>v4</code> when constructing the client.</p> <pre>s3 = Aws::S3::Client.new(signature_version: 'v4')</pre>
.NET SDK	<p>Add the following to the code before creating the S3 client.</p> <pre>AWSConfigs.S3UseSignatureVersion4 = true;</pre> <p>Or, add the following to the config file.</p> <pre><appSettings> <add key="AWS.S3.UseSignatureVersion4" value='true' /> </appSettings></pre>

Using the AWS SDK for Java

The AWS SDK for Java provides an API for the Amazon S3 bucket and object operations. For object operations, in addition to providing the API to upload objects in a single operation, the SDK provides API

to upload large objects in parts (see [Uploading Objects Using Multipart Upload API \(p. 155\)](#)). The API gives you the option of using a high-level or low-level API.

Low-Level API

The low-level APIs correspond to the underlying Amazon S3 REST operations, such as create, update, and delete operations that apply to buckets and objects. When you upload large objects using the low-level multipart upload API, it provides greater control such as letting you pause and resume multipart uploads, vary part sizes during the upload, or to begin uploads when you do not know the size of the data in advance. If you do not have these requirements, use the high-level API to upload objects.

High-Level API

For uploading objects, the SDK provides a higher level of abstraction by providing the `TransferManager` class. The high-level API is a simpler API, where in just a few lines of code you can upload files and streams to Amazon S3. You should use this API to upload data unless you need to control the upload as described in the preceding Low-Level API section.

For smaller data size the `TransferManager` API uploads data in a single operation. However, the `TransferManager` switches to using the multipart upload API when data size reaches certain threshold. When possible, the `TransferManager` uses multiple threads to concurrently upload the parts. If a part upload fails, the API retries the failed part upload up to three times. However, these are configurable options using the `TransferManagerConfiguration` class.

Note

When using a stream for the source of data, the `TransferManager` class will not do concurrent uploads.

The Java API Organization

The following packages in the AWS SDK for Java provide the API:

- **`com.amazonaws.services.s3`**—Provides the implementation APIs for Amazon S3 bucket and object operations.
For example, it provides methods to create buckets, upload objects, get objects, delete objects, and to list keys.
- **`com.amazonaws.services.s3.transfer`**—Provides the high-level API data upload.
This high-level API is designed to further simplify uploading objects to Amazon S3. It includes the `TransferManager` class. It is particularly useful when uploading large objects in parts. It also include the `TransferManagerConfiguration` class which you can use to configure the minimum part size for uploading parts and the threshold in bytes of when to use multipart uploads.
- **`com.amazonaws.services.s3.model`**—Provides the low-level API classes to create requests and process responses.
For example, it includes the `GetObjectRequest` class to describe your get object request, the `ListObjectRequest` class to describe your list keys requests, and the `InitiateMultipartUploadRequest` and `InitiateMultipartUploadResult` classes when initiating a multipart upload.

For more information about the AWS SDK for Java API, go to [AWS SDK for Java API Reference](#).

Testing the Java Code Examples

The easiest way to get started with the Java code examples is to install the latest AWS Toolkit for Eclipse. For information on setting up your Java development environment and the AWS Toolkit for Eclipse, go to [Install a Java Development Environment](#).

The following tasks guide you through the creation and testing of the Java code examples provided in this guide.

General Process of Creating Java Code Examples

1	Create an AWS credentials profile file as described in the AWS SDK for Java topic Set up your AWS Credentials for Use with the SDK for Java .
2	Create a new AWS Java project in Eclipse. The project is pre-configured with the AWS SDK for Java.
3	Copy the code from the section you are reading to your project.
4	Update the code by providing any required data. For example, if uploading a file, provide the file path and the bucket name.
5	Run the code. Verify that the object is created by using the AWS Management Console. For more information about the AWS Management Console, go to http://aws.amazon.com/console/ .

Using the AWS SDK for .NET

Topics

- [The .NET API Organization \(p. 547\)](#)
- [Testing the .NET Code Examples \(p. 547\)](#)

The AWS SDK for .NET provides the API for the Amazon S3 bucket and object operations. For object operations, in addition to providing the API to upload objects in a single operation, the SDK provides the API to upload large objects in parts (see [Uploading Objects Using Multipart Upload API \(p. 155\)](#)). The API gives you the option of using a high-level or low-level API.

Low-Level API

The low-level APIs correspond to the underlying Amazon S3 REST operations, including the create, update, and delete operations that apply to buckets and objects. When you upload large objects using the low-level multipart upload API (see [Uploading Objects Using Multipart Upload API \(p. 155\)](#)) it provides greater control, such as letting you pause and resume multipart uploads, vary part sizes during the upload, or to begin uploads when you do not know the size of the data in advance. If you do not have these requirements, use the high-level API for uploading objects.

High-Level API

For uploading objects, the SDK provides a higher level of abstraction by providing the `TransferUtility` class. The high-level API is a simpler API, where in just a few lines of code you can upload files and streams to Amazon S3. You should use this API to upload data unless you need to control the upload as described in the preceding Low-Level API section.

For smaller data size the `TransferUtility` API uploads data in a single operation. However, the `TransferUtility` switches to using the multipart upload API when data size reaches certain threshold. By default, it uses multiple threads to concurrently upload the parts. If a part upload fails, the API retries the failed part upload up to three times. However, these are configurable options.

Note

When using a stream for the source of data, the `TransferUtility` class will not do concurrent uploads.

The .NET API Organization

When writing Amazon S3 applications using the AWS SDK for .NET, you use the AWSSDK.dll. The following namespaces in this assembly provide the multipart upload API:

- **Amazon.S3.Transfer**—Provides the high-level API to upload your data in parts. It includes the TransferUtility class that enables you to specify a file, directory, or stream for uploading your data. It also includes the TransferUtilityUploadRequest and TransferUtilityUploadDirectoryRequest classes to configure advanced settings such as the number of concurrent threads, part size, object metadata, the storage class (STANDARD, REDUCED_REDUNDANCY) and object ACL.
- **Amazon.S3**—Provides the implementation for the low-level APIs. It provides methods that correspond to the Amazon S3 REST multipart upload API (see [Using the REST API for Multipart Upload \(p. 190\)](#)).
- **Amazon.S3.Model**—Provides the low-level API classes to create requests and process responses. For example, it provides the InitiateMultipartUploadRequest and InitiateMultipartUploadResponse classes you can use when initiating a multipart upload, and the UploadPartRequest and UploadPartResponse classes when uploading parts.

For more information about the AWS SDK for .NET API, go to [AWS SDK for .NET Reference](#).

Testing the .NET Code Examples

The easiest way to get started with the .NET code examples is to install the AWS SDK for .NET. For more information, go to [AWS SDK for .NET](#).

Note

The examples in this guide are AWS SDK for .NET version 2.0 compliant.

The following tasks guide you through creating and testing the C# code samples provided in this section.

General Process of Creating .NET Code Examples

1	Create a credentials profile for your AWS credentials as described in the AWS SDK for .NET topic Configuring AWS Credentials .
2	Create a new Visual Studio project using the <i>AWS Empty Project</i> template.
3	Replace the code in the project file, Program.cs, with the code in the section you are reading.
4	Run the code. Verify that the object is created using the AWS Management Console. For more information about AWS Management Console, go to http://aws.amazon.com/console/ .

Using the AWS SDK for PHP and Running PHP Examples

The AWS SDK for PHP provides access to the API for Amazon S3 bucket and object operations. The SDK gives you the option of using the service's low-level API or using higher-level abstractions.

The SDK is available at [AWS SDK for PHP](#), which also has instructions for installing and getting started with the SDK.

Note

The setup for using the AWS SDK for PHP depends on your environment and how you want to run your application. To set up your environment to run the examples in this documentation, see the [AWS SDK for PHP Getting Started Guide](#).

AWS SDK for PHP Levels

Low-Level API

The low-level APIs correspond to the underlying Amazon S3 REST operations, including the create, update, and delete operations on buckets and objects. The low-level APIs provide greater control over these operations. For example, you can batch your requests and execute them in parallel, or when using the multipart upload API (see [Uploading Objects Using Multipart Upload API \(p. 155\)](#)), you can manage the object parts individually. Note that these low-level API calls return a result that includes all the Amazon S3 response details.

High-Level Abstractions

The high-level abstractions are intended to simplify common use cases. For example, for uploading large objects using the low-level API, you must first call `Aws\S3\S3Client::createMultipartUpload()`, then call the `Aws\S3\S3Client::uploadPart()` method to uploads object parts and then call the `Aws\S3\S3Client::completeMultipartUpload()` method to complete the upload. Instead, you could use the higher-level `Aws\S3\Model\MultipartUpload\UploadBuilder` object that simplifies creating a multipart upload.

Another example of using a higher-level abstraction is when enumerating objects in a bucket you can use the iterators feature of the AWS SDK for PHP to return all the object keys, regardless of how many objects you have stored in the bucket. If you use the low-level API the response returns only up to 1,000 keys and if you have more than a 1,000 objects in the bucket, the result will be truncated and you will have to manage the response and check for any truncation.

Running PHP Examples

The following procedure describes how to run the PHP code examples in this guide.

To Run the PHP Code Examples

1	Download and install the AWS SDK for PHP , and then verify that your environment meets the minimum requirements as described in the AWS SDK for PHP Getting Started Guide .
2	Install the AWS SDK for PHP according to the instructions in the AWS SDK for PHP Getting Started Guide . Depending on the installation method that you use, you might have to modify your code to resolve dependencies among the PHP extensions. All of the PHP code samples in this document use the Composer dependency manager that is described in the AWS SDK for PHP Getting Started Guide . Each code sample includes the following line to include its dependencies: <pre>require 'vendor/autoload.php';</pre>
3	Create a credentials profile for your AWS credentials as described in the AWS SDK for PHP topic Using the AWS credentials file and credential profiles . At run time, when you create a new Amazon S3 client object, the client will obtain your AWS credentials from the credentials profile.

4	<p>Copy the example code from the document to your project. Depending upon your environment, you might need to add lines to the code example that reference your configuration and SDK files.</p> <p>For example, to load a PHP example in a browser, add the following to the top of the PHP code, and then save it as a PHP file (extension .php) in the Web application directory (such as www or htdocs):</p> <pre style="background-color: #f0f0f0; padding: 10px;"><?php header('Content-Type: text/plain; charset=utf-8'); // Include the AWS SDK using the Composer autoloader require 'vendor/autoload.php';</pre>
5	<p>Test the example according to your setup.</p>

Related Resources

- [AWS SDK for PHP for Amazon S3](#)
- [AWS SDK for PHP Documentation](#)

Using the AWS SDK for Ruby

The AWS SDK for Ruby provides an API for Amazon S3 bucket and object operations. For object operations, you can use the API to upload objects in a single operation or upload large objects in parts (see [Uploading Objects Using Multipart Upload](#)). However, the API for a single operation upload can accept large objects as well and behind the scenes manage the upload in parts for you thereby reducing the amount of script you need to write.

The Ruby API Organization

When creating Amazon S3 applications using the AWS SDK for Ruby, you must install the SDK for Ruby gem. For more information, see the [AWS SDK for Ruby Getting Started Guide](#). Once installed, you can access the API, including the following key classes:

- **AWS::S3**—Represents the interface to Amazon S3 for the Ruby SDK.

The `S3` class provides the `#buckets` instance method for accessing existing buckets or creating new ones.

- **AWS::S3::Bucket**—Represents an Amazon S3 bucket.

The `Bucket` class provides the `#objects` instance method for accessing the objects in a bucket as well as methods to delete a bucket and return information about a bucket such as the bucket policy.

- **AWS::S3::S3Object**—Represents an Amazon S3 object identified by its key.

The `S3Object` class provides methods for getting and setting properties of an object, specifying the storage class for storing objects, and setting object permissions using access control lists. The `S3Object` class also has methods for deleting, uploading and copying objects. When uploading objects in parts, this class provides options for you to specify the order of parts uploaded and the part size.

For more information about the AWS SDK for Ruby API, go to [AWS SDK for Ruby API Reference](#).

Testing the Ruby Script Examples

The easiest way to get started with the Ruby script examples is to install the latest AWS SDK for Ruby gem. For information about installing or updating to the latest gem, go to <http://aws.amazon.com/sdkforruby/>. The following tasks guide you through the creation and testing of the Ruby script examples assuming that you have installed the AWS SDK for Ruby.

General Process of Creating and Testing Ruby Script Examples

1	<p>Create a new SDK for Ruby script and add the following lines to the top of the script.</p> <pre>#!/usr/bin/env ruby require 'rubygems' require 'aws-sdk'</pre>
	<p>The first line is the interpreter directive and the two <code>require</code> statements import two required gems into your script.</p>
3	Copy the code from the section you are reading to your script.
4	Update the code by providing any required data. For example, if uploading a file, provide the file path and the bucket name.
5	Run the script. Verify changes to buckets and objects by using the AWS Management Console. For more information about the AWS Management Console, go to http://aws.amazon.com/console/ .

Using the AWS SDK for Python (Boto)

Boto is a Python package that provides interfaces to AWS including Amazon S3. For more information about Boto, go to the [AWS SDK for Python \(Boto\)](#). To get started with the SDK for Python, go to the [Getting Started with Boto](#).

Appendices

This Amazon Simple Storage Service Developer Guide appendix include the following sections.

Topics

- [Appendix A: Using the SOAP API \(p. 551\)](#)
- [Appendix B: Authenticating Requests \(AWS Signature Version 2\) \(p. 554\)](#)

Appendix A: Using the SOAP API

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

This section contains information specific to the Amazon S3 SOAP API.

Note

SOAP requests, both authenticated and anonymous, must be sent to Amazon S3 using SSL. Amazon S3 returns an error when you send a SOAP request over HTTP.

Common SOAP API Elements

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

You can interact with Amazon S3 using SOAP 1.1 over HTTP. The Amazon S3 WSDL, which describes the Amazon S3 API in a machine-readable way, is available at: <http://doc.s3.amazonaws.com/2006-03-01/AmazonS3.wsdl>. The Amazon S3 schema is available at <http://doc.s3.amazonaws.com/2006-03-01/AmazonS3.xsd>.

Most users will interact with Amazon S3 using a SOAP toolkit tailored for their language and development environment. Different toolkits will expose the Amazon S3 API in different ways. Please refer to your specific toolkit documentation to understand how to use it. This section illustrates the Amazon S3 SOAP operations in a toolkit-independent way by exhibiting the XML requests and responses as they appear "on the wire."

Common Elements

You can include the following authorization-related elements with any SOAP request:

- *AWSAccessKeyId*: The AWS Access Key ID of the requester
- *Timestamp*: The current time on your system
- *Signature*: The signature for the request

For information about endpoints, see [Request Endpoints \(p. 13\)](#).

Authenticating SOAP Requests

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

Every non-anonymous request must contain authentication information to establish the identity of the principal making the request. In SOAP, the authentication information is put into the following elements of the SOAP request:

- Your AWS Access Key ID

Note

When making authenticated SOAP requests, temporary security credentials are not supported. For more information about types of credentials, see [Making Requests \(p. 11\)](#).

- *Timestamp*: This must be a dateTime (go to <http://www.w3.org/TR/xmlschema-2/#dateTime>) in the Coordinated Universal Time (Greenwich Mean Time) time zone, such as 2009-01-01T12:00:00.000Z. Authorization will fail if this timestamp is more than 15 minutes away from the clock on Amazon S3 servers.
- *Signature*: The RFC 2104 HMAC-SHA1 digest (go to <http://www.ietf.org/rfc/rfc2104.txt>) of the concatenation of "AmazonS3" + OPERATION + Timestamp, using your AWS Secret Access Key as the key. For example, in the following CreateBucket sample request, the signature element would contain the HMAC-SHA1 digest of the value "AmazonS3CreateBucket2009-01-01T12:00:00.000Z":

For example, in the following CreateBucket sample request, the signature element would contain the HMAC-SHA1 digest of the value "AmazonS3CreateBucket2009-01-01T12:00:00.000Z":

Example

```
<CreateBucket xmlns="http://doc.s3.amazonaws.com/2006-03-01">
  <Bucket>quotes</Bucket>
  <Acl>private</Acl>
  <AWSAccessKeyId>AKIAIOSFODNN7EXAMPLE</AWSAccessKeyId>
  <Timestamp>2009-01-01T12:00:00.000Z</Timestamp>
  <Signature>Iuyz3d3P0aTou39dzqbqaEXAMPLE=</Signature>
</CreateBucket>
```

Note

SOAP requests, both authenticated and anonymous, must be sent to Amazon S3 using SSL. Amazon S3 returns an error when you send a SOAP request over HTTP.

Important

Due to different interpretations regarding how extra time precision should be dropped, .NET users should take care not to send Amazon S3 overly specific time stamps. This can be accomplished by manually constructing `DateTime` objects with only millisecond precision.

Setting Access Policy with SOAP

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

Access control can be set at the time a bucket or object is written by including the "AccessControlList" element with the request to `CreateBucket`, `PutObjectInline`, or `PutObject`. The `AccessControlList` element is described in [Managing Access Permissions to Your Amazon S3 Resources \(p. 269\)](#). If no access control list is specified with these operations, the resource is created with a default access policy that gives the requester `FULL_CONTROL` access (this is the case even if the request is a `PutObjectInline` or `PutObject` request for an object that already exists).

Following is a request that writes data to an object, makes the object readable by anonymous principals, and gives the specified user `FULL_CONTROL` rights to the bucket (Most developers will want to give themselves `FULL_CONTROL` access to their own bucket).

Example

Following is a request that writes data to an object and makes the object readable by anonymous principals.

Sample Request

```
<PutObjectInline xmlns="http://doc.s3.amazonaws.com/2006-03-01">
  <Bucket>quotes</Bucket>
  <Key>Nelson</Key>
  <Metadata>
    <Name>Content-Type</Name>
    <Value>text/plain</Value>
  </Metadata>
  <Data>aGetaGE=</Data>
  <ContentLength>5</ContentLength>
  <AccessControlList>
    <Grant>
      <Grantee xsi:type="CanonicalUser">
        <ID>75cc57f09aa0c8caeab4f8c24e99d10f8e7faebf76c078efc7c6caea54ba06a</ID>

        <DisplayName>chriscustomer</DisplayName>
      </Grantee>
      <Permission>FULL_CONTROL</Permission>
    </Grant>
    <Grant>
      <Grantee xsi:type="Group">
        <URI>http://acs.amazonaws.com/groups/global/AllUsers<URI>
      </Grantee>
      <Permission>READ</Permission>
    </Grant>
  </AccessControlList>
  <AWSAccessKeyId>AKIAIOSFODNN7EXAMPLE</AWSAccessKeyId>
  <Timestamp>2009-03-01T12:00:00.183Z</Timestamp>
  <Signature>Iuyz3d3P0aTou39dzbqaEXAMPLE=</Signature>
</PutObjectInline>
```

Sample Response

```
<PutObjectInlineResponse xmlns="http://s3.amazonaws.com/doc/2006-03-01">
  <PutObjectInlineResponse>
    <ETag>"828ef3fdfa96f00ad9f27c383fc9ac7f"</ETag>
    <LastModified>2009-01-01T12:00:00.000Z</LastModified>
  </PutObjectInlineResponse>
</PutObjectInlineResponse>
```

The access control policy can be read or set for an existing bucket or object using the `GetBucketAccessControlPolicy`, `GetObjectAccessControlPolicy`, `SetBucketAccessControlPolicy`, and `SetObjectAccessControlPolicy` methods. For more information, see the detailed explanation of these methods.

Appendix B: Authenticating Requests (AWS Signature Version 2)

Topics

- [Authenticating Requests Using the REST API \(p. 556\)](#)
- [Signing and Authenticating REST Requests \(p. 557\)](#)
- [Browser-Based Uploads Using POST \(AWS Signature Version 2\) \(p. 568\)](#)

Note

This topic explains authenticating requests using Signature Version 2. Amazon S3 now supports the latest Signature Version 4, which is supported in all regions; it is the only version supported for new AWS regions. For more information, go to [Authenticating Requests \(AWS Signature Version 4\)](#) in the *Amazon Simple Storage Service API Reference*.

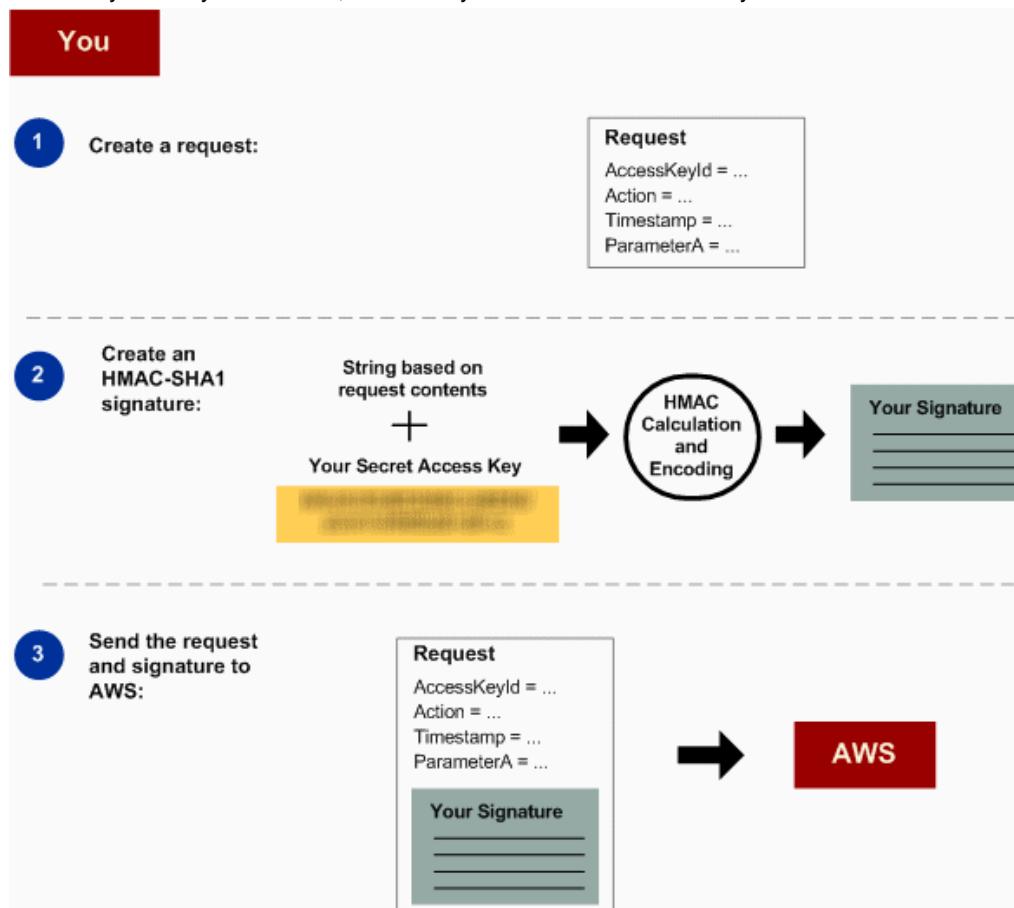
Authenticating Requests Using the REST API

When accessing Amazon S3 using REST, you must provide the following items in your request so the request can be authenticated:

Request Elements

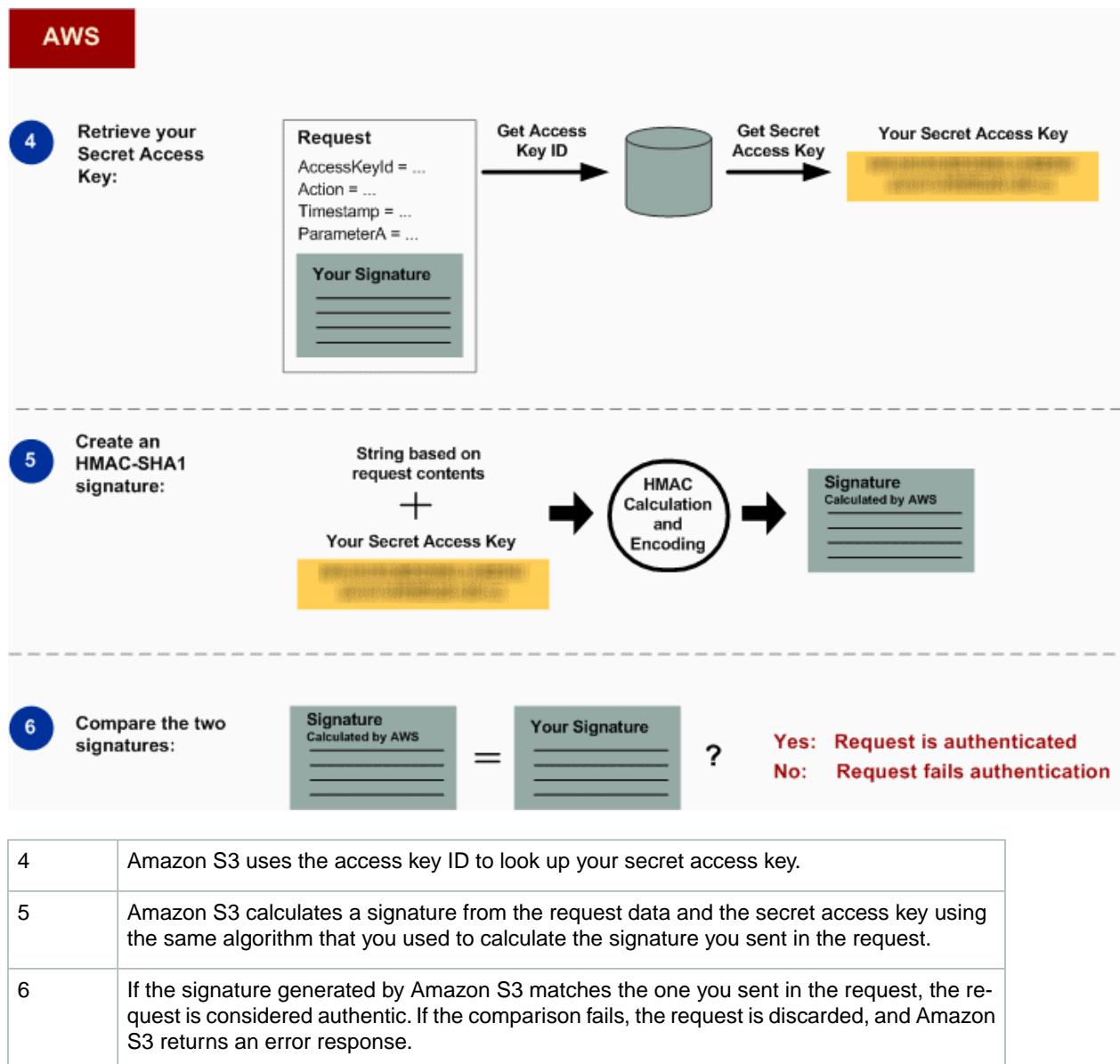
- **AWS Access Key Id** – Each request must contain the access key ID of the identity you are using to send your request.
- **Signature** – Each request must contain a valid request signature, or the request is rejected. A request signature is calculated using your secret access key, which is a shared secret known only to you and AWS.
- **Time stamp** – Each request must contain the date and time the request was created, represented as a string in UTC.
- **Date** – Each request must contain the time stamp of the request.
Depending on the API action you're using, you can provide an expiration date and time for the request instead of or in addition to the time stamp. See the authentication topic for the particular action to determine what it requires.

Following are the general steps for authenticating requests to Amazon S3. It is assumed you have the necessary security credentials, access key ID and secret access key.



1 Construct a request to AWS.

2	Calculate the signature using your secret access key.
3	Send the request to Amazon S3. Include your access key ID and the signature in your request. Amazon S3 performs the next three steps.



Detailed Authentication Information

For detailed information about REST authentication, see [Signing and Authenticating REST Requests \(p. 557\)](#).

Signing and Authenticating REST Requests

Topics

- [Using Temporary Security Credentials \(p. 559\)](#)
- [The Authentication Header \(p. 559\)](#)
- [Request Canonicalization for Signing \(p. 560\)](#)
- [Constructing the CanonicalizedResource Element \(p. 560\)](#)
- [Constructing the CanonicalizedAmzHeaders Element \(p. 561\)](#)
- [Positional versus Named HTTP Header StringToSign Elements \(p. 561\)](#)
- [Time Stamp Requirement \(p. 561\)](#)
- [Authentication Examples \(p. 562\)](#)
- [REST Request Signing Problems \(p. 566\)](#)
- [Query String Request Authentication Alternative \(p. 566\)](#)

Note

This topic explains authenticating requests using Signature Version 2. Amazon S3 now supports the latest Signature Version 4. This latest signature version is supported in all regions and any new regions after January 30, 2014 will support only Signature Version 4. For more information, go to [Authenticating Requests \(AWS Signature Version 4\)](#) in the *Amazon Simple Storage Service API Reference*.

Authentication is the process of proving your identity to the system. Identity is an important factor in Amazon S3 access control decisions. Requests are allowed or denied in part based on the identity of the requester. For example, the right to create buckets is reserved for registered developers and (by default) the right to create objects in a bucket is reserved for the owner of the bucket in question. As a developer, you'll be making requests that invoke these privileges, so you'll need to prove your identity to the system by authenticating your requests. This section shows you how.

Note

The content in this section does not apply to HTTP POST. For more information, see [Browser-Based Uploads Using POST \(AWS Signature Version 2\) \(p. 568\)](#).

The Amazon S3 REST API uses a custom HTTP scheme based on a keyed-HMAC (Hash Message Authentication Code) for authentication. To authenticate a request, you first concatenate selected elements of the request to form a string. You then use your AWS secret access key to calculate the HMAC of that string. Informally, we call this process "signing the request," and we call the output of the HMAC algorithm the signature, because it simulates the security properties of a real signature. Finally, you add this signature as a parameter of the request by using the syntax described in this section.

When the system receives an authenticated request, it fetches the AWS secret access key that you claim to have and uses it in the same way to compute a signature for the message it received. It then compares the signature it calculated against the signature presented by the requester. If the two signatures match, the system concludes that the requester must have access to the AWS secret access key and therefore acts with the authority of the principal to whom the key was issued. If the two signatures do not match, the request is dropped and the system responds with an error message.

Example Authenticated Amazon S3 REST Request

```
GET /photos/puppy.jpg HTTP/1.1
Host: johnsmith.s3.amazonaws.com
Date: Mon, 26 Mar 2007 19:37:58 +0000

Authorization: AWS AKIAIOSFODNN7EXAMPLE:frJIUN8DYpKDtOLCwo//y1lqDzg=
```

Using Temporary Security Credentials

If you are signing your request using temporary security credentials (see [Making Requests \(p. 11\)](#)), you must include the corresponding security token in your request by adding the `x-amz-security-token` header.

When you obtain temporary security credentials using the AWS Security Token Service API, the response includes temporary security credentials and a session token. You provide the session token value in the `x-amz-security-token` header when you send requests to Amazon S3. For information about the AWS Security Token Service API provided by IAM, go to [Action](#) in the *AWS Security Token Service API Reference Guide*.

The Authentication Header

The Amazon S3 REST API uses the standard HTTP Authorization header to pass authentication information. (The name of the standard header is unfortunate because it carries authentication information, not authorization.) Under the Amazon S3 authentication scheme, the Authorization header has the following form:

```
Authorization: AWS AWSAccessKeyId:Signature
```

Developers are issued an AWS access key ID and AWS secret access key when they register. For request authentication, the `AWSAccessKeyId` element identifies the access key ID that was used to compute the signature and, indirectly, the developer making the request.

The `Signature` element is the RFC 2104 HMAC-SHA1 of selected elements from the request, and so the `Signature` part of the Authorization header will vary from request to request. If the request signature calculated by the system matches the `Signature` included with the request, the requester will have demonstrated possession of the AWS secret access key. The request will then be processed under the identity, and with the authority, of the developer to whom the key was issued.

Following is pseudogrammar that illustrates the construction of the `Authorization` request header. (In the example, `\n` means the Unicode code point U+000A, commonly called newline).

```
Authorization = "AWS" + " " + AWSAccessKeyId + ":" + Signature;

Signature = Base64( HMAC-SHA1( YourSecretAccessKeyID, UTF-8-Encoding-Of(
StringToSign ) ) );

StringToSign = HTTP-Verb + "\n" +
Content-MD5 + "\n" +
Content-Type + "\n" +
Date + "\n" +
CanonicalizedAmzHeaders +
CanonicalizedResource;

CanonicalizedResource = [ "/" + Bucket ] +
<HTTP-Request-URI, from the protocol name up to the query string> +
[ subresource, if present. For example "?acl", "?location", "?logging", or
"?torrent" ];

CanonicalizedAmzHeaders = <described below>
```

HMAC-SHA1 is an algorithm defined by [RFC 2104 - Keyed-Hashing for Message Authentication](#). The algorithm takes as input two byte-strings, a key and a message. For Amazon S3 request authentication,

use your AWS secret access key (`YourSecretAccessKeyID`) as the key, and the UTF-8 encoding of the `StringToSign` as the message. The output of HMAC-SHA1 is also a byte string, called the digest. The `Signature` request parameter is constructed by Base64 encoding this digest.

Request Canonicalization for Signing

Recall that when the system receives an authenticated request, it compares the computed request signature with the signature provided in the request in `StringToSign`. For that reason, you must compute the signature by using the same method used by Amazon S3. We call the process of putting a request in an agreed-upon form for signing *canonicalization*.

Constructing the CanonicalizedResource Element

`CanonicalizedResource` represents the Amazon S3 resource targeted by the request. Construct it for a REST request as follows:

Launch Process

1	Start with an empty string ("").
2	<p>If the request specifies a bucket using the HTTP Host header (virtual hosted-style), append the bucket name preceded by a "/" (e.g., "/bucketname"). For path-style requests and requests that don't address a bucket, do nothing. For more information about virtual hosted-style requests, see Virtual Hosting of Buckets (p. 48).</p> <p>For a virtual hosted-style request "https://johnsmith.s3.amazonaws.com/photos/puppy.jpg", the <code>CanonicalizedResource</code> is "/johnsmith".</p> <p>For the path-style request, "https://s3.amazonaws.com/johnsmith/photos/puppy.jpg", the <code>CanonicalizedResource</code> is "".</p>
3	<p>Append the path part of the un-decoded HTTP Request-URI, up-to but not including the query string.</p> <p>For a virtual hosted-style request "https://johnsmith.s3.amazonaws.com/photos/puppy.jpg", the <code>CanonicalizedResource</code> is "/johnsmith/photos/puppy.jpg".</p> <p>For a path-style request, "https://s3.amazonaws.com/johnsmith/photos/puppy.jpg", the <code>CanonicalizedResource</code> is "/johnsmith/photos/puppy.jpg". At this point, the <code>CanonicalizedResource</code> is the same for both the virtual hosted-style and path-style request.</p> <p>For a request that does not address a bucket, such as GET Service, append "/".</p>
4	<p>If the request addresses a subresource, such as <code>?versioning</code>, <code>?location</code>, <code>?acl</code>, <code>?torrent</code>, <code>?lifecycle</code>, or <code>?versionid</code>, append the subresource, its value if it has one, and the question mark. Note that in case of multiple subresources, subresources must be lexicographically sorted by subresource name and separated by '&', e.g., <code>?acl&versionId=value</code>.</p> <p>The subresources that must be included when constructing the <code>CanonicalizedResource</code> Element are <code>acl</code>, <code>lifecycle</code>, <code>location</code>, <code>logging</code>, <code>notification</code>, <code>partNumber</code>, <code>policy</code>, <code>requestPayment</code>, <code>torrent</code>, <code>uploadId</code>, <code>uploads</code>, <code>versionId</code>, <code>versioning</code>, <code>versions</code>, and <code>website</code>.</p> <p>If the request specifies query string parameters overriding the response header values (see Get Object), append the query string parameters and their values. When signing, you do not encode these values; however, when making the request, you must encode these parameter values. The query string parameters in a GET request include <code>response-content-type</code>, <code>response-content-language</code>, <code>response-expires</code>, <code>response-cache-control</code>, <code>response-content-disposition</code>, and <code>response-content-encoding</code>.</p> <p>The <code>delete</code> query string parameter must be included when you create the <code>CanonicalizedResource</code> for a multi-object Delete request.</p>

Elements of the CanonicalizedResource that come from the HTTP Request-URI should be signed literally as they appear in the HTTP request, including URL-Encoding meta characters.

The *CanonicalizedResource* might be different than the HTTP Request-URI. In particular, if your request uses the HTTP Host header to specify a bucket, the bucket does not appear in the HTTP Request-URI. However, the *CanonicalizedResource* continues to include the bucket. Query string parameters might also appear in the Request-URI but are not included in *CanonicalizedResource*. For more information, see [Virtual Hosting of Buckets \(p. 48\)](#).

Constructing the CanonicalizedAmzHeaders Element

To construct the CanonicalizedAmzHeaders part of *StringToSign*, select all HTTP request headers that start with 'x-amz-' (using a case-insensitive comparison), and use the following process.

CanonicalizedAmzHeaders Process

1	Convert each HTTP header name to lowercase. For example, 'X-Amz-Date' becomes 'x-amz-date'.
2	Sort the collection of headers lexicographically by header name.
3	Combine header fields with the same name into one "header-name:comma-separated-value-list" pair as prescribed by RFC 2616, section 4.2, without any whitespace between values. For example, the two metadata headers 'x-amz-meta-username: fred' and 'x-amz-meta-username: barney' would be combined into the single header 'x-amz-meta-username: fred,barney'.
4	"Unfold" long headers that span multiple lines (as allowed by RFC 2616, section 4.2) by replacing the folding whitespace (including new-line) by a single space.
5	Trim any whitespace around the colon in the header. For example, the header 'x-amz-meta-username: fred,barney' would become 'x-amz-meta-username:fred,barney'
6	Finally, append a newline character (U+000A) to each canonicalized header in the resulting list. Construct the CanonicalizedResource element by concatenating all headers in this list into a single string.

Positional versus Named HTTP Header StringToSign Elements

The first few header elements of *StringToSign* (Content-Type, Date, and Content-MD5) are positional in nature. *StringToSign* does not include the names of these headers, only their values from the request. In contrast, the 'x-amz-' elements are named. Both the header names and the header values appear in *StringToSign*.

If a positional header called for in the definition of *StringToSign* is not present in your request (for example, Content-Type or Content-MD5 are optional for PUT requests and meaningless for GET requests), substitute the empty string ("") for that position.

Time Stamp Requirement

A valid time stamp (using either the HTTP Date header or an x-amz-date alternative) is mandatory for authenticated requests. Furthermore, the client timestamp included with an authenticated request must be within 15 minutes of the Amazon S3 system time when the request is received. If not, the request will fail with the *RequestTimeTooSkewed* error code. The intention of these restrictions is to limit the possibility that intercepted requests could be replayed by an adversary. For stronger protection against eavesdropping, use the HTTPS transport for authenticated requests.

Note

The validation constraint on request date applies only to authenticated requests that do not use query string authentication. For more information, see [Query String Request Authentication Alternative \(p. 566\)](#).

Some HTTP client libraries do not expose the ability to set the `Date` header for a request. If you have trouble including the value of the 'Date' header in the canonicalized headers, you can set the timestamp for the request by using an '`x-amz-date`' header instead. The value of the `x-amz-date` header must be in one of the RFC 2616 formats (<http://www.ietf.org/rfc/rfc2616.txt>). When an `x-amz-date` header is present in a request, the system will ignore any `Date` header when computing the request signature. Therefore, if you include the `x-amz-date` header, use the empty string for the `Date` when constructing the `StringToSign`. See the next section for an example.

Authentication Examples

The examples in this section use the (non-working) credentials in the following table.

Parameter	Value
AWSAccessKeyId	AKIAIOSFODNN7EXAMPLE
AWSSecretAccessKey	wJalrXUtnFEMI/K7MDENG/bPxRficyEXAMPLEKEY

In the example `StringToSigns`, formatting is not significant, and `\n` means the Unicode code point `U+000A`, commonly called newline. Also, the examples use "`+0000`" to designate the time zone. You can use "GMT" to designate timezone instead, but the signatures shown in the examples will be different.

Example Object GET

This example gets an object from the `johnsmith` bucket.

Request	StringToSign
<pre>GET /photos/puppy.jpg HTTP/1.1 Host: johnsmith.s3.amazonaws.com Date: Tue, 27 Mar 2007 19:36:42 +0000 Authorization: AWS AKIAIOSFODNN7EX AMPLE: bWq2s1WEIj+Ydj0vQ697zp+IXMU=</pre>	<pre>GET\n \n \n Tue, 27 Mar 2007 19:36:42 +0000\n /johnsmith/photos/puppy.jpg</pre>

Note that the CanonicalizedResource includes the bucket name, but the HTTP Request-URI does not. (The bucket is specified by the Host header.)

Example Object PUT

This example puts an object into the johnsmith bucket.

Request	StringToSign
<pre>PUT /photos/puppy.jpg HTTP/1.1 Content-Type: image/jpeg Content-Length: 94328 Host: johnsmith.s3.amazonaws.com Date: Tue, 27 Mar 2007 21:15:45 +0000 <i>Authorization: AWS AKIAIOSFODNN7EX AMPLE: MyyxerY7whkBe+bq8fHCL/2kKUg=</i></pre>	<pre>PUT\n \n image/jpeg\n Tue, 27 Mar 2007 21:15:45 +0000\n /johnsmith/photos/puppy.jpg</pre>

Note the Content-Type header in the request and in the StringToSign. Also note that the Content-MD5 is left blank in the StringToSign, because it is not present in the request.

Example List

This example lists the content of the johnsmith bucket.

Request	StringToSign
<pre>GET /?prefix=photos&max-keys=50&marker=puppy HTTP/1.1 User-Agent: Mozilla/5.0 Host: johnsmith.s3.amazonaws.com Date: Tue, 27 Mar 2007 19:42:41 +0000 <i>Authorization: AWS AKIAIOSFODNN7EXAMPLE: htDYFYduRNen8P9ZfE/s9SuKy0U=</i></pre>	<pre>GET\n \n \n Tue, 27 Mar 2007 19:42:41 +0000\n /johnsmith/</pre>

Note the trailing slash on the CanonicalizedResource and the absence of query string parameters.

Example Fetch

This example fetches the access control policy subresource for the 'johnsmith' bucket.

Request	StringToSign
<pre>GET /?acl HTTP/1.1 Host: johnsmith.s3.amazonaws.com Date: Tue, 27 Mar 2007 19:44:46 +0000 <i>Authorization: AWS AKIAIOSFODNN7EXAMPLE: c2WLPFtWHVgbEmeEG93a4cG37dM=</i></pre>	<pre>GET\n \n \n Tue, 27 Mar 2007 19:44:46 +0000\n /johnsmith/?acl</pre>

Notice how the subresource query string parameter is included in the CanonicalizedResource.

Example Delete

This example deletes an object from the 'johnsmith' bucket using the path-style and Date alternative.

Request	StringToSign
<pre>DELETE /johnsmith/photos/puppy.jpg HT TP/1.1 User-Agent: dotnet Host: s3.amazonaws.com Date: Tue, 27 Mar 2007 21:20:27 +0000 x-amz-date: Tue, 27 Mar 2007 21:20:26 +0000 <i>Authorization: AWS AKIAIOSFODNN7EX AMPLE:lx3byBScXR6KzyMaifNkardMwNk=</i></pre>	<pre>DELETE\n \n \n Tue, 27 Mar 2007 21:20:26 +0000\n /johnsmith/photos/puppy.jpg</pre>

Note how we used the alternate 'x-amz-date' method of specifying the date (because our client library prevented us from setting the date, say). In this case, the x-amz-date takes precedence over the Date header. Therefore, date entry in the signature must contain the value of the x-amz-date header.

Example Upload

This example uploads an object to a CNAME style virtual hosted bucket with metadata.

Request	StringToSign
<pre>PUT /db-backup.dat.gz HTTP/1.1 User-Agent: curl/7.15.5 Host: static.johnsmith.net:8080 Date: Tue, 27 Mar 2007 21:06:08 +0000 x-amz-acl: public-read content-type: application/x-download Content-MD5: 4gJE4saaMU4BqNR0kLY+lw== X-Amz-Meta-ReviewedBy: joe@johnsmith.net X-Amz-Meta-ReviewedBy: jane@johnsmith.net X-Amz-Meta-FileChecksum: 0x02661779 X-Amz-Meta-ChecksumAlgorithm: crc32 Content-Disposition: attachment; file name=database.dat Content-Encoding: gzip Content-Length: 5913339 <i>Authorization: AWS AKIAIOSFODNN7EXAMPLE: iIlyl83RwaSoYIEdixDQcA4OnAnc=</i></pre>	<pre>PUT\n 4gJE4saaMU4BqNR0kLY+lw==\n application/x-download\n Tue, 27 Mar 2007 21:06:08 +0000\n x-amz-acl:public-read\n x-amz-meta-checksumalgorithm:crc32\n x-amz-meta-filechecksum:0x02661779\n x-amz-meta-reviewedby: joe@johnsmith.net,jane@johns mith.net\n /static.johnsmith.net/db- backup.dat.gz</pre>

Notice how the 'x-amz-' headers are sorted, trimmed of whitespace, and converted to lowercase. Note also that multiple headers with the same name have been joined using commas to separate values.

Note how only the Content-Type and Content-MD5 HTTP entity headers appear in the *StringToSign*. The other Content-* entity headers do not.

Again, note that the *CanonicalizedResource* includes the bucket name, but the HTTP Request-URI does not. (The bucket is specified by the Host header.)

Example List All My Buckets

Request	StringToSign
<pre>GET / HTTP/1.1 Host: s3.amazonaws.com Date: Wed, 28 Mar 2007 01:29:59 +0000 <i>Authorization: AWS AKIAIOSFODNN7EXAMPLE:gGdzdER IC03wnaRNKh6OqZehG9s=</i></pre>	<pre>GET\n \n \n Wed, 28 Mar 2007 01:29:59 +0000\n /</pre>

Example Unicode Keys

Request	StringToSign
<pre>GET /diction ary/fran%C3%A7ais/pr%c3%a9f%c3%a8re HT TP/1.1 Host: s3.amazonaws.com Date: Wed, 28 Mar 2007 01:49:49 +0000 Authorization: AWS AKIAIOSFODNN7EX AMPLE:DNEZGsoieTZ92F3bUFSPQcbGmlM=</pre>	<pre>GET\n \n \n \n Wed, 28 Mar 2007 01:49:49 +0000\n /diction ary/fran%C3%A7ais/pr%c3%a9f%c3%a8re</pre>

Note

The elements in *StringToSign* that were derived from the Request-URI are taken literally, including URL-Encoding and capitalization.

REST Request Signing Problems

When REST request authentication fails, the system responds to the request with an XML error document. The information contained in this error document is meant to help developers diagnose the problem. In particular, the *StringToSign* element of the *SignatureDoesNotMatch* error document tells you exactly what request canonicalization the system is using.

Some toolkits silently insert headers that you do not know about beforehand, such as adding the header `Content-Type` during a PUT. In most of these cases, the value of the inserted header remains constant, allowing you to discover the missing headers by using tools such as Ethereal or tcpmon.

Query String Request Authentication Alternative

You can authenticate certain types of requests by passing the required information as query-string parameters instead of using the `Authorization` HTTP header. This is useful for enabling direct third-party browser access to your private Amazon S3 data without proxying the request. The idea is to construct a "pre-signed" request and encode it as a URL that an end-user's browser can retrieve. Additionally, you can limit a pre-signed request by specifying an expiration time.

Note

For examples of using the AWS SDKs to generating pre-signed URLs, see [Share an Object with Others \(p. 139\)](#).

Creating a Signature

Following is an example query string authenticated Amazon S3 REST request.

```
GET /photos/puppy.jpg
?AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE&Expires=1141889120&Signature=vjbyPxybdZaN
mGa%2ByT272YEAiv4%3D HTTP/1.1
Host: johnsmith.s3.amazonaws.com
Date: Mon, 26 Mar 2007 19:37:58 +0000
```

The query string request authentication method doesn't require any special HTTP headers. Instead, the required authentication elements are specified as query string parameters:

Query String Parameter Name	Example Value	Description
<i>AWSAccessKeyId</i>	AKIAIOSFODNN7EXAMPLE	Your AWS access key ID. Specifies the AWS secret access key used to sign the request and, indirectly, the identity of the developer making the request.
<i>Expires</i>	1141889120	The time when the signature expires, specified as the number of seconds since the epoch (00:00:00 UTC on January 1, 1970). A request received after this time (according to the server) will be rejected.
<i>Signature</i>	vjbyPxybdZaN-mGa%2ByT272YEAiv4%3D	The URL encoding of the Base64 encoding of the HMAC-SHA1 of <i>StringToSign</i> .

The query string request authentication method differs slightly from the ordinary method but only in the format of the *Signature* request parameter and the *StringToSign* element. Following is pseudo-grammar that illustrates the query string request authentication method.

```

Signature = URL-Encode( Base64( HMAC-SHA1( YourSecretAccessKeyID, UTF-8-Encoding-
Of( StringToSign ) ) ) );

StringToSign = HTTP-VERB + "\n" +
    Content-MD5 + "\n" +
    Content-Type + "\n" +
    Expires + "\n" +
    CanonicalizedAmzHeaders +
    CanonicalizedResource;

```

YourSecretAccessKeyID is the AWS secret access key ID that Amazon assigns to you when you sign up to be an Amazon Web Service developer. Notice how the *Signature* is URL-Encoded to make it suitable for placement in the query string. Note also that in *StringToSign*, the *HTTP Date* positional element has been replaced with *Expires*. The *CanonicalizedAmzHeaders* and *CanonicalizedResource* are the same.

Note

In the query string authentication method, you do not use the *Date* or the *x-amz-date* request header when calculating the string to sign.

Example Query String Request Authentication

Request	StringToSign
<pre>GET /photos/puppy.jpg?AWSAccessKey Id=AKIAIOSFODNN7EXAMPLE& Signature=NpgCjnDzrM%2BWFzoENXmpN DUsSn8%3D& Expires=1175139620 HTTP/1.1 Host: johnsmith.s3.amazonaws.com</pre>	<pre>GET\n \n \n 1175139620\n /johnsmith/photos/puppy.jpg</pre>

We assume that when a browser makes the GET request, it won't provide a Content-MD5 or a Content-Type header, nor will it set any x-amz- headers, so those parts of the *StringToSign* are left blank.

Using Base64 Encoding

HMAC request signatures must be Base64 encoded. Base64 encoding converts the signature into a simple ASCII string that can be attached to the request. Characters that could appear in the signature string like plus (+), forward slash (/), and equals (=) must be encoded if used in a URL. For example, if the authentication code includes a plus (+) sign, encode it as %2B in the request. Encode a forward slash as %2F and equals as %3D.

For examples of Base64 encoding, refer to the Amazon S3 [Authentication Examples \(p. 562\)](#).

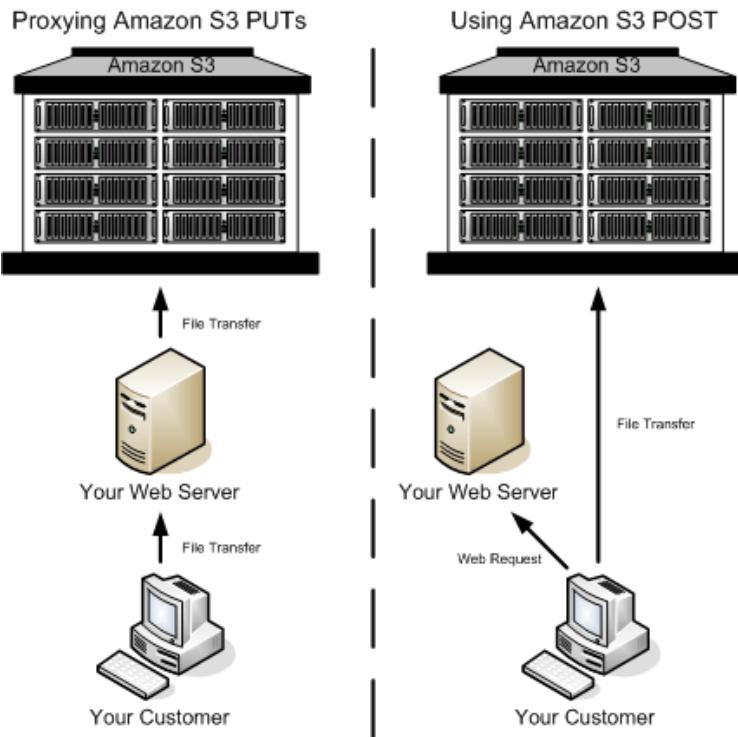
Browser-Based Uploads Using POST (AWS Signature Version 2)

Amazon S3 supports POST, which allows your users to upload content directly to Amazon S3. POST is designed to simplify uploads, reduce upload latency, and save you money on applications where users upload data to store in Amazon S3.

Note

The request authentication discussed in this section is based on AWS Signature Version 2, a protocol for authenticating inbound API requests to AWS services. Amazon S3 also supports Signature Version 4, in all AWS regions. If you are using Signature Version 4, go to [Authenticating Requests in Browser-Based Uploads Using POST \(AWS Signature Version 4\)](#) in the Amazon Simple Storage Service API Reference.

The following figure shows an upload using Amazon S3 POST.



Uploading Using POST

1	The user opens a web browser and accesses your web page.
2	Your web page contains an HTTP form that contains all the information necessary for the user to upload content to Amazon S3.
3	The user uploads content directly to Amazon S3.

Note

Query string authentication is not supported for POST.

HTML Forms (AWS Signature Version 2)

Topics

- [HTML Form Encoding \(p. 570\)](#)
- [HTML Form Declaration \(p. 570\)](#)
- [HTML Form Fields \(p. 571\)](#)
- [Policy Construction \(p. 573\)](#)
- [Constructing a Signature \(p. 576\)](#)
- [Redirection \(p. 576\)](#)

When you communicate with Amazon S3, you normally use the REST or SOAP API to perform put, get, delete, and other operations. With POST, users upload data directly to Amazon S3 through their browsers, which cannot process the SOAP API or create a REST `PUT` request.

Note

SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.

To allow users to upload content to Amazon S3 by using their browsers, you use HTML forms. HTML forms consist of a form declaration and form fields. The form declaration contains high-level information about the request. The form fields contain detailed information about the request, as well as the policy that is used to authenticate it and ensure that it meets the conditions that you specify.

Note

The form data and boundaries (excluding the contents of the file) cannot exceed 20 KB.

This section explains how to use HTML forms.

HTML Form Encoding

The form and policy must be UTF-8 encoded. You can apply UTF-8 encoding to the form by specifying it in the HTML heading or as a request header.

Note

The HTML form declaration does not accept query string authentication parameters.

The following is an example of UTF-8 encoding in the HTML heading:

```
<html>
  <head>
    ...
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    ...
  </head>
  <body>
```

The following is an example of UTF-8 encoding in a request header:

```
Content-Type: text/html; charset=UTF-8
```

HTML Form Declaration

The form declaration has three components: the action, the method, and the enclosure type. If any of these values is improperly set, the request fails.

The action specifies the URL that processes the request, which must be set to the URL of the bucket. For example, if the name of your bucket is "johnsmith", the URL is "http://johnsmith.s3.amazonaws.com/".

Note

The key name is specified in a form field.

The method must be POST.

The enclosure type (enctype) must be specified and must be set to multipart/form-data for both file uploads and text area uploads. For more information, go to [RFC 1867](#).

Example

The following example is a form declaration for the bucket "johnsmith".

```
<form action="http://johnsmith.s3.amazonaws.com/" method="post"
enctype="multipart/form-data">
```

HTML Form Fields

The following table describes fields that can be used within an HTML form.

Note

The variable \${filename} is automatically replaced with the name of the file provided by the user and is recognized by all form fields. If the browser or client provides a full or partial path to the file, only the text following the last slash (/) or backslash (\) will be used. For example, "C:\Program Files\directory1\file.txt" will be interpreted as "file.txt". If no file or file name is provided, the variable is replaced with an empty string.

Field Name	Description	Required
<i>AWSAccessKeyId</i>	The AWS Access Key ID of the owner of the bucket who grants an anonymous user access for a request that satisfies the set of constraints in the policy. This field is required if the request includes a policy document.	Conditional
<i>acl</i>	An Amazon S3 access control list (ACL). If an invalid access control list is specified, an error is generated. For more information on ACLs, see Amazon S3 ACLs (p. 8) . Type: String Default: private Valid Values: private public-read public-read-write authenticated-read bucket-owner-read bucket-owner-full-control	No
<i>Cache-Control</i> , <i>Content-Type</i> , <i>Content-Disposition</i> , <i>Content-Encoding</i> , <i>Expires</i>	REST-specific headers. For more information, see PUT Object .	No
<i>key</i>	The name of the uploaded key. To use the filename provided by the user, use the \${filename} variable. For example, if user Betty uploads the file lolcatz.jpg and you specify /user/betty/\${filename}, the file is stored as /user/betty/lolcatz.jpg. For more information, see Object Key and Metadata (p. 79) .	Yes

Field Name	Description	Required
<i>policy</i>	Security policy describing what is permitted in the request. Requests without a security policy are considered anonymous and will succeed only on publicly writable buckets.	No
<i>success_action_redirect</i> , <i>redirect</i>	<p>The URL to which the client is redirected upon successful upload. Amazon S3 appends the bucket, key, and etag values as query string parameters to the URL.</p> <p>If <i>success_action_redirect</i> is not specified, Amazon S3 returns the empty document type specified in the <i>success_action_status</i> field.</p> <p>If Amazon S3 cannot interpret the URL, it ignores the field.</p> <p>If the upload fails, Amazon S3 displays an error and does not redirect the user to a URL.</p> <p>For more information, see Redirection (p. 576).</p> <p>Note The redirect field name is deprecated and support for the redirect field name will be removed in the future.</p>	No
<i>success_action_status</i>	<p>The status code returned to the client upon successful upload if <i>success_action_redirect</i> is not specified.</p> <p>Valid values are 200, 201, or 204 (default).</p> <p>If the value is set to 200 or 204, Amazon S3 returns an empty document with a 200 or 204 status code.</p> <p>If the value is set to 201, Amazon S3 returns an XML document with a 201 status code. For information about the content of the XML document, see POST Object.</p> <p>If the value is not set or if it is set to an invalid value, Amazon S3 returns an empty document with a 204 status code.</p> <p>Note Some versions of the Adobe Flash player do not properly handle HTTP responses with an empty body. To support uploads through Adobe Flash, we recommend setting <i>success_action_status</i> to 201.</p>	No

Field Name	Description	Required
<i>signature</i>	The HMAC signature constructed by using the secret access key that corresponds to the provided AWSAccessKeyId. This field is required if a policy document is included with the request. For more information, see Using Auth Access .	Conditional
<i>x-amz-security-token</i>	A security token used by Amazon DevPay and session credentials If the request is using Amazon DevPay then it requires two <i>x-amz-security-token</i> form fields: one for the product token and one for the user token. For more information, go to Using DevPay . If the request is using session credentials, then it requires one <i>x-amz-security-token</i> form . For more information, go to Welcome in the <i>AWS Security Token Service</i> guide.	No
Other field names prefixed with <i>x-amz-meta-</i>	User-specified metadata. Amazon S3 does not validate or use this data. For more information, see PUT Object .	No
<i>file</i>	File or text content. The file or content must be the last field in the form. Any fields below it are ignored. You cannot upload more than one file at a time.	Yes

Policy Construction

Topics

- [Expiration \(p. 574\)](#)
- [Conditions \(p. 574\)](#)
- [Condition Matching \(p. 575\)](#)
- [Character Escaping \(p. 576\)](#)

The policy is a UTF-8 and Base64-encoded JSON document that specifies conditions that the request must meet and is used to authenticate the content. Depending on how you design your policy documents, you can use them per upload, per user, for all uploads, or according to other designs that meet your needs.

Note

Although the policy document is optional, we highly recommend it over making a bucket publicly writable.

The following is an example of a policy document:

```
{
  "expiration": "2007-12-01T12:00:00.000Z",
  "conditions": [
    {"content-length-range": [0, 10485760]}
  ]
}
```

```
{ "acl": "public-read" } ,  
{ "bucket": "johnsmith" } ,  
[ "starts-with", "$key", "user/eric/" ] ,  
]  
}
```

The policy document contains the expiration and conditions.

Expiration

The expiration element specifies the expiration date of the policy in ISO 8601 UTC date format. For example, "2007-12-01T12:00:00.000Z" specifies that the policy is not valid after midnight UTC on 2007-12-01. Expiration is required in a policy.

Conditions

The conditions in the policy document validate the contents of the uploaded object. Each form field that you specify in the form (except AWSAccessKeyId, signature, file, policy, and field names that have an x-ignore- prefix) must be included in the list of conditions.

Note

If you have multiple fields with the same name, the values must be separated by commas. For example, if you have two fields named "x-amz-meta-tag" and the first one has a value of "Ninja" and second has a value of "Stallman", you would set the policy document to Ninja,Stallman. All variables within the form are expanded before the policy is validated. Therefore, all condition matching should be performed against the expanded fields. For example, if you set the key field to user/betty/\${filename}, your policy might be ["starts-with", "\$key", "user/betty/"]. Do not enter ["starts-with", "\$key", "user/betty/\${filename}"]. For more information, see [Condition Matching \(p. 575\)](#).

The following table describes policy document conditions.

Element Name	Description
acl	Specifies conditions that the ACL must meet. Supports exact matching and <i>starts-with</i> .
content-length-range	Specifies the minimum and maximum allowable size for the uploaded content. Supports range matching.
Cache-Control, Content-Type, Content-Disposition, Content-Encoding, Expires	REST-specific headers. Supports exact matching and <i>starts-with</i> .
key	The name of the uploaded key. Supports exact matching and <i>starts-with</i> .

Element Name	Description
success_action_redirect, redirect	The URL to which the client is redirected upon successful upload. Supports exact matching and <i>starts-with</i> .
success_action_status	The status code returned to the client upon successful upload if success_action_redirect is not specified. Supports exact matching.
x-amz-security-token	Amazon DevPay security token. Each request that uses Amazon DevPay requires two x-amz-security-token form fields: one for the product token and one for the user token. As a result, the values must be separated by commas. For example, if the user token is <code>eW91dHViZQ==</code> and the product token is <code>b0hnNVNKWVJIQTA=</code> , you set the policy entry to: <code>{ "x-amz-security-token": "eW91dHViZQ==,b0hnNVNKWVJIQTA=" }</code> . For more information about Amazon DevPay, see Using DevPay .
Other field names prefixed with x-amz-meta-	User-specified metadata. Supports exact matching and <i>starts-with</i> .

Note

If your toolkit adds additional fields (e.g., Flash adds filename), you must add them to the policy document. If you can control this functionality, prefix `x-ignore-` to the field so Amazon S3 ignores the feature and it won't affect future versions of this feature.

Condition Matching

The following table describes condition matching types. Although you must specify one condition for each form field that you specify in the form, you can create more complex matching criteria by specifying multiple conditions for a form field.

Condition	Description
Exact Matches	<p>Exact matches verify that fields match specific values. This example indicates that the ACL must be set to public-read:</p> <pre>{ "acl": "public-read" }</pre> <p>This example is an alternate way to indicate that the ACL must be set to public-read:</p> <pre>["eq", "\$acl", "public-read"]</pre>
Starts With	<p>If the value must start with a certain value, use starts-with. This example indicates that the key must start with user/betty:</p> <pre>["starts-with", "\$key", "user/betty/"]</pre>

Condition	Description
Matching Any Content	To configure the policy to allow any content within a field, use starts-with with an empty value. This example allows any success_action_redirect: ["starts-with", "\$success_action_redirect", ""] ["starts-with", "\$success_action_redirect", ""]
Specifying Ranges	For fields that accept ranges, separate the upper and lower ranges with a comma. This example allows a file size from 1 to 10 megabytes: ["content-length-range", 1048579, 10485760] ["content-length-range", 1048579, 10485760]

Character Escaping

The following table describes characters that must be escaped within a policy document.

Escape Sequence	Description
\\	Backslash
\\$	Dollar sign
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\uxxxx	All Unicode characters

Constructing a Signature

#	Description
1	Encode the policy by using UTF-8.
2	Encode those UTF-8 bytes by using Base64.
3	Sign the policy with your secret access key by using HMAC SHA-1.
4	Encode the SHA-1 signature by using Base64.

For general information about authentication, see [Using Auth Access](#).

Redirection

This section describes how to handle redirects.

General Redirection

On completion of the POST request, the user is redirected to the location that you specified in the `success_action_redirect` field. If Amazon S3 cannot interpret the URL, it ignores the `success_action_redirect` field.

If `success_action_redirect` is not specified, Amazon S3 returns the empty document type specified in the `success_action_status` field.

If the POST request fails, Amazon S3 displays an error and does not provide a redirect.

Pre-Upload Redirection

If your bucket was created using `<CreateBucketConfiguration>`, your end users might require a redirect. If this occurs, some browsers might handle the redirect incorrectly. This is relatively rare but is most likely to occur right after a bucket is created.

Upload Examples (AWS Signature Version 2)

Topics

- [File Upload \(p. 577\)](#)
- [Text Area Upload \(p. 580\)](#)

File Upload

This example shows the complete process for constructing a policy and form that can be used to upload a file attachment.

Policy and Form Construction

The following policy supports uploads to Amazon S3 for the `johnsmith` bucket.

```
{ "expiration": "2007-12-01T12:00:00.000Z",
  "conditions": [
    { "bucket": "johnsmith" },
    [ "starts-with", "$key", "user/eric/" ],
    { "acl": "public-read" },
    { "success_action_redirect": "http://johnsmith.s3.amazonaws.com/successful_upload.html" },
    [ "starts-with", "$Content-Type", "image/" ],
    { "x-amz-meta-uuid": "14365123651274" },
    [ "starts-with", "$x-amz-meta-tag", "" ]
  ]
}
```

This policy requires the following:

- The upload must occur before 12:00 UTC on December 1, 2007.
- The content must be uploaded to the `johnsmith` bucket.
- The key must start with `"user/eric/"`.
- The ACL is set to public-read.
- The `success_action_redirect` is set to `http://johnsmith.s3.amazonaws.com/successful_upload.html`.
- The object is an image file.
- The `x-amz-meta-uuid` tag must be set to `14365123651274`.
- The `x-amz-meta-tag` can contain any value.

The following is a Base64-encoded version of this policy.

```
eyAiZXhwaxJhdGlvbiI6IClYMDA3LTEyLTAxVDEyOjAwOjAwMfoilAogICJjb25kaXRpb25zI
jogWwogICAgeyJidWNrZXQiOiaiam9obnNtaXRoIn0sCiAgICBbInN0YXJ0cy13aXRoIiwgIiRrZXkIL
CAidXNlc19lcmljLyJdLAogICAgeyJhY2wiOiaicHvibGljLXJ1YWQifSwKICAgIH
sic3VjY2Vzc19hY3Rpb25fcvkaXJ1Y3QiOiaHR0cDovL2pvaG5zbWl0aC5zMy5hb
WF6b25hd3MuY29tL3N1Y2N1c3NmfdWxfdBsb2FkLmh0bWwifSwKICAgIFsic3RhcnRzLXdpdGgiL
CAiJENvbnRlbnQtVHlwZSIisICJpbWFnZS8iXSwKICAgIHsieC1hbXotbWW0YS11dWlkIjogIjE0MzY1MT
IzNjUxMjc0In0sCiAgICBbInN0YXJ0cy13aXRoIiwgIiR4LWFteiltZXRhLXRhZyIsICIiXQo
gIF0KfQo=
```

Using your credentials create a signature, for example `0RavWzkygo6QX9caELEqKi9kDbU=` is the signature for the preceding policy document.

The following form supports a POST request to the `johnsmith.net` bucket that uses this policy.

```
<html>
  <head>
    ...
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    ...
  </head>
  <body>
    ...
    <form action="http://johnsmith.s3.amazonaws.com/" method="post" enctype="multipart/form-data">
      Key to upload: <input type="input" name="key" value="user/eric/" /><br />
      <input type="hidden" name="acl" value="public-read" />
      <input type="hidden" name="success_action_redirect" value="http://johnsmith.s3.amazonaws.com/successful_upload.html" />
      Content-Type: <input type="input" name="Content-Type" value="image/jpeg" /><br />
      <input type="hidden" name="x-amz-meta-uuid" value="14365123651274" />
      Tags for File: <input type="input" name="x-amz-meta-tag" value="" /><br />
      <input type="hidden" name="AWSAccessKeyId" value="AKIAIOSFODNN7EXAMPLE" />
      <input type="hidden" name="Policy" value="POLICY" />
      <input type="hidden" name="Signature" value="SIGNATURE" />
      File: <input type="file" name="file" /> <br />
      <!-- The elements after this will be ignored -->
      <input type="submit" name="submit" value="Upload to Amazon S3" />
    </form>
    ...
  </html>
```

Sample Request

This request assumes that the image uploaded is 117,108 bytes; the image data is not included.

```
POST / HTTP/1.1
Host: johnsmith.s3.amazonaws.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.10)
Gecko/20071115 Firefox/2.0.0.10
```

```
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Content-Type: multipart/form-data; boundary=9431149156168
Content-Length: 118698

--9431149156168
Content-Disposition: form-data; name="key"

user/eric/MyPicture.jpg
--9431149156168
Content-Disposition: form-data; name="acl"

public-read
--9431149156168
Content-Disposition: form-data; name="success_action_redirect"

http://johnsmith.s3.amazonaws.com/successful_upload.html
--9431149156168
Content-Disposition: form-data; name="Content-Type"

image/jpeg
--9431149156168
Content-Disposition: form-data; name="x-amz-meta-uuid"

14365123651274
--9431149156168
Content-Disposition: form-data; name="x-amz-meta-tag"

Some,Tag,For,Picture
--9431149156168
Content-Disposition: form-data; name="AWSAccessKeyId"

AKIAIOSFODNN7EXAMPLE
--9431149156168
Content-Disposition: form-data; name="Policy"

eyAiZXhwaXJhdGvbI6IClYMDA3LTEyLTAxVDEyOjAwOjAwLjAwMFOilAogICJjb25kaXRpb25zI
jogWwogICAgeyJidWNrZXQiOiaiam9obnNtaXRoIn0sCiAgICBbInN0YXJ0cy13aXRoIiwgIiRrZXkiL
CAidXNlc19lcmljLyJdLAogICAgeyJhY2wiOiAicHVibGljLXJlYWQifSwKICAgIH
sic3VjY2Vzc19hY3Rpb25fcmVkaXJlY3QiOiaHR0cDovL2pvaG5zbWl0aC5zMy5hb
WF6b25hd3MuY29tL3N1Y2Nlc3NmdWxfdXBsb2FkLmh0bWwifSwKICAgIFsic3RhcnRzLXdpdGgiL
CAiJENvbnRlbnQtVHlwZSIsICJpbWFnZS8iXSwKICAgIHsieC1hbXotbWW0YS1ldWlkijogIjE0MzY1MT
IzNjUxMjc0In0sCiAgICBbInN0YXJ0cy13aXRoIiwgIiR4LWFteiltZXRhLXRhZyIsICIiXQo
gIF0KfQo=
--9431149156168
Content-Disposition: form-data; name="Signature"

0RavWzkygo6QX9caELEqKi9kDbU=
--9431149156168
Content-Disposition: form-data; name="file"; filename="MyFilename.jpg"
Content-Type: image/jpeg

...file content...
```

```
--9431149156168
Content-Disposition: form-data; name="submit"

Upload to Amazon S3
--9431149156168--
```

Sample Response

```
HTTP/1.1 303 Redirect
x-amz-request-id: 1AEE782442F35865
x-amz-id-2: cxzFLJRaTHy+NGtaDFRR8YvI9BHmgLxjvJzNiGGICARZ/mVXHj7T+qQKhdpzHFh
Content-Type: application/xml
Date: Wed, 14 Nov 2007 21:21:33 GMT
Connection: close
Location: http://johnsmith.s3.amazonaws.com/successful_upload.html?bucket=john
smith&key=user/eric/MyPicture.jpg&etag="39d459df
bc0faabb5e179358dfb94c3"
Server: AmazonS3
```

Text Area Upload

Topics

- [Policy and Form Construction \(p. 580\)](#)
- [Sample Request \(p. 582\)](#)
- [Sample Response \(p. 583\)](#)

The following example shows the complete process for constructing a policy and form to upload a text area. Uploading a text area is useful for submitting user-created content, such as blog postings.

Policy and Form Construction

The following policy supports text area uploads to Amazon S3 for the johnsmith bucket.

```
{ "expiration": "2007-12-01T12:00:00.000Z",
  "conditions": [
    { "bucket": "johnsmith" },
    [ "starts-with", "$key", "user/eric/" ],
    { "acl": "public-read" },
    { "success_action_redirect": "http://johnsmith.s3.amazonaws.com/new_post.html" },
    [ "eq", "$Content-Type", "text/html" ],
    { "x-amz-meta-uuid": "14365123651274" },
    [ "starts-with", "$x-amz-meta-tag", "" ]
  ]
}
```

This policy requires the following:

- The upload must occur before 12:00 GMT on 2007-12-01.
- The content must be uploaded to the johnsmith bucket.
- The key must start with "user/eric/".
- The ACL is set to public-read.
- The success_action_redirect is set to http://johnsmith.s3.amazonaws.com/new_post.html.

- The object is HTML text.
- The x-amz-meta-uuid tag must be set to 14365123651274.
- The x-amz-meta-tag can contain any value.

Following is a Base64-encoded version of this policy.

```
eyAiZXhwaXJhdGlvbiI6IClYMDA3LTEyLTaxVDEyOjAwOjAwLjAwMFoiLAogICJjb25kaXRpb25zIjogWwogICAgeyJidWNrZXQiOiaiam9obnNtaXRoIn0sCiAgICBbInN0YXJ0cy13aXRoiwgIIrRzXkIlCAidXNlcj9lcmljLyJdLAogICAgeyJhY2wiOiaicHVibGljLXJ1YWQifSwKICAgIHsic3VjY2Vzc19hY3RpB25fcmVkaXJ1Y3QiOiaiaHR0cDovL2pvaG5zbW10aC5zMy5hbWF6b25hd3MuY29tL25ld19wb3N0Lmh0bWwifSwKICAgIFsiZXeILCAiJENvbnR1bnQtVHlwZSIisICJ0ZXh0L2h0bWwiXSxKIiCAGIHsieC1hbXotbWW0YS11dWlkIjogIjE0MzY1MTIzNjUxMjc0In0sCiAgICBbInN0YXJ0cy13aXRoiIwgIIr4LWFteiltZXrhLXRhZyIsIClIiXQogIF0KfQo=
```

Using your credentials, create a signature. For example, qA7FWXKq6VvU681I9KdveT1cWgF= is the signature for the preceding policy document.

The following form supports a POST request to the johnsmith.net bucket that uses this policy.

```
<html>
  <head>
    ...
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    ...
  </head>
  <body>
    ...
    <form action="http://johnsmith.s3.amazonaws.com/" method="post" enctype="multipart/form-data">
      Key to upload: <input type="input" name="key" value="user/eric/" /><br />
      <input type="hidden" name="acl" value="public-read" />
      <input type="hidden" name="success_action_redirect" value="http://johnsmith.s3.amazonaws.com/new_post.html" />
      <input type="hidden" name="Content-Type" value="text/html" />
      <input type="hidden" name="x-amz-meta-uuid" value="14365123651274" />
      Tags for File: <input type="input" name="x-amz-meta-tag" value="" /><br />

      <input type="hidden" name="AWSAccessKeyId" value="AKIAIOSFODNN7EXAMPLE" />

      <input type="hidden" name="Policy" value="POLICY" />
      <input type="hidden" name="Signature" value="SIGNATURE" />
      Entry: <textarea name="file" cols="60" rows="10">
        Your blog post goes here.

      </textarea><br />
      <!-- The elements after this will be ignored -->
      <input type="submit" name="submit" value="Upload to Amazon S3" />
    </form>
    ...
  </html>
```

Sample Request

This request assumes that the image uploaded is 117,108 bytes; the image data is not included.

```
POST / HTTP/1.1
Host: johnsmith.s3.amazonaws.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.10)
Gecko/20071115 Firefox/2.0.0.10
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Content-Type: multipart/form-data; boundary=178521717625888
Content-Length: 118635

-178521717625888
Content-Disposition: form-data; name="key"

ser/eric/NewEntry.html
--178521717625888
Content-Disposition: form-data; name="acl"

public-read
--178521717625888
Content-Disposition: form-data; name="success_action_redirect"

http://johnsmith.s3.amazonaws.com/new_post.html
--178521717625888
Content-Disposition: form-data; name="Content-Type"

text/html
--178521717625888
Content-Disposition: form-data; name="x-amz-meta-uuid"

14365123651274
--178521717625888
Content-Disposition: form-data; name="x-amz-meta-tag"

Interesting Post
--178521717625888
Content-Disposition: form-data; name="AWSAccessKeyId"

AKIAIOSFODNN7EXAMPLE
--178521717625888
Content-Disposition: form-data; name="Policy"
eyAiZXhwaxJhdGvbI6IClEyMDA3LTEyLTAxVDEyOjAwOjAwLjAwMFOilAogICJjb25kaXRpb25zI
jogWwogICAgeyJidWNrZXQiOiaiam9obnNtaXRoIn0sCiAgICBbInN0YXJ0cy13aXRoIiwgIiRrZXkIL
CAidXNlcj9lcmljLyJdLAogICAgeyJhY2wiOiAicHVibGljLXJ1YWQifSwKICAgIH
sic3VjY2Vzc19hY3Rpb25fcmVkaXJ1Y3QiOiaHR0cDovL2pvaG5zbW10aC5zMy5hb
WF6b25hd3MuY29tL25ld19wb3N0Lmh0bWwfSwKICAgIFsiZXEilCAiJENvbnRlbnQtVHlwZSis
ICJ0ZXh0L2h0bWwiXSwKICAgIHsieC1hbXotbWV0YS1ldWlkIjogIjE0MzY1MTIzN
jUxMjc0In0sCiAgICBbInN0YXJ0cy13aXRoIiwgIiR4LWFtei1tZXRhLXRhZyIsICIiXQogIF0KfQo=
--178521717625888
Content-Disposition: form-data; name="Signature"
```

```
qA7FWXKq6VvU68lI9KdveTlcWgF=
--178521717625888
Content-Disposition: form-data; name="file"

...content goes here...
--178521717625888
Content-Disposition: form-data; name="submit"

Upload to Amazon S3
--178521717625888--
```

Sample Response

```
HTTP/1.1 303 Redirect
x-amz-request-id: 1AEE782442F35865
x-amz-id-2: cxzFLJRatFHy+NGtaDFRR8YvI9BHmgLxjvJzNiGGICARZ/mVXHj7T+qQKhdpzHFh
Content-Type: application/xml
Date: Wed, 14 Nov 2007 21:21:33 GMT
Connection: close
Location: http://johnsmith.s3.amazonaws.com/new_post.html?bucket=johnsmith&key=user/eric/NewEntry.html&etag=40c3271af26b7f1672e41b8a274d28d4
Server: AmazonS3
```

POST with Adobe Flash

This section describes how to use POST with Adobe Flash.

Adobe Flash Player Security

By default, the Adobe Flash Player security model prohibits Adobe Flash Players from making network connections to servers outside the domain that serves the SWF file.

To override the default, you must upload a publicly readable crossdomain.xml file to the bucket that will accept POST uploads. The following is a sample crossdomain.xml file.

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM
"http://www.macromedia.com/xml/dtds/cross-domain-policy.dtd">
<cross-domain-policy>
<allow-access-from domain="*" secure="false" />
</cross-domain-policy>
```

Note

For more information about the Adobe Flash security model, go to the Adobe website.

Adding the crossdomain.xml file to your bucket allows any Adobe Flash Player to connect to the crossdomain.xml file within your bucket; however, it does not grant access to the actual Amazon S3 bucket.

Adobe Flash Considerations

The FileReference API in Adobe Flash adds the *Filename* form field to the POST request. When you build Adobe Flash applications that upload to Amazon S3 by using the FileReference API action, include the following condition in your policy:

```
[ 'starts-with', '$Filename', '' ]
```

Some versions of the Adobe Flash Player do not properly handle HTTP responses that have an empty body. To configure POST to return a response that does not have an empty body, set *success_action_status* to 201. Amazon S3 will then return an XML document with a 201 status code. For information about the content of the XML document, see [POST Object](#). For information about form fields, see [HTML Form Fields \(p. 571\)](#).

Amazon S3 Resources

Following is a table that lists related resources that you'll find useful as you work with this service.

Resource	Description
Amazon Simple Storage Service Getting Started Guide	The Getting Started Guide provides a quick tutorial of the service based on a simple use case.
Amazon Simple Storage Service API Reference	The API Reference describes Amazon S3 operations in detail.
Amazon S3 Technical FAQ	The FAQ covers the top questions developers have asked about this product.
Amazon S3 Release Notes	The Release Notes give a high-level overview of the current release. They specifically note any new features, corrections, and known issues.
AWS Developer Resource Center	A central starting point to find documentation, code samples, release notes, and other information to help you build innovative applications with AWS.
AWS Management Console	The console allows you to perform most of the functions of Amazon S3 without programming.
https://forums.aws.amazon.com/	A community-based forum for developers to discuss technical questions related to AWS.
AWS Support Center	The home page for AWS Technical Support, including access to our Developer Forums, Technical FAQs, Service Status page, and Premium Support.
AWS Premium Support	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.
Amazon S3 product information	The primary web page for information about Amazon S3.
Contact Us	A central contact point for inquiries concerning AWS billing, account, events, abuse etc.

Resource	Description
Conditions of Use	Detailed information about the copyright and trademark usage at Amazon.com and other topics.

Document History

The following table describes the important changes since the last release of the *Amazon Simple Storage Service Developer Guide*.

Relevant Dates to this History:

- **Current product version:** 2006-03-01
- **Last documentation update:** April 9, 2015

Change	Description	Date
Event notifications	Amazon S3 event notifications have been updated to support the switch to resource-based permissions for AWS Lambda functions. For more information, see Configuring Amazon S3 Event Notifications (p. 472) .	In this release
Cross-region replication	Amazon S3 now supports cross-region replication. Cross-region replication is the automatic, asynchronous copying of objects across buckets in different AWS regions. For more information, see Cross-Region Replication (p. 487) .	March 24, 2015
Event notifications	Amazon S3 now supports new event types and destinations in a bucket notification configuration. Prior to this release, Amazon S3 supported only the <code>s3:ReducedRedundancyLostObject</code> event type and an Amazon SNS topic as the destination. For more information about the new event types, see Configuring Amazon S3 Event Notifications (p. 472) .	November 13, 2014
Server-side encryption with customer-provided encryption keys	Server-side encryption with AWS Key Management Service (KMS) Amazon S3 now supports server-side encryption using AWS Key Management Service. This feature allows you to manage the envelope key through KMS, and Amazon S3 calls KMS to access the envelope key within the permissions you set. For more information about server-side encryption with KMS, see Protecting Data Using Server-Side Encryption with AWS Key Management Service .	November 12, 2014

Change	Description	Date
EU (Frankfurt) region	Amazon S3 is now available in the EU (Frankfurt) region.	October 23, 2014
Server-side encryption with customer-provided encryption keys	<p>Amazon S3 now supports server-side encryption using customer-provided encryption keys (SSE-C). Server-side encryption enables you to request Amazon S3 to encrypt your data at rest. When using SSE-C, Amazon S3 encrypts your objects with the custom encryption keys that you provide. Since Amazon S3 performs the encryption for you, you get the benefits of using your own encryption keys without the cost of writing or executing your own encryption code.</p> <p>For more information about SSE-C, see Server-Side Encryption (Using Customer-Provided Encryption Keys).</p>	June 12, 2014
Lifecycle support for versioning	Prior to this release, lifecycle configuration was supported only on nonversioned buckets. Now you can configure lifecycle on both nonversioned and versioning-enabled buckets. For more information, see Object Lifecycle Management (p. 86) .	May 20, 2014
Access control topics revised	Revised Amazon S3 access control documentation. For more information, see Managing Access Permissions to Your Amazon S3 Resources (p. 269) .	April 15, 2014
Server access logging topic revised	Revised server access logging documentation. For more information, see Server Access Logging (p. 530) .	November 26, 2013
.NET SDK samples updated to version 2.0	.NET SDK samples in this guide are now compliant to version 2.0.	November 26, 2013
SOAP Support Over HTTP Deprecated	SOAP support over HTTP is deprecated, but it is still available over HTTPS. New Amazon S3 features will not be supported for SOAP. We recommend that you use either the REST API or the AWS SDKs.	September 20, 2013
IAM policy variable support	<p>The IAM access policy language now supports variables. When a policy is evaluated, any policy variables are replaced with values that are supplied by context-based information from the authenticated user's session. You can use policy variables to define general purpose policies without explicitly listing all the components of the policy. For more information about policy variables, go to Policy Variables in the <i>AWS Identity and Access Management Using IAM</i> guide.</p> <p>For examples of policy variables in Amazon S3, see User Policy Examples (p. 342).</p>	April 3, 2013
Console support for Requester Pays	You can now configure your bucket for Requester Pays by using the Amazon S3 console. For more information, see Configure Requester Pays by Using the Amazon S3 Console (p. 73) .	December 31, 2012

Change	Description	Date
Root domain support for website hosting	<p>Amazon S3 now supports hosting static websites at the root domain. Visitors to your website can access your site from their browser without specifying "www" in the web address (e.g., "example.com"). Many customers already host static websites on Amazon S3 that are accessible from a "www" subdomain (e.g., "www.example.com"). Previously, to support root domain access, you needed to run your own web server to proxy root domain requests from browsers to your website on Amazon S3. Running a web server to proxy requests introduces additional costs, operational burden, and another potential point of failure. Now, you can take advantage of the high availability and durability of Amazon S3 for both "www" and root domain addresses. For more information, see Hosting a Static Website on Amazon S3 (p. 448).</p>	December 27, 2012
Console revision	<p>Amazon S3 console has been updated. The documentation topics that refer to the console have been revised accordingly.</p>	December 14, 2012
Support for Archiving Data to Amazon Glacier	<p>Amazon S3 now support a storage option that enables you to utilize Amazon Glacier's low-cost storage service for data archival. To archive objects, you define archival rules identifying objects and a timeline when you want Amazon S3 to archive these objects to Amazon Glacier. You can easily set the rules on a bucket using the Amazon S3 console or programmatically using the Amazon S3 API or AWS SDKs.</p> <p>For more information, see Object Lifecycle Management (p. 86).</p>	November 13, 2012
Support for Website Page Redirects	<p>For a bucket that is configured as a website, Amazon S3 now supports redirecting a request for an object to another object in the same bucket or to an external URL. For more information, see Configuring a Web Page Redirect (p. 460).</p> <p>For information about hosting websites, see Hosting a Static Website on Amazon S3 (p. 448).</p>	October 4, 2012
Support for Cross-Origin Resource Sharing (CORS)	<p>Amazon S3 now supports Cross-Origin Resource Sharing (CORS). CORS defines a way in which client web applications that are loaded in one domain can interact with or access resources in a different domain. With CORS support in Amazon S3, you can build rich client-side web applications on top of Amazon S3 and selectively allow cross-domain access to your Amazon S3 resources. For more information, see Enabling Cross-Origin Resource Sharing (p. 110).</p>	August 31, 2012
Support for Cost Allocation Tags	<p>Amazon S3 now supports cost allocation tagging, which allows you to label S3 buckets so you can more easily track their cost against projects or other criteria. For more information about using tagging for buckets, see Cost Allocation Tagging (p. 77).</p>	August 21, 2012

Change	Description	Date
Support for MFA-protected API access in bucket policies	<p>Amazon S3 now supports MFA-protected API access, a feature that can enforce AWS Multi-Factor Authentication for an extra level of security when accessing your Amazon S3 resources. It is a security feature that requires users to prove physical possession of an MFA device by providing a valid MFA code. For more information, go to AWS Multi-Factor Authentication. You can now require MFA authentication for any requests to access your Amazon S3 resources.</p> <p>To enforce MFA authentication, Amazon S3 now supports the <code>aws:MultiFactorAuthAge</code> key in a bucket policy. For an example bucket policy, see Adding a Bucket Policy to Require MFA Authentication (p. 339).</p>	July 10, 2012
Object Expiration support	You can use Object Expiration to schedule automatic removal of data after a configured time period. You set object expiration by adding lifecycle configuration to a bucket. For more information, see Object Expiration (p. 98) .	27 December 2011
New region supported	Amazon S3 now supports the South America (Sao Paulo) region. For more information, see Accessing a Bucket (p. 57) .	December 14, 2011
Multi-Object Delete	Amazon S3 now supports Multi-Object Delete API that enables you to delete multiple objects in a single request. With this feature, you can remove large numbers of objects from Amazon S3 more quickly than using multiple individual DELETE requests. For more information, see Deleting Objects (p. 228) .	December 7, 2011
New region supported	Amazon S3 now supports the US West (Oregon) region. For more information, see Buckets and Regions (p. 57) .	November 8, 2011
Documentation Update	Documentation bug fixes.	November 8, 2011
Documentation Update	<p>In addition to documentation bug fixes, this release includes the following enhancements:</p> <ul style="list-style-type: none"> <li data-bbox="605 1305 1274 1453">• New server-side encryption sections using the AWS SDK for PHP (see Specifying Server-Side Encryption Using the AWS SDK for PHP (p. 390)) and the AWS SDK for Ruby (see Specifying Server-Side Encryption Using the AWS SDK for Ruby (p. 392)). <li data-bbox="605 1463 1274 1522">• New section on creating and testing Ruby samples (see Using the AWS SDK for Ruby (p. 549)). 	October 17, 2011
Server-side encryption support	Amazon S3 now supports server-side encryption. It enables you to request Amazon S3 to encrypt your data at rest, that is, encrypt your object data when Amazon S3 writes your data to disks in its data centers. In addition to REST API updates, the AWS SDK for Java and .NET provide necessary functionality to request server-side encryption. You can also request server-side encryption when uploading objects using AWS Management Console. To learn more about data encryption, go to Using Data Encryption .	October 4, 2011

Change	Description	Date
Documentation Update	<p>In addition to documentation bug fixes, this release includes the following enhancements:</p> <ul style="list-style-type: none"> • Added Ruby and PHP samples to the Making Requests (p. 11) section. • Added sections describing how to generate and use pre-signed URLs. For more information, see Share an Object with Others (p. 139) and Uploading Objects Using Pre-Signed URLs (p. 191). • Updated an existing section to introduce AWS Explorers for Eclipse and Visual Studio. For more information, see Using the AWS SDKs and Explorers (p. 542). 	September 22, 2011
Support for sending requests using temporary security credentials	<p>In addition to using your AWS account and IAM user security credentials to send authenticated requests to Amazon S3, you can now send requests using temporary security credentials you obtain from AWS Identity and Access Management (IAM). You can use the AWS Security Token Service API or the AWS SDK wrapper libraries to request these temporary credentials from IAM. You can request these temporary security credentials for your own use or hand them out to federated users and applications. This feature enables you to manage your users outside AWS and provide them with temporary security credentials to access your AWS resources.</p> <p>For more information, see Making Requests (p. 11).</p> <p>For more information about IAM support for temporary security credentials, go to Granting Temporary Access to Your AWS Resources.</p>	August 3, 2011
Multipart Upload API extended to enable copying objects up to 5 TB	<p>Prior to this release, Amazon S3 API supported copying objects of up to 5 GB in size. To enable copying objects larger than 5 GB, Amazon S3 now extends the multipart upload API with a new operation, <code>Upload Part (Copy)</code>. You can use this multipart upload operation to copy objects up to 5 TB in size. For more information, see Copying Objects (p. 199).</p> <p>For conceptual information about multipart upload API, see Uploading Objects Using Multipart Upload API (p. 155).</p>	June 21, 2011
SOAP API calls over HTTP disabled	To increase security, SOAP API calls over HTTP are disabled. Authenticated and anonymous SOAP requests must be sent to Amazon S3 using SSL.	June 6, 2011

Change	Description	Date
IAM enables cross-account delegation	<p>Previously, to access an Amazon S3 resource, an IAM user needed permissions from both the parent AWS account and the Amazon S3 resource owner. With cross-account access, the IAM user now only needs permission from the owner account. That is, If a resource owner grants access to an AWS account, the AWS account can now grant its IAM users access to these resources.</p> <p>For more information, go to Enabling Cross-Account Access in <i>Using Identity and Access Management</i>.</p> <p>For more information on specifying principals in a bucket policy, see Specifying a Principal in a Policy (p. 314).</p>	June 6, 2011
New link	This service's endpoint information is now located in the AWS General Reference. For more information, go to Regions and Endpoints in AWS General Reference .	March 1, 2011
Support for hosting static websites in Amazon S3	Amazon S3 introduces enhanced support for hosting static websites. This includes support for index documents and custom error documents. When using these features, requests to the root of your bucket or a subfolder (e.g., <code>http://mywebsite.com/subfolder</code>) returns your index document instead of the list of objects in your bucket. If an error is encountered, Amazon S3 returns your custom error message instead of an Amazon S3 error message. For more information, see Hosting a Static Website on Amazon S3 (p. 448) .	February 17, 2011
Response Header API Support	The GET Object REST API now allows you to change the response headers of the REST GET Object request for each request. That is, you can alter object metadata in the response, without altering the object itself. For more information, see Getting Objects (p. 130) .	January 14, 2011
Large object support	Amazon S3 has increased the maximum size of an object you can store in an S3 bucket from 5 GB to 5 TB. If you are using the REST API you can upload objects of up to 5 GB size in a single PUT operation. For larger objects, you must use the Multipart Upload REST API to upload objects in parts. For more information, see Uploading Objects Using Multipart Upload API (p. 155) .	December 9, 2010
Multipart upload	Multipart upload enables faster, more flexible uploads into Amazon S3. It allows you to upload a single object as a set of parts. For more information, see Uploading Objects Using Multipart Upload API (p. 155) .	November 10, 2010
Canonical ID support in bucket policies	You can now specify canonical IDs in bucket policies. For more information, see Access Policy Language Overview (p. 312)	September 17, 2010
Amazon S3 works with IAM	This service now integrates with AWS Identity and Access Management (IAM). For more information, go to Integrating with Other AWS Products in <i>Using AWS Identity and Access Management</i> .	September 2, 2010

Change	Description	Date
Notifications	The Amazon S3 notifications feature enables you to configure a bucket so that Amazon S3 publishes a message to an Amazon Simple Notification Service (Amazon SNS) topic when Amazon S3 detects a key event on a bucket. For more information, see Setting Up Notification of Bucket Events (p. 472) .	July 14, 2010
Bucket policies	Bucket policies is an access management system you use to set access permissions across buckets, objects, and sets of objects. This functionality supplements and in many cases replaces access control lists. For more information, see Using Bucket Policies and User Policies (p. 312) .	July 6, 2010
Path-style syntax available in all regions	Amazon S3 now supports the path-style syntax for any bucket in the US Classic Region, or if the bucket is in the same region as the endpoint of the request. For more information, see Virtual Hosting (p. 48) .	June 9, 2010
New endpoint for EU (Ireland)	Amazon S3 now provides an endpoint for EU (Ireland): <code>http://s3-eu-west-1.amazonaws.com</code> .	June 9, 2010
Console	You can now use Amazon S3 through the AWS Management Console. You can read about all of the Amazon S3 functionality in the console in the <i>Amazon Simple Storage Service Console User Guide</i> .	June 9, 2010
Reduced Redundancy	Amazon S3 now enables you to reduce your storage costs by storing objects in Amazon S3 with reduced redundancy. For more information, see Reduced Redundancy Storage (p. 7) .	May 12, 2010
New region supported	Amazon S3 now supports the Asia Pacific (Singapore) region. For more information, see Buckets and Regions (p. 57) .	April 28, 2010
Object Versioning	This release introduces object versioning. All objects now can have a key and a version. If you enable versioning for a bucket, Amazon S3 gives all objects added to a bucket a unique version ID. This feature enables you to recover from unintended overwrites and deletions. For more information, see Versioning (p. 8) and Using Versioning (p. 422) .	February 8, 2010
New region supported	Amazon S3 now supports the US-West (Northern California) region. The new endpoint for requests to this Region is <code>s3-us-west-1.amazonaws.com</code> . For more information, see Buckets and Regions (p. 57) .	December 2, 2009
AWS SDK for .NET	AWS now provides libraries, sample code, tutorials, and other resources for software developers who prefer to build applications using .NET language-specific APIs instead of REST or SOAP. These libraries provide basic functions (not included in the REST or SOAP APIs), such as request authentication, request retries, and error handling so that it's easier to get started. For more information about language-specific libraries and resources, see Using the AWS SDKs and Explorers (p. 542) .	November 11, 2009

AWS Glossary

For the latest AWS terminology, see the [AWS Glossary](#) in the *AWS General Reference*.